Probabilistic Neural Network: Comparison of the Cross-validation Approach and a Fast Heuristic to Choose the Smoothing Parameters

M. Zhong (*), J. Hecker (*), I. Maidhof (*), P. Shibly (*), M. Georgiopoulos (*),
G. Anagnostopoulos (**), M. Mollaghasemi (*), and S. Richie (*)

(*) University of Central Florida, Orlando, FL, USA
(**) Florida Institute of Technology, Melbourne, FL, USA

Abstract

It is well known that the Bayesian Classifier is the optimal classifier provided that p(X|c), the probability density function (PDF) with input attribute X given class c is known for any possible X and c. In practice, however, these PDFs are seldom given. Therefore, the Bayesian Classifier usually serves merely as a theoretical reference during the evaluation of other practical approaches. On the other hand, the Probabilistic Neural Network (PNN) attempts to estimate the PDFs with the instances in the training database, as well as a preset "smoothing parameter", which might be uniform or variable among each attribute and/or each class. Although it is proved that the estimate will approach the real PDF asymptotically if the training database contains sufficient number of instances, the choice of the smoothing parameter still affects the estimate when the number of training instances is finite. Although the cross validation could help identify a good setting for the smoothing parameters, this approach is extremely expensive due to the time complexity of the PNN in its performance phase. In this paper, we discuss the relationship between this parameter and some of the data statistics, and attempt to develop a simple approach to determine this parameter without relying on cross validation tests that are time consuming. We explain why these parameters can vary in a certain range, while maintaining the network's accuracy, as most experiments have shown, and even provide the quantitative expression of this range. Finally, we show that our approach is superior to the cross validation approach by conducting a number of experiments.

1. Introduction

The Bayesian classifier is the best classifier that one can build in terms of minimizing the misclassification error on unseen data. This property though of the Bayesian classifier holds true under the assumption that the conditional probability distributions, referred to as class conditional probabilities, from which the data are drawn are known. In practice, these probabilities are not known but are estimated from the data given, by using legitimate approaches for estimating them. One such approach is the Parzen window approach (Parzen, 1962), extended by Cacoullos (1966), which estimates the class conditional probabilities as sum of Gaussian functions centered at the training points and having appropriately chosen widths (variances), designated from now on as smoothing parameters. Parzen's approach works well under some reasonable assumptions regarding the choice of the smoothing parameters and when the training data becomes very large the approximation becomes equal to the actual class conditional probabilities. In the case though where the training data sets are of finite size (as it is usually the case in practice) the right choice of the smoothing parameters is essential for the good performance of the classifier. PNN (Specht, 1990), invented by Specht, is actually Bayes classifier where the class conditional probabilities are approximated by using the Parzen's approach.

One of the major issues associated with the PNN classifier is how to choose the smoothing parameters involved in the Gaussian functions utilized to estimate the class conditional probabilities. Specht, suggested using cross validation to estimate the smoothing parameters (see Specht, 1992) This is a reasonable approach if we assume that there is only one smoothing parameter that we focus on optimizing. But if we want to use a different smoothing parameter per dimension of the input data and per class to which the input data belong, we end up having to optimize a large number of parameters and the problem quickly becomes exponentially complex. Another way of dealing with this issue of smoothing parameters is to cluster the training data and approximate the class conditional probabilities by Gaussian functions centered at the cluster points instead of the actual training points. The clustering of the data gives the additional capability of estimating the smoothing parameters of these Gaussian functions as the within-cluster standard deviations. Clustering procedures that have been used in the literature in relation to the PNN neural network are: Burrascano, 1991 (used LVQ approach), Traven, 1991 (used K-Means clustering), Tseng, 1991 (used mixture of Gaussians).

All of the above reported approaches for estimating the smoothing parameters are incurring a computational expense for estimating the parameters. For instance, a single run of the cross-validation has a complexity of $O(PT^T PT^V)$, where PT^T is the number of points in the training set and PT^V is the number of points in the validation set. Another problem with the cross-validation approach is that we have to do many such runs over a variety of smoothing parameter values to reliably estimate their optimal values.

Our approach, attempts to determine a good estimate of the optimal smoothing parameters with a time complexity of $O(PT^T)$. No cross-validation is required. This is a significant computational advantage and it also an advantage in practical situations, where the dataset given is small, and we do not have the luxury of defining a sizable cross-validation set.

2. PNN Preliminaries

The Bayes classifier is based on the following formula for finding the class that a datum $\mathbf{x}$ belongs.

$$P(c_j \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid c_j) P(c_j)}{p(\mathbf{x})}$$

In particular, we calculate the above probabilities for every class *j* of our pattern classification task, and we assign datum $\mathbf{x}$ to the class *j* that maximizes this probability. In order to calculate the above probabilities one needs to estimate the class conditional probabilities $p(\mathbf{x} \mid c_j)$ and the a-priori probabilities $P(c_j)$ for every class *j* (the calculation of $p(\mathbf{x})$ is not needed because it is a common factor in all of these probabilities and can be cancelled out). The a-priori probabilities $P(c_j)$ are calculated from the given training data. The class conditional probabilities $p(\mathbf{x} \mid c_j)$ can also be calculated from the training data by using the approximation suggested by Parzen (see Parzen, 1962). In particular, in Specht's PNN paper the approximation suggested by Parzen is utilized to estimate these class conditional probabilities, as follows:

$$p(\mathbf{x} \mid c_j) = \frac{1}{(2\pi)^{D/2} \sigma^D PT_j} \sum_{r=1}^{PT_j} \exp\left[-\frac{(\mathbf{x} - \mathbf{X}_r^j)^T (\mathbf{x} - \mathbf{X}_r^j)}{2\sigma^2} \right] \quad (1)$$

where D is the dimensionality of the input patterns, PT_j represents the number of training patterns belonging to class j, $\mathbf{X}_r^j$ denotes the r-th such training pattern, $\mathbf{x}$ is the input pattern to be classified, and σ is the smoothing parameter that we talked about earlier. It is pointed out in Specht, 1990, that this estimation will approach asymptotically the real PDF if:

$$\lim_{PT_j \to \infty} \sigma = 0 \quad (2)$$

and:

$$\lim_{PT_j \to \infty} PT_j \sigma = \infty \quad (3)$$

Equation (1) is the basis of the PNN classifier. The smoothing parameter σ should be selected properly. If σ is too small, the estimated PDF will be so non-linear that the PDF at a testing point will be almost zero if this point is not close enough to any one of the training points, thus reducing the network's capacity to generalize. If, on the other hand, σ is too large, then over a wide range of input values the estimated PDF will be almost constant (proportional to the number of training patterns belonging to the class), and in that case the actual values of the training and test patterns does not seem to play any role in the determination of which class the test input pattern belongs.

How to select this parameter, however, is still an issue of difficulty, especially when we use smoothing parameters that depend on the dimension and the class of the data. In that case, the above formula for the estimation of the class conditional probabilities becomes.

$$P(\mathbf{x} \mid c_j) = \frac{1}{(2\pi)^{D/2} \prod_{i=1}^{D} \sigma_{ij} PT_j} \sum_{r=1}^{PT_j} \exp\left[-\sum_{i=1}^{D} \frac{\left(x_i - X_{ir}^j\right)^2}{2\sigma_{ij}^{\ 2}} \right]$$

where σ_{ij} is the smoothing parameter across dimension *i* for the training points belonging to class *j*.

In the following we present four different approaches of selecting the smoothing parameters (one of them is the expensive cross validation approach, and another one is our inexpensive approach).

3. Candidate Solutions for Choosing the Smoothing Parameters

In this paper, we compare the following four approaches to find the smoothing parameters in the PNN:

a) Simply try the sigma parameters as the in-class standard deviation for each dimension and each class (that is, $\sigma_{ij} = STD\left(X_i^{\ j}\right)$, where *i* is the dimension index and *j* is the class index). As analyzed in Appendix I, this approach is risky, but we still run experiments following this method.

b) Evaluate $\sigma_{ij} = v_j STD\left(X_i^{\ j}\right)$ where v_j can be chosen from {0.5, 0.25} for each class. Although there are only two candidates, the number of combinations is as high as 2^J where J is the number of classes. Choose the best accuracy on the test set among all the 2^J combinations of the v_j's.

c) (Our approach) solve for the sigma parameters σ_{ij} as described in the section 4.

d) Based on the results given in option c), evaluate kσ_{ij} for k = 0.5, 1, 2 and pick the best one. This approach is different from option b), each time we multiply k to all σ_{ij}'s, and thus there are only 3 trials for each database.

4. Algorithm Description (Approach c)

As we have mentioned above, the formula used for the estimation of the class conditional probabilities is given below.

$$P(\mathbf{x} \mid c_j) = \frac{1}{(2\pi)^{D/2} \prod_{i=1}^{D} \sigma_{ij} PT_j} \sum_{r=1}^{PT_j} \exp\left[-\sum_{i=1}^{D} \frac{\left(x_i - X_{ir}^{j}\right)^2}{2\sigma_{ij}^{\ 2}} \right] \tag{4}$$

According to Appendix I, we should not set σ_{ij} to $STD\left(X_i^{\ j}\right)$, the unbiased standard deviation of X_{ir}^{j}. Nevertheless, we can set σ_{ij} proportional to $STD\left(X_i^{\ j}\right)$, or equivalently, we first standardize the patterns in each class (not among the whole database) and find out the optimal $\sigma_{ij} / STD\left(X_i^{\ j}\right)$ ratio. The benefit of this approach is that our result will be invariant of the scale of any dimension. Our analysis and experiments show that we should set $\sigma_{ij} = \min\left(4d_j, 0.5\right) STD\left(X_i^{\ j}\right)$, where d_j is the average value of the minimum distance between two input points belonging to class j. Computing d_j directly is too time-consuming. Therefore, we estimate d_j by sampling the points. We can sample $2N_j$ points and calculate N_j distances: the distance between the first sample and the second sample, the distance between the third and the fourth, etc. Let $\hat{d}_j$ denote the expected value of the minimum distance of the above N_j distances. When the PDF is fixed and N_j is sufficiently large, $\hat{d}_j$ depends on N_j as well as D_j, the actual dimensionality of the points in class j, according to the following equation (for more information about this equation, please refer to appendix III).

$$\hat{d}_j \approx d_j \left(\frac{PT_j}{N_j} \right)^{\frac{1}{D_j}}$$

Here D_j means the number of free dimensions, instead of the number of linearly independent vectors. For example, when the input points are distributed in a unity circle in a 2-D space, their attributes x and y are linearly independent; however, there is only one free dimension – the angle – among the points. Since the linear measures (such as the rank, the eigenvalues, etc) are unable to derive D_j directly, we repeat sampling with various N_j values and solve for d_j and D_j with the corresponding observed values $\hat{d}_j$. The pseudo code is included in Figure 1.

Notations and parameters:

PT_j : Number of input points in class j

D : Number of dimensions of the input points

$K_{\max}$: Maximum Repeat Times (natural number). Typical $K_{\max} = 4$

F_M, F_N : Sample Size Factor (small positive number). Typical $F_M = 8, F_N = 4$

Main Procedure:

For each class j

Compute $STD\left(X_i^{\,j}\right)$ for $i = 1,2,...,D$

$M_j = \min\left(PT_j, \lfloor F_M \sqrt{PT_j} \rfloor\right)$

$N_j = \min\left(PT_j, \lfloor F_N \sqrt{PT_j} \rfloor\right)$

$K_j = \min\left(K_{\max}, \left\lfloor \log_2 \frac{PT_j}{N_j} \right\rfloor\right)$

If $K_j < 1$ (which means PT_j is small) then

$d_j = \text{Mean-Min-Distance}(j, M_j, PT_j)$

Else

For $k = 0,1,2,...,K_j$

$\hat{d}_{jk} = \text{Mean-Min-Distance}(j, M_j, 2^k N_j)$

End For

Solve the equations $\frac{1}{D_j}\log\left(\frac{PT_j}{N_{jk}}\right) + \log d_j = \log \hat{d}_{jk}$, $k = 1,2,...,K_j$, with least square error, treating $\frac{1}{D_j}$ and $\log d_j$ as unknowns.

End If

$\sigma_{ij} = \min\left(d_j, 0.5\right) STD\left(X_i^{\,j}\right)$ for $i = 1,2,...,D$

End For

Subroutine Mean-Min-Distance (j, M, N)

For $m = 1,2,...,M$

Randomly pick a point $\mathbf{X}$ from class j.

For $n = 1,2,...,N$

Randomly pick a point $\mathbf{X}_n$ from class j, different from $\mathbf{X}$.

$$d_{mn}^{\,2} = \sum_{i=1}^{D}\left(\frac{X_{in} - X_i}{std_{ij}}\right)^2$$

End For

Find the smallest $2D$ elements from $d_{m1}^{\,2}, d_{m2}^{\,2}, ...d_{mN}^{\,2}$

Find the largest element (denoted by $d_m^{\,2}$) in the above $2D$ elements

End For

Return $\sqrt{mean_m\left[d_m^{\,2}\right]}$

Note: the smallest $2D$ elements for each m can be cached so that when we double N next time, only N more patterns have to be picked, which halves the computational complexity.

Figure 1: Algorithm of our solution to the smoothing parameters

The least-square-error solution to the linear equations can be explicitly given as:

$$\begin{bmatrix} 1/D_j \\ \log d_j \end{bmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{B} \qquad \mathbf{A} = \begin{bmatrix} \log\left(PT_j/N_{j1}\right) & 1 \\ \vdots & \vdots \\ \log\left(PT_j/N_{jK_j}\right) & 1 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} \log\hat{d}_{j1} \\ \vdots \\ \log\hat{d}_{jK_j} \end{bmatrix}$$

In practice, computing $\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{B}$ directly is not efficient in both time and space. A recursive algorithm can be applied, which is shown in Figure 2.

$$\mathbf{P}_0 = \begin{bmatrix} \log\left(PT_j/N_{j1}\right) & 1 \\ \log\left(PT_j/N_{j2}\right) & 1 \end{bmatrix}, \qquad \mathbf{P} = \left(\mathbf{P}_0{}^T\mathbf{P}_0\right)^{-1},\ \mathbf{Q} = \mathbf{A}_0{}^T\begin{bmatrix} \log\hat{d}_{j1} \\ \log\hat{d}_{j2} \end{bmatrix}$$

For $k = 3,...,K_j$

$$\mathbf{a} = \begin{bmatrix} \log\left(PT_j/N_{jk}\right) & 1 \end{bmatrix}$$

$$\mathbf{P} = \mathbf{P} - \left(\mathbf{P}\mathbf{a}^T\right)\left(\mathbf{a}\mathbf{P}\mathbf{a}^T + 1\right)^{-1}\left(\mathbf{a}\mathbf{P}\right)$$

$$\mathbf{Q} = \mathbf{Q} + \mathbf{a}^T \log\hat{d}_{jk}$$

End For

$$\begin{bmatrix} 1/D_j \\ \log d_j \end{bmatrix} = \mathbf{P}\mathbf{Q}$$

Note: the update of $\mathbf{P}$ is very simple because $\left(\mathbf{P}\mathbf{a}^T\right)$ is only 2-by-1, $\left(\mathbf{a}\mathbf{P}\mathbf{a}^T + 1\right)$ is a scalar, and $\left(\mathbf{a}\mathbf{P}\right)$ is 1-by-2.

Figure 2: Recursive LSE Solution Algorithm

5. Complexity Analysis of The Algorithmic Approach c

In the worst case, assume PT_j is large enough, or $F_M < \sqrt{PT_j}, F_N < \sqrt{PT_j}$, so that $M_j = O\left(F_M\sqrt{PT_j}\right), N_j = O\left(F_N\sqrt{PT_j}\right)$. Also assume $K_j = K_{\max} \geq 1$. The subroutine Mean-Min-Distance$\left(j, M, N\right)$ has a complexity of $O\left(DMN\right)$. This subroutine is called for $K_j + 1$ times, with $M = M_j$ and $N = N_j, 2N_j, ..., 2^{K_j}N_j$. The overall complexity is $O\left(DM_j 2^{K_{\max}} N_j\right) = O\left(DF_M F_N PT_j 2^{K_{\max}}\right)$. Solving the linear equation has a constant time complexity. Therefore, the whole algorithm has a computational complexity of $O\left(D \cdot PT \cdot 2^{K_{\max}} \cdot F_M \cdot F_N\right)$, where PT is the total number of points. Since $F_M, F_N, K_{\max}$ are small constants, this complexity is negligible in comparison with the complexity of the classification process of PNN.

6. Experiments

In our experiments, we compared the four approaches, mentioned in Section 3, of finding the smoothing parameters in the PNN. In particular, the four approaches are repeated below.

a) Set the sigma parameters as the in-class standard deviation for each dimension and each class. The time for finding the sigma parameters is ignored.
b) Evaluate $\sigma_{ij} = v_j STD\left(X_i{}^j\right)$ where v_j can be chosen from {0.5, 0.25} for each class. The time for finding the sigma parameters is defined as the total time for all the 2^J runs of the PNN algorithm.

c) Apply the sigma parameters σ_{ij} given from our approach directly. The time for finding the sigma parameters is defined as the time for running our approach (see also Section 5 for the calculation of the complexity of this approach).

d) Evaluate $k\sigma_{ij}$ for k = 0.5, 1, 2 and pick the best one. The time for finding the sigma parameters is defined as the total time for the 3 runs of the PNN algorithm.

6.1 Databases

The databases used in our experiments are listed below:

Database Name	# Training Patterns	# Test Patterns	# Numerical Attributes	# Classes	% Major Class
Grass and Trees	2000	4000	1	2	1/2
Modified Iris	500	4800	2	2	1/2
Page Blocks	500	2486	10	4(5)	0.832(0.8826)
Abalone	501	1838	7	3	1/3
Satellite	4435	2000	36	6	0.2417

Note: The numbers between parentheses reflects the test set, if different from the training set

Figure 3: Statistics of Databases

To demonstrate that the standard deviation is not directly related to the optimal sigma value, we created an artificial database "Grass and Trees" that contains only one attribute. The first class is uniformly distributed in [0, 1]. The second class has five clusters, centered at 0, 0.25, 0.5, 0.75, and 1, respectively. Each cluster has also uniform distribution with range 0.05. Both classes occupy 50% of the instances. It can be shown that the Bayes Classifier would have 90% accuracy on this database. We used 1000 points for training and another 2000 points for testing. The following figures show that it is difficult to guess the optimal sigma value using the standard deviation, while our approach produces very accurate estimates.

Figure 4: Estimates of the PDFs for Database "Grass and Trees"

The rest of the databases in the above table are the benchmark databases provided by UCI Repository (see Hettich et al, 1998). We selected the ones with sufficient size (but not too large since the enumeration would take a long time) in order to make our results statistically significant. The "Iris" database represents the classification problem of the iris plants, which can be easily solved using even linear classifiers. We introduced noise to generate enough data points, and removed the two attributes with least correlation to the class label.

"Page Blocks" database consists of the attributes of page layouts in a document with various block types. In the raw database, approximately 90% of the instances belong to the "text" block type. To increase the difficulty, we reduced the percentage of the major class and removed one class in the training set. The test set preserves all the 5 classes.

"Abalone" database is used for predicting the age of abalones. In our case, we removed the categorical attribute and grouped the targets into three classes: 8 and lower, 9-10, 11 and greater, as other researchers have

done in the past. The resulting classes, however, are still highly overlapped in the attribute space and current Machine Learning algorithms can attain only 50%~60% accuracy.

"Satellite" database is the largest tested. It provides the multi-spectral values extracted from satellite images corresponding to 6 types of soil (previously 7 types, one of which used to be "mixed soil" and was removed due to the doubt about its validity).

6.2 Experimental Results

Database Name	Time for a Single Run	Option a $\sigma = STD$	Option b Enumeration		Option c Solving σ_{ij}		Option d Variation of σ_{ij}	
		%Acc	Time	%Acc	Time	%Acc	Time	%Acc
Grass and Trees	1.422	66.23	5.738	66.13	0.06	89.62	4.947	89.62
Modified Iris	0.204	94.06	1.811	94.65	0.047	94.92	0.64	94.92
Page Blocks	0.203	85.89	16.206	92.00	0.016	86.85	0.625	89.22
Abalone	0.125	52.18	3.549	52.18	0.015	51.58	0.406	51.58
Satellite	3.734	82.95	2943.7	90.50	0.344	90.50	11.563	90.50

Time is measured in seconds
The highlighted columns correspond to our approach

Figure 5: Time and accuracy of four approaches.

The above table shows the performance of the four approaches. The first column is the time to run PNN only once, and it is simply used as a reference.

For option a) which simply sets the σ_{ij} as the standard deviation, the elapsed time for computing σ_{ij} can be ignored. However, this option is sometimes risky; its accuracy is close to that of wild guessing (see example above, referred to as "Grass and Trees").

Our solution is satisfactory (approaches c and d) in most of the cases. The time elapsed for solving σ_{ij} is much smaller than a single run of the PNN algorithm. Note that the accuracy on the first database is very close to the theoretical optimum, which means our approach is robust against clustering and noise. The accuracy on "Page Blocks" is not as good as the best one, but we believe that the main reason is the statistical discrepancy between the training set and the test set. Both the Bayes Classifier and the PNN rely on the priori probability of each class. PNN estimates it using the training set if it is not given; if the training set does not reflect the priori probabilities, the PNN cannot approximate the Bayes Classifier even if the smoothing parameter are appropriately optimized.

Enumeration of different scales of the standard deviation is comparable to our approach in accuracy except for the first database, but at a high computational cost. For the satellite database, it required almost 10,000 times longer than our approach, while achieving the same accuracy. Of course, one can argue that we evaluate only two σ_{ij} candidates, one for each class. Remember, the time complexity grows exponentially with the number of candidates per class. Even for a simple database such as "Grass And Trees", it is difficult to locate the approximate optimal range for σ_{ij}, which does not depend on the standard deviation only. Therefore, it is not practical to apply enumeration to optimize σ_{ij}.

As a compromise, it is reasonable to vary the smoothing parameter returned from our approach by a fixed number of scales (approach d). As shown in the above table, this option is more computationally complex than approach c, but reasonably so, and furthermore it improves the accuracy.

7. Summary

Setting the proper smoothing parameter in the PNN is important for achieving high accuracy. Searching for the optimal parameter by cross validation is usually too expensive, especially when the parameter depends on both the class label and the attribute (see approach b, discussed above). In this paper, we modeled the smoothing parameter as a diagonal matrix and proposed an effective approach to determine it. Our approach (approach c) is very computationally efficient (compared to the cross validation approach) and achieved high accuracy Experiments show that our approach is satisfactory in most cases, regardless of the number of instances/classes/dimensions. Even in case that the accuracy of our approach is not high enough, it can be improved by varying our estimated smoothing parameter (approach d).

Acknowledgment
This work was supported in part by a National Science Foundation (NSF) grant CRCD: 0203446. Georgios Anagnostopoulos and Michael Georgiopoulos acknowledge the partial support from the NSF grant CCLI 0341601.

References

Burrascano, P., "Learning vector quantization for the Probabilistic Neural Network," *IEEE Transactions on Neural Networks*, Vol. 2, pp. 458-461, July 1991

Cacoullos, T., "Estimation of a multi-variate density," *Annals of the Institute of mathematical Statistics (Tokyo)*, Vol. 18, No. 2, pp. 179-189, 1966.

Maloney P.S. et al., "Successful Applications of Expanded Probabilistic Neural Networks," *Intelligent Engineering Systems Through Artificial Neural Networks*, Eds. C. H. Dagli et al., ASME PRESS, New York, 1991

Hunter, A., "Feature Selection Using Probabilistic Neural Networks", *Neural Computing & Applications*, Vol. 9, Issue 2, pp 124-132, 2000.

Parzen, E., "On estimation of probability density function and mode," *Annals of Mathematical Statistics*, Vol. 33, pp. 1065-1073, 1962.

Specht, D.F., "Probabilistic Neural Networks and the Polynomial Adaline as Complementary Techniques for Classification", *IEEE Transactions on Neural Networks*, Vol.1, No.1, pp.111-121, March 1990

Specht, D.F., "Enhancements to Probabilistic Neural Networks", *Proceedings International Joint Conference on Neural Networks (IJCNN '92)*, 1, pp. 761-768

Specht, D.F., "Experience with Adaptive Probabilistic Neural Networks and Adaptive General Regression Neural Networks", *Proceedings of the IEEE World Congress on Computational Intelligence*, Vol. 2, pp. 1203-1208, 1994

Traven, H. G. C., "A neural network approach to statistical pattern classification by 'semi-parametric' estimation of probability density functions," *IEEE Transactions on Neural Networks*, Vol. 2, pp. 366-377, May 1991.

Tseng, M-L., "Integrating Neural Networks with Influence Diagrams for Multiple Sensor Diagnostic Systems," *Ph.D. Dissertation, University of California at Berkley*, August 1991

Washburne, T.P. et al., "Identification of Unknown Categories with Probabilistic Neural Networks", *IEEE International Conference on Neural Networks*, vol.1, pp. 434 -437, 1993

Hettich, S. & Blake, C.L. & Merz, C.J. UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

Appendix I – Relationship between σ and STD(X)

Consider the 1-dimensional case. The PDF is estimated as

$$p(x \mid c_j) = \frac{1}{\sqrt{2\pi}\sigma_j PT_j} \sum_{r=1}^{PT_j} \exp\left[-\frac{\left(x - X_r^j\right)^2}{2\sigma_j^{\,2}} \right]$$

Let $f_{\hat{X}}(x) = p(x \mid c_j)$ stand for the estimated PDF and $f_X(x)$ represent the real unknown PDF of the attribute X^j of class j. When PT_j is sufficiently large,

$$f_{\hat{X}}(x) = \frac{1}{\sqrt{2\pi}\sigma_j} E\left\{ \exp\left[-\frac{\left(x - X^j\right)^2}{2\sigma_j^{\,2}} \right] \right\}$$

$$= \frac{1}{\sqrt{2\pi}\sigma_j} \int_{-\infty}^{\infty} \exp\left[-\frac{(x-y)^2}{2\sigma_j^{\,2}} \right] f_X(y) dy$$

where $E[X]$ denotes the expected value of the random variable X. It can be easily proved that

$$E\left[\hat{X}\right] = E\left[X^j\right], \qquad VAR\left[\hat{X}\right] = VAR\left[X^j\right] + \sigma_j^{\,2}$$

Therefore, the estimated PDF is not the real PDF since the variance is different, unless σ_j is zero, which always over fits the network to the training set. In fact, if we seek for another kernel function $F(x, X)$ such that $f_X(x) = EX\{F(x, X)\}$, the only solution is

$$F(x, X) = \delta(x - X) = \lim_{\sigma \to 0} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-X)^2}{2\sigma^2} \right]$$

This observation leads to the constraint (3). It also suggests that $\sigma_j^{\,2} << VAR\left[X^j\right]$, or $\sigma_j << STD\left[X^j\right]$. The above result is also applicable to the multi-dimensional case.

Appendix II – Relationship between σ and d

Now we assume that the attributes have been standardized within each class, that is, $STD\left[X^j\right]$ is always one. An important requirement for the σ value of a single class can be derived from a simple model. Assume that the patterns in this class are uniformly distributed within a *D*-dimensional hyper cube whose range in every dimension is $(-L, L)$, and the training data set contains $PT = (2m+1)^D$ (the subscript j is omitted since only this class is discussed) exemplar points regularly located as an array in the hyper-box, as shown in Figure 6.

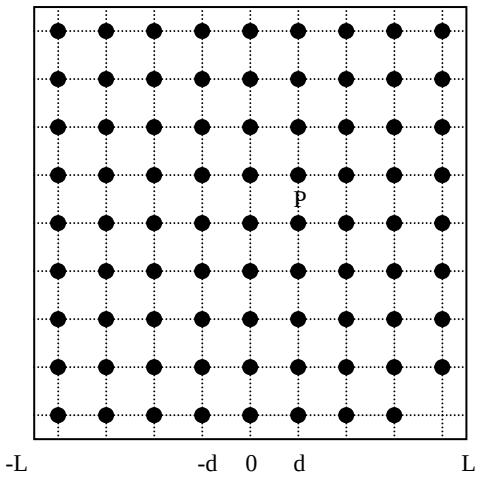

Figure 6: A simple model

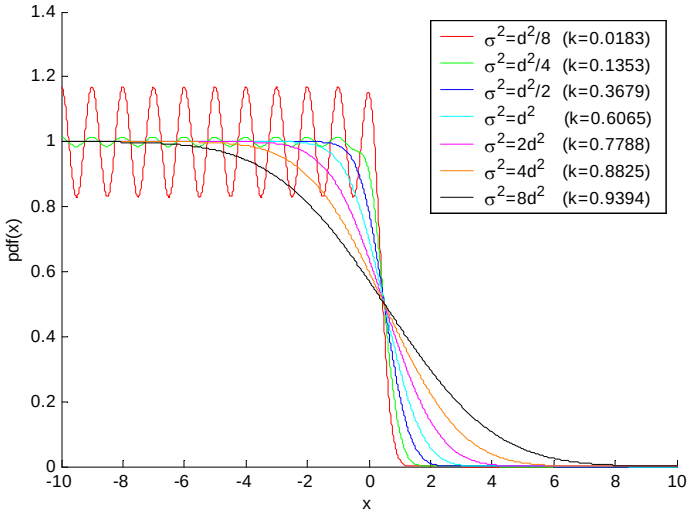

Figure 7: Estimated PDF with various σ^2

The figure reflects the case where $m = 4, D = 2$ for simplicity. Actually *m* and *n* can be much larger. One of the points is $(0,0,0,\cdots,0)$ and one of its neighbors is $(d,0,0,\cdots,0)$, where $d = 2L/(2m+1)$. This model may seem too ideal, but it will be shown that the results can be applied to more general cases.

Consider the center P in a "cell" box which has the following 2^D corners $(0,0,0,\cdots,0)$, $(d,0,0,\cdots,0)$, $(0,d,0,\cdots,0)$, … $(d,d,\cdots,d)$. Obviously $P = (d/2, d/2, \cdots, d/2)$, with equal distance to each corner: $d_P{}^2 = Dd^2/4$. Since the points are uniformly distributed, it is expected that the estimated PDF at P is the same as that at the origin:

$$\frac{1}{\sigma^D}\sum_{i_1=-m}^{m}\sum_{i_2=-m}^{m}\cdots\sum_{i_n=-m}^{m}\exp\left[-\frac{1}{2\sigma^2}\sum_{l=1}^{D}(i_l d-\frac{d}{2})^2\right]=\frac{1}{\sigma^D}\sum_{i_1=-m}^{m}\sum_{i_2=-m}^{m}\cdots\sum_{i_n=-m}^{m}\exp\left[-\frac{1}{2\sigma^2}\sum_{l=1}^{D}(i_l d)^2\right]$$

After some calculations, the above equality leads to the following equation:

$$\sum_{i=-\infty}^{\infty}k^{(i-0.5)^2}=\sum_{i=-\infty}^{\infty}k^{i^2}\qquad\text{where } k=\exp\left(-\frac{d^2}{2\sigma^2}\right)$$

Numerical solution of this equation yields $\sigma^2 \geq 0.476d^2$

Now consider a similar model in which the training patterns are distributed in a hyper cube with half volume as the previous one, but with the same density. That is, all the patterns with positive x_1 (coordinate in the first dimension) are removed. Then, consider the point $Q = (q \times d, 0, 0, \cdots 0)$ where *q* is positive. The estimated PDF at *Q* should be small enough because *Q* is outside the region where the training patterns are distributed. With derivations similar to the above ones, the following condition should be met:

$$0.476d^2 \leq \sigma^2 \leq 4.58d^2$$

Thus, the smoothing parameter is allowed to vary in a certain way. To show this more clearly, we plot the PDF with various σ^2 in Figure 7, assuming the training instances are {–50, -49, …, 0} (which means $d^2 = 1$).

Although $\sigma = 0.5d$ seems to approximate the PDF very well, we should set σ^2 higher. The reason is that we do not expect the PDF to drop sharply at the boundary of the training points, since the training set is only part of the data set. Therefore, we prefer to choose $\sigma = 2d$.

The above analysis is based on the regular model shown in Figure 6. In practice, however, the data points are seldom regularly distributed and thus the minimum distance of a uniformly distributed data set would be much smaller than d defined in this model, even when the number of points is the same. We can then do the following: For each point $\mathbf{X}$, find the $2D$ nearest points of $\mathbf{X}$, record the distance between $\mathbf{X}$ and its farthest neighbor among the $2D$ nearest ones, and estimate d as the average such distance among all $\mathbf{X}$'s. After estimating d, we should set $\sigma = 4d$ because: a) some of the points are relatively far from its nearest neighbor, and b) the standard deviation of the estimated distance is usually comparable to the distance itself.

Another issue is that setting $\sigma_{ij} = 4d_j STD\left(X_i{}^j\right)$ for each dimension might not be optimal. The reason is that by assuming diagonal $\mathbf{\Sigma}_j$, we imply that all the attributes are independent, which is not true in many cases. Consider an example where all the attributes are equal within any instance, and they are uniformly distributed with unity variance across the instances in the same class. Apparently, $d_j{}^2$ is proportional to the number of attributes. When the number of attributes is sufficiently large, d_j can exceed one. In this case, although the marginal PDFs are estimated very well, the PDF in the whole space is not accurate. Considering the constraint that $\sigma_{ij} << STD\left(X_i{}^j\right)$ given from Appendix I, we set $\sigma_{ij} = \min\left(4d_j, 0.5\right) STD\left(X_i{}^j\right)$, where the constant 0.5 was determined by experimentation.

Appendix III – Analysis on Sampling

For large data sets, it is not necessary to examine every pattern in order to compute d. Suppose N ($\sqrt{PT} \le N \le PT$) pairs of points are randomly picked and the expected value of the minimum distance between each pair is $\hat{d}$, then

$$\hat{d} \approx d\left(\frac{PT}{N}\right)^{\frac{1}{D}}$$

The first ideal model can help to explain this approximation. Suppose we have observed a value of $\hat{d}$, and now more points are then added to the model so that PT is 2^D times as large as before, doubling the density of the data set in every dimension and thus halving *d*. After this procedure is repeated for T times where T is small so that N/PT is not too small, PT becomes $2^{nT}PT$ and *d* turns $2^{-T}d$, while $\hat{d}$ and *N* remain as before, which verifies our approximation.

If σ is set proportional to d, it is not difficult to verify that (2) is true, and (3) is true when $n > 1$ ($n = 1$ seldom happens; (2) and (3) is a sufficient condition instead of a necessary one, as Figure 7 shows).

There is another reason why N should be large. If a class is separated into *C* clusters, then it is reasonable to require at least 4*C* patterns to be present in each cluster, which means $PT \ge 4C^2$. If $N = 4\sqrt{PT}$, then $N \ge 8C$. Assume that for each random selection all the candidate patterns have the same probability to be chosen, and that all clusters have the same number of patterns. When we choose a pattern $\mathbf{X}$ and N other patterns, the probability that none of the N patterns falls in the same cluster as $\mathbf{X}$ is:

$$P = (1 - \frac{1}{C})^{8C}$$

It can be proved that $P \le e^{-8} = 3.3546 \times 10^{-4}$ for any *C* >0, indicating that the result should be typical. A general designing rule can be derived: if $P \le e^{-a}$ is desired and at least b^2C patterns for each cluster can be assumed, where $a > 0, b > 0$, then $N = (a/b)\sqrt{PT}$ is sufficient.

Although *a* and *b* seem problem-dependent by definition (and thus new parameters are introduced to solve for the old one, σ), they are merely thresholds that can be predefined and do not need to be accurate. For example, if a/b is fixed to 8 when the PNN is designed, then a problem that requires $a = 16, b = 4$ can be solved with sufficient accuracy, because more patterns than the minimum will be chosen.