Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization

Min-Ling Zhang and Zhi-Hua Zhou, *Senior Member*, *IEEE*

Abstract—In multilabel learning, each instance in the training set is associated with a set of labels and the task is to output a label set whose size is unknown a priori for each unseen instance. In this paper, this problem is addressed in the way that a neural network algorithm named BP-MLL, i.e., Backpropagation for Multilabel Learning, is proposed. It is derived from the popular Backpropogation algorithm through employing a novel error function capturing the characteristics of multilabel learning, i.e., the labels belonging to an instance should be ranked higher than those not belonging to that instance. Applications to two real-world multilabel learning problems, i.e., functional genomics and text categorization, show that the performance of BP-MLL is superior to that of some well-established multilabel learning algorithms.

Index Terms—Machine learning, data mining, multilabel learning, neural networks, backpropagation, functional genomics, text categorization.

1 Introduction

Multilabel learning tasks are ubiquitous in real-world problems. For instance, in text categorization, each document may belong to several predefined topics, such as *government* and *health* [18], [28]; in bioinformatics, each gene may be associated with a set of functional classes, such as *metabolism*, *transcription* and *protein synthesis* [8]; in scene classification, each scene image may belong to several semantic classes, such as *beach* and *urban* [2]. In all these cases, instances in the training set are each associated with a set of labels and the task is to output the label set whose size is not known a priori for the unseen instance.

Traditional two-class and multiclass problems can both be cast into multilabel ones by restricting each instance to have only one label. On the other hand, the generality of multilabel problems inevitably makes them more difficult to solve. An intuitive approach to solving a multilabel problem is to decompose it into multiple independent binary classification problems (one per category). However, this kind of method does not consider the correlations between the different labels of each instance and the expressive power of such a system can be weak [8], [18], [28]. Fortunately, several approaches specially designed for multilabel learning tasks have been proposed, such as multilabel text categorization algorithms [12], [18], [28], [30], multilabel decision trees [4], [5], and multilabel kernel methods [2], [8], [16]. In this paper, a neural network algorithm named BP-MLL, i.e., Backpropagation for Multilabel Learning, is proposed, which is the first multilabel neural network algorithm. As its name implies, BP-MLL is derived from the popular Backpropagation algorithm [24] through replacing its error function with a new function defined to capture the characteristics of multilabel learning; that is, the labels belonging to an instance should be ranked higher than those not belonging to that instance. Applications to two real-world multilabel learning problems, i.e., functional genomics and text categorization, show that BP-MLL outperforms some well-established multilabel learning algorithms.

The rest of this paper is organized as follows: In Section 2, a formal definition of multilabel learning is given and previous works in this area are reviewed. In Section 3, BP-MLL is presented. In Section 4, evaluation metrics used in multilabel learning are briefly introduced. In Section 5 and Section 6, experiments of BP-MLL on two real-world multilabel learning problems are reported, respectively. Finally, in Section 7, the main contribution of this paper is summarized.

2 Multilabel Learning

Let $\mathcal{X} = \mathbb{R}^d$ denote the domain of instances and let $\mathcal{Y} = \{1, 2, \ldots, Q\}$ be the finite set of labels. Given a training set $T = \{(\mathbf{x}_1, Y_1), (\mathbf{x}_2, Y_2), \ldots, (\mathbf{x}_m, Y_m)\}$ $(\mathbf{x}_i \in \mathcal{X},\ Y_i \subseteq \mathcal{Y})$ i.i.d. drawn from an unknown distribution D, the goal of the learning system is to output a multilabel classifier $h : \mathcal{X} \rightarrow 2^{\mathcal{Y}}$, which optimizes some specific evaluation metric. In most cases, however, instead of outputting a multilabel classifier, the learning system will produce a real-valued function of the form $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$. It is supposed that, given an instance $\mathbf{x}_i$ and its associated label set Y_i, a successful learning system will tend to output larger values for labels in Y_i than those not in Y_i, i.e. $f(\mathbf{x}_i, y_1) > f(\mathbf{x}_i, y_2)$, for any $y_1 \in Y_i$ and $y_2 \notin Y_i$. The real-valued function $f(\cdot, \cdot)$ can be transformed to a ranking function $rank_f(\cdot, \cdot)$, which maps the outputs of $f(\mathbf{x}_i, y)$ for any $y \in \mathcal{Y}$ to $\{1, 2, \ldots, Q\}$ such that, if $f(\mathbf{x}_i, y_1) > f(\mathbf{x}_i, y_2)$, then $rank_f(\mathbf{x}_i, y_1) < rank_f(\mathbf{x}_i, y_2)$. Note that the corresponding multilabel classifier $h(\cdot)$ can also be derived from the

- *The authors are with the National Laboratory for Novel Software Technology, Nanjing University, Mailbox 419, Hankou Road 22, Nanjing 210093, China. E-mail: {zhangml, zhouzh}@lamda.nju.edu.cn.*

Manuscript received 15 Sept. 2005; revised 11 Mar. 2006; accepted 24 May 2006; published online 18 Aug. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0373-0905.

1041-4347/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

function $f(\cdot,\cdot)$: $h(\mathbf{x}_i) = \{y|f(\mathbf{x}_i, y) > t(\mathbf{x}_i),\ y \in \mathcal{Y}\}$, where $t(\cdot)$ is a threshold function, which is usually set to be the zero constant function.

As stated in the above section, the generality of multilabel problems inevitably makes them more difficult to solve than traditional single-label (two-class or multiclass) problems. Until now, only a few works on multilabel learning are available, which mainly concern the problems of text categorization [5], [12], [16], [18], [28], [30], bioinformatics [4], [8], and scene classification [2].

Research on multilabel learning was initially motivated by the difficulty of concept ambiguity encountered in text categorization, where each document may belong to several topics (labels) simultaneously. One famous approach to solving this problem is BOOSTEXTER, proposed by Schapire and Singer [28], which is, in fact, extended from the popular ensemble learning method ADABOOST [10]. In the training phase, BOOSTEXTER maintains a set of weights over both training examples and their labels, where training examples and their corresponding labels that are hard (easy) to predict correctly get incrementally higher (lower) weights. In 1999, McCallum [18] proposed a Bayesian approach to multilabel document classification, where a mixture probabilistic model (one mixture component per category) is assumed to generate each document and the EM algorithm [6] is utilized to learn the mixture weights and the word distributions in each mixture component. In 2003, Ueda and Saito [30] presented two types of probabilistic generative models for multilabel text, called parametric mixture models (PMM1, PMM2), where the basic assumption under PMMS is that multilabel text has a mixture of characteristic words appearing in single-label text that belong to each category of the multicategories. It is worth noting that the generative models used in [18] and [30] are both based on learning text frequencies in documents and are, thus, specific to text applications. Also in 2003, Comité et al. [5] extended alternating decision tree [9] to handle multilabel data, where the ADABOOST.MH algorithm proposed by Schapire and Singer [27] is employed to train the multilabel alternating decision trees.

In 2004, Gao et al. [12] generalized the maximal figure-of-merit (MFoM) approach [11] for binary classifier learning to the case of multiclass, multilabel text categorization. They defined a continuous and differentiable function of the classifier parameters to simulate specific performance metrics, such as precision and recall etc. (microaveraging F_1 in their paper). Their method assigns a uniform score function to each category of interest for each given test example and, thus, the classical Bayes decision rules can be applied. One year later, Kazawa et al. [16] converted the original multilabel learning problem of text categorization into a multiclass single-label problem by regarding a set of topics (labels) as a new class. To cope with the data sparseness caused by the huge number of possible classes (Q topics will yield 2^Q classes), they embedded labels into a similarity-induced vector space in which prototype vectors of similar labels would be placed close to each other. They also provided an approximation method in learning and efficient classification algorithms in testing to overcome the demanding computational cost of their method.

In addition to text categorization, multilabel learning has also manifested its effectiveness in other real-world applications, such as bioinformatics and scene classification. In 2001, Clare and King [4] adapted C4.5 decision tree [22] to handle multilabel data (gene expression in their case) through modifying the definition of entropy. They chose decision trees as the baseline algorithm because its output (equivalently, a set of symbolic rules) is interpretable and can be compared with existing biological knowledge. It is also noteworthy that their goal is to learn a set of accurate rules, not necessarily a complete classification. One year later, through defining a special cost function based on *ranking loss* (as shown in (24)) and the corresponding margin for multilabel models, Elisseeff and Weston [8] proposed a kernel method for multilabel classification and tested their algorithm on a yeast gene functional classification problem with positive results. In 2004, Boutell et al. [2] applied multilabel learning techniques to scene classification. They decomposed the multilabel learning problem into multiple independent binary classification problems (one per category), where each example associated with label set Y will be regarded as a positive example when building classifier for class $y \in Y$ while regarded as a negative example when building classifier for class $y \notin Y$. They also provided various labeling criteria to predict a set of labels for each test instance based on its output on each binary classifier. Note that, although most works on multilabel learning assume that an instance can be associated with multiple valid labels, there are also works assuming that only one of the labels associated with an instance is correct [14].[1]

As reviewed above, most of the existing multilabel learning algorithms are derived from traditional learning techniques such as probabilistic generative models [18], [30], boosting methods [28], decision trees [4], [5], and maximal margin methods [8], [2], [16]. However, as a popular and effective learning mechanism, there has not been any multilabel learning algorithm derived from a neural network model. In the following section, the first multilabel learning algorithm based on a neural network model, i.e., BP-MLL, is proposed.

3 BP-MLL

3.1 Neural Networks

As defined in the literature [17], neural networks are massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations, which are intended to interact with the objects of the real world in the same way that biological nervous systems do. The earliest work on neural networks dates back to McCulloch and Pitts's M-P model of a neuron [19], which is then followed by considerable work in the 1950s and 1960s on single-layer neural networks [23], [31]. Although single-layer neural networks were successful in classifying certain patterns, they had a number of limitations so that even simple functions such as XOR could hardly be learned [20]. Such limitations led to the decline of research on neural

[1] In this paper, only the former formalism of multilabel learning is studied.

networks during the 1970s. In the early 1980s, research on neural networks resurged largely due to successful learning algorithms for multilayer neural networks. Currently, diverse neural networks exist, such as multilayer feed-forward networks, radial basis function networks, adaptive resonance theory models, self-organizing feature mapping networks, etc. Neural networks provide general and practical techniques for learning from examples, which have been widely used in various areas.

In this paper, traditional multilayer feed-forward neural networks are adapted to learn from multilabel examples. Feed-forward networks have neurons arranged in layers, with the first layer taking inputs and the last layer producing outputs. The middle layers have no connection with the external world and, hence, are called hidden layers. Each neuron in one layer is connected (usually fully) to neurons on the next layer and there is no connection among neurons in the same layer. Therefore, information is constantly *fed forward* from one layer to the next one. Parameters of the feed-forward networks are learned by minimizing some error function defined over the training examples, which commonly takes the form of the sum of the squared difference between the network output values and the target values on each training example. The most popular approach to minimizing this sum-of-squares error function is the backpropagation algorithm [24], which uses gradient descent to update parameters of the feed-forward networks by propagating the errors of the output layer successively back to the hidden layers. More detailed information about neural networks and related topics can be found in textbooks such as [1] and [13].

Actually, adapting traditional feed-forward neural networks from handling *single-label* examples to *multilabel* examples requires two keys. The first key is to design some specific error function other than the simple sum-of-squares function to capture the characteristics of multilabel learning. Secondly, some revisions have to be made accordingly for the classical learning algorithm in order to minimize the newly designed error function. These two keys will be described in detail in the following two sections, respectively.

3.2 Architecture

Let $\mathcal{X} = \mathbb{R}^d$ be the instance domain and $\mathcal{Y} = \{1, 2, \dots, Q\}$ be the finite set of class labels. Suppose the training set is composed of m multilabel instances, i.e.,

$$\{(\mathbf{x}_1, Y_1), (\mathbf{x}_2, Y_2), \dots, (\mathbf{x}_m, Y_m)\},$$

where each instance $\mathbf{x}_i \in \mathcal{X}$ is a d-dimensional feature vector and $Y_i \subseteq \mathcal{Y}$ is the set of labels associated with this instance. Now, suppose a single-hidden-layer feed-forward BP-MLL neural network, as shown in Fig. 1, is used to learn from the training set. The BP-MLL neural network has d input units each corresponding to a dimension of the d-dimensional feature vector, Q output units each corresponding to one of the possible classes, and one hidden layer with M hidden units. The input layer is fully connected to the hidden layer with weights $\mathrm{V} = [v_{hs}]$ $(1 \leq h \leq d,\ 1 \leq s \leq M)$ and the hidden layer is also fully connected to the output layer with weights $\mathrm{W} = [w_{sj}]$ $(1 \leq s \leq M,\ 1 \leq j \leq Q)$. The bias parameters γ_s $(1 \leq s \leq M)$ of the hidden units are shown as weights from an extra input unit a_0 having a fixed value of 1. Similarly, the bias parameters θ_j $(1 \leq j \leq Q)$ of the output units are shown as weights from an extra hidden unit b_0, with activation again fixed at 1.

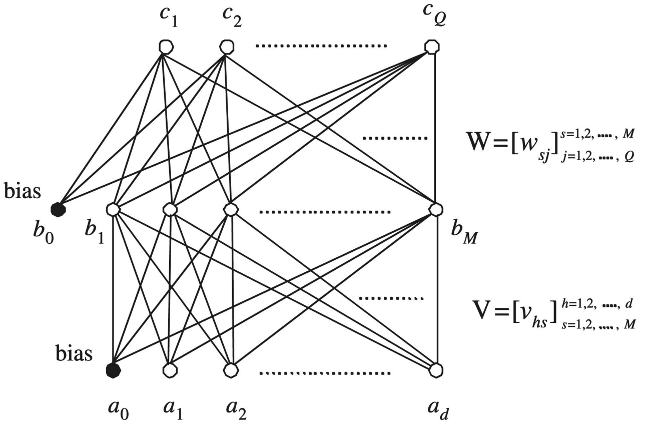

c_1
c_2
c_Q
$\mathrm{W}=[w_{sj}]^{s=1,2,\dots,M}_{j=1,2,\dots,Q}$
bias
b_0
b_1
b_M
$\mathrm{V}=[v_{hs}]^{h=1,2,\dots,d}_{s=1,2,\dots,M}$
bias
a_0
a_1
a_2
a_d

Fig. 1. Architecture of the BP-MLL neural network.

Since the goal of multilabel learning is to predict the label sets of unseen instances, an intuitive way to define the global error of the network on the training set could be

$$E = \sum_{i=1}^{m} E_i, \tag{1}$$

where E_i is the error of the network on $\mathbf{x}_i$, which could be defined as

$$E_i = \sum_{j=1}^{Q} (c_j^i - d_j^i)^2, \tag{2}$$

where $c_j^i = c_j(\mathbf{x}_i)$ is the actual output of the network on $\mathbf{x}_i$ on the jth class and d_j^i is the desired output of $\mathbf{x}_i$ on the jth class, which takes the value of either +1 $(j \in Y_i)$ or -1 $(j \notin Y_i)$.

Combining (2) with (1), various optimization methods can be directly applied to learn from the multilabel training instances. In this paper, the classical Backpropagation algorithm [24] is used to learn from this intuitive global error function and the resulting algorithm is named as BASICBP. However, although BASICBP is feasible, some important characteristics of multilabel learning are not considered by this method. Actually, the error function defined in (2) concentrates only on individual label discrimination, i.e., whether a particular label $j \in \mathcal{Y}$ belongs to the instance $\mathbf{x}_i$ or not; it does not consider the correlations between the different labels of $\mathbf{x}_i$, e.g., labels in Y_i should be ranked higher than those not in Y_i. In this paper, these characteristics of multilabel learning are appropriately addressed by rewriting the global error function as follows:

$$E = \sum_{i=1}^{m} E_i = \sum_{i=1}^{m} \frac{1}{|Y_i||\overline{Y}_i|} \sum_{(k,l) \in Y_i \times \overline{Y}_i} \exp(-(c_k^i - c_l^i)). \tag{3}$$

As regards the right-hand side of (3), the ith error term

$$\left(\frac{1}{|Y_i||\overline{Y}_i|} \sum\nolimits_{(k,l) \in Y_i \times \overline{Y}_i} \exp\left(-\left(c_k^i - c_l^i\right)\right) \right)$$

in the summation defines the error of the network on the ith multilabel training example $(\mathbf{x}_i, Y_i)$. Here, $\overline{Y}_i$ is the complementary set of Y_i in $\mathcal{Y}$ and $|\cdot|$ measures the cardinality of a set. Specifically, $c_k^i - c_l^i$ measures the difference between the outputs of the network on one label belonging to $\mathbf{x}_i$ ($k \in Y_i$) and one label not belonging to it ($l \in \overline{Y}_i$). It is obvious that the bigger the difference, the better the performance. Furthermore, the negation of this difference is fed to the exponential function in order to severely penalize the ith error term if c_k^i (i.e., the output on the label belonging to $\mathbf{x}_i$) is much smaller than c_l^i (i.e., the output on the label not belonging to $\mathbf{x}_i$). The summation in the ith error term takes account of the accumulated difference between the outputs of any pair of labels with one belonging to $\mathbf{x}_i$ and another not belonging to $\mathbf{x}_i$, which is then normalized by the total number of possible pairs, i.e. $|Y_i||\overline{Y}_i|$.[2] In this way, the correlations between different labels of $\mathbf{x}_i$, i.e., labels in Y_i should get larger network outputs than those in $\overline{Y}_i$, are appropriately addressed.

As analyzed above, (3) focuses on the difference between the network's outputs on labels belonging to one instance and on other labels not belonging to it. Therefore, minimization of (3) will lead the system to output larger values for labels belonging to the training instance and smaller values for those not belonging to it. When the training set sufficiently covers the distribution information of the learning problem, the well-trained neural network model encoding these information will also eventually give larger outputs for the labels belonging to the test instance than those labels not belonging to it. Actually, this error function is closely related to the *ranking loss* criterion (as shown in (24)) which will be introduced in Section 4.

In this paper, minimization of the global error function is carried out by gradient descent combined with the error backpropagation strategy [24], which is scrutinized in the following section.

3.3 Training and Testing

For training instance $\mathbf{x}_i$ and its associated label set Y_i, the actual output of the jth output unit is (omitting the superscript i without loss of generality)

$$c_j = f(netc_j + \theta_j), \tag{4}$$

where θ_j is the bias of the jth output unit; $f(x)$ is the activation function of the output units, which is set to be the "tanh" function:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \tag{5}$$

$netc_j$ is the input to the jth output unit:

$$netc_j = \sum_{s=1}^{M} b_s w_{sj}, \tag{6}$$

where w_{sj} is the weight connecting the sth hidden unit and the jth output unit; and M is the number of hidden units. b_s is the output of the sth hidden unit:

$$b_s = f(netb_s + \gamma_s), \tag{7}$$

where γ_s is the bias of the sth hidden unit and $f(u)$ is also the "tanh" function. $netb_s$ is the input to the sth hidden unit:

$$netb_s = \sum_{h=1}^{d} a_h v_{hs}, \tag{8}$$

where a_h is the hth component of $\mathbf{x}_i$ and v_{hs} is the weight connecting the hth input unit and the sth hidden unit.

Since the "tanh" function is differentiable, we can define the general error of the jth output unit as

$$d_j = -\frac{\partial E_i}{\partial netc_j}. \tag{9}$$

Considering $c_j = f(netc_j + \theta_j)$, we get

$$d_j = -\frac{\partial E_i}{\partial c_j}\frac{\partial c_j}{\partial netc_j} = -\frac{\partial E_i}{\partial c_j} f'(netc_j + \theta_j). \tag{10}$$

Then, considering $E_i = \frac{1}{|Y_i||\overline{Y}_i|}\sum_{(k,l)\in Y_i \times \overline{Y}_i} \exp(-(c_k - c_l))$, we get

$$\begin{aligned}\frac{\partial E_i}{\partial c_j} &= \frac{\partial\left[\frac{1}{|Y_i||\overline{Y}_i|}\sum_{(k,l)\in Y_i\times\overline{Y}_i}\exp(-(c_k - c_l))\right]}{\partial c_j} \\ &= \begin{cases} -\frac{1}{|Y_i||\overline{Y}_i|}\sum\limits_{l\in\overline{Y}_i}\exp(-(c_j - c_l)), & \text{if } j \in Y_i \\ \frac{1}{|Y_i||\overline{Y}_i|}\sum\limits_{k\in Y_i}\exp(-(c_k - c_j)), & \text{if } j \in \overline{Y}_i. \end{cases}\end{aligned} \tag{11}$$

Since $f'(netc_j + \theta_j) = (1 + c_j)(1 - c_j)$, then substituting this equation and (11) into (10), we get

$$d_j = \begin{cases} \left(\frac{1}{|Y_i||\overline{Y}_i|}\sum\limits_{l\in\overline{Y}_i}\exp(-(c_j - c_l))\right)(1 + c_j)(1 - c_j) \text{if } j \in Y_i \\ \left(-\frac{1}{|Y_i||\overline{Y}_i|}\sum\limits_{k\in Y_i}\exp(-(c_k - c_j))\right)(1 + c_j)(1 - c_j) \text{if } j \in \overline{Y}_i. \end{cases} \tag{12}$$

Similarly, we can define the general error of the sth hidden unit as

$$e_s = -\frac{\partial E_i}{\partial netb_s}. \tag{13}$$

Considering $b_s = f(netb_s + \gamma_s)$, we get

$$e_s = -\frac{\partial E_i}{\partial b_s}\frac{\partial b_s}{\partial netb_s} = -\left(\sum_{j=1}^{Q}\frac{\partial E_i}{\partial netc_j}\frac{\partial netc_j}{\partial b_s}\right) f'(netb_s + \gamma_s). \tag{14}$$

Then, considering $d_j = -\frac{\partial E_i}{\partial netc_j}$ and $netc_j = \sum\limits_{s=1}^{M} b_s w_{sj}$, we get

$$\begin{aligned} e_s &= \left(\sum_{j=1}^{Q} d_j \frac{\partial\left[\sum\limits_{s=1}^{M} b_s w_{sj}\right]}{\partial b_s}\right) f'(netb_s + \gamma_s) \\ &= \left(\sum_{j=1}^{Q} d_j w_{sj}\right) f'(netb_s + \gamma_s). \end{aligned} \tag{15}$$

[2] In this paper, the example $(\mathbf{x}_i, Y_i)$ is simply excluded from the training set for BP-MLL if either Y_i or $\overline{Y}_i$ is an empty set.

Since $f'(netb_s + \gamma_s) = (1 + b_s)(1 - b_s)$, then substituting this equation into (15), we get

$$e_s = \left(\sum_{j=1}^{Q} d_j w_{sj} \right)(1 + b_s)(1 - b_s). \tag{16}$$

In order to reduce error, we can use *gradient descent* strategy, i.e., make the change of the weights be proportional to negative gradient:

$$\begin{aligned} \Delta w_{sj} &= -\alpha \frac{\partial E_i}{\partial w_{sj}} = -\alpha \frac{\partial E_i}{\partial netc_j} \frac{\partial netc_j}{\partial w_{sj}} \\ &= \alpha d_j \left[\frac{\partial \left(\sum_{s=1}^{M} b_s w_{sj} \right)}{\partial w_{sj}} \right] = \alpha d_j b_s, \end{aligned} \tag{17}$$

$$\begin{aligned} \Delta v_{hs} &= -\alpha \frac{\partial E_i}{\partial v_{hs}} = -\alpha \frac{\partial E_i}{\partial netb_s} \frac{\partial netb_s}{\partial v_{hs}} \\ &= \alpha e_s \left[\frac{\partial \left(\sum_{h=1}^{d} a_h v_{hs} \right)}{\partial v_{hs}} \right] = \alpha e_s a_h. \end{aligned} \tag{18}$$

The biases are changed according to (by fixing the activations of the extra input unit a_0 and hidden unit b_0 at 1)

$$\Delta \theta_j = \alpha d_j, \qquad \Delta \gamma_s = \alpha e_s, \tag{19}$$

where α is the *learning rate* whose value is in the range of $(0.0, 1.0)$.

Therefore, based on the above derivation, the training procedure of BP-MLL can be conveniently set up. In detail, in each training epoch of BP-MLL, the training instances are fed to the network one by one. For each multilabel instance $(\mathbf{x}_i, Y_i)$, the weights (and biases) are modified according to (17) through (19). After that, $(\mathbf{x}_{i+1}, Y_{i+1})$ is fed to the network and the training process is iterated until the global error E doesn't decrease any more or the number of training epochs increases to a threshold.

When a trained BP-MLL network is used in prediction for an unseen instance $\mathbf{x}$, the actual outputs c_j $(j = 1, 2, \ldots, Q)$ will be used for label ranking. The associated label set for $\mathbf{x}$ is determined by a threshold function $t(\mathbf{x})$, i.e., $Y = \{j | c_j > t(\mathbf{x}),\ j \in \mathcal{Y}\}$. A natural solution is to set $t(\mathbf{x})$ to be the zero constant function. Nevertheless, in this paper, the threshold learning mechanism used in the literature [8] is adopted, which generalizes the above natural solution. In detail, $t(\mathbf{x})$ is modeled by a linear function $t(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \cdot c(\mathbf{x}) + b$, where $c(\mathbf{x}) = (c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_Q(\mathbf{x}))$ is the Q-dimensional vector whose jth component corresponds to the actual output of the trained network on $\mathbf{x}$ on the jth class. The procedure used to learn the parameters of $t(\mathbf{x})$ (i.e., the weight vector $\mathbf{w}^{\mathrm{T}}$ and bias value b) is described as follows: For each multilabel training example $(\mathbf{x}_i, Y_i)$ $(1 \le i \le m)$, let $c(\mathbf{x}_i) = (c_1^i, c_2^i, \ldots, c_Q^i)$ and set the target values $t(\mathbf{x}_i)$ as

$$t(\mathbf{x}_i) = \arg\min_t \left(|\{k | k \in Y_i,\ c_k^i \le t\}| + |\{l | l \in \overline{Y}_i,\ c_l^i \ge t\}| \right). \tag{20}$$

When the minimum is not unique and the optimal values are a segment, the middle of this segment is chosen. Based on the above process, the parameters of the threshold function can be learned through solving the matrix equation $\Phi \cdot \mathbf{w}' = \mathbf{t}$. Here, matrix Φ has dimensions $m \times (Q + 1)$ whose ith row is $(c_1^i, \ldots, c_Q^i, 1)$, $\mathbf{w}'$ is the $(Q + 1)$-dimensional vector $(\mathbf{w}, b)$, and $\mathbf{t}$ is the m-dimensional vector $(t(\mathbf{x}_1), t(\mathbf{x}_2), \ldots, t(\mathbf{x}_m))$. In this paper, the linear least squares method is then applied to find the solution of the above equation. When a test instance $\mathbf{x}$ is given, it is first fed to the trained network to get the output vector $c(\mathbf{x})$. After that, the threshold value for $\mathbf{x}$ is computed via $t(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \cdot c(\mathbf{x}) + b$.

It is worth noting that the number of computations needed to evaluate the derivatives of the error function scales linearly with the size of the network. In words, let W be the total number of weights and biases of the BP-MLL network, i.e., $W = (d + 1) \times M + (M + 1) \times Q$ (usually $d \gg Q$ and $M > Q$). The total number of computations needed mainly comes from three phases, i.e., the forward propagation phase (computing b_s and c_j), the backward propagation phases (computing d_j and e_s), and the weights and biases update phase (computing Δw_{sj}, Δv_{hs}, $\Delta \theta_j$, and $\Delta \gamma_s$). In the forward propagation phase, most computational cost is spent in evaluating the sums as shown in (6) and (8), with the evaluation of the activation functions as shown in (4) and (7) representing a small overhead. Each term in the sum in (6) and (8) requires one multiplication and one addition, leading to an overall computational cost which is $\mathcal{O}(W)$. In the backward phase, as shown in (12) and (16), computing each d_j and e_i both requires $\mathcal{O}(Q)$ computations. Thus, the overall computational cost in the backward propagation phase is $\mathcal{O}(Q^2) + \mathcal{O}(Q \times M)$, which is at most $\mathcal{O}(W)$. As for the weights and biases update phase, it is evident that the overall computational cost is again $\mathcal{O}(W)$. To sum up, the total number of computations needed to update the BP-MLL network on each multilabel instance is $\mathcal{O}(W)$, indicating that the network training algorithm is very efficient. Thus, the overall training cost of BP-MLL is $\mathcal{O}(W \cdot m \cdot n)$, where m is the number of training examples and n is the total number of training epochs. The issues of the total number of epochs before a local solution is obtained and the possibility of getting stuck in a "bad" local solution will be discussed in Section 5.2.

4 Evaluation Metrics

Before presenting comparative results of each algorithm, evaluation metrics used in multilabel learning are introduced in this section. The performance evaluation of a multilabel learning system is different from that of a classical single-label learning system. Popular evaluation metrics used in single-label systems include accuracy, precision, recall, and F-measure [29]. In multilabel learning, the evaluation is much more complicated. Adopting the same notations as used in the beginning of Section 2, for a test set $S = \{(\mathbf{x}_1, Y_1), (\mathbf{x}_2, Y_2), ..., (\mathbf{x}_p, Y_p)\}$, the following multilabel evaluation metrics proposed in [28] are used in this paper:

1. *Hamming loss* evaluates how many times an instance-label pair is misclassified, i.e., a label not belonging

to the instance is predicted or a label belonging to the instance is not predicted. The performance is perfect when $\text{hloss}_S(h) = 0$; the *smaller* the value of $\text{hloss}_S(h)$, the better the performance.

$$\text{hloss}_S(h) = \frac{1}{p}\sum_{i=1}^{p}\frac{1}{Q}|h(\mathbf{x}_i)\Delta Y_i|, \qquad (21)$$

where Δ stands for the symmetric difference between two sets and Q is the total number of possible class labels. Note that when $|Y_i| = 1$ for all instances, a multilabel system is in fact a multiclass single-label one and the hamming loss is $\frac{2}{Q}$ times the usual classification error.

While hamming loss is based on the multilabel classifier $h(\cdot)$, the following metrics are defined based on the real-valued function $f(\cdot,\cdot)$ and concern the ranking quality of different labels for each instance.

2. *One-error* evaluates how many times the top-ranked label is not in the set of proper labels of the instance. The performance is perfect when $\text{one-error}_S(f) = 0$; the *smaller* the value of $\text{one-error}_S(f)$, the better the performance.

$$\text{one-error}_S(f) = \frac{1}{p}\sum_{i=1}^{p}[\![[\arg\max_{y\in\mathcal{Y}} f(\mathbf{x}_i, y)] \notin Y_i]\!] \qquad (22)$$

where, for any predicate π, $[\![\pi]\!]$ equals 1 if π holds and 0 otherwise. Note that, for single-label classification problems, the *one-error* is identical to ordinary classification error.

3. *Coverage* evaluates how far we need, on average, to go down the list of labels in order to cover all the proper labels of the instance. It is loosely related to precision at the level of perfect recall. The *smaller* the value of $\text{coverage}_S(f)$, the better the performance.

$$\text{coverage}_S(f) = \frac{1}{p}\sum_{i=1}^{p}\max_{y\in Y_i} rank_f(\mathbf{x}_i, y) - 1. \qquad (23)$$

As mentioned in the beginning of Section 2, $rank_f(\cdot,\cdot)$ is derived from the real-valued function $f(\cdot,\cdot)$, which maps the outputs of $f(\mathbf{x}_i, y)$ for any $y \in \mathcal{Y}$ to $\{1, 2, \ldots, Q\}$ such that if $f(\mathbf{x}_i, y_1) > f(\mathbf{x}_i, y_2)$, then $rank_f(\mathbf{x}_i, y_1) < rank_f(\mathbf{x}_i, y_2)$.

4. *Ranking loss* evaluates the average fraction of label pairs that are reversely ordered for the instance. The performance is perfect when $\text{rloss}_S(f) = 0$; the *smaller* the value of $\text{rloss}_S(f)$, the better the performance.

$$\text{rloss}_S(f) = \frac{1}{p}\sum_{i=1}^{p}\frac{1}{|Y_i||\overline{Y}_i|}$$
$$|\{(y_1, y_2)|f(\mathbf{x}_i, y_1) \le f(\mathbf{x}_i, y_2), (y_1, y_2) \in Y_i \times \overline{Y}_i\}|, \qquad (24)$$

where $\overline{Y}$ denotes the complementary set of Y in $\mathcal{Y}$.

5. *Average precision* evaluates the average fraction of labels ranked above a particular label $y \in Y$, which, actually, are in Y. It was originally used in information retrieval (IR) systems to evaluate the document ranking performance for query retrieval [26]. The performance is perfect when $\text{avgprec}_S(f) = 1$; the *bigger* the value of $\text{avgprec}_S(f)$, the better the performance.

$$\text{avgprec}_S(f) =$$
$$\frac{1}{p}\sum_{i=1}^{p}\frac{1}{|Y_i|}\sum_{y\in Y_i}\frac{|\{y'|rank_f(\mathbf{x}_i, y') \le rank_f(\mathbf{x}_i, y), y' \in Y_i\}|}{rank_f(\mathbf{x_i}, y)}. \qquad (25)$$

Note that, in the rest of this paper, the performance of each multilabel learning algorithm is evaluated based on the above five metrics.

5 Application to Functional Genomics

5.1 Functional Genomics

Bioinformatics, or computational biology, is a new interdisciplinary field, where techniques from applied mathematics, informatics, and computer science are applied to biology in order to model systems, extract information, understand processes, etc. Major efforts in this field include sequence alignment, protein structure prediction, analysis of protein-protein interactions, functional genomics, etc. Among these, *functional genomics* is of great importance; it aims at characterizing the function of genes and the proteins they encode in determining traits, physiology, or development of an organism. With the steady growing of the rate of genome sequencing and increasing of the amount of available data, computational functional genomics becomes both possible and necessary. It uses high-throughput techniques like DNA microarrays, proteomics, metabolomics, and mutation analysis to describe the function and interactions of genes. The range of recent work in computational functional genomics includes improved sequence similarity search algorithms, microarray expression analysis, computational prediction of protein secondary structure, differential genome analysis, etc. [3].

In this paper, the effectiveness of multilabel learning algorithms is evaluated through predicting the gene functional classes of the yeast *Saccharomyces cerevisiae*, which is one of the best-studied organisms. Specifically, the yeast data set studied in [8] and [21] is investigated. Each gene is described by the concatenation of microarray expression data and phylogenetic profile and is associated with a set of functional classes whose maximum size can be potentially more than 190. In order to make it easier, Elisseeff and Weston preprocessed the data set where only the known structure of the functional classes is used. Actually, the whole set of functional classes is structured into hierarchies up to four levels deep.[3] In this paper, the same data set as used in [8] is adopted. In this data set, only functional classes in the top hierarchy (as depicted in Fig. 2) are considered. The resulting multilabel data set contains 2,417 genes, each represented by a 103-dimensional feature

[3] See http://mips.gsf.de/proj/yeast/catalogues/funcat/ for more details.

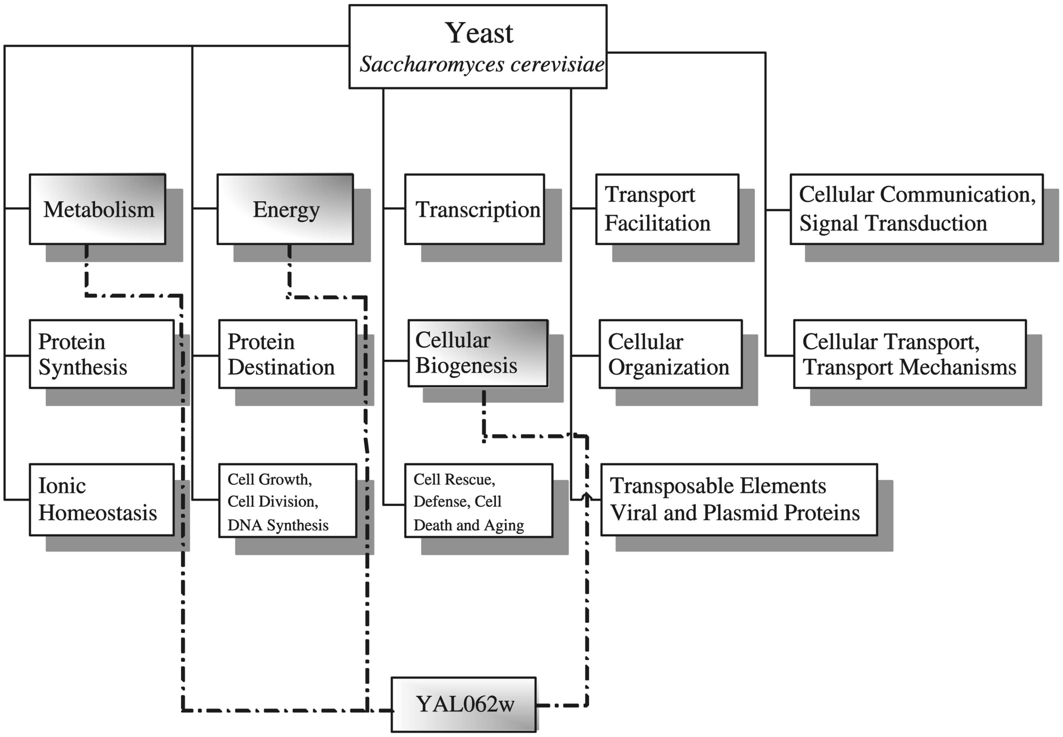

Yeast
Saccharomyces cerevisiae
Metabolism
Energy
Transcription
Transport Facilitation
Cellular Communication, Signal Transduction
Protein Synthesis
Protein Destination
Cellular Biogenesis
Cellular Organization
Cellular Transport, Transport Mechanisms
Ionic Homeostasis
Cell Growth, Cell Division, DNA Synthesis
Cell Rescue, Defense, Cell Death and Aging
Transposable Elements Viral and Plasmid Proteins
YAL062w

Fig. 2. First level of the hierarchy of the yeast gene functional classes. One gene, for instance, the one named YAL062w, can belong to several classes (shaded in gray) of the 14 possible classes.

vector. There are 14 possible class labels and the average number of labels for each gene is 4.24 ± 1.57.

5.2 Results

As reviewed in Section 2, there have been several approaches to solving multilabel problems. In this paper, BP-MLL is compared with the boosting-style algorithm BOOSTEXTER[4] [28], the multilabel decision tree ADTBOOST.MH[5] [5], and the multilabel kernel method RANK-SVM [8], which are all general-purpose multilabel learning algorithms applicable to various multilabel problems. In addition, BP-MLL is also compared with BASICBP, i.e., the intuitive implementation of neural networks for multilabel learning as described in Section 3, to see whether the more complex global error function as defined in (3) will perform better than the intuitive solution.

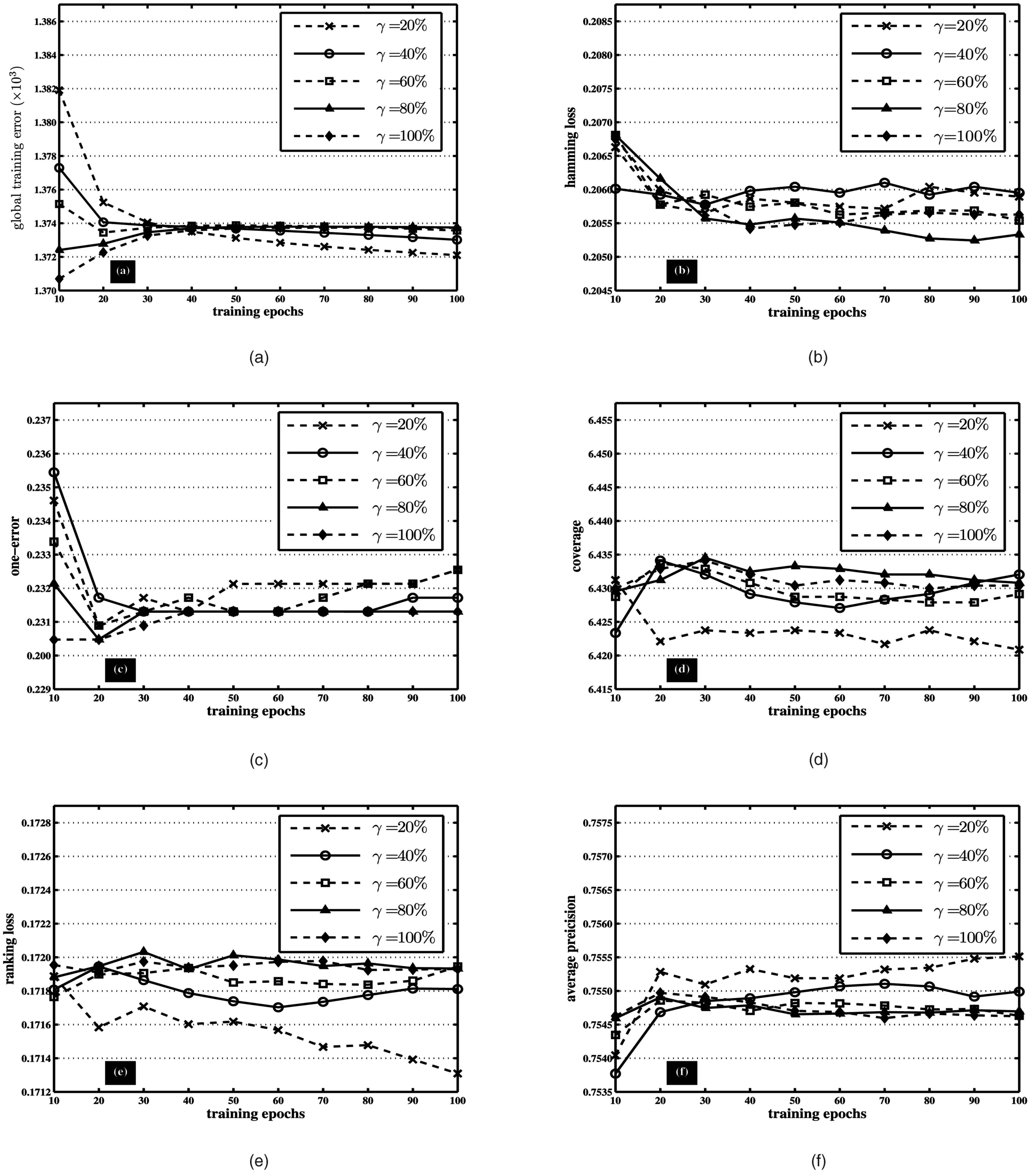
For BP-MLL, the learning rate is set to be 0.05. The number of hidden units of the network is set to be 20 percent to 100 percent of the number of input units with an interval of 20 percent, while the number of training epochs varies from 10 to 100 with an interval of 10. Furthermore, in order to avoid overfitting, a regularization term equal to one tenth of the sum of squares of all network weights and biases is added to the global error function. For BOOSTEXTER [28] and ADTBOOST.MH [5], the number of boosting rounds is set to be 500 and 50, respectively, because on the yeast data set (also the Reuters collection studied in the next section), the performance of these two algorithms will not significantly change after the specified boosting rounds. For RANK-SVM [8], polynomial kernels with degree 8 are used, which yield the best performance as shown in [8]. For BASICBP, the number of training epochs is set to be 1,500 and the number of hidden units is set to be four times of the number of input units to yield comparable results.

Tenfold cross validation is performed on this data set. In detail, the original data set is randomly divided into 10 parts, each with approximately the same size. In each fold, one part is held out for testing and the learning algorithm is trained on the remaining data. The above process is iterated 10 times so that each part is used as the test data exactly once, where the averaged metric values out of 10 runs are reported for the algorithm.

Fig. 3. illustrates how the global training error and various metric values of BP-MLL change as the number of training epochs increases. Different curves correspond to different number of hidden neurons ($= \gamma \times$ input dimensionality) used by BP-MLL. Fig. 3 shows that when γ is set to be 20 percent, BP-MLL performs comparable to other values of γ in terms of *hamming loss* and *one-error* (Figs. 3b and 3c), while slightly better than other values of γ in terms of *coverage*, *ranking loss*, and *average precision* (Figs. 3d, 3e, and 3f). Furthermore, after 40 epochs of training, the global training error (Fig. 3a) and those evaluation metric values (Figs. 3b, 3c, 3d, 3e, and 3f) of BP-MLL will not significantly change. Therefore, for the sake of computational cost, all the results of BP-MLL shown in the rest of this paper are obtained with the number of hidden units set to be 20 percent of the number of input units. The number of training epochs for BP-MLL is fixed to be 100.

[4] Program available at http://www.cs.princeton.edu/~schapire/boostexter.html.

[5] The algorithm and a graphical user interface are available at http://www.grappa.univ-lille3.fr/grappa/index.php3?info=logiciels. Furthermore, *ranking loss* is not provided by the outputs of this implementation.



(a)

(b)

(c)

(d)

(e)

(f)

Fig. 3. The performance of BP-MLL with different number of hidden neurons ($= \gamma \times$ input dimensionality) changes as the number of training epochs increases. (a) *Global training error.* (b) *Hamming loss.* (c) *One-error.* (d) *Coverage.* (e) *Ranking loss.* (f) *Average precision.*

Table 1 reports the experimental results of BP-MLL and other multilabel learning algorithms on the yeast data, where the best result on each metric is shown in bold face. To make a clearer view of the relative performance between each algorithm, a partial order "$\succ$ " is defined on the set of all comparing algorithms for each evaluation criterion, where $A1 \succ A2$ means that the performance of algorithm $A1$ is statistically better than that of algorithm $A2$ on the specific metric (based on two-tailed paired t-test at 5 percent significance level). The partial order on all the comparing algorithms in terms of each evaluation criterion is summarized in Table 2, where the p-value shown in the parentheses further gives a quantification of the significance level.

Note that the partial order "$\succ$ " only measures the relative performance between two algorithms $A1$ and $A2$ on one specific evaluation criterion. However, it is quite

TABLE 1
Experimental Results of Each Multilabel Learning Algorithm (Mean ± Std. Deviation) on the Yeast Data

EVALUATION CRITERION	ALGORITHM				
	BP-MLL	BOOSTEXTER	ADTBOOST.MH	RANK-SVM	BASICBP
HAMMING LOSS	**0.206±0.011**	0.220±0.011	0.207±0.010	0.207±0.013	0.209±0.008
ONE-ERROR	**0.233±0.034**	0.278±0.034	0.244±0.035	0.243±0.039	0.245±0.032
COVERAGE	6.421±0.237	6.550±0.243	**6.390±0.203**	7.090±0.503	6.653±0.219
RANKING LOSS	**0.171±0.015**	0.186±0.015	N/A	0.195±0.021	0.184±0.017
AVERAGE PRECISION	**0.756±0.021**	0.737±0.022	0.744±0.025	0.750±0.026	0.740±0.022

TABLE 2
Relative Performance between Each Multilabel Learning Algorithm on the Yeast Data

EVALUATION CRITERION	ALGORITHM
	$A1$-BP-MLL; $A2$-BOOSTEXTER; $A3$-ADTBOOST.MH; $A4$-RANK-SVM; $A5$-BASICBP
HAMMING LOSS	$A1 \succ A2$ ($p = 2.5 \times 10^{-4}$), $A3 \succ A2$ ($p = 8.4 \times 10^{-5}$), $A4 \succ A2$ ($p = 4.7 \times 10^{-3}$), $A5 \succ A2$ ($p = 6.1 \times 10^{-4}$)
ONE-ERROR	$A1 \succ A2$ ($p = 1.4 \times 10^{-3}$), $A1 \succ A5$ ($p = 1.4 \times 10^{-2}$), $A3 \succ A2$ ($p = 7.2 \times 10^{-4}$), $A4 \succ A2$ ($p = 4.4 \times 10^{-2}$), $A5 \succ A2$ ($p = 2.0 \times 10^{-3}$)
COVERAGE	$A1 \succ A4$ ($p = 7.0 \times 10^{-4}$), $A1 \succ A5$ ($p = 7.1 \times 10^{-5}$), $A2 \succ A4$ ($p = 8.4 \times 10^{-3}$), $A2 \succ A5$ ($p = 2.7 \times 10^{-2}$), $A3 \succ A2$ ($p = 8.9 \times 10^{-3}$), $A3 \succ A4$ ($p = 8.9 \times 10^{-4}$), $A3 \succ A5$ ($p = 7.4 \times 10^{-6}$), $A5 \succ A4$ ($p = 1.3 \times 10^{-2}$)
RANKING LOSS	$A1 \succ A2$ ($p = 1.3 \times 10^{-4}$), $A1 \succ A4$ ($p = 6.3 \times 10^{-3}$), $A1 \succ A5$ ($p = 1.0 \times 10^{-5}$)
AVERAGE PRECISION	$A1 \succ A2$ ($p = 1.3 \times 10^{-4}$), $A1 \succ A3$ ($p = 1.4 \times 10^{-3}$), $A1 \succ A5$ ($p = 6.9 \times 10^{-6}$)
TOTAL ORDER	BP-MLL(**11**)>ADTBOOST.MH(**4**)>{RANK-SVM(**-3**), BASICBP(**-3**)}>BOOSTEXTER(**-9**)

possible that $A1$ performs better than $A2$ in terms of some metrics but worse than $A2$ in terms of others. In this case, it is hard to judge which algorithm is superior. Therefore, in order to give an overall performance assessment of an algorithm, a score is assigned to this algorithm which takes account of its relative performance with other algorithms on all metrics. Concretely, for each evaluation criterion, for each possible pair of algorithms $A1$ and $A2$, if $A1 \succ A2$ holds, then $A1$ is rewarded by a positive score +1 and $A2$ is penalized by a negative score -1. Based on the accumulated score of each algorithm on all evaluation criteria, a total order ”> ” is defined on the set of all comparing algorithms as shown in the last line of Table 2, where $A1 > A2$ means that $A1$ performs better than $A2$ on the yeast data. The accumulated score of each algorithm is also shown in bold face in the parentheses.

Table 2 shows that BP-MLL performs fairly well in terms of all the evaluation criteria, where, on all the evaluation criteria, no algorithm has outperformed BP-MLL. Especially, BP-MLL outperforms all the other algorithms with respect to *ranking loss*[6] since minimization of the global error function of BP-MLL could be viewed as approximately optimizing the ranking loss criterion. Furthermore, BP-MLL outperforms BASICBP on all the evaluation criteria except hamming loss, on which the two algorithms are comparable. These facts illustrate that the more complex global error function employed by BP-MLL (as defined in (3)) really works better than the intuitive one employed by BASICBP (as defined in (1) and (2)). It is also worth noting that BOOSTEXTER performs quite poorly compared to other algorithms. As indicated in the literature [8], the reason may be that the simple decision function realized by this method is not suitable to learn from the yeast data set. On the whole (as shown by the total order), BP-MLL outperforms all the other algorithms on the multilabel learning problem of yeast functional genomics.

Table 3 reports the computation time consumed by each multilabel learning algorithm on the yeast data, where all experiments are conducted on an HP Server equipped with 4G RAM and four Intel Xeron™ CPUs each running at 2.80 GHz.[7] As shown in Table 3, BP-MLL consumes much more time in the training phase than BOOSTEXTER, ADTBOOST.MH and BASICBP, mainly due to its *complex* global error function, which needs to be optimized, and the *iterative* processing of training examples. On the other hand, although the training complexity of BP-MLL is high, the time cost of BP-MLL on testing unseen examples is quite trivial.

[6] Note that *ranking loss* is not provided in the outputs of the ADTBOOST.MH algorithm.

[7] Codes of BOOSTEXTER and ADTBOOST.MH are written in C language, while those of BP-MLL, RANK-SVM, and BASICBP are developed with MATLAB™. Note that programs written in C usually run several times faster than those written in MATLAB.

TABLE 3
Computation Time of Each Multilabel Learning Algorithm (Mean ± Std. Deviation) on the Yeast Data

Computation	Algorithm				
Time	Bp-mll	BoosTexter	Adtboost.MH	Rank-svm	BasicBp
Training Phase (Hours)	6.989±0.235	0.154±0.015	0.415±0.031	7.723±5.003	0.743±0.002
Testing Phase (Seconds)	0.739±0.037	1.100±0.123	0.942±0.124	1.255±0.052	1.302±0.030

Training time is measured in hours and testing time is measured in seconds.

As analyzed at the end of Section 3.3, the total cost of training a BP-MLL network scales to $\mathcal{O}(W \cdot m \cdot n)$. Here, W is the total number of weights and biases of the network, m is the number of training examples, and n is the total number of training epochs. In order to illustrate how many training epochs are needed before a local solution is obtained and the possibility of getting stuck in a "bad" local solution, the following experiments are conducted. In detail, 500 examples are randomly selected from the yeast data (in total, 2,417 examples) constituting the test set and the remaining 1,917 examples form the potential training set. After that, 200 runs of experiments are performed where, in each run, 1,000 examples are randomly selected from the potential training set to train a BP-MLL neural network and the trained model is then evaluated on the test set. The maximum number of training epochs is set to be 200 and the training process terminates as long as the global training error of BP-MLL does not decrease enough. Concretely, let E^t denote the global training error of BP-MLL at the tth training epoch, the training process will terminate before reaching the maximum number of training epochs if the condition of $E^t - E^{t+1} \leq \epsilon \cdot E^t$ is satisfied. It is obvious that the smaller the value of ϵ, the more training epochs are executed before termination. In this paper, ϵ is set to be 10^{-6} for illustration purpose.

Fig. 4 gives the quantile plot regarding the number of training epochs of BP-MLL out of 200 runs of experiments, where each point (x, y) in the plot means that the number of training epochs of BP-MLL will be smaller than or equal to y in $100 \cdot x$ percent cases out of the 200 runs. It was shown that the training process will terminate before 80 epochs in about 50 percent of cases and before 140 epochs in about 80 percent of cases. Furthermore, Table 4 summarizes the statistics of each evaluation criterion out of 200 runs of experiments, where the minimal, maximal, and mean (together with standard deviation) values of each metric are illustrated. In this paper, the metric values that fall one standard deviation outside of the mean value will be regarded as "bad" local solutions.[8] Based on this, the probability of getting stuck in a "bad" local solution with regard to a specific metric can be calculated as shown in the last column of Table 4. It is revealed that BP-MLL will get stuck in a "bad" local solution with no more than 20 percent probability in terms of any evaluation metric. Since there is no criterion available for judging whether the learning algorithm has terminated as a "bad" or "good" during the training phase, one possible solution is to train many BP-MLL neural networks based on different initial configurations and, then, combine their predictions. In this way, the power of ensemble learning [34] may be utilized to achieve strong generalization ability and it will be an interesting issue for future work as indicated in Section 7.

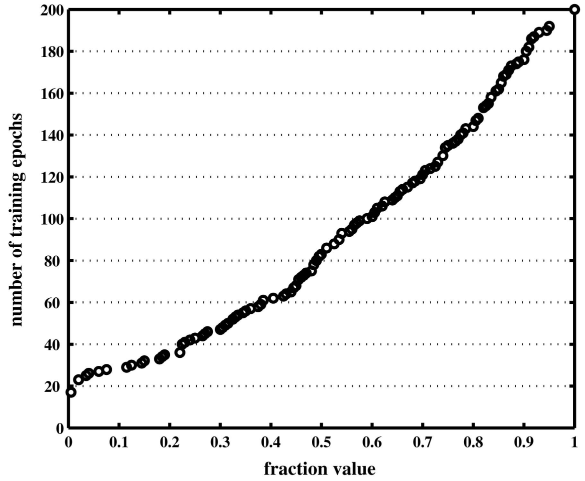


Fig. 4. Quantile plot regarding the number of training epochs of BP-MLL out of 200 runs of experiments.

6 Application to Text Categorization

6.1 Text Categorization

Text categorization (TC) is the task of building learning systems capable of classifying text (or hypertext) documents under one or more of a set of predefined categories or subject codes [15]. Due to the increased availability of ever larger numbers of text documents in digital form and the ensuing need to organize them for easier use, TC has become one of the key techniques for handling and organizing text data. TC is now being applied to a wide range of applications, including document organization, text filtering, automated metadata generation, word sense disambiguation, Web page categorization under hierarchical catalogs, etc. [29].

In the 1980s, the most popular approach to TC was based on *knowledge engineering* (KE) techniques, which aim at manually defining a set of logical rules encoding expert knowledge on how to classify documents under the given categories. Since the early 1990s, the *machine learning* (ML) approach to TC has gradually gained popularity, where a general inductive process is employed to automatically build a text classifier by learning from a set of preclassified documents. The advantages of the ML approach over the KE approach lie in the fact that the former can achieve comparable performance to that achieved by human experts

[8] For the metric *average precision*, "fall outside" means the value is more than one standard deviation smaller than the mean value. For the other four evaluation criteria, "fall outside" means the value is more than one standard deviation larger than the mean value.

TABLE 4
Statistics of Each Evaluation Criterion out of 200 Runs of Experiments,
Where Values Falling One Standard Deviation Outside of the Mean Value Are Regarded as "Bad" Local Solutions

Evaluation Criterion	Statistics out of 200 Runs of Experiments			
	Min	Max	Mean±Std. Deviation	Prob. of "Bad" Loc. Sol.
Hamming Loss	0.198	0.221	0.206±0.004	14.0%
One-error	0.200	0.244	0.228±0.008	14.0%
Coverage	6.354	6.890	6.542±0.106	15.0%
Ranking Loss	0.170	0.187	0.176±0.003	18.0%
Average Precision	0.736	0.762	0.749±0.004	14.5%

while at the same time considerably saves the experts' labor costs [29].

The first step in ML-based TC is to transform documents, which typically are strings of characters, into a representation suitable for the learning algorithm and the classification task. The most commonly used document representation is the so-called vector space model, where each document d is represented as a vector of term weights $\vec{d} =< w_1, w_2, \ldots, w_{|\mathcal{T}|} >$. Here, $\mathcal{T}$ is the set of terms (usually the set of words) that occur at least once in at least one document of the training set, and w_i approximately represents how much term $t_i \in \mathcal{T}$ contributes to the semantics of document d. Various approaches are available to determine the term weights, such as Boolean weighting (set w_i to 1 if term t_i occurs in d and 0 otherwise), frequency-based weighting (set w_i to the frequency of term t_i in d) and the widely used *tf-idf* (term frequency-inverse document frequency) weighting [29]. Note that the dimensionality of the vector space may be prohibitively high (the term set $\mathcal{T}$ could contain hundreds of thousands of terms) for any ML algorithm to efficiently build classifiers; *dimensionality reduction* (DR) techniques are necessary to reduce the size of the vector space from $|\mathcal{T}|$ to $|\mathcal{T}'| \ll |\mathcal{T}|$. A lot of DR methods have been proposed, such as *term selection* methods based on document frequency, information gain, mutual information, χ^2 statistic, etc., and *term extraction* methods based on term clustering and latent semantic indexing [33]. Various ML methods have been applied to solve TC problems, including decision trees, support vector machines, nearest neighbor classifiers, Bayesian probabilistic classifiers, inductive rule learning algorithms, and more [29]. In most cases, the multilabel learning problem of TC is decomposed into multiple independent binary classification problems where a separate classifier is built for each category. For more information about TC research, an excellent and comprehensive survey on this topic is given in [29].

6.2 Results

The Reuters collection is the most commonly used collection for TC evaluation and various versions of this collection have been studied in the TC community [29], [32]. In this paper, the Reuters-21578 Distribution 1.0[9] is used to further evaluate the performance of BP-MLL and other multilabel learning algorithms. Reuters-21578 consists of 21,578 Reuters newswire documents that appeared in 1987, where less than half of the documents have human-assigned topic labels. All documents without any topic label or with empty main text are discarded from the collection. Each remaining document belongs to at least one of the 135 possible topics (categories), where a "subcategory" relation governs categories and nine of them constitute the top level of this hierarchy. In this paper, only those top level categories are used to label each remaining document.

For each document, the following preprocessing operations are performed before experiments: All words were converted to lower case, punctuation marks were removed, and "function words" such as "of" and "the" on the SMART stop-list [25] were removed. Additionally, all strings of digits were mapped to a single common token. Following the same data set generation scheme as used in [28] and [5], subsets of the k categories with the largest number of articles for $k = 3, \ldots, 9$ are selected resulting in seven different data sets denoted as FIRST3, FIRST4, ..., FIRST9. The simple term selection method based on *document frequency* (the number of documents containing a specific term) is used to reduce the dimensionality of each data set. Actually, only the 2 percent of words with the highest document frequency are retained in the final vocabulary.[10] Note that other term selection methods such as *information gain* could also be adopted. Each document in the data set is described as a feature vector using the "*Bag-of-Words*" representation [7], i.e., each dimension of the feature vector corresponds to the number of times a word in the vocabulary appears in this document. Table 5 summarizes the characteristics of the preprocessed data sets.

Adopting the same validation mechanism as used in [28] and [5], threefold cross validation is performed on each data set. In detail, each data set is randomly divided into three parts, each with approximately the same size. In each fold, one part is held out for testing and the learning algorithm is trained on the remaining data. The above process is iterated three times so that each part is used as the test data exactly once, where the averaged metric values out of three runs are reported for the algorithm.

[9] Data set available at http://www.daviddlewis.com/resources/testcollections/reuters21578/.

[10] It is worth noting that principles used in document weighting and dimensionality reduction may have some differences. Although in several document weighting schemes, such as *tf-idf* weighting [29], words that occur in most documents are assumed to be less useful in representing individual documents. For dimensionality reduction, however, the words with the highest document frequency, excluding those "function words" which have already been removed from the vocabulary using the SMART stop-list [25], are representative in describing the information contained in the corpus. Actually, based on a series of experiments, Yang and Pedersen [33] have shown that, based on document frequency, it is possible to reduce the dimensionality by a factor of 10 with no loss in effectiveness and by a factor of 100 with just a small loss.

TABLE 5
Characteristics of the Preprocessed Data Sets

Data Set	Number of Categories	Number of Documents	Vocabulary Size	*PMC*	*ANL*
First3	3	7,258	529	0.74%	1.0074
First4	4	8,078	598	1.39%	1.0140
First5	5	8,655	651	1.98%	1.0207
First6	6	8,817	663	3.43%	1.0352
First7	7	9,021	677	3.62%	1.0375
First8	8	9,158	683	3.81%	1.0396
First9	9	9,190	686	4.49%	1.0480

PMC denotes the percentage of documents belonging to more than one category and ANL denotes the average number of labels for each document.

TABLE 6
Experimental Results of Each Multilabel Learning Algorithm on the Reuters-21578 Collection in Terms of Hamming Loss

Data	Algorithm				
Set	Bp-mll	BoosTexter	Adtboost.MH	Rank-svm	BasicBp
First3	0.0368	**0.0236**	0.0404	0.0439	0.0433
First4	0.0256	**0.0250**	0.0439	0.0453	0.0563
First5	**0.0257**	0.0260	0.0469	0.0592	0.0433
First6	0.0271	**0.0262**	0.0456	0.0653	0.0439
First7	0.0252	**0.0249**	0.0440	0.0576	0.0416
First8	0.0230	**0.0229**	0.0415	0.0406	0.0399
First9	0.0231	**0.0226**	0.0387	0.0479	0.0387
Average	0.0266	**0.0245**	0.0430	0.0514	0.0439

TABLE 7
Experimental Results of Each Multilabel Learning Algorithm on the Reuters-21578 Collection in Terms of One-Error

Data	Algorithm				
Set	Bp-mll	BoosTexter	Adtboost.MH	Rank-svm	BasicBp
First3	0.0506	**0.0287**	0.0510	0.0584	0.0558
First4	0.0420	**0.0384**	0.0730	0.0647	0.0847
First5	0.0505	**0.0475**	0.0898	0.0873	0.0842
First6	0.0597	**0.0569**	0.1024	0.1064	0.1055
First7	**0.0632**	0.0655	0.1206	0.1438	0.1147
First8	**0.0673**	0.0679	0.1249	0.0997	0.1422
First9	**0.0708**	0.0719	0.1383	0.1288	0.1489
Average	0.0577	**0.0538**	0.1000	0.0985	0.1051

TABLE 8
Experimental Results of Each Multilabel Learning Algorithm on the Reuters-21578 Collection in Terms of Coverage

Data	Algorithm				
Set	Bp-mll	BoosTexter	Adtboost.MH	Rank-svm	BasicBp
First3	0.0679	**0.0416**	0.0708	0.0869	0.0761
First4	0.0659	**0.0635**	0.1187	0.1234	0.1419
First5	0.0921	**0.0916**	0.1624	0.1649	0.1390
First6	**0.1363**	0.1397	0.2438	0.2441	0.2552
First7	**0.1488**	0.1635	0.2882	0.3301	0.2837
First8	**0.1628**	0.1815	0.3194	0.3279	0.4358
First9	**0.1905**	0.2208	0.3811	0.4099	0.4995
Average	**0.1235**	0.1289	0.2263	0.2410	0.2616

TABLE 9
Experimental Results of Each Multilabel Learning Algorithm on the Reuters-21578 Collection in Terms of Ranking Loss

Data	Algorithm				
Set	Bp-mll	BoosTexter	Adtboost.MH	Rank-svm	BasicBp
First3	0.0304	**0.0173**	N/A	0.0398	0.0345
First4	0.0172	**0.0164**	N/A	0.0363	0.0435
First5	0.0176	**0.0173**	N/A	0.0354	0.0302
First6	**0.0194**	0.0199	N/A	0.0406	0.0448
First7	**0.0177**	0.0198	N/A	0.0471	0.0407
First8	**0.0166**	0.0190	N/A	0.0393	0.0563
First9	**0.0166**	0.0197	N/A	0.0434	0.0563
Average	0.0193	**0.0185**	N/A	0.0403	0.0438

TABLE 10
Experimental Results of Each Multilabel Learning Algorithm on the Reuters-21578 Collection in Terms of Average Precision

Data	Algorithm				
Set	Bp-mll	BoosTexter	Adtboost.MH	Rank-svm	BasicBp
First3	0.9731	**0.9848**	0.9725	0.9673	0.9699
First4	0.9775	**0.9791**	0.9587	0.9615	0.9512
First5	0.9719	**0.9730**	0.9481	0.9491	0.9530
First6	0.9651	**0.9658**	0.9367	0.9345	0.9343
First7	**0.9629**	0.9603	0.9326	0.9134	0.9286
First8	**0.9602**	0.9579	0.9211	0.9336	0.9071
First9	**0.9570**	0.9540	0.9112	0.9149	0.8998
Average	0.9668	**0.9679**	0.9401	0.9392	0.9348

The experimental results on each evaluation criterion are reported in Tables 6, 7, 8, 9, and 10, where the best result on each data set is shown in bold face. Parameter configuration for each algorithm is the same as that used in Section 5. Similarly to the yeast data, the partial order "$\succ$ " (based on a two-tailed paired t-test at a 5 perecent significance level) and total order "$>$ " are also defined on the set of all comparing algorithms, which are shown in Table 11. Again, as in Table 2, p-value is given to indicate the level of significance and the accumulated score of each algorithm is shown in bold face in the parentheses in the last line.

Table 11 shows that BP-MLL and BOOSTEXTER are both superior to ADTBOOST.MH, RANK-SVM, and BASICBP on all evaluation criteria (*ranking loss* is not available for ADTBOOST.MH). Furthermore, as shown in Tables 7, 8, 9, and 10, BP-MLL is inferior to BOOSTEXTER when the number of categories is small (from FIRST3 to FIRST6). However, when the corresponding data sets get more difficult to learn from, i.e., the number of categories becomes larger and the portion of documents belonging to more than one category increases (from FIRST7 to FIRST9), BP-MLL outperforms BOOSTEXTER. In addition, the fact that BP-MLL outperforms BASICBP on all the evaluation criteria again proves that BP-MLL works better than BASICBP when the more complex global error function (as defined in (3)) is employed to learn from the multilabel training examples. On the whole (as shown by the total order), BP-MLL is comparable to BOOSTEXTER but is superior to all the other algorithms on the Reuters collection.

As with the yeast data, Table 12 reports the computation time consumed by each multilabel learning algorithm on

TABLE 11
Relative Performance between Each Multilabel Learning Algorithm on the Reuters-21578 Collection

EVALUATION CRITERION	ALGORITHM
	$A1$-BP-MLL; $A2$-BOOSTEXTER; $A3$-ADTBOOST.MH; $A4$-RANK-SVM; $A5$-BASICBP
HAMMING LOSS	$A1 \succ A3$ ($p = 3.2 \times 10^{-4}$), $A1 \succ A4$ ($p = 9.4 \times 10^{-4}$), $A1 \succ A5$ ($p = 6.7 \times 10^{-4}$), $A2 \succ A3$ ($p = 8.2 \times 10^{-8}$), $A2 \succ A4$ ($p = 1.2 \times 10^{-4}$), $A2 \succ A5$ ($p = 7.5 \times 10^{-5}$), $A3 \succ A4$ ($p = 2.4 \times 10^{-2}$)
ONE-ERROR	$A1 \succ A3$ ($p = 2.4 \times 10^{-3}$), $A1 \succ A4$ ($p = 4.0 \times 10^{-3}$), $A1 \succ A5$ ($p = 2.3 \times 10^{-3}$), $A2 \succ A3$ ($p = 1.7 \times 10^{-4}$), $A2 \succ A4$ ($p = 6.9 \times 10^{-4}$), $A2 \succ A5$ ($p = 3.1 \times 10^{-4}$)
COVERAGE	$A1 \succ A3$ ($p = 5.8 \times 10^{-3}$), $A1 \succ A4$ ($p = 5.4 \times 10^{-3}$), $A1 \succ A5$ ($p = 1.8 \times 10^{-2}$), $A2 \succ A3$ ($p = 1.6 \times 10^{-3}$), $A2 \succ A4$ ($p = 1.8 \times 10^{-3}$), $A2 \succ A5$ ($p = 1.1 \times 10^{-2}$), $A3 \succ A4$ ($p = 4.6 \times 10^{-2}$)
RANKING LOSS	$A1 \succ A4$ ($p = 1.5 \times 10^{-4}$), $A1 \succ A5$ ($p = 2.6 \times 10^{-3}$), $A2 \succ A4$ ($p = 1.4 \times 10^{-6}$), $A2 \succ A5$ ($p = 3.5 \times 10^{-4}$)
AVERAGE PRECISION	$A1 \succ A3$ ($p = 2.9 \times 10^{-3}$), $A1 \succ A4$ ($p = 2.7 \times 10^{-3}$), $A1 \succ A5$ ($p = 4.1 \times 10^{-3}$), $A2 \succ A3$ ($p = 3.5 \times 10^{-4}$), $A2 \succ A4$ ($p = 4.8 \times 10^{-4}$), $A2 \succ A5$ ($p = 1.0 \times 10^{-3}$)
TOTAL ORDER	{BP-MLL(**14**), BOOSTEXTER(**14**)}>ADTBOOST.MH(**-6**)>BASICBP(**-10**)>RANK-SVM(**-12**)

TABLE 12
Computation Time of Each Multilabel Learning Algorithm on the Reuters-21578 Collection

DATA SET	ALGORITHM									
	BP-MLL		BOOSTEXTER		ADTBOOST.MH		RANK-SVM		BASICBP	
	TrPhase	*TePhase*	*TrPhase*	*TePhase*	*TrPhase*	*TePhase*	*TrPhase*	*TePhase*	*TrPhase*	*TePhase*
FIRST3	44.088	4.552	0.115	2.938	0.776	2.561	2.873	28.594	6.395	7.094
FIRST4	57.442	6.891	0.202	3.785	1.055	2.720	5.670	37.328	12.264	6.969
FIRST5	60.503	8.547	0.237	5.575	1.188	3.933	8.418	48.078	20.614	12.969
FIRST6	69.615	9.328	0.277	7.331	1.539	4.966	15.431	50.969	20.274	13.766
FIRST7	73.524	14.083	0.321	8.305	1.739	5.837	16.249	55.016	22.792	18.922
FIRST8	74.220	15.292	0.343	9.841	1.940	6.945	26.455	55.141	20.927	17.219
FIRST9	75.291	17.922	0.373	11.817	2.107	7.494	28.106	48.141	23.730	23.578
AVERAGE	64.955	10.945	0.267	7.085	1.478	4.922	14.743	46.181	18.142	14.360

Training time (denoted as TrPhase) is measured in hours while testing time (denoted as TePhase) is measured in seconds.

the Reuters collection. As shown in Table 12, BP-MLL consumes much more time in the training phase than all the other algorithms but is quite efficient in the testing phase to predict labels for unseen examples.

7 CONCLUSION

In this paper, a neural network algorithm, named BP-MLL, which is the multilabel version of Backpropagation, is proposed. Through employing a new error function, BP-MLL captures the characteristics of multilabel learning, i.e., the labels belonging to an instance should be ranked higher than those not belonging to that instance. Applications to two real-world multilabel learning problems, i.e., functional genomics and text categorization, show that BP-MLL achieves superior performance to some well-established multilabel learning methods. Furthermore, as a common characteristic of neural network methods, the computational complexity of BP-MLL in the training phase is high while the time cost of making predictions based on the trained model is quite trivial.

Recent research has shown that neural network ensemble could significantly improve the generalization ability of neural network-based learning systems, which has become a hot topic in both machine learning and neural network communities [34]. So, it is interesting to see that whether better results could be obtained with ensembles of BP-MLL networks.

ACKNOWLEDGMENTS

The authors wish to express their gratitude toward the associate editor and the anonymous reviewers because their valuable comments and suggestions significantly improved this paper. The authors also want to thank A. Elisseeff and J. Weston for the yeast data and the implementation details of RANK-SVM. This work was supported by the National Science Foundation of China under Grant No. 60473046 and the National Science Fund for Distinguished Young Scholars of China under Grant No. 60325207.

REFERENCES

[1] C.M. Bishop, *Neural Networks for Pattern Recognition.* New York: Oxford Univ. Press, 1995.

[2] M.R. Boutell, J. Luo, X. Shen, and C.M. Brown, "Learning Multi-Label Scene Classification," *Pattern Recognition,* vol. 37, no. 9, pp. 1757-1771, 2004.

[3] A. Clare, "Machine Learning and Data Mining for Yeast Functional Genomics," PhD dissertation, Dept. of Computer Science, Univ. of Wales Aberystwyth, 2003.

[4] A. Clare and R.D. King, "Knowledge Discovery in Multi-Label Phenotype Data," *Lecture Notes in Computer Science,* L.D. Raedt and A. Siebes, eds., vol. 2168, pp. 42-53. Berlin: Springer, 2001.

[5] F.D. Comité, R. Gilleron, and M. Tommasi, "Learning Multi-Label Alternating Decision Tree from Texts and Data," *Lecture Notes in Computer Science,* P. Perner and A. Rosenfeld, eds., vol. 2734, pp. 35-49. Berlin: Springer, 2003.

[6] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistics Soc. B,* vol. 39, no. 1, pp. 1-38, 1977.

[7] S.T. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive Learning Algorithms and Representation for Text Categorization," *Proc. Seventh ACM Int'l Conf. Information and Knowledge Management (CIKM '98),* pp. 148-155, 1998.

[8] A. Elisseeff and J. Weston, "A Kernel Method for Multi-Labelled Classification," *Advances in Neural Information Processing Systems,* T.G. Dietterich, S. Becker, and Z. Ghahramani, eds., vol. 14, pp. 681-687, 2002.

[9] Y. Freund and L. Mason, "The Alternating Decision Tree Learning Algorithm," *Proc. 16th Int'l Conf. Machine Learning (ICML '99),* pp. 124-133, 1999.

[10] Y. Freund and R.E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Computer and System Sciences,* vol. 55, no. 1, pp. 119-139, 1997.

[11] S. Gao, W. Wu, C.-H. Lee, and T.-S. Chua, "A Maximal Figure-of-Merit Learning Approach to Text Categorization," *Proc. 26th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '03),* pp. 174-181, 2003.

[12] S. Gao, W. Wu, C.-H. Lee, and T.-S. Chua, "A MFoM Learning Approach to Robust Multiclass Multi-Label Text Categorization," *Proc. 21st Int'l Conf. Machine Learning (ICML '04),* pp. 329-336, 2004.

[13] S. Haykin, *Neural Networks: A Comprehensive Foundation,* second ed. Englewood Cliffs, N.J.: Prentice-Hall, 1999.

[14] R. Jin and Z. Ghahramani, "Learning with Multiple Labels," *Advances in Neural Information Processing Systems,* S. Becker, S. Thrun, and K. Obermayer, eds., vol. 15, pp. 897-904, 2003.

[15] T. Joachims and F. Sebastiani, guest editors' introduction, *J. Intelligent Information Systems,* special issue on automated text categorization, vol. 18, nos. 2/3, pp. 103-105, Mar.-May 2002.

[16] H. Kazawa, T. Izumitani, H. Taira, and E. Maeda, "Maximal Margin Labeling for Multi-Topic Text Categorization," *Advances in Neural Information Processing Systems,* L.K. Saul, Y. Weiss, and L. Bottou, eds., vol. 17, pp. 649-656, 2005.

[17] T. Kohonen, "An Introduction to Neural Computing," *Neural Networks,* vol. 1, no. 1, pp. 3–16, 1988.

[18] A. McCallum, "Multi-Label Text Classification with a Mixture Model Trained by EM," *Proc. Working Notes Am. Assoc. Artificial Intelligence Workshop Text Learning (AAAI '99),* 1999.

[19] W.S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bull. Math. Biophysics,* vol. 5, pp. 115-133, 1943.

[20] M. Minsky and S. Papert, *Perceptrons.* Cambridge, Mass.: MIT Press, 1969.

[21] P. Pavlidis, J. Weston, J. Cai, and W.N. Grundy, "Combining Microarray Expression Data and Phylogenetic Profiles to Learn Functional Categories Using Support Vector Machines," *Proc. Fifth Ann. Int'l Conf. Computational Molecular Biology (RECOMB '01),* pp. 242-248, 2001.

[22] J.R. Quinlan, *Programs for Machine Learning.* San Mateo, Calif.: Morgan Kaufmann, 1993.

[23] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Washington, D.C.: Spartan Books, 1962.

[24] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* D.E. Rumelhart and J. L. McClelland, eds., vol. 1, pp. 318-362, 1986.

[25] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Reading, Pa.: Addison-Wesley, 1989.

[26] G. Salton, "Developments in Automatic Text Retrieval," *Science,* vol. 253, pp. 974-980, 1991.

[27] R.E. Schapire and Y. Singer, "Improved Boosting Algorithms Using Confidence-Rated Predictions," *Proc. 11th Ann. Conf. Computational Learning Theory (COLT '98),* pp. 80-91, 1998.

[28] R.E. Schapire and Y. Singer, "BoosTexter: A Boosting-Based System for Text Categorization," *Machine Learning,* vol. 39, no. 2/3, pp. 135-168, 2000.

[29] F. Sebastiani, "Machine Learning in Automated Text Categorization," *ACM Computing Surveys,* vol. 34, no. 1, pp. 1-47 2002.

[30] N. Ueda and K. Saito, "Parametric Mixture Models for Multi-Label Text," *Advances in Neural Information Processing Systems,* S. Becker, S. Thrun, and K. Obermayer, eds., vol. 15, pp. 721-728, 2003.

[31] B. Widrow and M.E. Hoff, "Adaptive Switching Circuits," *IRE WESCON Convention Record,* vol. 4, pp. 96-104, 1960.

[32] Y. Yang, "An Evaluation of Statistical Approaches to Text Categorization," *Information Retrieval,* vol. 1, no. 1-2, pp. 69-90, 1999.

[33] Y. Yang and J.O. Pedersen, "A Comparative Study on Feature Selection in Text Categorization," *Proc. 14th Int'l Conf. Machine Learning (ICML '97),* pp. 412-420, 1997.

[34] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling Neural Networks: Many Could Be Better than All," *Artificial Intelligence,* vol. 137, no. 1-2, pp. 239-263, 2002.

Min-Ling Zhang received the BSc and MSc degrees in computer science from Nanjing University, China, in 2001 and 2004, respectively. Currently, he is a PhD candidate in the Department of Computer Science and Technology at Nanjing University and a member of the LAMDA Group. His main research interests include machine learning and data mining, especially in multi-instance learning and multi-label learning.

Zhi-Hua Zhou (S'00, M'01, SM'06) received the BSc, MSc and PhD degrees in computer science from Nanjing University, China, in 1996, 1998, and 2000, respectively, all with the highest honors. He joined the Department of Computer Science and Technology at Nanjing University as a lecturer in 2001 and is a professor and head of the LAMDA group at present. His research interests are in artificial intelligence, machine learning, data mining, pattern recognition, information retrieval, neural computing, and evolutionary computing. In these areas, he has published more than 60 technical papers in refereed international journals or conference proceedings. He has won the Microsoft Fellowship Award (1999), the National Excellent Doctoral Dissertation Award of China (2003), and the Award of National Science Fund for Distinguished Young Scholars of China (2003). He is an associate editor of *Knowledge and Information Systems* and serves on the editorial boards of *Artificial Intelligence in Medicine*, the *International Journal of Data Warehousing and Mining*, the *Journal of Computer Science & Technology*, and the *Journal of Software*. He served as program committee member for various international conferences and chaired a number of native conferences. He is a senior member of the China Computer Federation (CCF), the vice chair of the CCF Artificial Intelligence and Pattern Recognition Society, an executive committee member of the Chinese Association of Artificial Intelligence (CAAI), the vice chair and chief secretary of the CAAI Machine Learning Society, a member of the AAAI and the ACM, and a senior member of the IEEE and the IEEE Computer Society.