PROBABILISTIC NEURAL NETWORKS WITH COMPETITIVE LEARNING ALGORITHM

Mohammad Bagher Menhaj and Farshid Delgosha
Amirkabir University of Technology
Electrical Engineering Department
424 Hafez Ave., 15914 Tehran, Iran
Phone: +9821-6466009, Fax: +9821-6406469
email: menhaj@cic.aku.ac.ir

ABSTRACT: Baysian classifier is the optimum system because it minimizes the decision risk. The difficulty of this classifier is its need to the conditional Probability Density Functions (PDFs) of all classes. Parametric PDF estimators cannot fit their parameters to many actual random processes. Therefore, non-parametric estimators have been introduced. These estimators could be realized by a neural network structure called Probabilistic Neural Network (PNN). However, realizing PNN requires a large amount of memories. To remedy this problem, Gaussian mixture estimators have been developed. These estimators are actually semi-parametric. In this paper, we propose a PNN based Gaussian mixture estimator used for classification task. The parameters of this network are adjusted with a competitive learning type algorithm. The simulation results highlight the merit of the proposed method.

KEYWORDS: Probabilistic Neural Networks, Competitive Learning, Classification, Pattern Recognition

INTRODUCTION

From the decision theory, we know that Baysian criterion is the optimum solution to the classification problem since it minimizes the decision risk. Usually the difficulty of this criterion is the lack of knowledge about the probability density function (PDF) of each class conditioned on the data vector. Parzen has introduced a family of PDF estimators that are consistent and asymptotically normal [Parzen, (1962)]. These estimators are non-parametric and behave well towards the sample vectors of almost any random process. The Parzen PDF estimator has been realized with a neural network type structure called Probabilistic Neural Networks in [Specht, (1988, 1990)]. Because the size of this network is strongly dependent on the number of training data vectors, it requires a large amount of memories and extensive computational time. Specht in 1990 has also introduced the polynomial Adaline (Padaline) as a solution to this problem [Specht, (1990)]; but data clustering seems to be a more reasonable approach. In this approach, neighboring sample vectors form a cluster and each cluster is represented by a prototype vector having a nearly equivalent effect in the PDF estimation. In other words, instead of employing a non-parametric estimator a semi-parametric one can be used to limit the number of unknown parameters. Streit and Traven make use of mixture Gaussian estimator, which indeed represent a semi-parametric PDF estimator, and adjust its parameters with the maximum likelihood method [Streit, (1994)] [Traven, (1991)].
None of the above approaches takes advantage of the training capabilities of neural networks estimator for parameters adjustment. In the paper, we employ a mixture Gaussian estimator, realize by a neural network type of PNN structure as given in [Streit, (1994)] and propose a competitive learning type algorithm to adjust the parameters of the estimator. By this method, size of the network and storage memory, in fact time of computations, is remarkably reduced while at the same time the neural network learning capabilities are exploited.
The remainder of the paper is organized as follows. Section 2 is a glossary to some PDF techniques. In Section 3, the classification problem and the structure of PNN is introduced. The competitive learning type algorithm of this network is proposed in Section 4. Section 5, is devoted to experimental results. Finally, Section 6, concludes the paper.

SOME PDF ESTIMATION TECHNIQUES

Suppose that we have observed *n* sample vectors of a random process denoted by $\{X_1, X_2, \ldots, X_n\}$. Parzen estimates the PDF as follows [Parzen, (1962)]:

$$\hat{f}(X) = \frac{1}{nh^d}\sum_{i=1}^{n} K\left(\frac{X - X_i}{h}\right) \tag{1}$$

where d is the dimension of sample vectors, K is a real valued kernel function and h is a suitably chosen positive number. In the clustering method, neighboring sample vectors are grouped as clusters and each cluster is represented by a prototype vector. If m_j is the prototype vector of cluster j and Gaussian kernel function–a traditional choice–is used then, we may estimate the PDF as follows

$$\hat{f}(X) = \frac{1}{(2p)^{d/2}|\Sigma|^{1/2}}\sum_{j=1}^{m} p_j \exp\left\{-\frac{1}{2}(X - m_j)^T \Sigma^{-1}(X - m_j)\right\} \tag{2}$$

where m is the number of clusters, p_j is the proportion of cluster j in the estimation, Σ represents the covariance matrix, which is positive definite, and superscript T denotes transpose. The p_js are positive numbers between 0 and 1 and satisfy

$$\sum_{j=1}^{m} p_j = 1. \tag{3}$$

Note that the number of clusters, m, has to be less than n, the number of sample vectors.

PROBLEM STATEMENT AND PNN ARCHITECTURE

We want to classify the d-dimensional data vector X as one of M classes. Let $f_k(X)$ and a_k denote the PDF and a priori probability of class k, respectively. Let ℓ_{jk} denote the loss associated with classifying X into class j when the correct decision should have been class k. The risk of classifying X into class j is defined as

$$r_j(X) = \sum_{k=1}^{M} \ell_{jk}\, a_k\, f_k(X), \qquad j = 1, 2, \ldots, M. \tag{4}$$

Baysian classifier is the minimum risk decision rule; i.e., X belongs to the jth class if $j = \arg\min\{r_j(X)\}$. Equation (4) shows that when the a priori probability of all classes are equal then knowledge of $f_k(X)$ is adequate for classification (l_{jk}s, which are dependent on the type of classification problem, should be determined by the user). In this paper, the PDF of class j is approximated by the mixture Gaussian estimator of equation (2):

$$\hat{f}_j(X) = \sum_{i=1}^{C_j} p_{ij}\, p_{ij}(X), \qquad j = 1, 2, \ldots, M \tag{5}$$

where C_j is the number of components (clusters) of the jth class and $p_{ij}(X)$ is

$$p_{ij}(X) = \frac{1}{(2p)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(X - m_{ij})^T \Sigma^{-1}(X - m_{ij})\right\}, \qquad i = 1, 2, \ldots, C_j, \quad j = 1, 2, \ldots, M. \tag{6}$$

In the above equation, m_{ij} is the prototype vector of the ith component in class j.
It has been shown that the task of choosing the optimum class, with minimum risk, could be done by a neural network structure [Streit, (1994)]. This can be done as follows. Noting that because Σ in equation (6) is a positive definite matrix, it can be factorized as $\Sigma = LL^T$. Therefore, the expression inside the "exp" argument in equation (6) could be written as

$$(X - m_{ij})^T \Sigma^{-1}(X - m_{ij}) = \left\|Y - m'_{ij}\right\|^2, \tag{7}$$

where

$$Y = L^{-1}X, \tag{8}$$

$$m'_{ij} = L^{-1}m_{ij} \tag{9}$$

($\|.\|$ is the second norm operator). Considering equations (7) and (8), and ignoring the constant terms, the network shown in Fig. 1 with d neurons computes $p_{ij}(X)$. Note that in this figure, instead of the commonly used sigmoid function, exponential function is used as the neuron activation function. In fact, L^{-1} plays the role of whitening filter

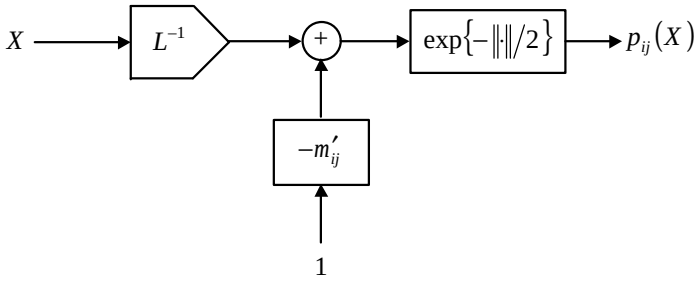

Figure 1: The network structure used for realizing $p_{ij}(X)$.

for the input vector space. As Fig. 2 shows, using C_j modules, each having the form depicted in Fig. 1, the value of $f_j(X)$ can be determined (j = 1, 2, …, M). To obtain $r_j(X)$s, we are required to establish a network with M subnetworks of the form given in Fig. 2.The final section of the network is a minimization block. This block is responsible for finding the neuron with minimum output. Consequently, PNN is a tree layered neural network; however if we have

$$\ell_{jk} = \begin{cases} 1, & j \neq k \\ 0, & j = k \end{cases} \tag{10}$$

then computation of the class risks is not required and PNN will have only two layers followed by a maximization block.

COMPETITIVE LEARNING APPLIED TO PNN

In order to train the network, the sample vector set $\Im_j = \{X_{ij}\}_{i=1}^{G_j}$ for the jth class containing G_j samples is available (j = 1, 2, …, M). The union of these sets is denoted by $\Im$

$$\Im = \bigcup_{j=1}^{M} \Im_j \,. \tag{11}$$

Before the training algorithm begins, some preprocessing must be done. At first, the covariance matrix of each cluster must be estimated. This can be done by the following equation

$$\hat{\Sigma} = \frac{1}{C^2 \#(\Im)} \sum_{X \in \Im} (X - \hat{m})(X - \hat{m})^T \tag{12}$$

where C is the total number of components

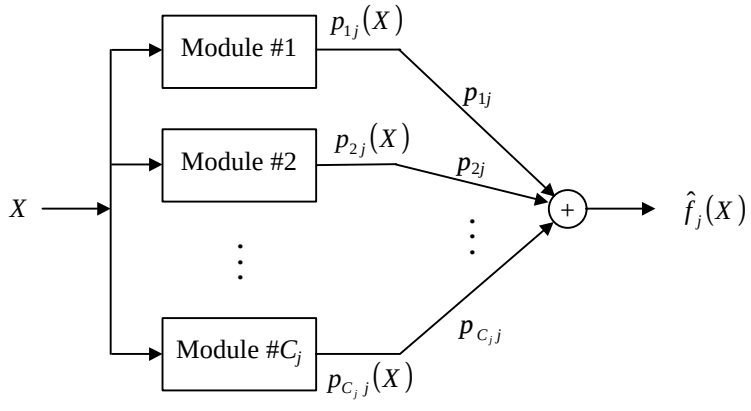

Figure 2: The network structure used for realizing $\hat{f}_j(X)$.

$$C = \sum_{j=1}^{M} C_j \,, \tag{13}$$

#(.) means the cardinality of the set and $\hat{m}$ is the estimation of total mean

$$\hat{m} = \frac{1}{\#(\Im)} \sum_{X \in \Im} X \,. \tag{14}$$

Thereafter, an estimation of the matrix square root of $\hat{\Sigma}$, denoted by $\hat{L}$, must be found. This can be obtained by one of the factorization methods such as Cholesky method.

The next step is obtaining the whitened training set for each class, denoted by $\Im'_j$. If $Y \in \Im'_j$ is the whitened vector corresponding to $X \in \Im_j$, then we may have

$$Y = \hat{L}^{-1} X \,. \tag{15}$$

Obtaining the whitened data vectors, training algorithm begins. Before that, the mean vectors $\hat{m}'_{ij}$ and component proportions $\hat{p}_{ij}$ must be initialized. The best initialization values for $\hat{m}'_{ij}$ s is something near the mean of each class

$$\hat{m}'_{ij} = \hat{m}'_j + \mathbf{n}\,, \qquad i = 1, 2, \ldots, C_j, \quad j = 1, 2, \ldots, M. \tag{16}$$

where

$$\hat{m}'_j = \frac{1}{\#(\Im'_j)} \sum_{Y \in \Im'_j} Y \,, \qquad j = 1, 2, \ldots, M. \tag{17}$$

and **n** is a vector whose elements are randomly selected in the interval [–1, 1] and normalized in such a way that the magnitude of **n** becomes a fraction, say 1/2, of the magnitude of $\hat{m}'_j$. The component proportions are equally initialized ($\hat{p}_{ij} = 1/C_j$, $i = 1, 2, \ldots, C_j$, $j = 1, 2, \ldots, M$). Relaxing the $\hat{p}_{ij}$ s from the condition of equation (3), softmax transformation is used as:

$$\hat{p}_{ij} = \frac{\exp(u_{ij})}{\sum_{i=1}^{C_j} \exp(u_{ij})}, \tag{18}$$

hence the u_{ij}s could be freely changed.

The training algorithm in addition to updating u_{ij}s must move the vectors $\hat{m}'_{ij}$ towards the center of each cluster. Suppose that Y is randomly chosen from the whitened data vector set $\Im'_j$ and is applied to the network. Also, suppose that the network has classified it as the j'th class. If $j' = j$ then classification is correctly performed and the closest $\hat{m}'_{ij}$ to Y from the set $\{\hat{m}'_{ij}\}_{i=1}^{C_j}$ is moved towards Y

$$(\hat{m}'_{ij})_{\text{new}} = (\hat{m}'_{ij})_{\text{old}} + h\left[Y - (\hat{m}'_{ij})_{\text{old}}\right] \tag{19}$$

(h, a positive real number between 0 and 1, is the learning rate parameter). In addition, the component proportion $\hat{p}_{ij}$ of that component is increased

$$(u_{ij})_{\text{new}} = (u_{ij})_{\text{old}} + \Delta u \tag{20}$$

(Δu is a small positive number, say 0.1). However, if $j' \neq j$, the network has made a mistake. In this condition, the closest $\hat{m}'_{ij'}$ to Y from the set $\{\hat{m}'_{ij'}\}_{i=1}^{C_{j'}}$ is moved away from Y as:

$$(\hat{m}'_{ij'})_{\text{new}} = (\hat{m}'_{ij'})_{\text{old}} - h\left[Y - (\hat{m}'_{ij'})_{\text{old}}\right], \tag{21}$$

and the closest $\hat{m}'_{ij}$ to Y from the set $\{\hat{m}'_{ij}\}_{i=1}^{C_j}$ is moved towards Y. In addition, $\hat{p}_{ij'}$ is decreased while $\hat{p}_{ij}$ increases

$$(u_{ij'})_{\text{new}} = (u_{ij'})_{\text{old}} - \Delta u \,. \tag{22}$$

The above training algorithm while the network has not correctly classified all the whitened training data vectors of $\Im'$ is repeated. The training steps are summarized in Fig. 3.

EXPERIMENTAL RESULTS

In order to show the ability of the proposed algorithm, we performed some computer simulations. The XOR problem, whose training data samples are illustrated in Fig. 4, is the first one. In the PNN classifier of this problem, we have considered two components for each class. Fig. 4a shows the initial and final positions of the prototype vectors. The decision boundary of the designed classifier is depicted in Fig. 4b.
The second example is a two-class problem, whose training data vectors are given in Fig. 5. As this figure shows, the two classes are twisted into each other and are not linearly separable. Single layer perceptron cannot separate this type of classes and multilayer perceptron can hardly learn to classify the data vectors. Nevertheless, a PNN that has two components for each class and is trained with the proposed algorithm quickly learns to classify all the data vectors. Initial and finial positions of the prototype vectors and the decision boundary are illustrated in parts (a) and (b) of Fig. 5.
The last example is related to the Optical Character Recognition (OCR) system. The task of this system is to recognize the character scanned by an optical scanner. We have designed an OCR system by the presented algorithm. For simplicity, the "F" and "E" characters are only considered. Some classification results are shown in Fig. 6. Part (a) of this figure shows the classification result when two noisy "F" characters are applied to the classifier. Part (b) shows the result when two noisy "E" characters are applied to the classifier. In each case we obtained the noisy characters from

1. Estimate the covariance matrix
$$\hat{\Sigma} = \frac{1}{C^2 \#(\Im)} \sum_{X \in \Im} (X - \hat{m})(X - \hat{m})^T$$
where
$$\hat{m} = \frac{1}{\#(\Im)} \sum_{X \in \Im} X ,$$
2. Find the matrix square root of $\hat{\Sigma}$ denoted by $\hat{L}$
$$\hat{\Sigma} = \hat{L}\hat{L}^T ,$$
3. Whiten all the vectors of training set
$$Y = \hat{L}^{-1} X ,$$
4. Randomly choose a whitened data vector, Y, from the jth class and apply it to the network. The network classifies it as j'th class.
If $j' = j$, then update $\hat{m}'_{ij}$ and u_{ij} as follows
$$\left(\hat{m}'_{ij}\right)_{\text{new}} = \left(\hat{m}'_{ij}\right)_{\text{old}} + h\left[Y - \left(\hat{m}'_{ij}\right)_{\text{old}}\right]$$
$$\left(u_{ij}\right)_{\text{new}} = \left(u_{ij}\right)_{\text{old}} + \Delta u$$
where $i = \arg\min \left\{ \left\| Y - \hat{m}'_{ij} \right\| \right\}_{i=1}^{C_j}$.
If $j' \neq j$, then update $\hat{m}'_{ij'}$ and $u_{ij'}$ as follows
$$\left(\hat{m}'_{ij'}\right)_{\text{new}} = \left(\hat{m}'_{ij'}\right)_{\text{old}} - h\left[Y - \left(\hat{m}'_{ij'}\right)_{\text{old}}\right]$$
$$\left(u_{ij'}\right)_{\text{new}} = \left(u_{ij'}\right)_{\text{old}} - \Delta u$$
where $i = \arg\min \left\{ \left\| Y - \hat{m}'_{ij'} \right\| \right\}_{i=1}^{C_{j'}}$. Also update $\hat{m}'_{ij}$ and u_{ij} as previously mentioned.
5. Terminate the training algorithm if all the training samples are correctly classified otherwise return to step 4.

Figure 3: Summary of the competitive learning algorithm applied to PNN.

the original character by adding a white Gaussian noise with zero mean and standard deviation 0.25.
The number of flops (per the number of training data vectors and per the vector dimension) and training time of the above examples are provided in Table 1.

CONCLUSION

In this paper we presented a new competitive learning type algorithm for PNNs. As mentioned in the paper, PNN is the realization of Baysian classifier with the Gaussian mixture model. The efficiency of classifiers realized by PNN and trained with the presented algorithm was examined with tree computer simulated examples. The results of the computer simulations proved the efficiency of this algorithm for solving complex classification problems.
A noteworthy point is the tradeoff between the classifier generalization and its susceptibility to noise: increasing the number of components for each class, the performance of the classifier increases. In the other hand, classifier becomes too susceptible to noise. Therefore, care must be taken in choosing the number of components for each class.

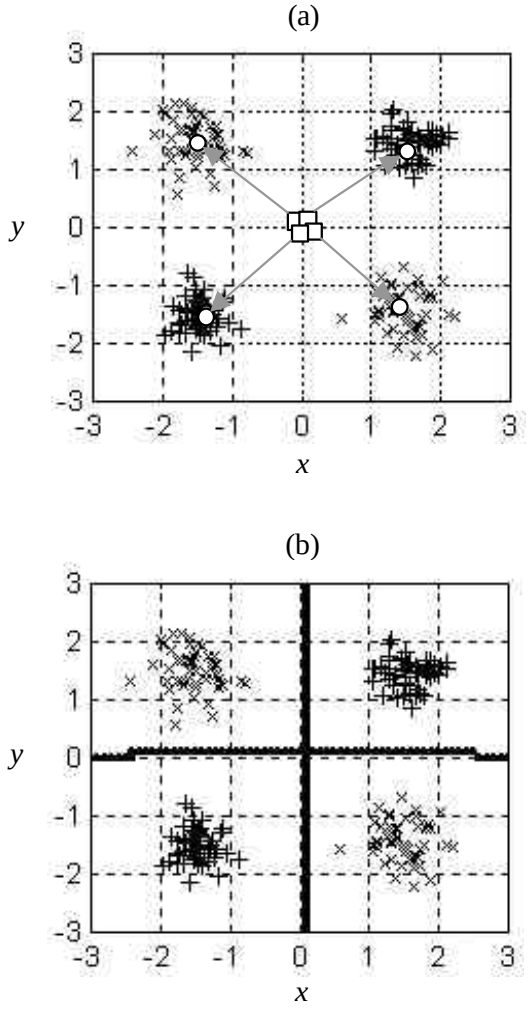

Figure 4: Data representation of the XOR example (+: first class, ×: second class). (a) Initial (□) and final (–) position of the prototype vectors, (b) Decision boundary.

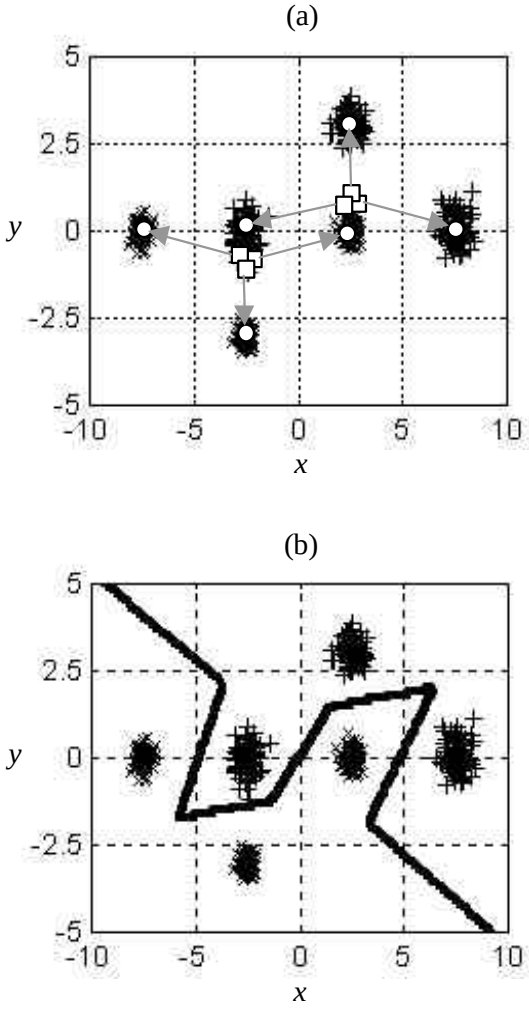

Figure 5: Data representation of the second example (+: first class, ×: second class). (a) Initial (□) and final (–) position of the prototype vectors, (b) Decision boundary.

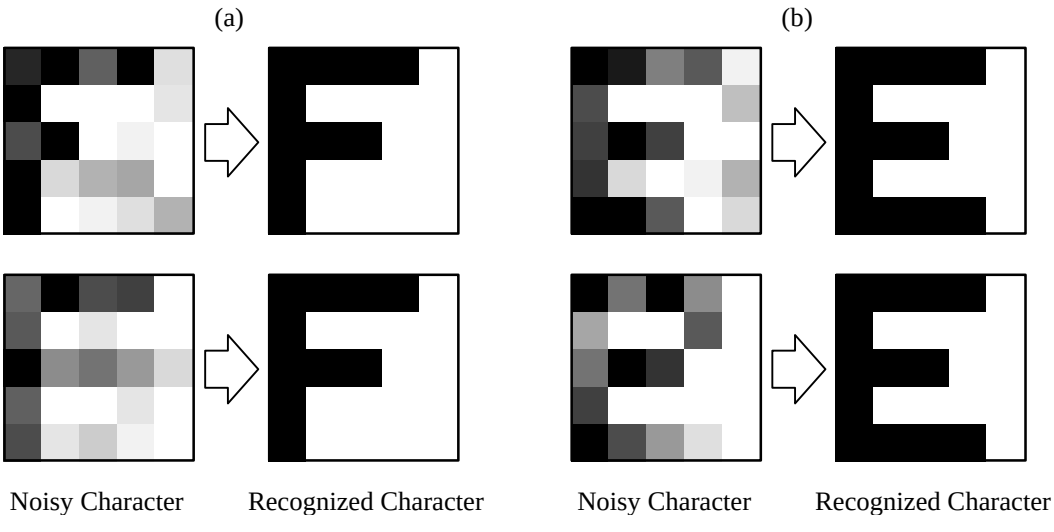

Figure 6: Results of an OCR system designed by PNN and trained by the proposed algorithm when noisy (a) “F”, (b) “E” characters are applied to the system.

	XOR	Example #2	OCR
Training Time (sec)	2.47	7.47	0.33
flops / ($C \times d$)	216.14	306.16	3433.4

Table 1: Training times and flops of the mentioned examples.

REFERENCES

Parzen, Emanuel, 1962, “On Estimation of Probability Density Function and Mode”, Ann. Math. Stat., vol. 33, pp. 1065 – 1076.

Specht, Donald F., 1988, “Probabilistic Neural Networks for Classification, Mapping, or Associative Memory”, Proceeding of the IEEE International Conference on Neural Networks, vol. 1, San Diego/CA, USA, pp. 525 – 532.

Specht, Donald F., 1990, “Probabilistic Neural Networks and Polynomial Adaline as Complementary Techniques for Classification”, IEEE Transactions on Neural Networks, vol. 1, No. 1, pp. 111 – 121.

Streit , Roy L.; Luginbuhl, Tod E., 1994, “Maximum Likelihood Training of Probabilistic Neural Networks”, IEEE Transactions on Neural Networks, vol. 5, No. 5, pp. 764 – 783.

Traven, Hans G. C., 1991, “A Neural Network Approach to Statistical Pattern Classification by Semiparametric Estimation of Probability Density Functions”, 1991, IEEE Transactions on Neural Networks, vol. 2, No. 3, pp. 366 – 377.