World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:6, 2007

Modeling Language for Machine Learning

Tsuyoshi Okita, and Tatsuya Niwa

***Abstract*—** For a given specific problem an efficient algorithm has been the matter of study. However, an alternative approach orthogonal to this approach comes out, which is called a reduction. In general for a given specific problem this reduction approach studies how to convert an original problem into subproblems. This paper proposes a formal modeling language to support this reduction approach. We show three examples from the wide area of learning problems. The benefit is a fast prototyping of algorithms for a given new problem.

***Keywords*—**Formal language, statistical inference problem, reduction.

I. Introduction

WHEN we use machine learning algorithms [8] as a software component in commercial applications, one of the typical difficulties has been the mechanism of achieving reliability. For example in an application such as speech recognition or handwritten digit recognition, the first question from public is usually related to the accuracies of their results. Traditionally this reliability has assessed by time consuming simulations. However recently, statistical learning methods, such as Support Vector Machines [4], kernel methods [12], and variational methods [5], made a revolution how to guarantee the reliability in a theoretical manner, which is achieved by bounds. Therefore in the perspectives of software reliability, statistical learning is the only promising clue at this moment, which should be emphasized. However, there are two immediate weaknesses.

Firstly the traditional approach to create a new algorithm for a new arising problem has disadvantageous in time and human power. Machine learning problems are continuously widening and deepening according as the expansion of application areas. All the more single problem requires various algorithms depending on the criteria, not only the efficiency in speed, but also efficiency in memory or power.

Secondly, the increase of the machine learning algorithms and target problems makes application designers more confused without using any synthetic design methodologies, such as the Unified Modeling Language (UML) in the area of software engineering. This is because there are several fundamental differences between the traditional methodology and that of machine learning. The first difference is the way that machine learning bases on the algorithm induction. As we do not know how to write algorithm directly, machine learning algorithm learns its hypothesis from data. The second difference is that this process has two steps: a learning phase and an application phase (or training phase and test phase). Similarly, there are crucial differences in terms of decomposition of problems and the number of samples as well.

The first problem of continuously arising problems can be solved by a recent appeared reduction approach [7]. Traditionally, for a given specific target problem, the creation of a new algorithm is the matter of study. A reduction approach, which is orthogonal to this traditional approach, breaks down a problem into decomposed problems that can be solved by already known algorithms, such as classification algorithms. Together with the statistical learning perspective, if we derive overall bounds from individual bounds of decomposed subproblems, this could be the definitive advantage over the traditional approach although this paper does not show this point. For the second problem of paradigms the related work is in Allison [2]. Allison formalizes the sample complexity using a functional language Haskell. However, this approach does not say anything about other aspects which we mentioned before.

The contribution of this paper is 1) to construct a formal modeling language in order to solve those two problems, and 2) to show several examples using this formal modeling language including new algorithms. The rest of the paper is organized as follows. In Section 2 we show briefly our new programming paradigm and our summary of language in the following Section, including syntax and semantics of our modeling language. In Section 4 we show several examples using our modeling language and conclude.

II. Inductive Inference Programming Paradigm

We introduce an inductive inference programming paradigm as the background of our formal modeling language. Firstly, we should separate an algorithm (or function) from an input, while in the procedural language an algorithm is directly connected with an input. This would be natural because machine learning learns from data, whereas these data consist of not only input data but also output data. At the same time, this would be natural because we are in difficulties in writing algorithms directly, which is the reason why we use machine learning methods to 'learn' them. Secondly, we should represent both of two phases, which consist of a training phase and a test phase. The learning algorithms learn hypotheses from data, but usually the second phase that uses these hypotheses fulfills the function in an application. In sum, a schema 1) shows a form of a traditional procedural language and 2) shows that of ours.

1) Output algorithm(Input)

2) $\left\langle \begin{array}{l} \text{Algorithm} \\ \text{TestOutput} \mid \text{TestInput} \sim \text{TrainingInput.} \end{array} \right.$

T. Okita and T. Niwa are with Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel, Belgium (phone: +32.2.629.37.24; fax: +32.2.629.37.08; e-mail: {Tsuyoshi.Okita, tniwa}@vub.ac.be).

III. Our Modeling Language

The aim of our modeling language is to describe (inductive / statistical) inference problems in the area of machine learning, especially to describe the necessary and sufficient input / output information. Therefore based on the inductive inference programming paradigm we introduced above, we can formalize this problem description as a specification of a problem. The core of this modeling language is to introduce A) the notation of data structures, B) relationships between those data structures, especially inference relations, C) rewriting rules between problems, and D) semantics of rewriting rules, especially related to the number of samples.

A. Data Structures

We introduce a sequence and a (directed) graph. Although we do not describe an undirected graph in here for the sake of brevity, it is a direct extension of our framework. For a sequence we denote by $\mathbf{\Pi}_{i=1}^{n} X$ a sequence $\{X_1, X_2, \ldots, X_n\}$. For a (directed) graph we denote by $(\mathbf{G}.P)$ a pair of a DAG $\mathbf{G} = (X, E)$ and a joint probability distribution P of the random variables in some set X, where X is a finite and nonempty set whose elements are called nodes, and E is a set of ordered pairs of distinct elements of X. Sometimes in order to emphasize that the nodes in a DAG comprise X_i, we write $(\mathbf{G}(X_i).P)$ instead of $(\mathbf{G}.P)$.

TABLE I
Data Structures in Our Modeling Language

symbols	meaning
data structures	
X_i	i-th sample of X
$X_{(i)}$	i-th variable in X
$\mathbf{\Pi} X_i$	a sequence of observation
$(G(X_i).P)$	a pair of a DAG of samples X in topological order and correspondent probabilities.
$(G(X_{(i)}).P)$	a pair of a DAG of variables in X in topological order and correspondent probabilities.
$\bigcup X_{(i)}$	union of variables

B. Relationships between Data Structures (Problem)

The summary of notation is shown in Table I and II. Among them we define by '$\sim$' the (inductive / statistical) inference relations, which is a natural extension of '$x \sim \mathbf{N}(\mu, \sigma^2)$'. This notation compresses a training and a test process. Firstly, training examples are put at the right-hand side and a resulting hypothesis is put at the left-hand side. We will use capital letters for training examples (which signifies random variables) and lower case letters for test examples (which signifies values of random variables). Secondly, a test example is shown in the second element at the left-hand side and the name of the output by this test example is in the first element. This notation does not lose the implication of the number of samples. Now we show several examples.

Example 1 (Classification problem): A classification problem is defined as '$y|x \sim X|Y$'.

TABLE II
Relationships between Data Structures

symbols	meaning
inductive inference relations	
$a \sim B$	an hypothesis a is (inductively / statistically) inferred by the given training set B.
$a_1 \| a_2 \sim B$	an hypothesis $a_1 \| a_2$ which Outputs a_1 for the given test example a_2 is (inductively / statistically) inferred by the given training set B.
$a \sim B_1 \| B_2$	an hypothesis a is (inductively / statistically) inferred by the training set B_1, each of which is given with a label B_2 or a model B_2.
relations between observations	
$\vee$	OR of observation
,	multi-view of observation
independencies	
$\perp\!\!\!\perp$	independencies in observation
$(G(X_i).P) \perp\!\!\!\perp_M$	$(G(X_i).P)$ satisfies the Markov condition.

In this case, an hypothesis $y|x$ which outputs y for the given test example x is (inductively / statistically) inferred by the given training set X, each of which is given with the label Y.

A clustering problem is to place a label y for the given test example x based on the observation X [12] [11], usually with assuming the resulting cluster number c. It is noted that some classical clustering algorithm such as k-means clustering may not place labels on new test example, but only on training examples, which may easily be extended.

Example 2 (Clustering): A clustering problem is defined as '$y|x \sim X$'. ($y \in Y$, $Y = \{1, \ldots, c\}$ when c is known).

A sequential prediction problem is defined in Conditional Random Fields (CRF) [6]. In a training phase, we take a sequence of words with labels $\mathbf{\Pi}(X|Y)$ as training examples. In a test phase, we predict a sequence of labels $\mathbf{\Pi} y$ based on the test sequence of words $\mathbf{\Pi} x$. In summary,

Example 3 (Sequential prediction): A sequential prediction problem is defined as '$\mathbf{\Pi}(y|x) \sim \mathbf{\Pi}(X|Y)$'.

Using Hidden Markov Models (HMMs) we can solve a couple of different kinds of problems. One problem is solved by the Viterbi algorithm [10], which associates an optimal sequence of states to a sequence of observations, given the parameters of a model of HMMs. More formally, for a given model $(\mathbf{G}(X_i).P)$ and a sequence of observations $\mathbf{\Pi}_{y_i \in Y} y_i$, we choose the optimal state sequence $\mathbf{\Pi}_{x_i \in X} x_i$.

Example 4 (Path selection): A path selection problem is defined as '$(\mathbf{\Pi}_{x_i \in X} x_i) | (g(x_i).p) \sim \mathbf{\Pi}_{Y_i \in Y} Y_i | (\mathbf{G}(X_i).P)$'.

C. Syntax of Rewriting Rules

From now on we consider a problem with an algorithm. Rewriting rules are shown in Table III and IV, where an arrow shows a transition. There are several categories of rewriting rules: 1) $\vee$ (logical 'OR'), 2) independencies, and 3) dimensionality transformation. An algorithm might be altered when examples are split in the case of the rewriting rule (1), and when different hypotheses are merged in the case of the rewriting rule (13) and (14). It is noted that one famous example that uses sequence dimensional reduction is CRFs [6].

TABLE III
REWRITING RULES WHERE WE HANDLE PROBLEMS WITH ALGORITHMS

	rewriting rules
1	$\left\langle \begin{array}{l}\text{algorithm A}\\ a \sim (B \vee C)\end{array} \right. \rightarrow$ $\left\langle \begin{array}{l}\text{algorithm A}\\ H_1 = (a \sim B)\end{array} \right. \vee \left\langle \begin{array}{l}\text{algorithm A}\\ H_2 = (a \sim C)\end{array} \right. \vee \left\langle \begin{array}{l}\text{algorithm B}\\ H = H_1 \vee H_2\end{array} \right.$ [OR separation]
2	$\left\langle \begin{array}{l}\text{algorithm A}\\ a \sim B\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ a \sim (B \vee C)\end{array} \right.$
3	$\left\langle \begin{array}{l}\text{algorithm A}\\ a \sim B\|C\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ a \sim ((B_1\|C_1) \vee (B_2\|C_2)), (B_1 \bigcup B_2 = B)\end{array} \right.$ [OR separation]
4	$\left\langle \begin{array}{l}\text{algorithm A}\\ \text{problem I}\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm B}\\ \text{problem I}\end{array} \right.$ [algorithm conversion]
5	$\left\langle \begin{array}{l}\text{algorithm A}\\ y\|x_{(i)} \sim X\|Y\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ y\|x \sim X\|Y\end{array} \right.$ [equalization of training and test set]
6	$\left\langle \begin{array}{l}\text{algorithm A}\\ \mathbf{\Pi}(y\|x) \sim \mathbf{\Pi}(X\|Y)\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ y\|x \sim \mathbf{\Pi}(X\|Y)\end{array} \right.$ [sequence dim-reduction]
7	$\left\langle \begin{array}{l}\text{algorithm A}\\ (y\|x) \sim \mathbf{\Pi}(X\|Y)\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ \mathbf{\Pi}(y\|x) \sim \mathbf{\Pi}(X\|Y)\end{array} \right.$ [sequence dim-augmentation]
8	$\left\langle \begin{array}{l}\text{algorithm A}\\ \mathbf{\Pi}y\|(G\mathbf{\Pi}x_i) \sim \mathbf{\Pi}(X\|Y)\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ y\|x \sim X\|Y\end{array} \right.$ [graph dim-reduction]
9	$\left\langle \begin{array}{l}\text{algorithm A}\\ y\|x \sim X\|Y\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ \mathbf{\Pi}y\|G(\mathbf{\Pi}x_i) \sim \mathbf{\Pi}(X\|Y)\end{array} \right.$ [graph dim-augmentation]

TABLE IV
REWRITING RULES CONTINUED FROM THE PREVIOUS TABLE

	rewriting rules
10	$\left\langle \begin{array}{l}\text{algorithm A}\\ (\mathbf{\Pi}_1^n y_i)\|x \sim X\|\mathbf{\Pi}_1^n Y_i)\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ ((\mathbf{\Pi}_1^2 y_i)\|x \sim X\|\mathbf{\Pi}_1^2 Y_i)) \vee \ldots \vee\\ \quad ((\mathbf{\Pi}_{n-1}^n y_i)\|x \sim X\|\mathbf{\Pi}_{n-1}^n Y_i))\end{array} \right.$ with$G(X_i)\perp\!\!\!\perp_M$ [decomposition by Markov property]
11	$\left\langle \begin{array}{l}\text{algorithm A}\\ (\bigcup_1^n y_{(i)})\|x \sim X\|Y\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ (y_{(1)}\|x \sim X\|Y) \vee \ldots \vee\\ \quad (y_{(n)}\|x \sim X\|Y)\end{array} \right.$ with $G(X_{(i)})\perp\!\!\!\perp_M$ [decomposition by d-separation]
12	$\left\langle \begin{array}{l}\text{algorithm A}\\ y\|x, p \sim X\|Y, P\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ y\|x \sim X\|Y\end{array} \right.$ [marginalization of hidden variables]
13	$\left\langle \begin{array}{l}\text{algorithm A}\\ ((y_i\|x_i) \sim \mathbf{\Pi}(X_i\|Y_i)) \vee \ldots \vee ((y_k\|x_k) \sim \mathbf{\Pi}(X_k\|Y_k))\end{array} \right.$ $\rightarrow \left\langle \begin{array}{l}\text{algorithm B}\\ \mathbf{\Pi}_{i=i}^k(y\|x) \sim \mathbf{\Pi}(X_i\|Y_i)\end{array} \right.$ [merge sequences]
14	$\left\langle \begin{array}{l}\text{algorithm A}\\ ((y_{(i)}\|x_{(i)}) \sim \mathbf{\Pi}(X_{(i)}\|Y_{(i)})) \vee \ldots \vee\\ \quad ((y_{(k)}\|x_{(k)}) \sim \mathbf{\Pi}(X_{(k)}\|Y_{(k)}))\end{array} \right.$ $\rightarrow \left\langle \begin{array}{l}\text{algorithm B}\\ \bigcup_{i=i}^k(y\|x) \sim \bigcup_{i=i}^k X_i\|\bigcup_{i=i}^k Y_i\end{array} \right.$ [merge variables]
15	$\left\langle \begin{array}{l}\text{algorithm A}\\ a \sim \bigcup X_{(i)}\|\bigcup Y_{(k)}(X_{(i)}, X_{(k)} \subseteq X)\end{array} \right. \rightarrow \left\langle \begin{array}{l}\text{algorithm A}\\ a \sim X\end{array} \right.$ [superset]

D. Semantics of Rewriting Rules

Rewriting rules related to decomposition and merge has two semantics : 1) decomposition semantics and 2) semantics of the number of samples.

Decomposition semantics relates to the fact that we decompose the input or the output examples instead of algorithms. This is due to the fact that an algorithm in machine learning consists of learned hypotheses, which are usually not able to decompose, which is different from the traditional decomposition of algorithms, such as parallelization of algorithms. In this reason, correspondent algorithms when decomposed should be carefully chosen based on the original algorithm. Assume that we divide an original training set into two sets. After learning an individual training set, we have to choose carefully how to merge those two hypotheses, where there are many possibilities in this merge. Similarly, correspondent algorithms when merged should be carefully chosen. Semantics of the number of samples can be used for the calculation of overall bounds from individual bounds.

IV. EXAMPLE OF REWRITING

In this section we show several examples of reduction of a source problem into a destination problem. The first example shows a parallelization of SVMs [9]. It is noted that there are many other ways to decompose this problem other than our examples.

Example 5: (Parallelization of SVMs) A problem $y|x \sim X|Y$ with $X_i \perp\!\!\!\perp X_j$, $X_{(i)} \perp\!\!\!\perp X_{(j)}$ with SVMs is equivalent to the following decomposed problems.

$$\left\{ \begin{array}{l}\text{SVMs algorithm}\\ H_1 = y|x_1 \sim X_1|Y_1,\\ \text{SVMs algorithm}\\ H_2 = y|x_2 \sim X_2|Y_2,\\ \text{boosting algorithm}\\ y|x \sim H_1 \vee H_2,\end{array} \right.$$

with $X_i \perp\!\!\!\perp X_j$, $X_{(i)} \perp\!\!\!\perp X_{(j)}$ and with $X_1 \Leftrightarrow X_2$, where Y_1 is a label for X_1 and Y_2 is a label for X_2, and $X = X_1 \bigcup X_2$.

Rewriting Procedure 5: First we decompose examples X into X_1 and X_2. It is noted that in the schema the number shows the number of rewriting rules in Tables.

$$\left\langle \begin{array}{l}\text{SVMs algorithm}\\ y|x \sim X|Y\end{array} \right.$$
$$\rightarrow_{(4)} \left\langle \begin{array}{l}\text{SVMs algorithm}\\ y|(x_1 \vee x_2) \sim ((X_1|Y_1) \vee (X_2|Y_2))\end{array} \right.$$
$$\rightarrow_{(3)} \left\{ \begin{array}{l}\text{SVMs algorithm}\\ H_1 = y|x_1 \sim X_1|Y_1,\\ \text{SVMs algorithm}\\ H_2 = y|x_2 \sim X_2|Y_2,\\ \text{boosting algorithm}\\ y|x \sim H_1 \vee H_2,\end{array} \right.$$

Next example rewrites from a path selection problem into a sequential prediction problem.

Example 6: A path selection problem $(\mathbf{\Pi}_{x_i \in X} x_i)|(g(x_i).p) \sim \mathbf{\Pi}Y_i|(G(X_i).P)$ $(G(X_i)\perp\!\!\!\perp_M)$ with the Viterbi algorithm can be reduced into a sequential prediction problem $PI(y|x) \sim \mathbf{\Pi}(X|Y)$ $(G(X_{(i)})\perp\!\!\!\perp_M)$ with CRF algorithm.

Rewriting Procedure 6: Using the rewriting rule (9), a path selection problem can be decomposed into the following problems:

$$\left\langle \begin{array}{l} \text{Viterbi algorithm} \\ (\mathbf{\Pi}_{x_i \in X} x_i)|(g(x_i).p) \sim \mathbf{\Pi} Y_i|(G(X_i).P) \end{array} \right.$$

$$\rightarrow_{(defG)} \left\langle \begin{array}{l} \text{Viterbi algorithm} \\ \mathbf{\Pi}(x_i)|\mathbf{\Pi}(x_i|p_i) \sim \mathbf{\Pi} Y_i|\mathbf{\Pi}(X_i|P_i) \end{array} \right.$$

$$\rightarrow_{D(10)} \left\langle \begin{array}{l} \text{Viterbi algorithm} \\ \left\{ \begin{array}{l} x_{i+1}|(x_i|p_i) \sim Y_i|(X_{i+1}|P_{i+1}) \\ \ldots \\ x_k|(x_{k-1}|p_{k-1}) \sim Y_{k-1}|(X_k|P_k) \end{array} \right. \end{array} \right.$$

$$\rightarrow_{M(13)} \left\langle \begin{array}{l} \text{CRF algorithm} \\ \mathbf{\Pi}(x_{i+1}|x_i, p_i) \sim (\mathbf{\Pi} Y_i|X_{i+1}, P_{i+1})) \end{array} \right.$$

$$\rightarrow_{(12)} \left\langle \begin{array}{l} \text{CRF algorithm} \\ \mathbf{\Pi}(x_{i+1}|x_i) \sim (\mathbf{\Pi}(Y_i|X_{i+1})) \end{array} \right.$$

We define a structure discovery (plus inference) problem is the combination of two phases: the first phase is to learn Bayesian networks from data and the second phase is to determine various probabilities of interest based on the constructed Bayesian networks. This problem can be decomposed into multi-class classification problems as follows.

Example 7: A structure discovery problem $\bigcup_{j=l}^{k} x_{(j)}|\bigcup_{i=n}^{m} x_{(i)} \sim X$ $(G(X_i)\perp\!\!\!\perp_M)$ with an arbitrary structure discovery algorithm is equivalent to $\#(k-l)$ number of multi-class classification (regression) problems where we observe them in $\bigcup X_{(i)}$ with a classification algorithm (and the hypotheses merge with an arbitrary boosting algorithm).

Rewriting Procedure 7: Using the rewriting rule (10), a structure discovery problem $\bigcup x_{(j)}|\bigcup x_{(n)} \sim X$ can be decomposed in a following manner.

$$\left\langle \begin{array}{l} \text{arbitrary structure discovery algorithm} \\ \bigcup x_{(j)}|\bigcup x_{(i)} \sim X \end{array} \right.$$

$$\rightarrow_{D(11)} \left\langle \begin{array}{l} \text{arbitrary structure discovery algorithm} \\ \left\{ \begin{array}{l} \left\{ \begin{array}{l} x_{(l)}|x_{(n)} \sim X \\ \ldots \\ x_{(l)}|x_{(m)} \sim X \end{array} \right. \\ \ldots \\ \left\{ \begin{array}{l} x_{(k)}|x_{(n)} \sim X \\ \ldots \\ x_{(k)}|x_{(m)} \sim X \end{array} \right. \end{array} \right. \end{array} \right.$$

$$\rightarrow_{(5)} \left\langle \begin{array}{l} \text{arbitrary structure discovery algorithm} \\ \left\{ \begin{array}{l} \left\{ \begin{array}{l} x_{(l)}|x \sim X|X_{(l)} \\ \ldots \\ x_{(l)}|x \sim X|X_{(l)} \end{array} \right. \\ \ldots \\ \left\{ \begin{array}{l} x_{(k)}|x \sim X|X_{(k)} \\ \ldots \\ x_{(k)}|x \sim X|X_{(k)} \end{array} \right. \end{array} \right. \end{array} \right.$$

$$\rightarrow_{M(14)} \left\{ \begin{array}{l} \left\langle \begin{array}{l} \text{multiclass classification /regression} \\ H_j = x_{(l)}|x \sim X|X_{(l)} \end{array} \right. \\ \ldots \\ \left\langle \begin{array}{l} \text{multiclass classification / regression} \\ H_k = x_{(k)}|x \sim X|X_{(k)} \\ \text{arbitrary merge algorithm} \\ H = H_j \vee \ldots \vee H_k \end{array} \right. \end{array} \right.$$

V. Conclusion

The contribution of this paper is to construct a formal modeling language for machine learning. The syntax and semantics of this formal modeling language are shown in Tables. Due to the space restriction we could not explain enough about each rewriting rule, but we implement the interesting aspects in this formal language, which we named the inductive inference programming paradigm, such as training / test phase separation, decomposition semantics, and separation of an algorithm and input / output. Then we show three rewriting examples. These examples show that for a given new problem we might be able to rewrite a wide range of new problems into easier problems, such as classification problems. Therefore this reduction approach enables us to do a fast prototyping of algorithms for a new problem.

As is mentioned in Introduction, our further study is to develop this formal modeling language as the basis for reduction. Then as is shown in the paper the result of rewriting boils down to the combination of easy algorithms in statistical learning, such as classification and boosting. Hence we can combine individual bounds of classification and boosting into overall bounds, which enables us to guarantee the overall reliability. This framework is related to the semantics of number of sample of this modeling language.

References

[1] Abney, S. (2002). *Bootstrapping*. The 40th Annual Meeting of the Association for Computational Linguistics.
[2] Allison, L. (2003). *Types and Classes of Machine Learning and Data Mining*. Twenty-Six Australasian Computer Science Conference (ACSC2003), pp.207-215, Australia.
[3] Bartlett, P. L., Collins, M., McAllester, D., and Taskar, B. (2004). *Large margin methods for structured classification: Exponentiated Gradient algorithms and PAC-Bayesian generalization bounds*. NIPS Conference.
[4] Cristianini, N., Shawe-Taylor, J. (2000). *Introduction to Support Vector Machines*. Cambridge University Press.
[5] Jaakkola, T. (2000) *Tutorial on Variational Approximation Method*. In Advanced Mean Field Methods: Theory and Practice, MIT Press.
[6] Lafferty, J., McCallum, A., Pereira, F. (2001). *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*. International Conference on Machine Learning (ICML).
[7] Langford, J., Beygelzimer, A. (2002). *Sensitive Error Correcting Output Codes*.
[8] Mitchell, T. (1997). *Machine Learning*. McGraw Hills.
[9] Okita, T., Manderick, B. (2003). *Support Vector Learning in Distributed Environments (poster)*, Conference On Learning Theory and Kernel Machines, Washington.
[10] Rabiner, L. R. (1989) *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.* Proceedings of the IEEE, VOL. 77, No. 2, February 1989.
[11] Scholkopf, B., Williamson, R.C., Smola, A.J., Shawe-Taylor, J. (2000). *Support Vector Method for Novelty Detection*. In Neural Information Processing Systems.
[12] Shawe-Taylor, J., Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.