2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Text Classifiers Evolved on a Simulated DNA Computer

Sun Kim, Min-Oh Heo, and Byoung-Tak Zhang

Abstract— **The use of synthetic DNA molecules for computing provides various insights to evolutionary computation. A molecular computing algorithm to evolve DNA-encoded genetic patterns has been previously reported in [1], [2]. Here we improve on the previous work by studying the convergence behavior of the molecular evolutionary algorithm in the context of text classification problems. In particular, we study the error reduction behavior of the evolutionary learning algorithm, both theoretically and experimentally. The individuals represent decision lists of variable length and the whole population takes part in making probabilistic decisions. The evolutionary process is to change each individual towards correct classification of training data, which is based on an error minimization strategy. The evolved molecular classifiers show a performance competitive to the standard algorithms such as naïve Bayes and neural network classifiers on the data set we studied. The possibility of molecular implementation by use of DNA-encoded individuals combined with simple molecular operations on a very big population distinguishes this approach from other existing evolutionary algorithms.**

I. Introduction

DNA computing is a computational paradigm that uses synthetic or natural DNA molecules as information storage media [3]. DNA molecules for computing applications were suggested by Adleman [4] and have now become a well-established field of biomolecular computing [5]. The techniques of molecular biology, such as polymerase chain reaction (PCR), gel electrophoresis, and enzymatic reactions, are used as computational operators for copying, sorting, and splitting/concatenating the information in the DNA molecules, respectively. Potential applications range from massively parallel computations, to new manufacturing techniques in nano-technology [6], [7], [8], [9], the creation of memories that can store very large data sets in minuscule spaces, and biological data processing in vitro. This differs significantly from the conventional computers, where basic operations are very fast, but each operation is executed sequentially. DNA computing performs each operation slowly, but in a massively parallel manner. The enormous capacity of DNA (over million fold compared to current electronic media) and the advances in recombinant biotechnology to manipulate DNA *in vitro* in the last 20 years, make this approach attractive and potentially very promising [10].

Classification is a typical problem in machine learning with a wide range of applications such as data mining, information retrieval, and bioinformatics. Text classification is the task of assigning one or more predefined categories (or class) to text documents [11]. In automatic text classification using machine learning techniques, the classifiers are learned using training documents and then assign labels to new documents. Due to the properties of document set, the text classification have the following issues [12], [13], [14]: (1) *High-dimensional feature space.* If distinct words occurring in the training documents are all used, text classification problems with a few thousand training examples can lead to 30,000 and more attributes. (2) *Sparse document vectors.* While there are huge number of features, each document contains only a small number of attributes. (3) *Heterogeneous use of terms.* Even though two documents are in the same class, the usage of terms can be different. (4) *High level of redundancy.* While there are many different attributes, often some of them is partly redundant. Based on the issues, DNA computing has the strength to find the category concepts from text documents, by operating given tasks in a massively parallel manner on a large hypothesis space derived from high storage density.

If one builds a molecular library which represents the probability distribution of training data, the library can be utilized as a classifier. In the library, training examples are encoded as DNA molecules by certain rules, and the frequency of molecules is maintained to be proportional to the probability of observed features from the examples. The probabilistic library model [1], [2] is based on this idea. The molecular library can be used as a probabilistic classifier by maintaining the library to represent the joint probability distribution of training examples and their class. The class label of new data is decided by computing the conditional probability of each class, and it is automatically performed by the library's own nature. To find the probabilistic distribution from training examples, they provide a learning procedure based on the Bayesian evolutionary update rule [15].

Here, we further study the convergence behavior of the probabilistic library model (PLM) in terms of classification problems. We propose an evolutionary learning method to converge into the minimum error for the probabilistic molecular classifiers. The individuals of the library represent decision lists of variable length and the whole population takes part in making probabilistic decisions. The evolutionary process is to adjust the number of multiple copies of each individual towards perfect classification. This strategy is based on the error minimization by stochastic gradient descent, which ensures the convergence to the local minimum error.

We have performed text classification tasks using Reuters-21578 test collection on a simulated DNA computer. The experimental results show the molecular classifiers using our evolutionary learning approach provides a comparable performance to other conventional methods such as naïve Bayes and neural network classifiers.

The authors are with the School of Computer Science and Engineering, Seoul National University, Seoul 151-744, Korea (email: skim@bi.snu.ac.kr; moheo@bi.snu.ac.kr; btzhang@bi.snu.ac.kr).

0-7803-9487-9/06/$20.00/©2006 IEEE

The remainder of the paper is organized as follows. In Section 2, the probabilistic molecular classifiers in DNA computing are described. The new evolutionary learning method for the DNA molecular classifiers is described in Section 3. In Section 4, experimental results on text data sets are reported. Conclusions are drawn in Section 5.

II. Probabilistic Molecular Classifiers

A classifier is to build a decision-making system f that outputs a label y given an input $\mathbf{x} = (x_1, \ldots, x_n)$. It is convenient to assume there exists a (unknown) target system f^* as an ideal model for f. Since we do not know the exact form of f^* the only information we can get is training examples collected from the input-output pairs of f^*. Given training data D of K labeled examples in the form

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{K} \quad (1)$$

$$\mathbf{x}_i = (x_{i_1}, x_{i_2}, \ldots, x_{i_n}) \in \{0,1\}^n \quad (2)$$

$$y_i \in \{0,1\}, \quad (3)$$

where $\mathbf{x}_i$ represents the DNA markers (set of features) in example i and y_i is its associated label.

Here, a library of DNA molecules represents individuals or decision rules and the library is maintained to classify new examples. Each individual is a conjunction of binary variables x_i and a label y, where we can refer to as a decision list. For instance, the individual $\mathbf{z} = (\mathbf{x}, y) = (x_1 = 1, x_2 = 0, x_3 = 1, y = 1)$, where the commas are interpreted as logical ANDs. The order of an individual is defined as the number of input variables in it. Thus, the individual $\mathbf{z}$ is of order 3.

We can represent each input variable as a sequence of nucleotides (A, T, G, and C). The output label can also be encoded as a DNA sequence. For example. if we use 10-mer to encode each binary variable and if there are 30 variables for inputs and one variable for class label, then DNA molecules of length 310-mer can represent an instance of training data or decision lists. The whole population consists of multiple copies of the decision lists and the number of copies is proportional to the importance of the decision list. The learning procedure of the molecular classifier is to update the number of copies to correctly discriminate training examples. The classifier works like a look-up table, but the probabilistic distribution of the data in the library facilitates classification computation. Keeping multiple copies of data items can also contribute to the robustness and fault-tolerance of the molecular computing system [16], [17].

Essentially, the DNA library represents the joint probability $P(X, Y)$ of the input pattern X and the output class Y. Since each individual, i.e. decision list, has a class label, given a query the decision can be made based on the individuals. Here, the whole population of decision lists can be considered as a single genetic program. By the genetic interpretation, the final decision is made by a consensus of the decisions of the individuals in the population. Since each individual is labeled either 0 or 1, the whole population is partitioned into two clusters. Given a new example, the class is determined by matching it against each and every decision list in the population and taking its majority class. This approach naturally makes use of the huge number of decision lists produced by the molecular genetic programming process to make decisions robust.

- 1. Let the library L represent the current empirical distribution $P(X, Y)$.
- 2. Present an input pattern $\mathbf{x}$.
- 3. Classify $\mathbf{x}$ using L as follows:
  - 3.1 Extract all library molecules matching with $\mathbf{x}$ into M.
  - 3.2 From M separate the molecules into classes:
    - Extract the molecules with label $Y = 0$ into M^0.
    - Extract the molecules with label $Y = 1$ into M^1.
  - 3.3 Compute $y^* = \arg\max_{Y \in \{0,1\}} |M^Y|/|M|$.

Fig. 1. The procedure of the molecular classifier to determine class labels based on DNA-encoded genetic programs.

The class is determined by computing the probability of each class conditional on the input $\mathbf{x}$, and then determining the class whose conditional probability is the highest, i.e.

$$y^* = \arg\max_{Y \in \{0,1\}} P(Y|\mathbf{x}) \quad (4)$$

$$= \arg\max_{Y \in \{0,1\}} \frac{P(Y, \mathbf{x})}{P(\mathbf{x})}, \quad (5)$$

where $P(Y, \mathbf{x}) = P(Y|\mathbf{x})P(\mathbf{x})$ and Y represents the candidate classes.

A method for realizing Equation (4) is to initialize the library with nth order decision lists and evolve their distributions. That is, the empirical probability distribution $P(X, Y)$ can be represented by a set of point estimators that constitute the DNA library L of decision lists:

$$P(X, Y) \approx \frac{1}{|L|} \sum_{i=1}^{|L|} f_i^{(n)}(X_1, X_2, \ldots, X_n, Y), \quad (6)$$

where $f_i^{(n)}(X_1, X_2, \ldots, X_n, Y)$ is the ith decision list of order n and $|L|$ is the library size. This approximation can be made arbitrarily accurate by increasing the library size $|L|$. More theoretical background of the probabilistic classifier can be found in [2].

Figure 1 summarizes the procedure for decision making using the molecular genetic programs in a test tube. Given an input $\mathbf{x}$ all the molecules that match the input is extract from the library. These molecules will have class labels from which the majority label is decided as the class of the input pattern. A class label is a sequence appended to denote the class to which the pattern belongs. *In silico* implementation of this method, the given query has to be matched against each and every element of the library. *In*

vitro molecular computation this can be done in a massively parallel fashion. Instead of a single $\mathbf{x}$, multiple copies (up to the number of population size) of it is used so that they can be matched with library elements in parallel. In addition, the input pattern $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ can be chopped into n DNA pieces representing $x_1, x_2, \ldots, x_n$, respectively, so that each of them can be matched separately to decision lists. The decision can be made by comparing the number of elements in class 1 with those in class 0.

In Step 3.1, the count $c(\mathbf{x})$ of $\mathbf{x}$ in M approximates the probability of observing the pattern:

$$c(\mathbf{x})/|L| = |M|/|L| \approx P(\mathbf{x}). \tag{7}$$

Step 3.2 essentially computes the frequencies $c(Y|\mathbf{x})$ of molecules belonging to different classes Y. These are an approximation of the conditional probabilities given the pattern, i.e. a posteriori probabilities:

$$c(Y|\mathbf{x})/|M| = |M^Y|/|M| \approx P(Y|\mathbf{x}). \tag{8}$$

Thus, the protocol computes the maximum a posteriori (MAP) criterion:

$$\begin{aligned} y^* &= \arg\max_{Y\in\{0,1\}} c(Y|\mathbf{x})/|M| && (9)\\ &= \arg\max_{Y\in\{0,1\}} c(Y|\mathbf{x}) && (10)\\ &\approx \arg\max_{Y\in\{0,1\}} P(Y|\mathbf{x}). && (11) \end{aligned}$$

It is worth noting that for classification purposes only the relative frequency or concentration of the molecular labels are important.

III. Evolutionary Learning Methods for Improved Classification Performance

In the previous section it is assumed that the library represents the proper probability distribution of input patterns and their classes. The probabilistic update procedure has been proposed by Zhang and Jang [1], [2]. Our method basically follows the same procedure, which is motivated from *in vitro* evolution [18], [19], [20]. *In vitro* evolution starts with a library of molecules and evaluates their goodness. Then, the fitter ones are selected as the basis for generating mutants that build the next generation of library. The iteration of the selection-amplification cycle can come up with the identification of molecules that best fits to the target function.

The key idea in our approach is to use gradient descent to find a proper probability distribution of the library that best fits in the training examples. In the molecular classifiers, each individual is maintained in multiple copies, and the learning procedure is to adjust the number of copies to reduce the magnitude of the classification error. It is a similar problem to find optimal weight vectors in machine learning methods.

For the input pattern $\mathbf{x}_i$ and its label y_i, the total quantity c for a class Y is given by

$$c(Y|\mathbf{x}_i) = \sum_{j=1}^{|L|} (c_j + \delta_j) I_{\mathbf{z}_j=(\mathbf{x},Y)} + \varepsilon, \tag{12}$$

where $I_{\mathbf{z}_j=(\mathbf{x},Y)} \in \{0,1\}$ denotes the indication function that is one if the individual $\mathbf{z}_j = (\mathbf{x}, Y)$, and zero otherwise, c_j is the number of copies for the $\mathbf{z}_j$, δ_j is the cross-hybridization error occurred by mismatches between the input example and the library elements due to the potential for formation of double-stranded DNA duplexes, and ε is the detection error occurred by inaccurate quantity measurement of hybridized elements. In particular, δ_j is closely related to DNA sequence design, and it can be reduced by finding non-crosshybridizing sequences [21]. To make the description simple, we here assume δ_j and ε are very small enough to be ignored, then the total quantity $c(Y|\mathbf{x}_i)$ is simplified to $\sum_{j=1}^{|L|} c_j I_{\mathbf{z}_j=(\mathbf{x},Y)}$. By introducing a weight vector $\mathbf{w}$, we can approximate the conditional probability given the input $\mathbf{x}_i$ as follows:

$$P(Y|\mathbf{x}_i) \approx \frac{c(Y|\mathbf{x}_i)}{|M|} = \sum_{j=1}^{|L|} w_j I_{\mathbf{z}_j=(\mathbf{x},Y)}, \tag{13}$$

where $w_j = c_j/|M|$.

For the training example $\mathbf{x}_i$ and the class Y, an error e_i is defined as follows:

$$e_i = P^*(Y|\mathbf{x}_i) - P(Y|\mathbf{x}_i), \tag{14}$$

where $P^*(Y|\mathbf{x}_i) = I_{y_i=Y} \in \{0,1\}$ is the target probability for the training example $\mathbf{x}_i$, where $I_{y_i=Y}$ is the indication function that is one if $y_i = Y$, and zero otherwise. Our objective is to find the weight vector $\mathbf{w}$ minimizing the cost function $E(\mathbf{w})$, defined in terms of the error e_i as follows:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i\in D} e_i^2. \tag{15}$$

The cost function $E(\mathbf{w})$ can be minimized by using the gradient descent approach, i.e.

$$w_j \leftarrow w_j + \Delta w_j, \tag{16}$$

where

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}, \tag{17}$$

and η is the learning rate determining the strength of reproduction for each generation. Since the $\frac{\partial E}{\partial w_j} = \sum_{i\in D}(P^*(Y|\mathbf{x}_i) - P(Y|\mathbf{x}_i))(-I_{\mathbf{z}_{ji}=(\mathbf{x},Y)})$, the update rule for gradient descent is defined as follows:

$$\Delta w_j = \eta \sum_{i\in D} (P^*(Y|\mathbf{x}_i) - P(Y|\mathbf{x}_i)) I_{\mathbf{z}_{ji}=(\mathbf{x},Y)}. \tag{18}$$

Additionally, we want to perform the weight updates for each training example. It is easily approached by stochastic gradient descent, which is to approximate the gradient descent by updating $\mathbf{w}$ incrementally, following the calculation of the error for each example. The modified update process is given by

$$\begin{aligned} \Delta w_j &= \eta(P^*(Y|\mathbf{x}) - P(Y|\mathbf{x}))I_{\mathbf{z}_j=(\mathbf{x},Y)} \quad (19) \\ &\approx \Delta c_j, \quad (20) \end{aligned}$$

where $P^*(Y|\mathbf{x})$, $P(Y|\mathbf{x})$, and $I_{\mathbf{z}_j=(\mathbf{x},Y)}$ are the target value, the system output, and the indicator for jth individual in the library respectively. Therefore, the update is to increase or decrease the number of copies c_j with a certain amount of value, i.e. Δc_j towards the opposite direction of the incorrect output.

Since the library includes decision lists for input variables, given the input $\mathbf{x}$ and its label y the library L is divided into four groups, i.e.

$$L = c(\mathbf{x}, y) + c(\overline{\mathbf{x}}, \overline{y}) + c(\mathbf{x}, \overline{y}) + c(\overline{\mathbf{x}}, y), \quad (21)$$

where $\overline{\mathbf{x}}$ and $\overline{y}$ are the complement elements of $\mathbf{x}$ and y, and the library is updated in four different ways based on the groups. For example, if the input $(x_1 = 0, y = 0)$ is misclassified, i.e. system output $y^* = 1$, the copies of $(x_1 = 0, y = 0)$ and $(x_1 = 1, y = 1)$ should be increased and the copies of $(x_1 = 0, y = 1)$ and $(x_1 = 1, y = 0)$ should be decreased by Equation (13) and Equation (19). Thus, the update rule for $y = 0$ is given by

$$\Delta L = \Delta c(\mathbf{x}, 0) + \Delta c(\overline{\mathbf{x}}, 1) - \Delta c(\mathbf{x}, 1) - \Delta c(\overline{\mathbf{x}}, 0). \quad (22)$$

If the input $(x_1 = 0, y = 1)$ is misclassified, i.e. $y^* = 0$, the copies of $(x_1 = 0, y = 1)$ and $(x_1 = 1, y = 0)$ should be increased and the copies of $(x_1 = 0, y = 0)$ and $(x_1 = 1, y = 1)$ should be decreased in the same way. Thus, the update rule for $y = 1$ is given by

$$\Delta L = \Delta c(\mathbf{x}, 1) + \Delta c(\overline{\mathbf{x}}, 0) - \Delta c(\mathbf{x}, 0) - \Delta c(\overline{\mathbf{x}}, 1). \quad (23)$$

According to Equation (22) and Equation (23), the error correction process is summarized as follows:

$$L \leftarrow L + \Delta L, \quad (24)$$

where

$$\Delta L = \Delta c(\mathbf{x}, y) + \Delta c(\overline{\mathbf{x}}, \overline{y}) - \Delta c(\mathbf{x}, \overline{y}) - \Delta c(\overline{\mathbf{x}}, y). \quad (25)$$

Addition process for Δc can be implemented by polymerase chain reaction (PCR) and removal can be done by extraction of the corresponding molecules. The update process relies upon the reliability of DNA extraction technology.

The update of the library is similar to evolutionary computation with the additional feature that the presentation of a training example proceeds one generation of the library (or population). This is also a learning procedure since the library improves its classification performance as new examples are presented. It would be interesting that the probabilistic molecular model can be seen as a neural network to minimize an error function by stochastic gradient descent. In a broader view, patterns appearing on the input or output nodes of a network are viewed as samples from probability densities, and a network is viewed as a probabilistic model that assigns probabilities to the patterns [22]. The learning of a network is thereby to find weights that look probable in the light of observed data. However, the probabilistic molecular classifiers based on ensemble approaches are different from neural networks by decision making process.

- 1. Let the library L represent the current empirical distribution $P(X, Y)$.
- 2. Get a training example $(\mathbf{x}, y)$.
- 3. Classify $\mathbf{x}$ using L as described in the previous section. Let this class be y^*.
- 4. Update L if $y^* \neq y$
  - $L_n \leftarrow L_{n-1} + \Delta c(\mathbf{x}, y)$,
  - $L_n \leftarrow L_{n-1} + \Delta c(\overline{\mathbf{x}}, \overline{y})$,
  - $L_n \leftarrow L_{n-1} - \Delta c(\mathbf{x}, \overline{y})$,
  - $L_n \leftarrow L_{n-1} - \Delta c(\overline{\mathbf{x}}, y)$.
- 5. Goto step 2 unless the termination condition is met.

Fig. 2. The evolutionary algorithm to adjust the library elements of DNA-encoded genetic patterns.

Figure 2 shows the evolutionary process to adjust the library elements of DNA-encoded genetic patterns, which is based on the stochastic gradient descent. We start with a random collection of DNA strands. Each DNA sequence represents an instance $(\mathbf{x}, y)$ of a vector (X, Y) of random variables of interest in the problem domain. Without any prior knowledge the DNA sequences are generated to represent uniform distribution of input variables. As a new training example $(\mathbf{x}, y)$ is observed, the patterns matching $\mathbf{x}$ is extracted from the library. The class y^* of $\mathbf{x}$ is determined by the classification procedure described in the previous section. Then, the matching patterns are modified in their frequency depending on their contribution to the correct or incorrect classification of $\mathbf{x}$. If the label y^* is correct, no action is performed because current library classifies the examples correctly. If the label y^* is incorrect, the library is modified according to Equation (25).

IV. Experiments Results

A. Data Set

For experiments, we performed text filtering tasks using Reuters-21578 test collection [23], which is widely used benchmark in text categorization research. The documents are manually labeled with the 135 pre-defined categories in 'TOPICS' set. The number of documents for each category have been counted and sorted by descending order. The top-ranked classes, 'acq' and 'earn' were then chosen to

TABLE I

EVALUATION CONTINGENCY TABLE.

Answer	Output as 'positive'	Output as 'negative'
Positive	a	c
Negative	b	d

create two data set. There were 2,362 documents in 'acq' class and 3,944 documents in 'earn' class, and they were pooled into two data set separately as positive examples. For negative examples, the same number of documents were randomly selected among the documents which did not belong to any pre-defined class. As a result, 'acq' data set has total 4,724 examples including 2,362 positive examples and 2,362 negative examples, and 'earn' data set has total 7,888 examples including 3,944 positive examples and 3,944 negative examples.

For document classification, a word is basically considered as a variable, hence we preprocessed each data set as follows:

1) Remove stop list: remove often used words with no specific meaning such as 'a', 'an', or 'the'.
2) Perform stemming: reduce words to their stem or root forms.
3) Count the word frequencies in the examples.
4) Remove any word that has below 100 of total frequency in the data set.
5) Set '1' as the word count if the word frequency is not zero, '0' otherwise.

As a result, we obtained 589 of input variables for 'acq' data set and 714 of input variables for 'earn' data set. Each data set naturally has a matrix form that has words (or input variables) as column and the word counts as row.

B. Performance Measure

When binary scales are used for both answer and system output, a table can be established showing how the document set is divided by these two measures (Table I). By the table, the classification performance is evaluated as follows:

$$\begin{aligned} accuracy &= \frac{a+d}{a+b+c+d} \\ precision &= \frac{a}{a+b} \\ recall &= \frac{a}{a+c} \end{aligned}$$

C. Simulation Results

For DNA classifier simulations, the data set was divided into training and test set. 70% of documents was used for training data and the rest was used for test data. Both training and test set contained same amount of positive and negative examples. The simulation was repeated ten times and the fitness and performance results were averaged.

In the original framework of the probabilistic molecular classifiers, the library elements are initialized to contain each and every conjunction of all possible order for input

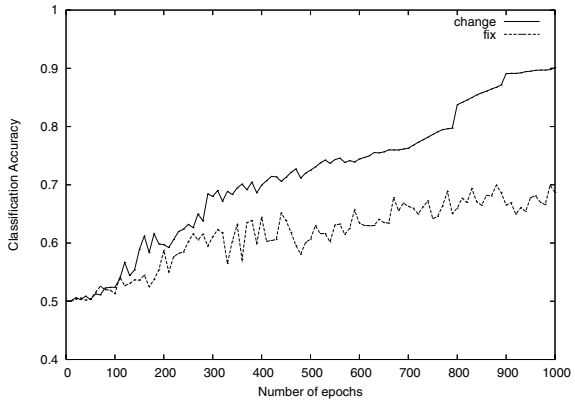

Fig. 3. Fitness evolution of the population for 'acq' data set. 'change' indicates the copy rates are changed over generations and 'fix' indicates the copy rates are fixed over generations.

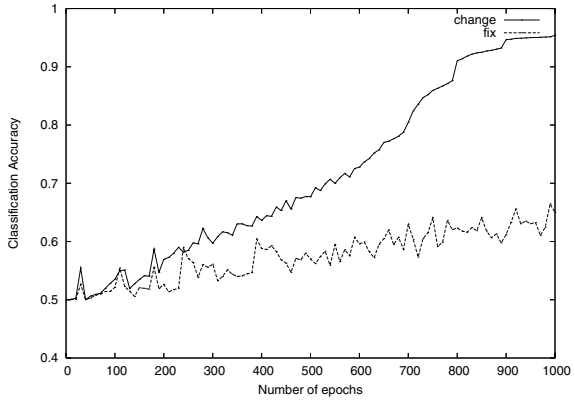

Fig. 4. Fitness evolution of the population for 'earn' data set. 'change' indicates the copy rates are changed over generations and 'fix' indicates the copy rates are fixed over generations.

variables. Thus, the total number of the different library elements is $\sum_{k=1}^{n} {}_nC_k \cdot 2^k \cdot 2$, where ${}_nC_k$ denotes the number of combinations to choose k variables out of n. Each element is a decision list, and whole population consists of the multiple copies of the different library elements. However, our focus is to present a locally converged molecular algorithm, hence we only use order one variables for the experiments. The population including order one variables means that the input variables are conditionally independent for the target value. Therefore, the population includes $(x_1 = 0, y = 0), (x_1 = 0, y = 1), (x_1 = 1, y = 0), (x_1 = 1, y = 1), (x_2 = 0, y = 0), (x_2 = 0, y = 1)$,.... Since the order one variables are used to present the whole population, we adapt an additional individual called *bias* to increase the flexibility and performance of linear representation. The *bias* individual always votes for '$y = 1$' in every step, while the copies of *bias* are modified in the same manner with other individuals.

Setting the learning parameter Δc is important to balance the adaptability and stability of the molecular library as a probabilistic model for the training data. The larger Δc is, the larger gets the change of the distribution, and the smaller Δc is, the smaller gets the change of the distribution. We

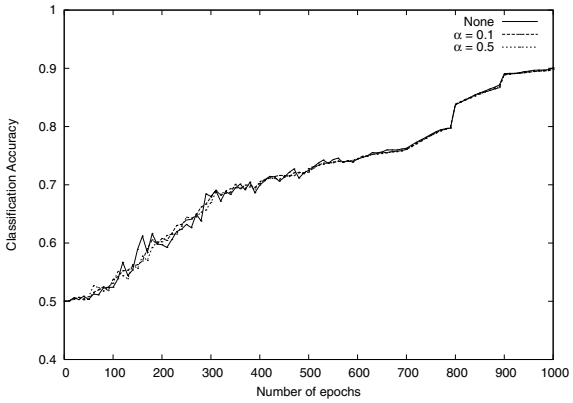

Fig. 5. Fitness evolution of the population for 'acq' data set when there exists the copy noise α in the experiments.

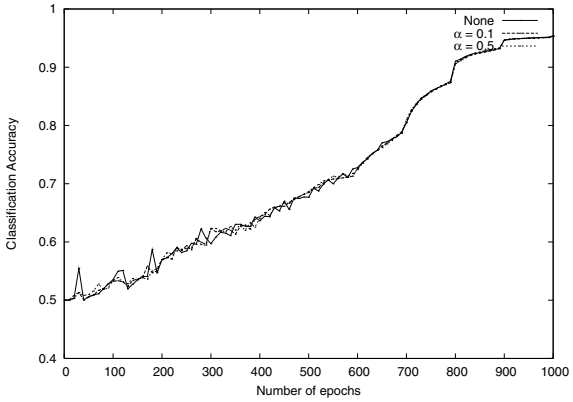

Fig. 6. Fitness evolution of the population for 'earn' data set when there exists the copy noise α in the experiments.

define Δc as the rate of the number of copies to be amplified or removed. In the experiments, Δc was started from 0.01 of copy rate, and it was decreased to half percent of previous copy rate in every 100 epoch. The number of copies for each element was set to 10^6.

Figure 3 and Figure 4 depict the evolution of the fitness as generation goes on. The graphs show that the fitness changes are stabilized after 800 epochs and converged to certain points by our copy rate policy. The fitness curves are increased gradually by decreasing the copy rate over generations, and almost fixed after 900 epochs. For the fixed copy rate policy, we set the copy rate $\Delta c = 0.01$ and the learning curves show poor classification accuracy. The fitness is slowly increased with fluctuations for both data sets, but the convergence cannot be confirmed. It shows the adjustment of copy rate is important to achieve both convergence and high accuracy.

Figure 5 and Figure 6 show the fitness changes over generations when there exists the copy noise α, i.e. a noise occurred by the incomplete adjustments for the target elements. The copy rate is randomly generated between 0 and $(1+\alpha)\Delta c$ in every generation. The experimental results show no difference in any noise level and the library is converged into almost same points for all cases. It is because the evolutionary learning tries to fix noise errors by the stochastic gradient descent method. Even though there exists mismatch or detection noise, our approach can provide robust results according to its learning behavior.

TABLE II

PERFORMANCE COMPARISON OF THE MOLECULAR CLASSIFIER AND CONVENTIONAL CLASSIFIERS FOR 'ACQ' DATA SET

Performance	Molecular	naïve Bayes	BPNN
Accuracy	0.86	0.87	0.87
Precision	0.86	0.85	0.82
Recall	0.85	0.89	0.95

TABLE III

PERFORMANCE COMPARISON OF THE MOLECULAR CLASSIFIER AND CONVENTIONAL CLASSIFIERS FOR 'EARN' DATA SET

Performance	Molecular	naïve Bayes	BPNN
Accuracy	0.88	0.88	0.90
Precision	0.85	0.84	0.86
Recall	0.94	0.95	0.97

Table II and Table III present the performance comparison of the molecular classifier and other conventional methods, naïve Bayes and backpropagation neural network (BPNN) classifiers. For 'acq' data set, the molecular classifier shows 86% of accuracy, 86% of precision and 85% of recall. It is slightly better performance than other classifiers in precision and less performance in recall, but the overall performance is similar, and has no significant difference. For 'earn' data set, the molecular classifier shows 88% of accuracy, 85% of precision and 94% of recall. We observe that the molecular classifier still provides comparable performance in precision and accuracy for 'earn' data set, whereas the recall rate is relatively low than other methods. As a result, the molecular classifier tends to focus on improving precision rather than recall. But, we need to perform further experiments to ensure it since the difference is very small and there are possibilities for refinement. Also, for 'earn' data set, the molecular classifiers could overfit the training examples, showing over 95% of classification accuracy. It may cause less performance on test examples. It should be mentioned that we only used order one variables as individuals. One strength of molecular classifiers is to allow the use of huge population size, hence if higher order variables are used, it would provide better performance than current results.

To summarize, the simulation results show that the learning behavior is stable and converged using the copy rate change policy. The fixed copy rates cause instability of the convergence behavior, also small learning rates can lead very slow convergence. The proposed method shows a robustness under the copy noise, which comes from the error correction process. The performance comparison of the molecular classifier and other conventional approaches shows similar performance. It provides the possibility of *in vitro* molecular computing as classifiers.

V. Conclusions

We presented an evolutionary learning method for the molecular classifiers on a simulated DNA computer. Using the stochastic gradient descent method, we derived a learning procedure for the probabilistic molecular classifiers, which provides a theoretical background for error minimization in the classification framework.

The method was used to solve text filtering problems on the Reuters-21578 test collection. Various policies for biasing individuals and changing the copy rate were examined to improve the classification performance. While the results showed some fluctuations caused by on-line learning, the fitness curve stabilized after a certain number of epochs. Our analysis shows that even though the individuals, i.e. decision lists are simple, their population has a powerful representation capability equivalent to a disjunctive normal form. The molecular classifier achieved a comparable performance to other classifiers such as naïve Bayes and neural networks. It is remarkable that the decision lists are composed of boolean variables and the actual output is also boolean. Nonetheless, the performance results are relatively robust against incomplete information. Note that naïve Bayes and neural network classifiers handle real values inside. The redundancy of DNA-coded genetic patterns and the error correction process provide a robustness against noise such as imperfect hybridization and detection errors.

Acknowledgment

This work was supported by the Korea Ministry of Science and Technology through National Research Lab (NRL) project, by the Ministry of Industry and Commerce through the Molecular Evolutionary Computing (MEC) Project, and the Ministry of Education and Human Resources Development under the BK21-IT Program.

References

[1] B.-T. Zhang and H.-Y. Jang, A Bayesian Algorithm for In Vitro Molecular Evolution of Pattern Classifiers, *DNA Computing 10*, LNCS 3384, pp. 458–467, 2005.

[2] B.-T. Zhang and H.-Y. Jang, Molecular Programming: Evolving Genetic Programs in a Test Tube, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*, 2, pp. 1761–1768, 2005.

[3] J.Y. Lee, S.-Y. Shin, T.H. Park, and B.-T. Zhang, Solving Traveling Salesman Problems with DNA Molecules Encoding Numerical Values, *Biosystems*, 78, pp. 39–47, 2004.

[4] L.M. Adleman, Molecular Computation of Solutions to Combinatorial Problems, *Science*, 266, pp. 1021–1024, 1994.

[5] A. Neel, M. Garzon, and P. Penumatsa, Soundness and Quality of Semantic Retrieval in DNA-based Memories with Abiotic Data, *Proceedings of the IEEE Conference on Evolutionary Computation (CEC-2004)*, pp. 1889–1895, 2004.

[6] R.S.Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund, and L. Adleman, Solution of a 20-Variable 3-SAT Problem on a DNA Computer, *Science*, 296, pp. 499–502, 2002.

[7] C.C. Maley, DNA Computation: Theory, Practice, and Prospects, *Evolutionary Computation*, 6, pp. 201–229, 1998.

[8] E. Winfree, F. Liu, L.A. Wenzler, and N.C. Seeman, Design and Self-assembly of Two-dimensional DNA Crystals, *Nature*, 394, pp. 539–544, 1998.

[9] S. Liao and N.C. Seeman, Translation of DNA Signals into Polymer Assembly Instructions, *Science*, 306, pp. 2072–2074, 2004.

[10] M.H. Garzon, K.V. Bobba, and A. Neel, Efficiency and Reliability of Semantic Retrieval in DNA-Based Memories, *DNA Computing 9*, LNCS 2943, pp. 157–169, 2004.

[11] N. Kushmerick, E. Johnston, and S. McGuinness, Information Extraction by Text Classification, *In IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, 2001.

[12] T. Joachims, A Statistical Learning Model of Text Classification for Support Vector Machines, *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR-01)*, pp. 128–136, 2001.

[13] F. Sebastiani, Machine Learning in Automated Text Categorization, *ACM Computing Surveys*, 34, pp. 1–47, 2002.

[14] T. Liu, Z. Chen, B. Zhang, W. Ma, and G. Wu, Improving Text Classification using Local Latent Semantic Indexing, *Proceedings of the IEEE International Conference on Data Mining (ICDM-2004)*, pp. 162–169, 2004.

[15] B.-T. Zhang, A Unified Bayesian Framework for Evolutionary Learning and Optimization, *Advances in Evolutionary Computation*, Chapter 15, pp. 393–412, 2003.

[16] J. Reif and T. LaBean, Computationally Inspired Biotechnologies: Improved DNA Synthesis and Associative Search Using Error-Correcting Codes and Vector-Quantization, *Lecture Notes in Computer Science*, 2054, pp. 145–172, 2001.

[17] E. Winfree and R. Bekbolatov, Proofreading Tile Sets: Error Correction for Algorithmic Self-Assembly, *DNA Computing 9*, LNCS 2943, pp. 126–144, 2004.

[18] M.C. Wright and G.F. Joyce, Continuous In Vitro Evolution of Catalytic Function, *Science*, 276, pp. 614–617, 1997.

[19] D.S. Wilson and J.W. Szostak, In Vitro Selection of Functional Nucleic Acids, *Annual Review of Biochemistry*, 68, pp. 611–647, 1999.

[20] J. Chen, E. Antipov, B. Lemieux, W. Cedeño, and D.H. Wood, DNA Computing Implementing Genetic Algorithms, *Proceedings of the DIMACS Workshop on Evolution as Computation*, pp. 39–49, 1999.

[21] M.H. Garzon and R.J. Deaton, Codeword Design and Information Encoding in DNA Ensembles, *Natural Computing*, 3, pp. 253–292, 2004.

[22] M.I. Jordan and C.M. Bishop, Neural Networks, *ACM Computing Surveys*, 28, pp. 73–75, 1996.

[23] D.D. Lewis, Reuters-21578 text categorization test collection, Distribution 1.0, http://www.daviddlewis.com/resources/testcollections.