AN IMPROVED FAST BRAIN LEARNING ALGORITHM

Shuo Xu [1], Xin An [2], Lan Tao [3 *]

[1] *College of Information and Engineering, China Agricultural University, Qinghua Donglu 17, Haidian, Beijing, 100083, P.R. China. E-mail: pzczxs@gmail.com. Tel: +86-10-62736755*

[2] *School of International Trade and Economics, University of International Business and Economics, Huixin Dongjie 10, Chaoyang, Beijing, 100029, P.R. China. E-mail: anxin927@gmail.com*

[3*] *College of Information Engineering, Shenzhen University, Nanhai Avenue 3688, Shenzhen, Guangdong, 518060, P.R. China. E-mail: taolan@szu.edu.cn. Tel: +86-755-26535078, Corresponding author*

Abstract: In this paper, an underlying problem on the fast BRAIN learning algorithm is pointed out, which is avoided by introducing the quantity *count* (·, ·). In addition, its speed advantage can still be enjoyed only at a cost of a little additional space. The improved fast BRAIN learning algorithm is also given.

Key words: BRAIN, Numerical Computation

1. INTRODUCTION

Given a labeled dataset (training dataset) $(\mathbf{x}_i, y_i)$, $i = 1, 2, \ldots, l$, $\mathbf{x}_i \in \{0, 1\}^n$, $y_i \in \{-1, +1\}$, where these data points are drawn randomly and independently according to some underlying but unknown probability distribution. We assume this dataset to be self-consistent, i.e., an instance cannot be positive and negative at the same time. The goal is to find a classification rule (hypothesis or function): f: $\{0, 1\}^n \rightarrow \{-1, +1\}$ using this dataset such that f will correctly classify a new instance $(\mathbf{x}, y)$, that is, $f(\mathbf{x}) = y$ for this new instance, which is generated from the same underlying probability distribution as the training data. For convenience, we denote

$\mathbf{x}_i^+ = \left(x_{i,1}^+, x_{i,2}^+, \cdots, x_{i,n}^+\right)^t$ for positive instances, where $x_{i,k}^+ \in \{0,1\}$, i = 1, 2, …, l^+, k = 1, 2, …, n, and similarly, denote $\mathbf{x}_j^- = \left(x_{j,1}^-, x_{j,2}^-, \cdots, x_{j,n}^-\right)^t$ for negative instances, where $x_{j,k}^- \in \{0,1\}$, j = 1, 2, …, l^-, k = 1, 2, …, n. And t denotes the transpose of a vector. Of course, $l = l^+ + l^-$.

Indeed, there are many approaches that can solve this problem, such as NN (Neural Network) (Haykin, 1999), SVM (Support Vector Machine) (Vapnik, 1998; 1999), and many others. However, from an entirely different perspective, Rampone (1998) put forward the BRAIN (Batch Relevance-based Artificial INtelligence) learning algorithm. The aim of the algorithm is to infer a consistent DNF (Disjunction Normal Form) classification rule of minimum syntactic complexity from a set of instances, i.e., training dataset. Here, the minimum syntactic complexity means the minimum number of *clauses*, each one with the minimum number of *literals*. A Boolean classification rule g is consistent with a training dataset if, and only if, it matches every positive instance and no negative instance in the set. That is, g is consistent with a training dataset if, and only if, $g(\mathbf{x}) = 1$ for all positive instances, $g(\mathbf{x}) = 0$ for all negative instances. Once such Boolean classification rule is found, then our final classification rule is $f(\mathbf{x}) = 2g(\mathbf{x}) - 1$.

The major advantages of this algorithm are the low error rates and high correlation coefficient, the explicit classification rules description as a DNF formula, a polynomial (cubic) computational complexity, and robust and stable "one shot" learning. However, the space and time complexity of this algorithm are very high, which heavily limit the range of its application in real world. On the other hand, by many reasons, errors may be present in the training dataset. That is, the training dataset may be contaminated by noise to some extent. The structural risk minimization (SRM) principle (Vapnik, 1998; 1999) tells us that a function which makes a few errors on the training set might have a better generalization ability than a larger function (with more *literal*s and more *clause*s) which makes zero empirical error.

Soon, Rampone (2004) realized this point, and gave a fast BRAIN learning algorithm with an error tolerance, which will be described in next section. From theoretical viewpoint, there are no problems, but from the viewpoint of numerical computation, there may be a problem, which will be analyzed in section 3. Finally, an improvement will be given in section 4.

2. FAST BRAIN LEARNING ALGORITHM

Rampone(2004) found that building the sets $S_{i,j}$ was a main computational drawback, whose time complexity is $O(n \times l^+ \times l^-)$, and space complexity is $O(l^+ \times l^-)$. By definition of $S_{i,j}$, the sets $S_{i,j}$ can be derived from the given positive and negative instances. When a new *literal* $e_k \leftarrow \arg\max R(e_k)$ (If there is a tie, that is, the *literal*s that reach the maximum value are not just one. At this time, we prefer the *literal* with lower subscript and the one with true form.) is selected, the following two steps are performed:

(1) Delete the $S_{i,j}$ sets for $j = 1, 2, \ldots, l^-$ if $e_k \notin S_i$;

(2) Delete the $S_{i,j}$ sets if $e_k \in S_{i,j}$.

In fact, the $S_{i,j}$ update step (1) can be done by deleting $\mathbf{x}_i^+$ having 0 in position k if e_k is in true form, or $\mathbf{x}_i^+$ having 1 in position k if e_k is in negated form, i.e., the positive instances whose indices belong to

$$II = \left\{ i \left| \begin{array}{c} x_{i,k}^+ = 0, e_k \text{ is in true form} \\ \mathbf{x}_i^+ \in S, i = 1,2,\cdots,l^+ \end{array} \right. \right\} \cup \left\{ i \left| \begin{array}{c} x_{i,k}^+ = 1, e_k \text{ is in negated form} \\ \mathbf{x}_i^+ \in S, i = 1,2,\cdots,l^+ \end{array} \right. \right\} \quad (1)$$

And the $S_{i,j}$ update step (2) can be done by deleting $\mathbf{x}_j^-$ having 0 in position k if e_k is in true form, or $\mathbf{x}_j^-$ having 1 in position k if e_k is in negated form, i.e., the negative instances whose indices belong to

$$JJ = \left\{ j \left| \begin{array}{c} x_{j,k}^- = 0, e_k \text{ is in true form} \\ \mathbf{x}_j^- \in S, j = 1,2,\cdots,l^- \end{array} \right. \right\} \cup \left\{ j \left| \begin{array}{c} x_{j,k}^- = 1, e_k \text{ is in negated form} \\ \mathbf{x}_j^- \in S, j = 1,2,\cdots,l^- \end{array} \right. \right\} \quad (2)$$

In this way, we can substitute the $l^+ \times l^-$ sets $S_{i,j}$ for a set S containing at most $l^+ + l^-$ instances. The space complexity can be dramatically reduced.

Now, the extended relevance can be evaluated by

$$R(e_k) = \sum_{i \in I} \sum_{j=1}^{l^-} \frac{\delta_{i,j}(e_k, S)}{d_{i,j}} \quad (3)$$

where $I = \left\{ i \left| \mathbf{x}_i^+ \in S, i = 1,2,\cdots,l^+ \right. \right\}$, $d_{i,j}$ is the Hamming distance between $\mathbf{x}_i^+$ and $\mathbf{x}_j^-$, which can be calculated once and used for all, and

$$\delta_{i,j}(x_k, S) = \begin{cases} 1, \text{ if } x_{i,k}^{+} = 1 \text{ and } x_{j,k}^{-} = 0 \\ 0, \text{ otherwise} \end{cases}$$
$$\delta_{i,j}(\overline{x}_k, S) = \begin{cases} 1, \text{ if } x_{i,k}^{+} = 0 \text{ and } x_{j,k}^{-} = 1 \\ 0, \text{ otherwise} \end{cases} \quad (4)$$

These quantities can be calculated just once for each *clause*. In fact, by using Eq. 3, it is easy to see that, when we update the sets $S_{i,j}$, the corresponding extended relevance update is for step (1):

$$R^{new}(e_k) = R^{old}(e_k) - \sum_{i \in II} \sum_{j \in J} \frac{\delta_{i,j}(e_k, S)}{d_{i,j}} \quad (5)$$

where $J = \left\{ j \middle| \mathbf{x}_j^{-} \in S, j = 1, 2, \cdots, l^{-} \right\}$, and for step (2):

$$R^{new}(e_k) = R^{old}(e_k) - \sum_{i \in I \wedge i \notin II} \sum_{j \in JJ} \frac{\delta_{i,j}(e_k, S)}{d_{i,j}} \quad (6)$$

3. A PROBLEM ON FAST BRAIN LEARNING ALGORITHM

From theoretical viewpoint, iterative formula Eq. 5 and Eq. 6 can work well. But from the viewpoint of numerical computation, there may be some problems, especially when there is a tie for $e_k \leftarrow \arg\max R(e_k)$. In what follows, we will give the analysis.

Though there are several different representations of real numbers (Matula and Kornerup, 1985), by far the floating-point representation is widely used in computer system, from PCs to supercomputers. However, most floating-point calculations have rounding error anyway. So the IEEE standard (IEEE, 1987) requires that the result of addition, subtraction, multiplication and division be exactly rounded. That is, the result must be calculated exactly and then rounded to the nearest floating-point number (using round to even). According to theorem 2 in (Goldberg, 1991), the relative rounding error in the result for addition and subtraction with one guard digit is less than or equal to 2ε, where ε is machine epsilon. That is, each addition or subtraction operation can potentially introduce an relative rounding error as large as 2ε, then a sum involving thousands of terms can have quite a bit of rounding error . The iterative formula Eq. 5 and Eq. 6 introduce much more floating-point addition or subtraction operations than necessary (see below), that is, they introduce quite a bit of rounding error.

More specially, let's consider a simple example $\sum_{i=1}^{n} x_i$, assuming the calculation is being done in double precision. If the naive formula $\sum_{i=1}^{n} x_i$ is utilized, then the computed sum is equal to $\sum_{i=1}^{n} x_i (1+\delta_i)$, where $| \delta_i | \leq (n - i)\varepsilon$, that is, each summand is perturbed by as large as $n\varepsilon$ (Goldberg. 1991). Though there is a much more efficient method which dramatically improves the accuracy of sums, namely, the Kahan summation formula, the relative rounding error is still related to the number of operations. Since at this time the calculated sum is equal to $\sum_{i=1}^{n} x_i (1+\delta_i) + O(n\varepsilon^2) \sum_{i=1}^{n} |x_i|$, where $| \delta_i | \leq 2\varepsilon$ (Goldberg, 1991). In this way, it is very possible that for the two *literal*s, say e_1, e_2, it should be $R(e_1) = R(e_2)$ (or $R(e_1) \neq R(e_2)$), but it becomes $R(e_1) \neq R(e_2)$ (or $R(e_1) = R(e_2)$) after several update calculations using Eq. 5 and Eq. 6. Eventually, it will possibly result in the Boolean classification rule obtained by the fast BRAIN learning algorithm (Rampone, 2004) is not same as the one derived by the origin BRAIN learning algorithm (Rampone, 1998).

4. AN IMPROVEMENT ON FAST BRAIN LEARNING ALGORITHM

According to above analysis, let's consider how to avoid this underlying problem. Assume the training dataset is self-consistent, the Hamming distance between $\mathbf{x}_i^+$ and $\mathbf{x}_j^-$ must be at the interval [1, n], i.e., $1 \leq d_{i,j} \leq n$. Thus we can count the number of each Hamming distance for each *literal* e_k and denote it as *count* $(e_k, d_{i,j})$, $i \in I$, $j = 1, 2, \ldots, l^-$. Now, the extended relevance can be evaluated by

$$R(e_k) = \sum_{i=1}^{n} \frac{count(e_k, i)}{i} \tag{7}$$

It is not difficult to see that *count* $(\cdot, \cdot)$ can also be calculated just once for each *clause*. In fact, Eq. 5 can be replaced by Eq. 8 and Eq. 7, and Eq. 6 by Eq. 9 and Eq. 7.

$$count^{new}(e_k, d_{i,j}) = count^{old}(e_k, d_{i,j}) - 1, i \in II,\ j \in J \tag{8}$$

$$count^{new}(e_k, d_{i,j}) = count^{old}(e_k, d_{i,j}) - 1, i \in I \wedge i \notin II,\ j \in JJ \tag{9}$$

Because the results of integer addition and subtraction calculations are exact so long as operands and result are not out of range represented by computer system, and it is nearly impossible to reduce further the number of

floating-point operations in Eq. 7, the underlying problem on the fast BRAIN learning algorithm can be overcome. Compared with the version of Rampone (Rampone, 2004), the cost that we pay is the additional space (to be precise, $2n^2$) for *count* (·, ·). But in general, for many applications, e.g., splice sites prediction, the order of magnitude of l^+, especially l^- is usually several orders of magnitude of n. That is, the additional space for *count* (·, ·) is negligible. In addition, since Eq. 8 and Eq. 9 are similar to Eq. 5 and Eq. 6, respectively, and Eq. 8 and Eq. 9 are only involving integer addition or subtraction operations, we can still enjoy the speed advantage of the fast BRAIN learning algorithm.

In what follows, the improved fast BRAIN learning algorithm can be sketched:

Improved Fast BRAIN Learning Algorithm

Input: $X^+ = \left\{\mathbf{x}_1^+, \mathbf{x}_2^+, \cdots, \mathbf{x}_{l^+}^+\right\}$ and $X^- = \left\{\mathbf{x}_1^-, \mathbf{x}_2^-, \cdots, \mathbf{x}_{l^-}^-\right\}$ for positive instances and negative instances, respectively, where $\mathbf{x}_i^+ \in \{0, 1\}^n$, $\mathbf{x}_i^- \in \{0, 1\}^n$, and ε^+, ε^- for the error tolerant parameters;

Output: a Boolean classification rule or a consistent DNF formula $g(\mathbf{x})$;

Initialize: $g(\mathbf{x}) \leftarrow FALSE$, $l_r^+ \leftarrow l^+$;

Calculate the Hamming distance $d_{i,j}$, $i = 1, 2, \ldots, l^+$, $j = 1, 2, \ldots, l^-$;

While $\left(l_r^+ / l^+\right) > \varepsilon^+$

$S \leftarrow X = X^+ \cup X^-$;

Count the number of each Hamming distance for each *literal* e_k, i.e., *count* $(e_k, d_{i,j})$, $i \in I, j = 1, 2, \ldots, l^-$;

$c \leftarrow TRUE$;

$l_r^- \leftarrow l^-$;

Build the *clause* c: *While* $\left(l_r^- / l^-\right) > \varepsilon^-$

Calculate the extended relevance $R(e_k)$ by Eq. 7;

$e_k \leftarrow \arg\max R(e_k)$;

$c \leftarrow c \wedge e_k$;

Let II the set of indexes [Eq. 1], and update *count* (·, ·) by Eq. 8;

Delete from S the instances $\mathbf{x}_i^+$, $\forall i \in II$;

Let JJ the set of indexes [Eq. 2], and update *count* (·, ·) by Eq. 9;

Delete from S the instances $\mathbf{x}_j^-$, $\forall j \in JJ$;

$l_r^- \leftarrow l_r^- - \left|JJ\right|$;

End

$g(\mathbf{x}) \leftarrow g(\mathbf{x}) \vee c$;

Delete from X^+ the positive instances matching c, and update l_r^+;

End

5. CONCLUSION

In this paper, we analyze the reasons that an underlying computational problem on the fast BRAIN learning algorithm may occur, and give an improved algorithm, which is numerically more stable. Furthermore, its speed advantage can still be enjoyed only at a cost of a little additional space. In the end, since the algorithm will give an explicit classification rule description as a DNF formula, we think it can be utilized for feature selection with binary value data, thus the data will be compressed heavily in some cases.

AVAILABILITY

The source code implementing the improved fast BRAIN algorithm is available from the authors upon request for academic use.

ACKNOWLEDGEMENTS

This research is partially supported by the National Natural Science Foundation under Grant No. 60673122.

REFERENCES

Aykin S. Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice-Hall, Inc., 1999.

Goldberg D. What Every Computer Scientist Should Know about Floating-Point Arithmetic. ACM Computing Survey, 1991, 23(1): 5-48.

IEEE. IEEE Standard 754-1985 for Binary Floating-Point Arithmetic, IEEE, 1985. Reprinted in SIGPLAN, 1987, 22(2): 9-25.

Matula D.W. and Kornerup P. Finite Precision Rational Arithmetic: Slash Number Systems. IEEE Transaction on Computers, 1985, C-34(1): 3-18.

Rampone S. An Error Tolerant Software Equipment for Human DNA Characterization. IEEE Transactions on Nuclear Science, 2004, 52(5): 2018-2026.

Rampone S. Recognition of Splice Junctions on DNA Sequence by BRAIN Learning Algorithm. Bioinformatics, 1998, 14(8): 676-684.

Vapnik V.N. Statistical Learning Theory, Wiley, New York, 1998.

Vapnik V.N. The Nature of Statistical Learning Theory, 2nd Edition, Springer Verlag, New York, 1999.