Dynamic Learning Algorithm of Multi-Layer Perceptrons for Letter Recognition

Qin Feng and Gao Daqi

Abstract—The classical back-propagation learning algorithms of neural networks suffer from a major disadvantage that of excessive computational burden encountered by processing all the data. Relatively speaking, the samples near the separating boundary have a more important influence on the final weights than those far. This paper presents a dynamic back-propagation algorithm which is just based on those decision boundary samples. The dynamic back-propagation algorithm using those boundary samples to update weights can not only greatly improve the learning speed, but also can improve the classification correction. The experimental results for the Letter data set verified that the proposed method is effective. It is far faster than classical learning algorithm and gets 91.1% classification correction.

I. INTRODUCTION

Neural network[3] can be viewed as massively parallel computing systems consisting of an extremely large number of simple processors with many interconnections and have been widely used in pattern classification such object recognition, handwritten digit recognition[7], speech tagging and recognition[8], text categorization and information retrieval[9]. The main characteristics of neural networks are that they have the ability to learn complex nonlinear input-output relationships. Among various neural networks, multi-layer perceptron (MLP) [1] are deeply researched in theory, and are more widely applied into pattern classification, non-linear mappings and other fields. This neural network takes back-propagation (BP) [2] algorithm as their mainly learning algorithm. The theoretical results given in [2,3] show that if an MLP with one hidden layer is capable of realizing arbitrary continuous functions defined on a hypercube, the number of hidden neurons needed is infinite. BP is a very popular learning algorithm because it is conceptually simple, computationally efficient. However, despite many advances, for MLP to find general applicability in real world problems, several questions must still be answered. One question is how to improve the learning speed of MLP? The greatest drawbacks to the MLPs which using BP algorithm in real-world application are the slow learning speed. Even on small scale data set, BP will take hundreds or thousands of times to get the network weights as it iteratively updates weights between neurons. Because of the slow convergence, MLP usually suitable for small scale data set can’t simply generalized to large scale data set. Generally speaking, there are three approaches to improve the learning speed: high speed computer or on faster computer. A number of groups are working on faster implementations, including a group at CMU that using the 10-processor Wrap machine [10]. This works well, however, a slow algorithm directly using hardware may limit the range of problems it can tackle. So our attention mainly focuses on the BP algorithm itself.

Besides gradient descent methods, conjugate gradient and Newton [3, 11] in optimal theory are used to accelerate the learning speed. Newton’s method needs to calculate the inverse of Hessian matrix $H^{-1}$ and may be useless when $H$ is singular. Quasi-Newton method requires too much memory storage capacity. These methods have drawbacks and can only be limited to certain problem. BP uses the chain rule to compute the influence of each weight in the network with respect to an arbitrary error function [12]. Obviously, the learning rate $\eta$ has an important effect on the time needed to converge. If it is too small, many iteration steps are needed to train the network; on the contrary, a large learning rate will possibly lead to oscillation, preventing the error to fall below certain. There are a lot of literatures researched in the learning rate. These methods usually change the $\eta$ with the number of iterations and $\eta$ is regarded as a network parameter to optimize so that at each iteration step it can be optimal. Most of these schemes decrease the learning rate when the weight vector “oscillates”, and increase it when the weight vector follows a relatively direction. Beyond choosing a global learning rate, it is clear that picking a different learning rate $\eta_i$ for each weight can improve the convergence.

BP suffers from a major disadvantage, namely, that of excessive computational burden encountered by processing all the data. For each pattern in training set, BP will calculate each layer’s output according to connected weights. When updating weights, each pattern will also be used to calculate new weights. So when the original data sets consist of huge number of samples, BP algorithm will iterate slowly. Boosting technique trains successive component classifier with a subset of training data that is “most informative” given the current set of component classifiers [2]. Can we only use “most informative” patterns to adjust the connected weights? The Support Vector Machine (SVM) has sprung up in the last
decades or so. The main aim of SVM is to make a learning machine have as high accuracy as possible and as good generalization performances as possible with limited learning samples. The weight vectors for optimal separate hyperplane are given by \( \mathbf{w} = \sum_{i=1}^{N} \lambda_i \mathbf{x}_i \) when the samples are linearly separated. It is obvious that the optimal hyperplane is determined by only a few patterns called support vector which is the most informative. Rather than using all the patterns, we can greatly improve learning speed just using these informative patterns which are around the decision boundary.

The main motivation of this paper is to introduce a new dynamic BP algorithm based on those informative or boundary samples to greatly improve the learning speed of MLP.

II. RELATED WORK

A. Back-propagation algorithm

Although a single-hidden-layer is adequate to enable neural networks to approximate any given function, but the number of hidden neurons is infinite. From a more practical perspective, it has been shown in [13] that single-hidden-layer neural networks are superior to networks with more than one hidden layer with the same level of complexity mainly due to the fact that the latter are more prone to fall into local poor minima. This paper is also focused on single-hidden-layer neural networks.

The number of input, hidden and output nodes is \( m, s \) and \( n \) respectively, and the activation function of output is sigmoid. Suppose the weight vector between all input nodes and hidden node \( h \) at the \( k \)th iteration step is

\[
\mathbf{w}_h(k) = (w_{h1}(k),...,w_{hn}(k)) \in \mathbb{R}^n
\]

(1)

the weight between all the hidden nodes and the output node \( j \) is

\[
w_j(k) = (w_{j1}(k),...,w_{jn}(k)) \in \mathbb{R}^n
\]

(2)

For the \( p \)th input pattern \( \mathbf{x}_p \in \mathbb{R}^m \), the actual output of node \( h \) is \( z_{ph}(k) = f(w_h^T(k)\mathbf{x}_p + \theta_h(k)) \), the actual output of node \( j \) is \( y_{pj} = f(w_j^T(k)z_{ph}(k) + \theta_j(k)) \). Therefore, the sum-of-squares error of network is written as

\[
E_k = \frac{1}{2} \sum_{p=1}^{N} \sum_{j=1}^{n} (t_{pj} - y_{pj}(k))^2 = \frac{1}{2} \| T - Y(k) \|^2
\]

(3)

Generally, the batch learning form is adopted. If the activation function is \( f(x) = (1 + \exp(-x))^{-1} \), the gradient components of weights may be obtained by the chain derivation rule. So the gradient component between hidden node \( h \) and output unit \( j \) in the following form:

\[
\frac{\partial E(k)}{\partial \mathbf{w}_h} = \sum_{p=1}^{N} (t_{pj} - y_{pj}(k))z_{ph}(k)y_{pj}(k)(1 - y_{pj}(k))
\]

(4)

And the gradient component between input unit \( i \) and hidden node \( h \) is

\[
\frac{\partial E(k)}{\partial \mathbf{w}_{hi}} = \sum_{p=1}^{N} (t_{pj} - y_{pj}(k))z_{ph}(k)y_{pj}(k)(1 - y_{pj}(k))
\]

(5)

The gradient component of thresholds is in the same form as above two formulas. The uniformly updating formula of weights and thresholds is

\[
w(k+1) = w(k) - n\frac{\partial E(k)}{\partial \mathbf{w}^T(k)} + \alpha(w(k) - w(k-1))
\]

(6)

Where \( k \) is the iteration step, \( n \) is the learning rate and \( \alpha \) is moment factor. The third term on the right hand side of (6) is dynamic one. If \( E(k) > \varepsilon^c \), we repeatedly use BP algorithm to adjust the weights until \( E(k) > \varepsilon^c \) or the present number of iterations is reached.

B. Input Variable Scales

Assume \( x_1 \in \mathbb{R}^m \) and \( x_2 \in \mathbb{R}^m \) are two samples from different class. We can calculate two samples Euclid distance

\[
\Delta d = \| x_1 - x_2 \|
\]

(7)

in the input space,

\[
\Delta d_{ph} = |\phi_{1h} - \phi_{2h}| = \| (x_1 - x_2)^T \mathbf{w}_h \| \times \cos \gamma
\]

(8)

in the hidden node \( h \) input space where \( \phi_{1h} \) and \( \phi_{2h} \) are the input value of node \( h \),

\[
\Delta d_h = 0.25(x_1^T - x_2^T)\mathbf{w}_h = 0.25d_{ph}
\]

(9)

For example, \( x_1 = (0.0.35)^T \) and \( x_2 = (0.36)^T \) are two samples from training set. Then \( \Delta d = 0.36 - 0.35 = 0.01 \) if \( \mathbf{w}_h = (86.0071, 86.0071)^T \), \( \theta_h = -31.1029 \), we can calculate \( \Delta d_{ph} = 0.1942 \), \( \Delta d_h / \Delta d = 19.42 \) is about 20 times larger than \( \Delta d_{ph} \).

If network is not oscillation, the learning speed will improve as \( \frac{\partial E(k)}{\partial \mathbf{w}^T(k)} \) grows. The general form of standard sigmoid-type function is \( f(x) = \gamma (1 + \exp(-\beta x))^{-1} \) and [13] analysis the first-order derivatives between different action functions and the standard one. This paper adopts \( f(x) = 3(1 + \exp(-3x))^{-1} \) as the action function, the input and target components are scaled in proportion to the ranges \([0, 6]\) and \([0, 3]\) respectively.

C. Modular Single-hidden Neural Network for Multi-class Learning

Multi-class learning problem is a problem of building a system that accurately maps an input feature space to an output space of more than two patterns classes. Neural network algorithm such as BP algorithm were developed for two-class
classification problem and theoretical studies of learning have focused almost entirely on learning binary functions. The extensions for two-class problems to multi-class problems are non-trivial, and often lead to unexpected complexity or weaker performance. Experiment shows that network will easily drop to local minimum when there are over 10 classes. In fact, present literatures seldom use one single-hidden neural network solving more than 10-class samples. The actual output of output node $j$ is

$$y_{pj} = (1 + \exp(-\sum_{h} w_{ph}(\tau)(1 + \exp(-\sum_{h} w_{hj}(\tau)x_{h}))^{-1})^{-1}$$ (11)

and is related to all the weights between hidden and input layer, but only related to the weights between hidden and output nodes $j$. The weights between hidden and input layer are shared, but the weights between hidden and output layer are private. In order to solve multi-class problem with one single-hidden neural network, we can decompose the problem into n binary class problem and each one single output neural network responses for solving a binary class problem. Assume that single output neural network net$j$ represents class $w_j$ and responses for forming $w_j$ decision boundary. In order to train net$j$, we only need all samples from class $w_j$ and limited samples from class $\sim w_j$ which are near to the class $w_j$.

Based on that, we can first decompose a complicated n-class problem into n simpler two-class problem, and then get rid of those futile patterns for determining the decision boundary for a certain class $w_j$. Firstly, we can draw an initial hyper-dimensional oblique ellipsoid, which center and half axis directs coincide with those, and which sizes of major and minor axes are determined by the max Mahalanobis distance $d_{maj}$ between samples from $w_j$ and the center $u_j$. Then calculate the number $N_{ij}$ of samples that are included within the initial ellipsoid and from the other classes and determine the max radius of extended ellipsoid. Finally, form the subset $\Xi'$ which consists of all the samples within extended ellipsoid. The detailed description can be found in [15].

D. Amendment of Outputs Many-to-one MLP

The classification results of perceptrons are often either this or that, which is the reason why neural network are sometimes considered not able to say “NO”. Ref. [16] pointed out that the necessary condition for a single-hidden-layer perceptron to form the closed decision boundaries is that its number of hidden nodes is over that of input units. But most of the practical MLP do not meet the condition, and often no need to. As a result, the decision boundaries formed by MLP are often open. In other words, the real output $y_j$ of MLP $j$ should be added a correction coefficient. The real output of a one-output MLP is related to both the input-to-hidden and the hidden-to-output weights. Here, we add an amended term to the real output, namely

$$\rho_j(x) = \exp(-(y_j - \bar{y}_j)^2) \exp \left(-\frac{1}{2} \sum_{i} \left( \frac{x_i - H_j}{\sigma_j} \right)^2 \right)$$ (12)

where $\bar{y}_j$ is the output of MLP $j$ corresponding to the mean vector $u_j$ of class $\omega_j$ and $\sigma_j$ is the $i$th mean variance component of $\omega_j$.

E. Imbalance Problem

Without a doubt, if the number of samples in a two-class problem, the decision boundary of corresponding MLP will be closer to the central sections margin. In fact, the number of samples in the training subsets is often unequal. Under the situation, the final decision boundaries will be closer to the classed with fewer samples [15].

We can generate and add some real samples for the minority class. The harmfulness to do like this is unduly large storage requirement and subsequently long learning time. We can add some virtual samples to the small side to make the number samples virtually balance.

For $p=1$: $N^{(0)}$

If $x_p^{(i)} \in N^{(i)}$

$$\Delta w^{(i)}(\tau) \leftarrow \Delta w^{(i)}(\tau) - \frac{N_{ij}}{N_j} \frac{\partial E_j(\tau)}{\partial w_p^{(i)}(\tau)}$$

Else

$$\Delta w^{(i)}(\tau) \leftarrow \Delta w^{(i)}(\tau) - \frac{\partial E_j(\tau)}{\partial w_p^{(i)}(\tau)}$$

In this way, there is hardly any additionally computation burden for training MLP $j$ with unbalanced subset $\Xi' = \{X', X''\}$.

III. DYNAMIC BACK-PROPAGATION

Back-propagation algorithm often suffers from a major disadvantage, namely, that of excessive computaional burden encountered by processing all the data. Given a training set containing N samples, where a generic training vector $x_p \in R^m$ , our aim is to find an optimal decision boundary than can correctly classify different classes to make the total error $E(k)$ minimal.

The success of a classification scheme and learning speed may be directly associated with an appropriate data preprocessing. Two important aspects of preprocessing treatment are related to: the choice of attribute feature and to input data selection. A considerable amount of work can be found in the machine learning literature on input attributes choice and manipulation. It’s well-known than redundant or uninformative attributes are, in general, harmful and frequently over shadow performance.

SVM has proved to process the capability of extracting vectors and support the boundary between any two classes. It seems to be intuitively true that samples near the separating boundary between classes play more important roles than those which are interior in the feature space. Vectors near the boundaries between the classes have to be considered to be more significant. So in designing a MLP classifier, can we
only use these informative or boundary samples to adjust network parameters?

Suppose two samples \( x_1 \) at point 0.6 and \( x_2 \) at point 3 from class \( o_i \) are in 1-dimensional space. We take a single one-to-one neuron to simply illustrate that samples around the decision boundary play more important roles to update weights. When the decision equation is \( x - 0.5 = 0 \). According to the weight adjust equation

\[
\Delta w = \sum_{p=1}^{n}(t_p - y_p(x))(1 - y_p(x))x_p(x) \quad (13)
\]

We can calculate \( \Delta w_1 = 0.0711 \) and \( \Delta w_2 = 0.0160 \). This simple problem shows that the final weight adjust value can approximately related to the samples around decision boundary and samples far from decision boundary is irrelevant, so we can remove them from the training set.

Let \( T = \{x_1, x_2, \ldots, x_n\} \) be training set with \( n \) labeled samples \( x_i \in R^n \) in class \( \{o_1, o_2\} \). As a multi-class problem can be decomposed to binary problems, we can just research in binary problem, that is to say, with single output node. Let \( T_{\text{boundary}} = \{y_1, y_2, \ldots, y_n\} \) be boundary training set with \( n_1 \) labeled samples and \( T_{\text{boundary}} = \{y_1, y_2, \ldots, y_n\} \) stands for training set away from decision boundary with \( n_2 \) labeled samples, where \( n = n_1 + n_2 \) and \( y_i \in R^n \) is in the same representation space. It is shown that all the classes which appear in \( T_{\text{boundary}} \) and \( T_{\text{boundary}} \) are adequately appeared in \( T \).

In MLP classifier, samples around decision boundary will have a large output error \( e_p = |t_p - y_p| \) which denotes the error sample \( x_p \) makes. According to output error \( e_p \), training set \( T \) can be divided into two subsets \( T_{\text{boundary}} \) and \( T_{\text{boundary}} \). Let \( \gamma \) be error threshold value. If \( e_p \geq \gamma \), \( x_p \) belongs to \( T_{\text{boundary}} \), else \( x_p \) belongs to \( T_{\text{boundary}} \). Fig.1 shows that value \( \gamma \) has a great effect on the selection of boundary training set. \( T_{\text{boundary}} \)

\[
0.0871 = \Delta \approx \Delta = \Delta + \Delta = \Delta + \Delta \quad (14)
\]

MLP can just use boundary samples to iteratively update weight adjustment.

The selection of error threshold value \( \gamma \) has a great effect on the learning speed and classification rate. If we set \( \gamma \) too large, number of samples in \( T_{\text{boundary}} \) will be small. Though in this way, the network learning speed will be greatly improved than those with small \( \gamma \), the number of samples in \( T_{\text{boundary}} \) is too little to exactly represent the whole training set. However, if we set \( \gamma \) too small, \( T_{\text{boundary}} \) will almost be equal to \( T \) and our network learning speed can’t be improved. In our experiment, we set \( \gamma \) to 0.1 and the result shows that this value can not only greatly improved the learning speed, but also the classification rate remains unchanged.

As discussed above, this paper proposes a newly dynamic BP algorithm based on boundary samples to greatly improve network learning speed. The detail descriptions are as follows:

1. Initialize \( n_{\text{sample}} \) as number of training set, \( n_{\text{iteration}} \) as number of iterations to shuffle \( T_{\text{train}} \), \( \gamma \) as the error threshold value to judge boundary samples, \( T_{\text{boundary}} \) and \( T_{\text{boundary}} \) are initially empty set.

2. Randomly select \( n_{\text{sample}} \) samples from training set \( T \) as \( T_{\text{train}} \).

3. Use \( T_{\text{train}} \) as training set to update weight adjustment and number of iterations equal to \( n_{\text{iteration}} \), if \( E(k) < \varepsilon \) or the present number of iteration is reached then go to 5.

4. Do the following:
   For all samples \( x_p \in T \), calculate corresponding output error \( e_p = |t_p - y_p| \):
   (a) if \( e_p \geq \gamma \) then \( x_p \) is added to \( T_{\text{boundary}} \)
   (b) else \( x_p \) is added to \( T_{\text{boundary}} \).
   \( T_{\text{train}} = T_{\text{boundary}} \), go to 3.

5. The process terminates.

IV. EXPERIMENT–LETTER RECOGNITION

The experiments are conducted by a PC with 2.6G CPU 8cores, 4G RAM.

The Letter problem [14] is to recognize each of a large number of black-and-white rectangular pixel displays as one
of 26 letters in the English alphabet. The character images were based on 20 different fonts and were randomly distorted to produce 20,000 unique stimuli. Each image was scanned, pixel by pixel, to extract 16 numerical attributes. Each attribute was then scaled in the INT range from 0 to 15. The first 16,000 samples are used as the training set $X_{tr} \in \mathbb{R}^{16000 \times 16}$ and the remaining as the test set $X_{te} \in \mathbb{R}^{4000 \times 16}$. The task is especially challenging both because of the wide diversity among the fonts and because of the primitive nature of the attributes. Theoretically, a 16-s-26 MLP is feasible to solve the problem. However, given the termination criteria of training are that maximum iteration epoch $\tau_{\text{max}} = 50000$, almost all 16-s-26 MLP with action function $f(x) = (1 + \exp(-x))^{-1}$ do not converge in the training stage when taking all the 26 letters and 16000 samples as a whole and normalizing the input component, no matter how careful we select the number $s$ of hidden nodes and the learning rate $\eta$.

In our experiment, we decompose a $c$-class pattern classification problem into $c$ binary neural networks with a decision module that integrates the results from $c$ binary neural networks. Here, we take class “A” as an example to go into details on the learning of our dynamic BP algorithm. There are $N_{A} = 633$ samples in class “A” and the number of samples from other classes is $N_{\neq A} = 15367$. We randomly select 500 samples from class “A” and 500 samples from the other classes as the initial training set $T_{\text{train}}$, so $n_{\text{sample}} = 1000$. Now a few samples of the whole training set $T$ are used to train the network with initial iterations, here we set the initial iterations to 500. After 500 iterations, every $n_{\text{iteration}}$ iterations the network will shuffle the training set $T_{\text{train}}$ by calculating the output error $e_{p}$ with every sample in the whole training set $T$, here we set $n_{\text{iteration}}$ to 50. Those samples around the decision boundary will be added to the $T_{\text{boundary}}$ and we will use $T_{\text{boundary}}$ as $T_{\text{train}}$ to train the network in the next 50 iterations.

![Fig. 2. The relationship between number of samples $n_{\text{sample}}$ in $T_{\text{train}}$ and the iteration $k$ with different error threshold value $\gamma$. (a) $\gamma = 0.05$, (b) $\gamma = 0.1$, (c) $\gamma = 0.2$, (d) $\gamma = 0.5$.](image)

Table I gives the final classification results and training time when $\gamma = 0.1$ of 26 single-output MLP with the dynamic back-propagation algorithm for the letter data set. $n_{\text{sample}}$ represents average number of samples used to train MLP. Using those boundary samples, the final classification correct rates are 91.1%, however, the learning time is 547.25s.

![Fig. 3. The relationship between total final error $E(k)$ and the iteration $k$ with different error threshold value $\gamma$: (a) $\gamma = 0.1$, (b) $\gamma = 0.2$, (c) $\gamma = 0.3$, (d) $\gamma = 0.5$.](image)

V. CONCLUSION

The greatest drawbacks to the widespread use of MLP which using BP algorithm in real-world application is the slow learning speed. It takes the whole training set to update network weights. However, the most informative samples are around the decision boundary. They play a more important role to the weight adjustment. The dynamic back-propagation algorithm proposed just uses those boundary samples to iteratively update connected weights. By this way, we can greatly improve network learning speed.
REFERENCES


<table>
<thead>
<tr>
<th>Letter</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training subset</td>
<td>n_j</td>
<td>633</td>
<td>630</td>
<td>594</td>
<td>638</td>
<td>616</td>
<td>622</td>
<td>609</td>
<td>583</td>
<td>590</td>
<td>599</td>
<td>593</td>
<td>604</td>
</tr>
<tr>
<td>Average training set</td>
<td>n_sample</td>
<td>223.9</td>
<td>827.7</td>
<td>425.7</td>
<td>368</td>
<td>504.5</td>
<td>683.1</td>
<td>643.9</td>
<td>1301.1</td>
<td>425.1</td>
<td>340.9</td>
<td>551.9</td>
<td>413.3</td>
</tr>
<tr>
<td>Learning time(s)</td>
<td>time</td>
<td>10.83</td>
<td>17.37</td>
<td>20.68</td>
<td>22.62</td>
<td>22.33</td>
<td>27.35</td>
<td>26.37</td>
<td>48.21</td>
<td>17.89</td>
<td>16.39</td>
<td>22.13</td>
<td>18.55</td>
</tr>
<tr>
<td>Test subset</td>
<td>No.miscalss</td>
<td>3</td>
<td>11</td>
<td>12</td>
<td>8</td>
<td>15</td>
<td>11</td>
<td>16</td>
<td>30</td>
<td>16</td>
<td>22</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Recrate (%)</td>
<td>98.8</td>
<td>91.91</td>
<td>91.55</td>
<td>95.21</td>
<td>90.13</td>
<td>92.81</td>
<td>90.24</td>
<td>80.13</td>
<td>90.3</td>
<td>91.89</td>
<td>84.93</td>
<td>90.45</td>
<td>94.93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>N</th>
<th>O</th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
<th>T</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training subset</td>
<td>n_j</td>
<td>617</td>
<td>614</td>
<td>635</td>
<td>615</td>
<td>597</td>
<td>587</td>
<td>645</td>
<td>645</td>
<td>628</td>
<td>613</td>
<td>628</td>
<td>641</td>
<td>576</td>
</tr>
<tr>
<td>Average training set</td>
<td>n_sample</td>
<td>386</td>
<td>545.1</td>
<td>351.9</td>
<td>375.3</td>
<td>545.5</td>
<td>708.4</td>
<td>431</td>
<td>322.8</td>
<td>441.4</td>
<td>245.9</td>
<td>446</td>
<td>361.3</td>
<td>256.7</td>
</tr>
<tr>
<td>Learning time(s)</td>
<td>time</td>
<td>19.73</td>
<td>27.46</td>
<td>18.74</td>
<td>19.74</td>
<td>23.14</td>
<td>25.79</td>
<td>19.31</td>
<td>17.86</td>
<td>18.44</td>
<td>14.04</td>
<td>20.21</td>
<td>18</td>
<td>16.26</td>
</tr>
<tr>
<td>Test subset</td>
<td>No.miscalss</td>
<td>18</td>
<td>11</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>22</td>
<td>10</td>
<td>13</td>
<td>8</td>
<td>4</td>
<td>15</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>Recrate (%)</td>
<td>87.5</td>
<td>92.09</td>
<td>88.69</td>
<td>89.29</td>
<td>89.44</td>
<td>85.43</td>
<td>92.26</td>
<td>90.44</td>
<td>94.12</td>
<td>97.12</td>
<td>90.57</td>
<td>88.28</td>
<td>93.67</td>
<td>91.1</td>
</tr>
</tbody>
</table>