A Hierarchical Approach to Classification For Systems with Complex Low-Level Interactions

Ricardo Vilalta and Muralikrishna Achari
Department of Computer Science
University of Houston
4800 Calhoun Rd., Houston TX 77204
E-mail: {vilalta,amkchari}@cs.uh.edu

***Abstract*— Learning in multiple steps or layers is useful when the system under study is characterized by the complex interaction of low level components. In this case it is convenient to decompose the classification problem into different layers of complexity, starting at the bottom with all low-level features, and progressing to the top through the construction of more abstract terms. In this paper we propose a hierarchical approach to classification where each layer is an attempt to improve the predictive accuracy of our classifier through the construction of new terms. We perform an experimental study of this algorithm in eighteen real-world domains; a comparison with decision trees denotes an advantage in predictive accuracy, especially when the complexity of the domain requires the construction of multiple hierarchical layers.**

I. Introduction

Classification is prone to failure when one attempts to figure the relations among the low-level components of a complex system in a single pass. A more successful approach consist of dividing the classification process into several steps, each step unveiling part of the complex system relations. How much we should simplify the classification problem varies according to the quality of the feature set characterizing the system under study. On the one side, high quality features convey much information about the system; in this case, few steps suffice to produce good results. In contrast, low quality features must be combined with many other features to unveil system structure; here features can interact in many ways and multiple steps are needed to discover important feature combinations. As an example, predicting traffic patterns, signaling a fault in a communications network, or selecting the next best action in a robotic environment is hard because the features characterizing those systems interact in non-obvious ways.

Since the feature representation is typically fixed a priori, how can we develop new classification algorithms that are designed to modify the quality of the representation automatically? We answer this question by introducing a form of hierarchical concept learning where a hypothesis to the target function is built through multiple layers of increasing complexity. Each layer is made of partial terms that can be reused by upper layers; a partial term constitutes a new constructed feature that is integrated into the pool of previous features to facilitate the representation of more complex hypotheses. Each layer is associated to a hypothesis of the target function and the construction of new layers continues until the estimated accuracy of the most recent hypothesis starts to decrease. We provide an empirical study of our proposed algorithm over eighteen real-world domains.

Recent work has shown the advantages of learning in multiple steps or layers, by acquiring the ability to perform low-level tasks before striving for more complex tasks. Layered learning has proved useful in areas like robotic soccer [17] and for acquiring concepts from a stream of unstructured information [19]. Our approach is similar to layered learning in that each hierarchical layer is fully learned before attempting to create a new layer. This contrasts with methods that learn the parameters of a pre-defined hierarchical representation simultaneously (e.g., backpropagation in neural networks). In layered learning, however, the task decomposition of a problem is assumed a priori whereas in our case a hierarchy of partial terms is built automatically. Finally, the discovery of partial terms during learning is key to understanding the nature of a target function, and provides us with useful pieces of knowledge that can be reused if placed in a form of lifelong learning [18].

This paper is organized as follows. Section II introduces definitions and notation. Section III details the mechanism of our hierarchical learner. Section IV discusses related work. Section V describes our empirical analysis. Lastly, Section VI provides our conclusions and discussion.

II. Preliminaries

Let $\mathbf{A} = (A_1, A_2, \cdots, A_n)$ be an n-component vector-valued random variable, where each A_i represents an attribute or feature. The space of all possible feature vectors is called the input space $\mathcal{X}$. A classifier receives as input a set of training examples $T = \{(\mathbf{x}, y)\}$, where $\mathbf{x} = (a_1, a_2, \cdots, a_n)$ is an instance of the input space and $y \in \{0, 1\}$ is an instance of the output space (for simplicity we consider the two-class problem exclusively). The outcome of the classifier is a function h (or hypothesis) mapping the input space to the output space, $h : \mathcal{X} \rightarrow \mathcal{Y}$. Function h can then be used to predict the class of previously unseen feature vectors.

Learning Hierarchical Structures

Rather than learning hypothesis h in one step, we advocate the construction of many intermediate steps before the right degree of complexity is determined. We approximate a target function in a hierarchical fashion, by building a structure in

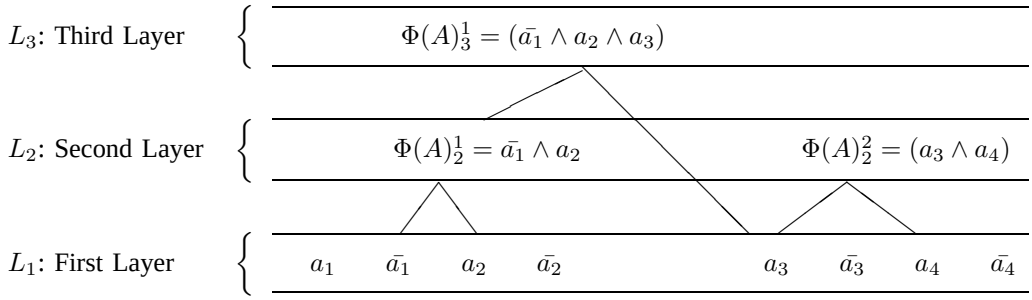

Fig. 1. A hierarchy of original and constructed features; each layer can be used to approximate concept $f = (\bar{a_1} \wedge a_2) \vee (a_3 \wedge a_4)$. Layer L_2 has the highest quality terms.

which lower layers denote the most simple relations, and upper layers denote increasingly more complex relations. We introduce the following notation:

- The hierarchical structure is made of a sequence of layers $\{L_1, L_2, \cdots, L_k\}$ of increasing complexity.
- Each layer L_i can be built by using the set of original features $\mathbf{A}$ and all constructed features from previous layers.
- We denote as $\{\Phi(\mathbf{A})_i^j\}$ the set of constructed features at layer L_i.
- Each layer L_i is associated to a hypothesis h_i.

As an illustration, consider a target function expressed as the logical combination of Boolean features. For example, the DNF expression $f = (\bar{a_1} \wedge a_2) \vee (a_3 \wedge a_4)$ can be formed by first finding the terms $(\bar{a_1} \wedge a_2)$ and $(a_3 \wedge a_4)$, followed by applying logical disjunction (we use the symbol $\wedge$ to mean logical conjunction and $\vee$ to mean logical disjunction). Figure 1 shows a hierarchical structure in which original features —and their negations— occupy the first layer (L_1), and more complex terms intermediate layers (L_2 and L_3). Layer L_2 embeds high quality terms and is closest in structure to f. Layer L_3 overspecializes the first term in f and misses the second term. The goal of our approach is to build enough hierarchical layers until a layer can be used to construct a hypothesis that is close to the target function. We explain how to build new terms and associate a hypothesis to each layer next.

III. The Hierarchical Concept Learner

The general strategy behind our hierarchical concept learner, for short HCL, is as follows[1]. At each iteration HCL builds a layer of new terms L_i and a hypothesis h_i. HCL estimates the accuracy of h_i, denoted as $\lambda(h_i)$, and compares it to the one corresponding to the level below, $\lambda(h_{i-1})$ (evidently we skip this comparison for the first layer). If $\lambda(h_i) > \lambda(h_{i-1})$, then each of the newly constructed terms in L_i is incorporated into the set of previously defined terms to serve as operands for the construction of new layers. If $\lambda(h_i) \leq \lambda(h_{i-1})$ the cycle stops and the algorithm outputs h_{i-1} as the final hypothesis.

[1]A detailed account of the algorithm is provided by [20].

Algorithm 1: The HCL Algorithm
Input: Training set T
Output: Final Hypothesis H
HCL(T)
(1) $L_1 \leftarrow$ original features
(2) $h_1 \leftarrow$ BuildHypothesis(L_1, T)
(3) Operands $\leftarrow L_1$
(4) $i = 2$
(5) **while** (**true**)
(6) $L_i \leftarrow$ NewTerms(Operands,T)
(7) $h_i \leftarrow$ BuildHypothesis(L_i, T)
(8) **if** ($\lambda(h_i) \leq \lambda(h_{i-1})$)
(9) Exit-Loop
(10) Operands $\leftarrow$ Operands $\cup L_i$
(11) **end while**
(12) **return** h_{i-1}

Fig. 2. The HCL algorithm.

An overview of the algorithm is shown in Figure 2. Previous constructed terms and original features are kept in memory (line 10), and used for the construction of new terms (line 6). Section III-A explains the mechanism behind the construction of new terms and Section III-B explains the construction of each new hypothesis (lines 2 and 7). We measure the predictive accuracy of each hypothesis (i.e., $\lambda(h_i)$) using a resampling technique known as stratified 10-fold cross validation [9]. HCL stops generating more hypotheses when a decrease in accuracy is detected, an indication of data overfitting (line 8).

Since the number of hierarchical layers is automatically adjusted according to the characteristics displayed by the system under study, our algorithm is *adaptive*. The adaptation process is based on a search for a maximum on a function that maps hypotheses (ordered by increasing complexity) against predictive accuracy.

A. Constructing A New Term

We first explain how to construct a single new term. We assume our system characterization is made of Boolean features. This is attained by dividing numeric features into intervals [3], and by mapping each nominal value into a Boolean feature. A new constructed term is a logical conjunction of Boolean

Algorithm 2: Search For a New Term
Input: Operands O, Training set T
Output: Best term F_{best}
NEWTERM(O, T)
(1) $I_{\text{original}} \leftarrow$ set of literals from O
(2) $I_{\text{beam}} \leftarrow$ best α literals in I_{original}
(3) **while** (**true**)
(4) $I_{\text{new}} \leftarrow$ Systematically form the conjunction
(5) of every $x \in I_{\text{beam}}$ with every $y \in I_{\text{original}}$
(6) Apply pruning into I_{new}
(7) **if** $I_{\text{new}} = \emptyset$
(8) $\Phi_{\text{best}} \leftarrow$ global best combination
(9) Exit-Loop
(10) $I_{\text{beam}} \leftarrow$ best α combinations in I_{new}
(11) **end while**
(12) **return** Φ_{best}

Fig. 3. The search for a new term.

features or their complements, also known as a monomial (e.g., $\bar{a_1} \wedge a_2$).

The algorithm conducts a beam search over the space of all monomials. The approach works as follows. Assume we are positioned at hierarchical layer L_i and have available the set of all original features and previously constructed terms from lower layers $\{L_1, L_2, \cdots, L_{i-1}\}$. Each of these terms is considered a Boolean feature. The algorithm explores the space of all logical combinations that can be obtained from these Boolean features and their negations; the output of the algorithm is a single combination with the best overall ranking according to an information-theoretic metric M. In our experiments we used Gain Ratio [10] but other metrics could potentially be used (e.g., gini, χ^2, Laplace).

Figure 3 outlines the general mechanism to construct a new term. A list of literals is created by forming a set of Boolean features (original and previously constructed terms) and their negations (line 1). The algorithm evaluates each of these literals keeping only the α best according to metric M (line 2; α is the size of the beam); these become now the best current retained terms. The space of monomials is explored by adding one literal at a time to the best previously retained terms in the beam (lines 4-5). Adding literals extends the depth of the search; retaining the best terms limits the width of the search. The process continues while keeping track of the best term. To avoid exploring all possible combinations, the size of the search space can be constrained (line 6) using known pruning techniques [15], [21], [11]. The search terminates when no more terms are left on the beam, at which point the algorithm outputs the best term.

B. Constructing A New Hypothesis

Assume we are at level L_i of our hierarchical structure, and have constructed our first term $\Phi(A)_i^1$. Once the first term is constructed we face two problems: 1) how to generate more terms substantially different from the rest, but relevant to the target function, and 2) how to combine terms to create a hypothesis h_i. Our approach to solve both problems is to apply a resampling technique over the training set known as boosting [6], [5], which we explain next.

After the first term has been constructed, we generate a variant or replicate of the original training set T by randomly sampling with replacement examples from T. Let's call T' the new replicate. The sampling is generated under a new distribution that gives more attention to those examples incorrectly classified by the most recent term. Thus after generating the first term $\Phi(A)_i^1$, the algorithm builds a new training set T' where those examples incorrectly classified by $\Phi(A)_i^1$ have higher chance of being selected. A new term $\Phi(A)_i^2$ is obtained by invoking the module to construct new terms (Figure 3; Algorithm 2) over T'. As the process continues we end up with a set of new terms for layer L_i; a new hypothesis h_i is simply a weighted linear combination of the set of new terms:

$$h_i = \sum_j w_j \Phi(A)_i^j + w_0 \quad (1)$$

where each weight w_j is proportional to the accuracy of its corresponding term $\Phi(A)_i^j$, and $w_0 = -(0.5 \cdot \sum_i w_i)$. The class predicted by h_i is either true if $h_i \geq 0$ or false if $h_i < 0$.

Figure 4 explains the mechanism to construct a new hypotheses h_i. A new term is obtained by invoking Algorithm 2 over the replicate training set (line 4); a new distribution is generated by giving higher probability to those examples misclassified by the new term (line 8). The process continues until the error is below a predefined value epsilon (line 6), after which the algorithm outputs a linear combination of all newly constructed terms.

Our mechanism to construct a new hypothesis has the advantage of using all available examples when looking for a diverse set of new terms. This contrasts to divide-and-conquer methods (e.g, decision trees) where the continuous splitting of the data reduces the statistical support of new terms. After a hypothesis is constructed, each new term is added into a pool of operands (Figure 2) and can potentially be used to construct new terms at upper layers of the hierarchical structure.

IV. RELATED WORK

We now compare our approach with previous work. One of the first attempts to dynamically search for the best structure of a hierarchical framework is the cascade-correlation architecture [4], where new hidden units are integrated to the connectionist network to maximize the correlation between the new unit's output and the error observed at each network-output unit. A different approach is to adjust the topology of the neural network dynamically by varying the number of hidden units using back-propagation [8]. One may also use evolutionary computation, [13], [14] to increase the size of a neural network by adding new hidden units, each competing to attain best training performance. These systems are similar to HCL in that new units are added to the network dynamically, and the process continues until enough error reduction is observed. HCL is different in that each layer represents a

Algorithm 3: Constructing a New Hypothesis
Input: Training set T
Output: Hypothesis h_i
NEWHYPOTHESIS(T)
(1) $D_1 \leftarrow$ uniform distribution in T
(2) **while** (**true**)
(3) $T' \leftarrow$ replicate from T according to D_i
(4) $\Phi(A)_i^j \leftarrow$ Apply NEWTERM to T'
(5) $w_i \leftarrow$ Error of Φ_i^j on T'
(6) **if** $w_i <$ epsilon
(7) Exit-Loop
(8) $D_i \leftarrow$ new distribution based on Φ_i^j
(9) **end while**
(10) $h_i \leftarrow \sum_j w_j \Phi(A)_i^j + w_0$
(11) **return** h_i

Fig. 4. Constructing a new hypothesis.

different hypothesis, and the estimation of the true error is done via a resampling technique (i.e., via cross-validation).

A hierarchical structure has been used in symbolic methods as well, by using partial knowledge to build a hierarchical knowledge base [7], or by searching for information compression [12]. HCL differs in its adaptive mechanism that looks for several hypotheses of increasingly complexity.

Feature construction using a hierarchical representation has been used in learning failure diagnosis models for robotic operations [16]. The mechanism learns increasingly more complex concepts using domain-dependent feature constructive techniques; this contrasts with HCL where feature construction is domain independent.

HCL is also related to layered learning [17], [19] where acquiring the ability to perform low-level tasks is used to strive for more complex tasks. In layered learning, however, the hierarchical decomposition of a problem is assumed before learning starts, whereas in our case the hierarchical structure is built automatically.

V. EXPERIMENTS

We now report on an experimental study comparing the predictive accuracy of HCL with a standard decision-tree algorithm. The decision tree, C4.5, is described by [10]. For each dataset, our experiments use 50% of the available examples for training and the rest for testing. Each reported value is the average over 50 runs. Real-world domains can be obtained from the UCI repository [2], except for the star-cluster domain, (obtained from [1]). Because of HCL's expensive mechanism, an upper bound on the size of the training set was set to 1000 examples. Runs were performed on a Sparc 10.

A. Predictive Accuracy

Table 1 displays our results. The first column describes the domains used for our experiments. The second and third columns report on the accuracy of HCL and C4.5 respectively (numbers enclosed in parentheses represent standard deviations). To obtain a clear view of the difference in accuracy between both methods and its relation to the number of hierarchical layers produced by HCL, we took the results in Table 1 and plot them in Figure 5. The x-axis shows the average number of hierarchical levels for the best hypothesis output by HCL. The y-axis shows the difference in predictive accuracy between HCL and C4.5. As shown in Figure 5, below three hierarchical layers the performance of HCL and C4.5 is relatively the same. Within the range of $[3, 4.3]$ layers the results are mixed. We notice there is a loss in accuracy for domains where noise or representation inadequacy are present (e.g., bankruptcy). For other domains the advantage of HCL is significant (e.g., lymphography, promoters). Above 4.3 hierarchical layers HCL tends to dominate; the corresponding domains are characterized precisely by low-level features that interact in complex ways (e.g., board games).

TABLE I
PREDICTIVE ACCURACY ON REAL-WORLD DOMAINS FOR HCL AND DECISION TREE C4.5. NUMBERS ENCLOSED IN PARENTHESES REPRESENT STANDARD DEVIATIONS.

Domain	HCL	Decision Tree
Bankruptcy	64.63 (5.00)	68.86 (5.59)
Bupa	61.76 (3.38)	63.06 (4.37)
Cancer	95.28 (0.99)	95.00 (1.27)
Credit	69.70 (5.05)	71.29 (4.45)
Diabetes	72.20 (2.11)	72.97 (2.19)
Heart	75.86 (3.63)	73.97 (3.69)
Hepatitis	78.75 (4.78)	79.25 (4.88)
Ionosphere	90.40 (2.23)	89.63 (2.06)
Kr-vs-Kp	99.11 (0.23)	97.25 (0.85)
Lymphography2	80.56 (4.28)	78.45 (4.95)
Lymphography3	81.45 (4.66)	77.18 (4.80)
Mushroom	99.54 (0.40)	99.59 (0.35)
New-thyroid-hyper	96.29 (2.01)	95.76 (1.81)
New-thyroid-hypo	95.96 (1.94)	96.07 (1.39)
Promoters	79.39 (5.24)	74.08 (4.52)
Star-cluster	96.38 (2.29)	95.98 (2.09)
TicTacToe	93.51 (1.61)	82.06 (2.58)
Zoo	95.67 (2.74)	94.2 (2.74)

Our approach shows a clear advantage in performance when compared to a decision tree, especially when the complexity of the domain requires the construction of multiple layers of new terms. Nevertheless, HCL appears sensitive to noise and did not show the same improvement over algorithms better suited for noisy domains (e.g., C4.5 rules and neural networks). Clearly our hierarchical design is still in need of refinements; we discuss limitations of our approach and future work in the next section.

VI. CONCLUSIONS AND DISCUSSION

We propose a new hierarchical approach to classification where each layer of the hierarchical structure is associated with a hypothesis to the target function; higher layers correspond to more complex hypotheses. We claim this approach can be extremely valuable in control systems when the features describing the low-level components interact in complex ways. Our results show significant performance improvement when the number of hierarchical layers is large (above 4-5 layers).

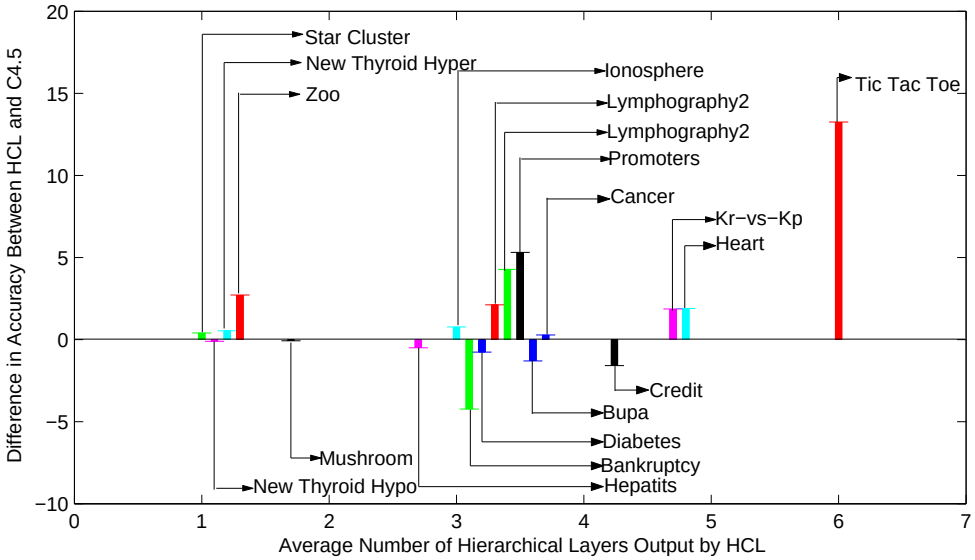

Fig. 5. A comparison of HCL and C4.5 trees.

Our approach is not exempt from limitations. First, the transition from one hypothesis, h_i, to the next upper layer, h_{i+1}, may be too coarse. In other words, it is possible that the right complexity in the representation lies somewhere between h_i and h_{i+1}, in which case we would fail to produce an accurate classifier. Second, every hypothesis is a linear combination of logical terms; such family of hypotheses is not very expressive (i.e., denotes low bias). In contrast, those same terms could be used to construct a more sophisticated algorithm at each hierarchical level, which may result in better levels of predictive accuracy. Third, our approach appears sensitive to noise; we plan to address this through the design of efficient pruning techniques.

As future work we plan to extend our approach to become a general framework for feature construction. The idea is to use the hierarchical structure to represent new terms, and to let any classification algorithm improve its predictions by using that particular layer that results in best performance. Each layer is in fact augmenting the dimensionality of the feature space; classification algorithms can then choose the representation that best fits their inductive bias.

ACKNOWLEDGMENT

We are grateful to Larry Rendell for his valuable suggestions. This work was supported by a scholarship from CONACYT, by Fulbright foundation, and by the University of Houston.

REFERENCES

[1] D.F Andrews and A.M. Herzberg: "Data: A Collection of Problems from Many Fields for the Student and Research Worker". Springer-Verlag (1985).

[2] C.L. Blake C.J. Merz: "UCI, Repository of Machine Learning Databases". University of California, Irvine, Dept. of Information and Computer Sciences. http://www.ics.uci.edu/∼mlearn/MLRepository.html (1998).

[3] J. Catlett: "On Changing Continuous Attributes Into Ordered Discrete Attributes". Proceedings of the European Conference on Machine Learning pp. 164-178. Springer-Verlag (1998).

[4] S. E. Fahlman and C. Lebiere: "The Cascade-Correlation Learning Architecture". Advances in Neural Information Processing Systems, Ed. D. S. Touretzky, Morgan Kaufmann (1990).

[5] Y. Freund and R. E. Schapire: "A Decision-Theoretic Generalization of On-Line Learning and am Application to Boosting". Computational Learning Theory, Second European Conference, pp. 23–37. Sprimger Verlag (1995).

[6] Y. Freund and R. E. Schapire: "Experiments with a New Boosting Algorithm", Proceedings of the International Conference on Machine Learning pp. 148–156, Morgan Kaufmann (1996).

[7] L. Fu and B. Buchanan: "Learning Intermediate Concepts in Constructing a Hierarchical Knowledge Base". Proceedings of the International Joint Conference on Artifi cial Intelligence, pp. 659–666 (1985).

[8] Y. Hirose, K. Yamashita, and S. Hijiya: "Back-Propagation Algorithm Which Varies the Number of Hidden Units". Neural Networks vol. 4, pp. 61–66 (1991).

[9] R. Kohavi: "A Study of Cross Validation and boostrap for Accuracy Estimation and Model Selection", Proceedings of the International Joint Conference on Artifi cial Intelligence, pp. 1137–1143, Morgan Kaufmann (1995).

[10] J. R. Quinlan: "C4.5: Programs for Machine Learning", Mogan Kaufmann (1994).

[11] J. R. Quinlan: "OverSearching and Layered Search in Empirical Learning" in Proceedings of the International Joint Conference on Artifi cial Intelligence, pp. 1019–1024, Morgan Kaufmann (1995).

[12] L. Rendell: "Substantial constructive induction using layered information compression: Tractable feature formation in search", Proceedings of the International Joint Conference on Artifi cial Intelligence, pp. 650–658 (1985).

[13] S. G. Romaniuk: "Learning To Learn: Automatic Adaptation of Learning Bias". Proceedings of the American Association for Artifi cial Intelligence pp. 871–876, MIT Press (1994).

[14] S. G. Romaniuk: "Learning to Learn with Evolutionary Growth Perceptrons" In "Genetic Algorithms for Pattern Recognition", Ed. K. Shankar and P. Wang, Ch. 9, pp. 179–211, CRC Press (1996).

[15] R. Rymon: "An SE-tree Based Characterization of the Induction Problem". International Conference on Machine Learning, pp. 268–275 Morgan Kaufmann (1993).

[16] L. Seabra-Lopes and L. Camarinha-Matos: "Feature Transformation Strategies for a Robot Learning Problem", In "Feature extraction, construction, and selection: a data mining perspective", Ed. L. Huan and M. Hiroshi, pp. 375–391, Kluwer Academic Publishers (1998).

[17] P. Stone and M. Veloso: ”Layered Learning”, Proceedings of the 11th European Conference on Machine Learning, pp. 369–381 (2000).
[18] T. Sebastian: ”Lifelong Learning Algorithms”, In ”Learning to Learn” pp. . 181–209, Kluwer Academic Publsihers (1998).
[19] P. E. Utgoff and D. J. Stracuzzi: ”Many-Layered Learning”, Neural Networks vol. 14 pp. 2497–2529, MIT Press Journals (2002).
[20] R. Vilalta: ”On the Development of Inductive Learning Algorithms: Generating Flexible and Adaptable Concept Representations”, Ph.D. thesis, University of Illinois at Urbana-Champaign (1998).
[21] G. I. Webb: ”OPUS: An Effi cient Admissible Algorithm for Unordered Search”, Journal of Artifi cial Intelligence Research vol. 3 pp. . 431–435 (1995).