A Note on the Fast BRAIN Learning Algorithm

Shuo Xu¹, Lan Tao², Xin An³, and Lin Li¹*
¹College of Information and Electrical Engineering, China Agricultural University, Beijing 100083, P.R. China
E-mail: {pzczxs, lilincau}@gmail.com
²College of Information Engineering, Shenzhen University, Shenzhen 518060, P.R. China
E-mail: taolan@szu.edu.cn
³School of International Trade and Economics, University of International Business and Economics, Beijing 100083, P.R. China
E-mail: anxin927@gmail.com

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, and the experiments on NN269 dataset validate our analysis.

1. Introduction

Given a labeled dataset (training dataset) \((x, y), i = 1, ..., l, x_i \in \{0, 1\}^n, y_i \in \{-1, +1\}\), 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 \((x, y)\), that is, \(f(x) = y\) for this new instance, which is generated from the same distribution as the training data. For convenience, we denote \(x_i^+ = (x_{i,1}^+, x_{i,2}^+, ..., x_{i,n}^+)^T\) for positive instances and \(x_i^- = (x_{i,1}^-, x_{i,2}^-, ..., x_{i,n}^-)^T\) for negative ones, where \(x_{i,k}^+ \in \{0, 1\}, x_{i,k}^- \in \{0, 1\}, i = 1, ..., l, j = 1, ..., \Gamma, k = 1, ..., n\). Here \(\Gamma\) denotes the transpose of a vector. Of course, \(I^T = I + \Gamma\).

Indeed, there are many approaches that can solve this problem, such as NN [1], SVM [2, 3], and many others. However, from an entirely different perspective, Rampone [4] put forward the BRAIN (Batch Relevance-based Artificial INtelligence) learning algorithm (BRAIN in short). The aim of the algorithm is to infer a consistent DNF (Disjunction Normal Form) classification rule of minimum syntactic complexity from 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, \(g(x) = 1\) for all positive instances, \(g(x) = 0\) for all negative ones. Once such Boolean classification rule is found, then our final classification rule is \(f(x) = 2g(x) - 1\).

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 principle [2, 3] 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 literals and more clauses) which makes zero empirical error.

* Corresponding author
Soon, Rampone [5] realized this point, and gave a fast BRAIN learning algorithm (FastBRAIN in short) 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. After that, an improved algorithm will be given in section 4. Finally, conducted experiments in section 5 validate our analysis.

2. FastBRAIN algorithm

Rampone [5] found that building the sets $S_{i,j}$ was a main computational drawback, whose time complexity is $O(n \times \Gamma \times \Gamma)$, and space complexity is $O(\Gamma \times \Gamma)$. 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 $e_k \leftarrow \text{arg max } R(e_k)$ is selected, the following two steps are performed:

(a) Delete the $S_{i,j}$ sets for $j = 1, \ldots, \Gamma$ if $e_k \not\in S_i$;
(b) Delete the $S_{i,j}$ sets if $e_k \in S_{i,j}$.

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

$$H = \left\{ i \left| x_{i,k}^+ = 0, e_k \text{ is in true form} \right. \right\} \cup \left\{ i \left| x_{i,k}^+ = 1, e_k \text{ is in negated form} \right. \right\}$$

(1)

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

$$JJ = \left\{ j \left| x_{j,k}^- = 0, e_k \text{ is in true form} \right. \right\} \cup \left\{ j \left| x_{j,k}^- = 1, e_k \text{ is in negated form} \right. \right\}$$

(2)

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

Now, the extended relevance can be evaluated by

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

(3)

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

$$\delta_{i,j}(x_i, S) = \begin{cases} 1, & \text{if } x_{i,k}^+ = 1 \text{ and } x_{j,k}^- = 0 \\ 0, & \text{otherwise} \end{cases}$$

(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 (a):

$$R^{\text{new}}(e_k) = R^{\text{old}}(e_k) - \sum_{i \in I} \sum_{j=1}^{l^+} \frac{\delta_{i,j}(e_k, S)}{d_{i,j}}$$

(5)

where $J = \left\{ j \left| x_j^- \in S, j = 1, \ldots, l^- \right. \right\}$, and for step (b):

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

(6)

3. A problem on FastBRAIN 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 $e_k \leftarrow \text{arg max } R(e_k)$, that is, the literals that reach the maximum value are not just one. In what follows, we will give the analysis.

Though there are several different representations of real numbers [6], 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 [7] 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 [8], the relative rounding error in the result for addition and subtraction with one guard digit is less than or equal to $2\epsilon$, where $\epsilon$ is machine epsilon. That is, each addition or subtraction operation can potentially introduce an absolute rounding error as large as $2\epsilon$, 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)e, that is, each summand is perturbed by as large as ne [8]. 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 \[ \sum_{i=1}^{n} x_i (1 + \delta_i) + O(ne^2 \sum_{i=1}^{n} |x_i|) \], where |\[ \delta_i \]| \leq 2e [8]. In this way, it is very possible that for the two literals, 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. Thereby, it will possibly result in the Boolean classification rule obtained by FastBRAIN [5] is not same as the one derived by BRAIN [4]. Eventually, it will possibly lead to lower prediction performance.

4. Improved FastBRAIN 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 \( x_i^+ \) and \( x_j^- \) 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 \( count(e_k, d_{i,j}) \), \( i \in I \), \( j = 1, \ldots, \Gamma \). Now, the extended relevance can be evaluated by

\[
R(e_k) = \frac{\sum_{i=1}^{n} count(e_k, i)}{i}
\]

(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.

\[
\begin{align*}
\text{count}_{\text{new}}(e_k, d_{i,j}) &= \text{count}_{\text{old}}(e_k, d_{i,j}) - 1, \\
& i \in I, j \in J \\
\text{count}_{\text{new}}(e_k, d_{i,j}) &= \text{count}_{\text{old}}(e_k, d_{i,j}) - 1, \\
& i \in I \land i \notin J, j \in JJ
\end{align*}
\]

(8)

(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 FastBRAIN algorithm can be overcome. Compared with FastBRAIN, the cost that we pay is the additional space (to be precise, \( 2n^2 \)) for \( count(\cdot, \cdot) \). But in general, for many applications, e.g., splice sites prediction, the order of magnitude of \( \Gamma \), especially \( \Gamma \) is usually several orders of magnitude of \( n \). That is, the additional space for \( count(\cdot, \cdot) \) 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 FastBRAIN.

In what follows, improved FastBRAIN algorithm (FastBRAIN-2 in short) can be sketched:

FastBRAIN-2 Algorithm

Step 0: Input: \( \mathcal{A}^+ = \{x_1^+, x_2^+, \ldots, x_n^+\} \) for positive instances and \( \mathcal{A}^- = \{x_1^-, x_2^-, \ldots, x_n^-\} \) for negative ones, where \( x_i^+ \in \{0, 1\}^n \), \( x_i^- \in \{0, 1\}^n \), and \( \epsilon^+, \epsilon^- \) for the error tolerant parameters;

Step 1: Initialize: \( g(x) \leftarrow FALSE, I_+^0 \leftarrow I^+ \); Step 2: Calculate the Hamming distance \( d_{i,j}, i = 1, \ldots, \Gamma, j = 1, \ldots, \Gamma \); Step 3: While \( \left( I_+^0/I_+^1 - \epsilon^+ \right) > \epsilon^- \)

\[
3.1 \ S \leftarrow \mathcal{A}^+ = \mathcal{A}^+ \cup \mathcal{A}^-;
3.2 \ \text{count the number of each Hamming distance for each literal} e_i, \text{i.e.,} \ |	ext{count}(e_i, d_{i,j}), i \in I, j = 1, \ldots, \Gamma|;
3.3 \ c \leftarrow TRUE;
3.4 \ I_+^0 \leftarrow I^-;
3.5 \ \text{Build the clause} c: \text{While} \left( \frac{I_+^0}{I_+^1} - \epsilon^- \right) > \epsilon^-\]

3.5.1 Calculate the extended relevance \( R(e_k) \) by Eq. 7;
3.5.2 \( e_k \leftarrow \arg \max R(e_k) \);
3.5.3 \( c \leftarrow c \land e_k \);
3.5.4 Let \( I \) the set of indexes [Eq. 1], and update \( count(\cdot, \cdot) \) by Eq. 8;
3.5.5 Delete from \( S \) the instances \( x_i^+, \forall i \in I \); 3.5.6 Let \( JJ \) the set of indexes [Eq. 2], and update \( count(\cdot, \cdot) \) by Eq. 9;
3.5.7 Delete from \( S \) the instances \( x_j^-, \forall j \in JJ \); 3.5.8 \( I_+^1 \leftarrow I_+^0 - |JJ| \); End

\[\text{In our experiments on NN269 (see below), if there is a tie, we prefer the literal whose script is closer to junction (GT/AG) and the one in true form.}\]
3.6 \( g(x) \leftarrow g(x) \lor c; \)

3.7 Delete from \( \mathcal{I}^+ \) the positive instances matching \( c \), and update \( I^+_r \);

End

Step 4: Output: a Boolean classification rule or a consistent DNF formula \( g(x) \);

Step 5: \( f(x) = 2g(x) - 1 \);

5. Material and data

![Exon region](Exon region) ![Intron region](Intron region)

**Figure 1.** The length of sequence around candidate donor and acceptor sites.

The referring material is the NN269 data set [9] which was collected to develop and test algorithm for human splice sites identification in GENIE system [10, 11]. It consists of 1324 confirmed true donor sites, 1324 confirmed true acceptor sites, 4922 false donor sites and 5553 false acceptor sites collected from 269 human genes. However, there is an ambiguity in splice.train-real.D (No. 903, HSG17G_3388) and splice.train-false.D (No. 358, HUMA1GLY2_3604), which are excluded from our experiments. Each of the false donor/acceptor sites also has GT/AG in the splicing site but is not a real splice site according to the annotation. The window size for a donor is 15 nucleotides \((l_1 = 7, l_2 = 6)\), corresponding to 60 binary attributes, for an acceptor 90 nucleotides \((l_1 = 68, l_2 = 20)\), corresponding to 360 binary attributes, as shown in Figure 1.

**Table 1.** NN269 instance distribution over each set.

<table>
<thead>
<tr>
<th></th>
<th>Donor Sites</th>
<th>Acceptor Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td><strong>Train</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(84.28%)</td>
<td>(84.11%)</td>
<td>(84.29%)</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(15.72%)</td>
<td>(15.89%)</td>
<td>(15.71%)</td>
</tr>
<tr>
<td><strong>sum</strong></td>
<td>1323</td>
<td>4921</td>
</tr>
</tbody>
</table>

Note that the nucleotides different from A or C or G or T are present in this dataset. Before converting each splice site to binary attributes, we filter each splice site in order to minimize the likelihood of a stop codon on the primary strand (e.g., R (A or G) \( \rightarrow \) G, S (C or G) \( \rightarrow \) C, N (anything) \( \rightarrow \) C, etc.). This dataset is further split into a training set and a testing set. Table 1 shows the instance distribution over each set.

Since splice sites prediction can be divided into two subtasks: donor sites prediction and acceptor sites prediction, either of which can be formally stated as a binary classification problem: \{donor site (+1), non-donor site (−1)\} and \{acceptor site (+1), non-acceptor site (−1)\}, two separate classifiers are constructed, one for donor sites and the other for acceptor sites.

6. Results and discussion

We have implemented these three algorithms in C on a common Intel Pentium IV based PC. To make the comparison meaningful, the error parameters, i.e., \( \varepsilon^+ \) and \( \varepsilon^- \), are set to zero. The corresponding source code can be available from the authors upon request for academic use.

6.1. Evaluation measures

To illustrate that quite a bit of rounding error introduced by the iterative formula Eq. 5 and Eq. 6 leads possibly to the Boolean classification rule obtained by FastBRAIN is not same as the one derived by BRAIN, we take the number of clauses \(#\text{ clauses} \) and the number of clauses in common with BRAIN \(#\text{ common} \) as indicators.

To understand how the Boolean classification rule obtained by FastBRAIN affects the prediction performances, the error rate and correlation coefficient \( (CC) \) are utilized. The error rate is the ratio between the number of instances in the test set erroneously classified and the test set size. The correlation coefficient \( (CC) \) is a measure that takes the relationship between correctly predicted positives and negatives as well as false positives and negatives:

\[
CC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}
\]

where \( TP \) and \( TN \) are the number of the correctly predicted positive and negative instances, respectively, and \( FP \) and \( FN \) are similarly the number of the incorrectly predicted positive and negative ones, respectively.

Finally, we will consider the ratio between running times of FastBRAIN and FastBRAIN-2 to verify whether FastBRAIN-2 will still enjoy the speed advantage of FastBRAIN.

6.2. The Boolean classification rule
From the results in Table 2 and Table 3, it is not difficult to see that the Boolean classification rule obtained by FastBRAIN-2 is completely same as the one derived by BRAIN, while FastBRAIN makes a great difference with BRAIN. In other words, our algorithm successfully overcomes the underlying computational problem on FastBRAIN.

**Table 2. Comparison of performance for three algorithms on donor sites prediction.**

<table>
<thead>
<tr>
<th>algorithm</th>
<th># clauses (# common)</th>
<th>error rate</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRAIN</td>
<td>88</td>
<td>0.0717</td>
<td>0.7859</td>
</tr>
<tr>
<td>FastBRAIN</td>
<td>85 (59)</td>
<td>0.0768</td>
<td>0.7713</td>
</tr>
<tr>
<td>FastBRAIN-2</td>
<td>88 (88)</td>
<td>0.0717</td>
<td>0.7859</td>
</tr>
</tbody>
</table>

**Table 3. Comparison of performance for three algorithms on acceptor sites prediction.**

<table>
<thead>
<tr>
<th>algorithm</th>
<th># clauses (# common)</th>
<th>error rate</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRAIN</td>
<td>81</td>
<td>0.1414</td>
<td>0.5292</td>
</tr>
<tr>
<td>FastBRAIN</td>
<td>81 (57)</td>
<td>0.1396</td>
<td>0.5337</td>
</tr>
<tr>
<td>FastBRAIN-2</td>
<td>81 (81)</td>
<td>0.1414</td>
<td>0.5292</td>
</tr>
</tbody>
</table>

Although the Boolean classification rule obtained by FastBRAIN is not same as the one obtained by BRAIN, its prediction performance on acceptor sites is better than that of BRAIN. In our opinion, the main reason is that the solution found by generalized greedy method is just an approximate, not optimal [4].

### 6.3. Speed advantage

**Table 4. Execution time ratio between the FastBRAIN-2 and FastBRAIN on NN269.**

<table>
<thead>
<tr>
<th>junction</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Donor Sites</td>
<td>1.0401</td>
</tr>
<tr>
<td>Acceptor Sites</td>
<td>1.0094</td>
</tr>
<tr>
<td>avg. ratio</td>
<td>1.0248</td>
</tr>
</tbody>
</table>

According to [5], FastBRAIN is about 14 times faster than BRAIN, which agrees with our observations. To illustrate that FastBRAIN-2 can still enjoy the speed advantage of FastBRAIN, the execution time of FastBRAIN-2 is then compared to that of FastBRAIN. The results, reported in Table 4, show an average ratio of 1.0248. Namely, the execution time of FastBRAIN-2 matches that of FastBRAIN, which is again in accordance with our analysis above.

### 7. Conclusion

In this paper, we analyze the reasons that an underlying computational problem on FastBRAIN 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 subset selection for Boolean domains, thus the data will be compressed heavily in some cases.

### 8. Acknowledgement

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

### 9. References


[9] NN269 is available online from http://www.fruitfly.org/datasets/Human/GENIE_96/splicesets/
