Turk. J. Math. Comput. Sci.
16(2)(2024) 507–517
© MatDer
DOI : 10.47000/tjmcs.1333685

Speed Optimizations to Parzen Window Classifier Using Probability Approximation, Discretization and Compression

Ibrahim Cem Baykal

Department of Computer Engineering, Faculty of Computer and Informatics, Adana Alparslan Türkeş Science and Technology University, 01250, Adana, Turkiye.

Received: 27-07-2023 • Accepted: 12-07-2024

Abstract. Parzen window estimators can model any type of complicated probability density manifolds. However, when it comes to real life applications, they are not as popular as the Artificial Neural Networks or the Support Vector Machines. That is mainly because Parzen window classifiers require long and complex calculations during the classification process. This article introduces speed optimization methods for Parzen window classifier that makes this classifier faster than any other convergent classifier at a small performance cost. The method includes, discretization, look-up tables, approximation, and probabilistic compression. Experiments conducted on both computer generated and real-life data prove that the resultant classifier is only slightly less accurate than Artificial Neural Networks and Support Vector Machines while immensely faster.

2020 AMS Classification: 68T10

Keywords: Probability density estimation, machine learning, speed optimization, Parzen window classifier.

1. Introduction

Pattern classifiers are increasingly becoming a part of our daily lives. Artificial intelligence and machine learning are being mentioned in every field from engineering to sociology. As their popularity increase, so do the amount of data they process. That brings a burden, especially on mobile devices such as robots and drones. With the increasing amount of data, the processing power requirements increase as well. Unfortunately, processing all this data requires energy, and that is very limited on battery operated devices such as drones and robots. Therefore, effort must be spend to make these classifiers computationally simpler.

There are many real life examples that Parzen Window (PW) classifier can compete and sometimes even beat Artificial Neural Networks (ANNs) or Support Vector Machines (SVMs). For example Shaikh and Alftieh [12] experimented on brain waves and found out that the Parzen Window classifier performs the best when it comes to interpreting the EEG signals of people with neuromuscular impairment. That is because the PW classifier can converge to predict probability density functions (PDFs) of multidimensional manifolds. Figure 1 shows a complicated PDF used as an example in [13].

In order to calculate the PDF at a given point, Parzen window classifier uses training samples at the vicinity of our target point. Therefore, the name “window” is used to define the size of that “vicinity.” Usually windowing and smoothing is obtained using a kernel function. There are many kernel functions in the literature [2]. However, it is not the goal of this article to discuss them. Although any kernel can be used with the proposed methods, we will use the

Email address: icbaykal@atu.edu.tr (I.C. Baykal)

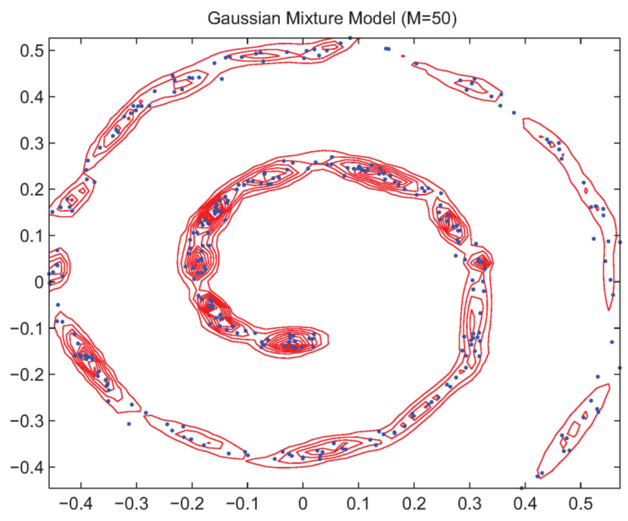


FIGURE 1. Parzen window estimation of a complex PDF using the training samples.

Gaussian kernel throughout this article. The PDF of a given point is calculated as a superposition of Gaussian kernels at every training point:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Where mu is the mean (or the center,) and sigma is the covariance. For an l-multidimensional process, the PDF of a given vector x is calculated as:

$$g(x) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2\pi^{l/2}h^l} exp\left(-\frac{(x-x_i)^T(x-x_i)}{2h^2}\right), \tag{1.1}$$

where N is the number of samples in the training set. If there are 100 samples in the training set, this calculation will take quite a long time. Unfortunately, this calculation must be repeated every time for every sample in the test set and for each class to find out which class's PDF is the highest. There are methods [13] to speed up this process but unfortunately none of them comes close to beating the speed of ANNs or SVMs.

In this article, a new method based on discretization, look-up tables and approximation will be introduced that considerably decreases the processing power needed by the Parzen window classifier. The method will be proven first using computer generated data and then using the age of abalone. Then, further optimization will be performed using probabilistic compression, which not only compresses the look-up tables but also decreases the processing power demand further.

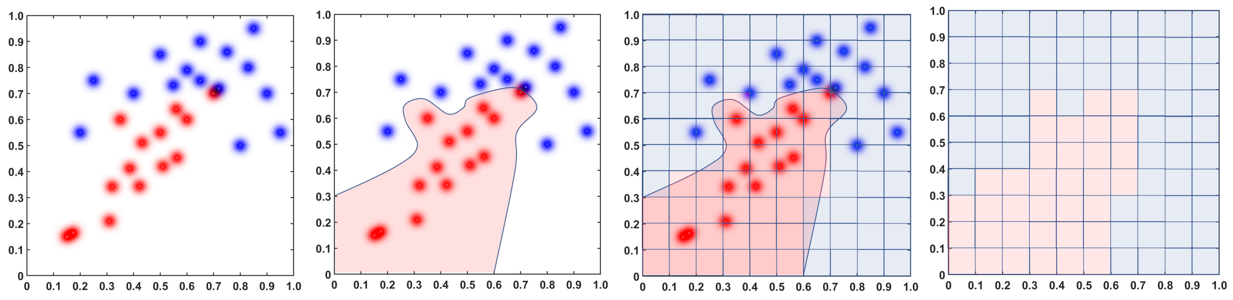
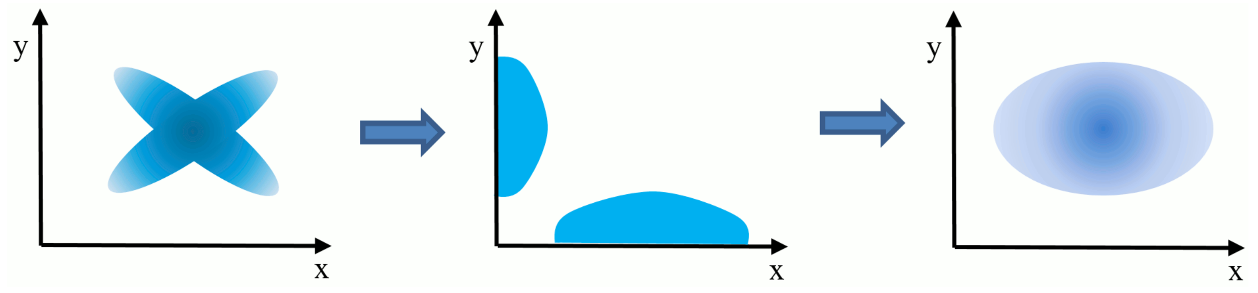


FIGURE 2. Discrete Parzen window estimation of PDF using training samples.



(a) Actual PDF (b)Projection of the PDF on x & y axes (c)Reconstructed 2D PDF

Figure 3. (a) The actual 2D PDF of the process. (b) 1D projections of the PDF shown in (a) on x & y axes. (c) Reconstructed 2D PDF from 1D projections.

2. Methodology

Multidimensional Parzen window requires that a multidimensional Gaussian is superimposed on every sample point in the training space. Figure 2-a shows a two dimensional simplified example with only two processes (classes). Each sample is marked with a white cross and the samples belonging to the first process have red auroras around them indicating the Gaussian probability field emanating from them. The second process have blue probability fields. Keep in mind that a Gaussian field extends to infinity, but because it decreases very fast, the auroras seem to disappear. Therefore, blue and red probability spaces are defined at every point in this space. The problem is figuring out which one is greater at a given point. As indicated in Eqn. (1.1), one must perform lengthy calculations to figure out probability densities for each of the classes, and then assign that sample's class based on which process's density is higher at that specific point. These calculations must be performed for each sample that needs to be classified. The result of this procedure is that we essentially divide the probability space into two regions where samples belong to one of the two classes as shown in Fig 2-b.

What if we calculate the PDF for the entire space and then store that information? That way, we can perform these calculations offline, like a training procedure, and then use this stored data online while classifying the samples in real time? That would greatly speed up the classification process. The most straightforward way to store this data is to divide the probability space into tiles using a grid. If we used one tenth of the range of each feature value as our grid spacing, then we would divide the probability space to 100 tiny squares as shown in Fig 2-c. Since there are two classes, we will need to store 200 values to approximate two different PDFs. What we will end up is small squares which approximate the PDF in that tiny region. The membership boundary of the probability space is therefore discretized as shown in Fig 2-d.

Unfortunately, such a discretization process would be cumbersome for processes with more than four dimensions. Even if we divide each feature space to 100 values, in five dimensions, the required memory would be 2x10^5=200,000 values. If we use single precision floating point format to store these values, we would need 800,000 bytes. Many of the real world classification problems have dozens of features making this approach impractical.

One solution to this problem is separating the dimensions of the PDF. Equation (1.1) can be separated into dimensions (features). For example, two dimension of the processes shown in Fig 2 can be separated. Eqn. (1.1) can be rewritten as:

$$g(x,y) = \frac{1}{N^2}\sum_{i=1}^{N}\left[\frac{1}{\sigma_x\sqrt{2\pi}}exp\left(-\frac{(x-x_i)^2}{2\sigma_x^2}\right)\times\frac{1}{\sigma_y\sqrt{2\pi}}exp\left(-\frac{(y-y_i)^2}{2\sigma_y^2}\right)\right],$$

$$g(x,y) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{\sigma_x\sqrt{2\pi}}exp\left(-\frac{(x-x_i)^2}{2\sigma_x^2}\right)\right]\times\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{\sigma_y\sqrt{2\pi}}exp\left(-\frac{(y-y_i)^2}{2\sigma_y^2}\right)\right].$$

Because of the rapid decline of the Gaussian PDF, such an approximation would not be too far off. The reader must keep in mind that this is an approximation. It does not provide the exact PDF in two dimensions but a very close (for most cases) approximation. The main drawback of this assumption is that it cannot predict the PDF as well defined or as precisely as the actual two dimensional PDF. An example of this is shown in Fig. 3. Figure 3-a shows the actual

PDF of a two dimensional process. If we calculate the one dimensional PDFs in x and y dimensions, we would get the curves shown in Figure 3-b. If we reconstruct the two dimensional PDF from these one dimensional PDFs, we would get Figure 3-c, which is a crude approximation of Figure 3-a.

Under such circumstances, discretization must be applied to the two dimensional PDF at the expanse of memory consumption. When there are more than two features, those that create such complex shape PDFs must be paired together. However, even using one dimensional PDFs, our classifier would still correctly predict %70-%90 of the samples. Some readers might think that this is a very low rate. Keep in mind that neither neural networks nor the support vector machines were invented in one day. It took decades of research and thousands of articles to bring their success rate to where they are today. As a matter of fact, once upon a time, neural networks were deemed useless. Also, as we will see in the next sections, the approach proposed here is far faster than the ANN or the SVM, which might be preferable for some applications. Because of the high speed, more features can be included to the classifier to compensate for the loss of accuracy.

3. Tests on Computer Generated Data

In order to test the approach proposed in this article, computer generated pseudo random numbers will be used. The test has two features and six different and very challenging classes. The mean and the covariance matrices of these processes are shown below:

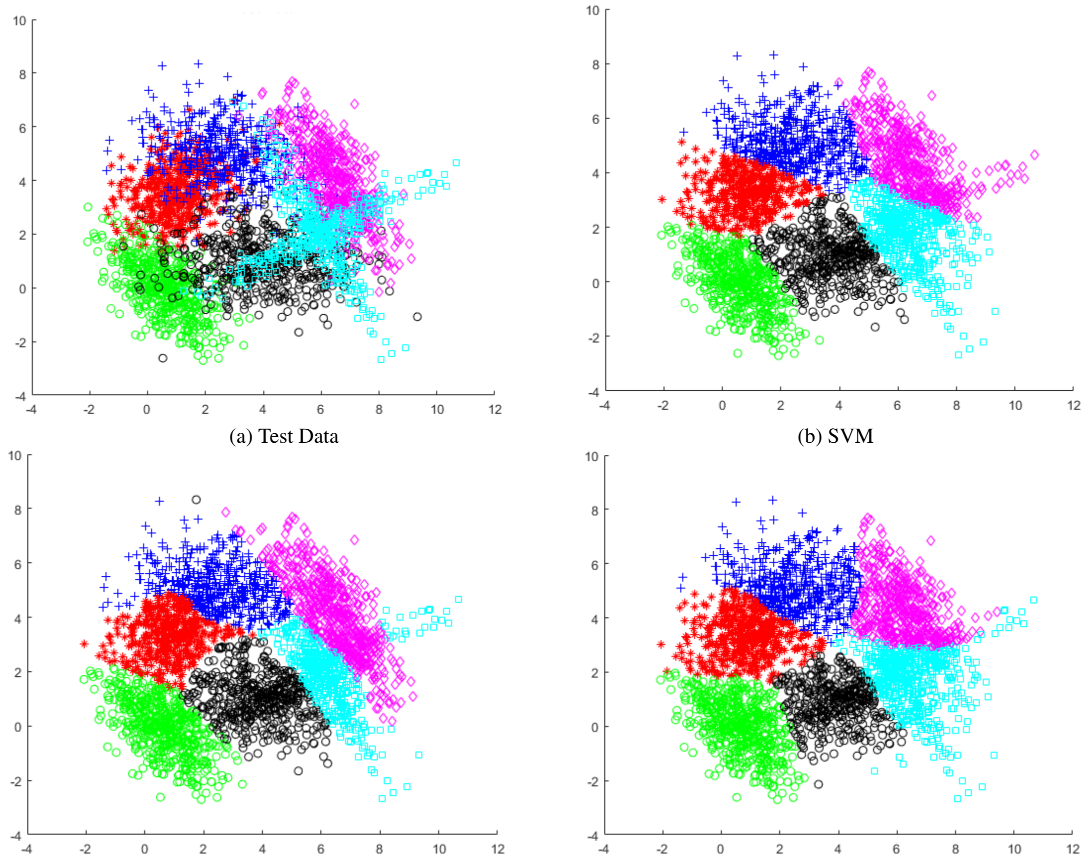
1. $\mu = [1 \quad 3.5] \qquad \sigma = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 2 \end{bmatrix} \quad (color\ red)$
2. $\mu = [2.65 \quad 5] \qquad \sigma = \begin{bmatrix} 2 & -0.3 \\ -0.3 & 1 \end{bmatrix} \quad (color\ blue)$
3. $\mu = [0.5 \quad 0] \qquad \sigma = \begin{bmatrix} 1.1 & -0.7 \\ -0.7 & 1.1 \end{bmatrix} \quad (color\ green)$
4. $\mu = [4 \quad 1] \qquad \sigma = \begin{bmatrix} 3 & -0.1 \\ -0.1 & 1 \end{bmatrix} \quad (color\ black)$
5. $\mu = [6.5 \quad 4] \qquad \sigma = \begin{bmatrix} 0.9 & -0.9 \\ -0.9 & 2 \end{bmatrix} \quad (color\ magenta)$

In order to create a star shaped PDF as discussed in the previous section:

6. $Half\ of\ the\ samples\ using:\ \mu = [6 \quad 2] \qquad \sigma = \begin{bmatrix} 1 & 1.6 \\ 1.6 & 1 \end{bmatrix} \quad (color\ cyan)$

   $and\ the\ other\ half\ using:\ \mu = [6 \quad 2] \qquad \sigma = \begin{bmatrix} 1 & -1.6 \\ -1.6 & 1 \end{bmatrix} \quad (color\ cyan)$

We create 100 samples from each process (total of 600 samples) to be used as the training set and then generate 500 samples from each process (total of 3000 samples) for testing. The experiment is conducted five times, each time regenerating the training and the test sets. Because the center (mean) of these processes are geometrically very close to each other, many of the samples fall inside the boundaries of the other processes as shown in Figure 4-a. In order to discretize the PDF we use 1000 samples for each of the dimensions. This gives us a matrix of 6x2x1000=12000 values. Another important issue is the selection of the variance σ of the Gaussian kernel we will use. There are many articles in the literature investigating the optimum value for σ. The scope of our research does not include those studies. Therefore, we will conduct a few experiments to figure out the optimum value for σ. We will conduct our experiment with three round numbers: 0.05, 0.10, and 0.15. After measuring the success rate of the algorithm using these kernels, we will further adjust σ to find a near optimal value. We also use ANN and SVM to classify these datasets do that we can measure the relative success rate of our algorithm compared to these well-established classifiers. In order to be fair, we will use the default parameters of the MATLAB program while training the ANNs and SVMs. If we use different parameters, readers might think that we altered the parameters to make our algorithm look more successful. The default ANN training algorithm in Matlab uses Levenberg-Marquardt algorithm [8, 9]. The start value of momentum is 0.001 and the maximum number of validation checks is 6. The default training algorithm for the SVM is Iterative Single Data Algorithm (ISDA) [7]. The feedforward ANN has one unavoidable parameter: The number of neurons in hidden



(a) Test Data (b) SVM

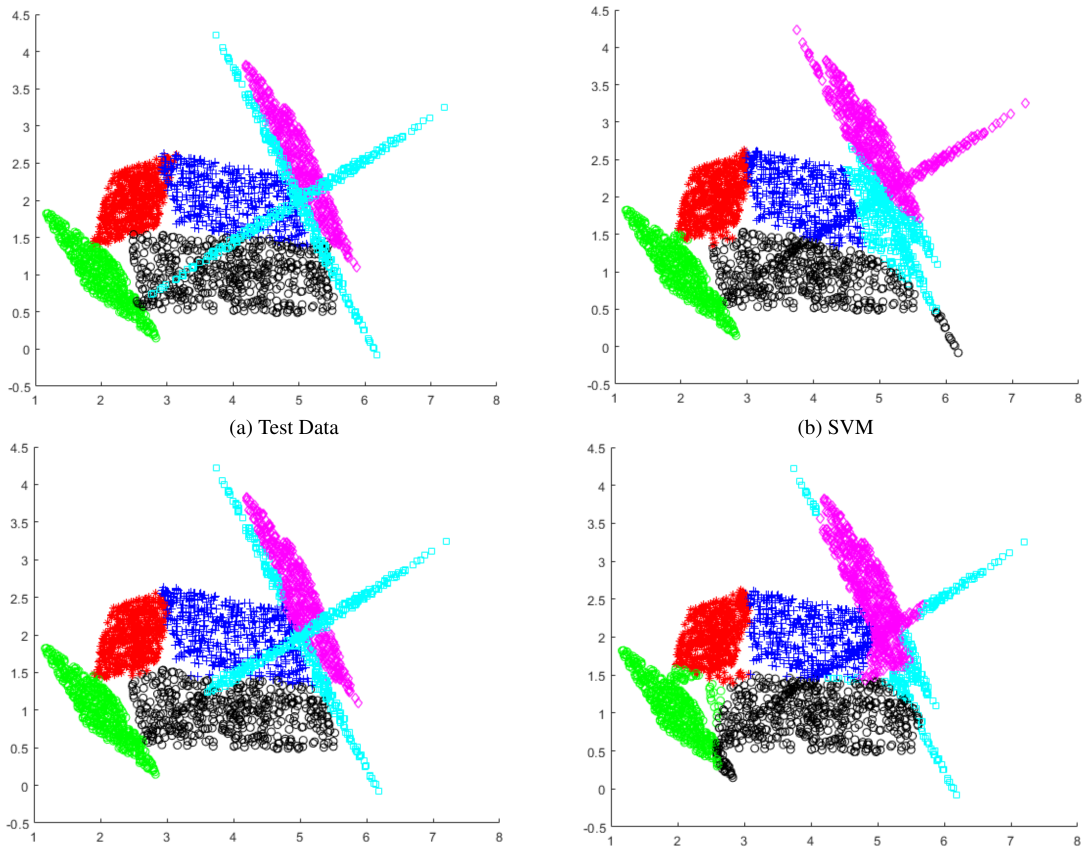
(c) ANN with 30 neurons in the hidden layer (d) Discrete Parzen Window (DPW)

Figure 4. Computer generated pseudo random test data with Gaussian distribution and the classification results of different classifiers for experiment 1.

layers. Since we want to test the speed as well as the performance, three different number of neurons in hidden layers were tested: 7, 15 and 30. Figure 4 shows the results for the dataset 1. Figure 4-a is the test set. As mentioned before, centers of the processes are geometrically very close. The star shape of the sixth process, which is marked with cyan color squares is also very noticeable. These results show that the feedforward ANN with 30 neurons in the hidden layer is more successful than the SVM and the PW classifier. ANN could distinguish process 5 (magenta) and 6 (star shaped cyan) better than the others. Table 1 shows the results of this experiment. Because it is such a challenging dataset, even the feedforward ANN and the SVM have difficulty classifying it correctly. According to the table, the ANN have a marginally higher success rate. The ANN with 15 nodes in the hidden layer and the ANN with 30 nodes performed

Table 1. Accuracy results of classification experiments.

Test	SVM	ANN(7)	ANN(15)	ANN(30)	DPW(σ=0.05)	DPW(σ=0.1)	DPW(σ=0.15)	DPW(σ=0.06)
Exp.1	0.761	0.7607	**0.7837**	0.7783	0.7553	0.7510	0.7447	0.7547
Exp.2	0.776	0.7773	**0.7873**	0.7730	0.7397	0.7367	0.7293	0.7410
Exp.3	0.783	0.7747	**0.7927**	0.7847	0.7680	0.7600	0.7550	0.7700
Exp.4	**0.786**	0.7750	0.7843	0.7837	0.7753	0.7663	0.7637	0.7727
Exp.5	0.777	0.7653	0.7760	**0.7843**	0.7560	0.7627	0.7593	0.7587
Aver.	0.776	0.7706	**0.7848**	0.7808	0.7589	0.7553	0.7504	0.7594



(a) Test Data

(b) SVM

(c) ANN with 30 neurons in the hidden layer

(d) Discrete Parzen Window (DPW)

Figure 5. Computer generated pseudo random test data with uniform distribution and the classification results of different classifiers for experiment 2.

almost the same, implying that 15 nodes are enough to classify the data. Since, even on the same training and test sets, ANN performs slightly differently every time it is retrained, ANN(15) surpassed ANN(30) by luck.

When we look at the performance of the Discrete Parzen Window (DPW) Classifier, we see that the initial values of the variance values of the Gaussian kernel (σ= 0.05, 0.10 and 0.15) performed very similar. However there is an obvious trend showing that, as the(sigma)increases, the accuracy of the classifier falls. When the experiments were performed using σ= 0.03 and σ= 0.06, it was noticed that the success rate is highest at σ= 0.06. Therefore, results for that experiment were also included in the table. These experiments proved that DPW using one dimensional discretization does not perform much worse than the feedforward ANN or the SVM. On the other hand, as we will prove in the next sections, it is far faster.

4. Uniform and Nonparametric Data

The DPW is essentially a probabilistic classifier. This implies that it works best on data that has generalized normal distribution [2]. However, thanks to the windowing property, it will work on nonsymmetric or uniformly distributed data as long as the PDF of the data has local maximums. In other words, data can be accumulated inside windows. Figure 5 shows a process similar to the one shown in the previous section. But this time, the data is uniformly distributed. The covariance matrices are kept the same. They were also used as the limits for processes. The mean of each process has been adjusted to bring the centers of each process closer, thus making the classification attempt more challenging. The new mean vectors are listed below:

1. $\mu = [2.5 \quad 2]$
2. $\mu = [4 \quad 2]$
3. $\mu = [2 \quad 1]$
4. $\mu = [4 \quad 1]$
5. $\mu = [5 \quad 2.5]$
6. $\mu = [5 \quad 2]$

Figure 5-a shows the new test data where each process has uniform PDF. Because of that, their distribution looks like a parallelogram rather than an ellipse. Their centers were brought close to each other so that either their sides or corners intersect. The sixth process, which is shown with the color cyan, is a cross shaped distribution that overlaps three other processes (magenta, blue and black). As shown in Fig. 5 and in Table 3, ANN with 30 neurons in the hidden layer again outperforms both the SVM and the DPW. In the previous example with Gaussian PDF, the ANN was only %1 more successful than the SVM. This time, the ANN is %13.5 more successful than the SVM and %12 more successful than the DPW. Among the classification results shown in Fig. 5, the ANN is the only one that recovered the cross shape of the process 6 (cyan). The training parameters of neither the ANN nor the SVM were altered. Therefore, this experiment indicates that the default training parameters chosen by the Matlab program does not fair the SVM when the PDF of the data is uniform. In the previous example, the SVM was %2 more successful than the DPW. This time the DPW is %2 more successful than the SVM. Table 3 indicates that the optimum value of σ is between 0.2 and 0.3.

As with the other probabilistic classifiers, using nonparametric data with the DPW can result in poor performance. For example, in a recent study [4], performances of several classifiers were compared using Parkinson's Disease data which included nonparametric ones. He found that Parzen Window Classifier and the SVM performed the worst. However, if most of the dimensions of the input data are parametric, and only one or two of them are nominal, then a different DPW matrix must be created for each nominal cases as we will do for the Abalone Classifier in the next section. The abalone data has seven parametric data and one nominal data (Male, Female, Infant). In order to achieve high classification rate, these three cases were independently trained and classified.

Our sample had only one nominal dimension and that dimension had only three cases which simplified our task. If the data had more dimensions and cases, then a simple decision tree would have to be implemented to assign different matrices for training and classifying. Unfortunately, this will complicate the implementation and increase memory requirements considerably. Therefore, if the data has many nonparametric features, using the DPW is not recommended.

5. Abalone Age Prediction

Abalone is a type of shellfish. There is a public dataset in Kaggle [3] which has become very popular among the machine learning scientist [1, 5, 6, 10, 11]. The dataset contains 9 features of male, female, and infant abalone such as length, diameter, height, weight etc. The goal is to predict the age of the abalone which is related to the number of rings on its shell. The number of rings in the abalone shell ranges between 1 and 29. All of the researchers in the literature divide this number to three categories and the classifier is considered successful if it predicts age of the abalone based on these three categories. For example, Güney et al. [1] used ring numbers 1-7 as the first category, 7-16 as the second and 17-29 as the third. They used 2923 of 4177 samples for training and the rest of them for testing. This is the same as Misman's [10] experiment. They tried many different classifiers such as KNN, Decision Tree, Random Forest, SVM and ANN. Their methods achieved an accuracy around %87, with the ANN being the most successful with an accuracy of %88.25. However, the ANN they used was much bigger than our examples, with three hidden layers each having 70 nodes in them.

Table 2. Accuracy results of classifications for experiments with uniform distribution.

Test	SVM	ANN(15)	ANN(30)	DPW(σ=0.1)	DPW(σ=0.2)	DPW(σ=0.3)	DPW(σ=0.4)
Exp.1	0.7793	0.8833	**0.9213**	0.7823	**0.8000**	0.7983	0.7923
Exp.2	0.7630	**0.9143**	0.8923	0.7913	**0.7990**	0.7960	0.7803
Exp.3	0.7967	0.9077	**0.9140**	0.7947	**0.8087**	0.8043	0.7963
Exp.4	0.7867	0.9213	**0.9287**	0.7657	0.7883	**0.7957**	0.7930
Exp.5	0.7700	0.9030	**0.9200**	0.7783	**0.7857**	0.7813	0.7647
Aver.	0.7791	0.9059	**0.9153**	0.7825	**0.7963**	0.7951	0.7853

Table 3 summarizes Güney's results as well as ours. In order to obtain a reference for success rate, Güney's experiment was repeated with the SVM. The SVM with default parameters classified %88.84 of the test samples correctly. This value is very similar to Güney's maximum success rate but the fact that ours is slightly higher means that either Matlab's default parameters for the SVM are very successful, or Güney chose 2293 training and 1254 test samples randomly instead of choosing the first 2293 as the training and the last 1254 as the testing samples. Then, we used the Discrete Parzen Window (DPW) algorithm to repeat Güney's experiment. The success rate is %88.6, which is less than the SVM but greater than Güney's maximum accuracy.

Another classification experiment was conducted by Şahin et al. [11] He labeled the ages of abalone between 1 and 8 as the first, 9 and 18 as the second and the rest as the third category which is more consistent with the categorization that the biologists use [6]. They used the first 3341 samples for training and the remaining 836 for testing. They classified these samples using Deep Learning algorithms and achieved a success rate of (accuracy) approximately %79. Their ANN had 65 neurons in each of the two hidden layers. Just as we did for Güney's experiments we first repeated Şahin's experiment using the SVM to get an impartial range for the possible maximum accuracy. The SVM classified %75 of the samples correctly which is consistent with Şahin's results, considering the fact that he used a large ANN. Then we repeated the experiment using the DPW and obtained an accuracy of %78.8. These results are summarized in table 2 as well. At this point we need to discuss a slight inconvenience of using the DPW. The DPW cannot use categorical features such as gender as input. While applying the one dimensional Discrete Parzen Window algorithm, we had to create three different matrices for three different sex categories: Male, female and infant. While SVM and ANN work based on clear cut boundaries, the fuzzy nature of the Gaussian PDF disallow us to use categorical features such as the sex as a feature. Therefore, these categories had to have their own PDF matrices and if the sample belongs to a male abalone then we had use PDF matrix for male to predict the age of the abalone.

6. Comparison of Classifier Complexities

As shown in Fig. 5, during the classification of a single sample, a feedforward neural network with one hidden layer has to perform I×H+H×O multiplications and (I-1)×H+(H-1)×O additions due to the weights of the ANN and H+O additions due to the biases, if it has I nodes at the input, H nodes in the hidden layer and O nodes at the output layer. At each of the hidden and output layers, a sigmoid function must be employed. Most of the well designed ANNs employ a look up table instead of calculating the sigmoid to save processing power therefore, we will assume that there are H+O table lookups. The number of additions can be formulized as:

$$(I-1)H + (H-1)O + H + O = IH + HO = H(I+O)$$

On the other hand, for DPW, there will be number of features times number of classes, which is actually same as the number of input nodes times number of output nodes, or shortly, I×O table lookups. Since the PDF of each feature is multiplied, that will require (I-1)×O multiplications.

For example, for the computer generated tests, while the ANN with 15 hidden layers will require 15×[(2×6)+6]=270 instructions, the DPW will require 6×(4-1)=18 instructions per every classification. In other words, a computer running DPW will be 15 times (%1500) faster. In Table 1, the ANN with 7 nodes in the hidden layer achieved an accuracy of 0.77, while DPW (σ=0.06) achieved an accuracy of 0.76. Their success rates are comparable. But the ANN with 7 nodes in the hidden layer would require 7×[(2×6)+6]= 144 instructions making it 8 times (%800) slower than the DPW.

Table 3. Accuracy results of abalone classification experiments.

Experiment	Accuracy%	ANN Hidden Layer Sizes	Ring Labels	Train /Test Samples
Güney[5]	88.25	70×70×70	[1-7] [8-16] [17-29]	1-2923 / 2924-4177
SVM	88.84	N/A	[1-7] [8-16] [17-29]	1-2923 / 2924-4177
DPW	88.60	N/A	[1-7] [8-16] [17-29]	1-2923 / 2924-4177
Şahin[7]	79.07	65×65	[1-8] [9-18] [19-29]	1-3341 / 3342-4177
SVM	75.12	N/A	[1-8] [9-18] [19-29]	1-3341 / 3342-4177
DPW	73.80	N/A	[1-8] [9-18] [19-29]	1-3341 / 3342-4177

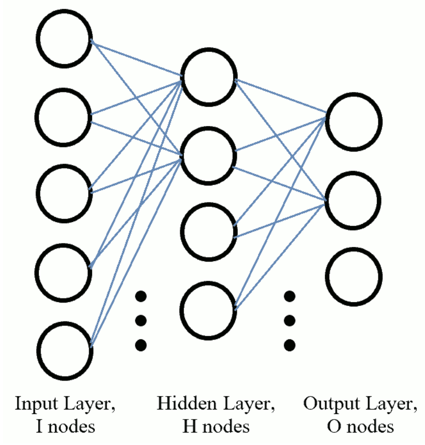


Figure 6. The schematic of an Artifical Neural Network with single hidden layer.

If a neural network has more than one hidden layer, as was the case for Güney's and Şahin'e experiments using the abalone samples, the number of instruction per each classification increases exponentially. They both have 8 input and 3 output layers. Güney used three hidden layers with 70 neurons requiring 8×703×3 multiplications and additions. Şahin used two hidden layers with 65 neurons requiring 8×652×3 multiplications and additions. For both networks, the number of table lookups performed is insignificant compared to these numbers. On the other hand, DPW requires only 7×3 multiplications and 8×3 table lookups. In other words DPW is expected to run 360 thousand times faster than Güney's ANN and 4500 times faster than Şahin's ANN.

7. Probabilistic Compression

If the required pattern classifier has I features and O outputs (classes), then the discretized PDF of the DPW is a (O×I, P) element matrix, where P is the number of bins to store the discretized one dimensional PDF of a feature per output. For example, the experiment described in section 3 will have a matrix with (2×6,1000) elements as shown below:

$$\begin{bmatrix} Px(1) & Px(2) & Px(3) & Px(4) & Px5) & .... & Px(1000) \\ Py(1) & Py(2) & Py(3) & Py(4) & Py5) & .... & Py(1000) \\ Px(1) & Px(2) & Px(3) & Px(4) & Px5) & .... & Px(1000) \\ Py(1) & Py(2) & Py(3) & Py(4) & Py5) & .... & Py(1000) \\ Px(1) & Px(2) & Px(3) & Px(4) & Px5) & .... & Px(1000) \\ Py(1) & Py(2) & Py(3) & Py(4) & Py5) & .... & Py(1000) \\ Px(1) & Px(2) & Px(3) & Px(4) & Px(5) & .... & Px(1000) \\ Py(1) & Py(2) & Py(3) & Px(4) & Py(5) & .... & Py(1000) \\ Px(1) & Px(2) & Px(3) & Px(4) & Px5) & .... & Px(1000) \\ Py(1) & Py(2) & Py(3) & Py(4) & Py5) & .... & Py(1000) \\ Px(1) & Px(2) & Px(3) & Px(4) & Px5) & .... & Px(1000) \\ Py(1) & Py(2) & Py(3) & Py(4) & Py5) & .... & Py(1000) \end{bmatrix}$$

During the classification, the algorithm receives (normalized) x and y values, and then calculates the probability of each output by multiplying the Px and Py values. This results in six different probability values. The output with the

Table 4. Number of insructions needed by ANN and DPW per each classification.

Problem	Multiplications	Additions	Table Lookups	Total Instructions
ANN	H×(I+O)	H×(I+O)	H+O	2H×[(I×O)]+H+O
DPW	(I-1)×O	0(zero)	I×O	O×(2I-1)

highest probability is selected as the outcome for that given x and y values. This matrix consists of 12000 double precision floating point values requiring 96,000 Bytes.

For pattern classification problems with more than three outcomes, this matrix can be compressed by storing only the outcomes with the highest three probabilities. Let's say, for the first feature (x), when the value is 17, outcome-4 has the highest probability, outcome-6 has the second highest probability and outcome-2 has the third highest probability. On the other hand, for the feature-2 (y), when the value is 93, outcome-3 has the highest probability, outcome-4 has the second highest probability and outcome-6 has the third highest probability. In that case the compressed PDF matrix would look like:

$$\begin{bmatrix} Px(1) & Px(2) & .... & Px(17) & .... & .... & Px(1000) \\ Px(1) & Px(2) & .... & Px(17) & .... & .... & Px(1000) \\ Px(1) & Px(2) & .... & Px(17) & .... & .... & Px(1000) \\ Px(1) & Px(2) & .... & .... & Py(93) & .... & Px(1000) \\ Px(1) & Px(2) & .... & .... & Py(93) & .... & Px(1000) \\ Px(1) & Px(2) & .... & .... & Py(93) & .... & Px(1000) \end{bmatrix}$$

Naturally, in order to calculate the membership probabilities of a sample at (x=17, y=93) (normalized values), we would need both Px and Py values and then multiply them. Unfortunately, for a given point, this matrix might have only one of those values. For example, the green one has only Px and the blue one has only Py. However that is not a problem. Because we want the highest probability, instead of multiplying, we can add the values. If the other value is not there, it means that its probability is so low that it couldn't make the list any way. Therefore we can assume a very low probability for the nonexistent values and add that value. If we add the probability values for the point at P(17,93) we would probably get outcome-4 as the highest because its Px is the largest and its Py is the second largest. Therefore, the outcome of the classification would be process-4. This type of probabilistic compression decreases the matrix size to (I×D, P) where D is the depth of the matrix. In our example the depth is 3 because we store the highest three probabilities. Note that this type of compression becomes more efficient if the classification problem has many outcomes because the size of the matrix is not dependent on the number of classes. In this example, instead of 12000 values we use 6000. However, we need another matrix to hold the index of the classes. Since we cannot write the probability values using different colors, we need another integer matrix to hold the indexes. Fortunately this matrix can be a Byte type matrix instead of double precision floating point. The total memory consumption will be 6000×8+6000=54,000 Bytes compared to 96000. Such compression would also increase the speed if the classification problem consists of many classes.

Naturally, there is a price to be paid for the probabilistic compression. When we repeat the experiments conducted in section three using the compressed matrices, which we call Compressed Discrete Parzen Window (CPWD) classifier, the accuracies fall further as listed in Table 4.

Table 4 shows that the highest accuracy achieved by DPW classifier is 0.7594 while CDPW is 0.7362. As a result of the compression, %2 accuracy is lost and that is the price we pay for using a smaller lookup table (matrix). However, Table 4 reveals an unexpected trend as well. In section 3, we discovered that the optimum σ value for the Gaussian kernel was 0.06 for DPW. As we increased the value of σ, the accuracy of the DPW fell. On the other hand, Table 4 clearly shows that the accuracy of CDPW classifier increases as the σ increases. As a matter of fact,further experimentation revealed that the optimum σ is at around 0.27 where an average accuracy of approximately 0.745 is achieved by the CDPW. Unfortunately this author lacks the math knowledge to explain that behavior.

TABLE 5. Accuracy results of classification experiments of DPW vs. CDPW.

Test	DPW(σ=0.1)	**CDPW(σ=0.1)**	DPW(σ=0.15)	**CDPW(σ=0.15)**	DPW(σ=0.06)	**CDPW(σ=0.06)**
Exp.1	0.7510	0.7273	0.7447	0.7347	0.7547	0.7097
Exp.2	0.7367	0.7180	0.7293	0.7250	0.7410	0.6967
Exp.3	0.7600	0.7323	0.7550	0.7410	0.7700	0.7097
Exp.4	0.7663	0.7437	0.7637	0.7513	0.7727	0.7167
Exp.5	0.7627	0.7217	0.7593	0.7390	0.7587	0.6900
Aver.	0.7553	0.7286	0.7504	**0.7362**	**0.7594**	0.7046

8. Conclusion

In this article, an approximation and a discretization method is introduced to speed up classification process of the Parzen Window Classifier which we call the Discretized Parzen Window Classifier (DPW). This faster version is tested on two challenging data sets. The first one uses computer generated pseudo random numbers to simulate six different processes. The DPW's performance was compared to ANN with different number of neurons in a single hidden layer and SVM. The DPW performed only %3 worse than the best ANN classifier while being at least 8 times faster. Then the performance of the DPW was compared to ANNs with more hidden layers designed by other researchers and SVM using abalone age data in two different configurations. In the worst case, DPW performed %5 worse than the deep ANN while being more than 4000 times faster. Unfortunately, on uniform data, the DPW performs %12 worst than the ANN classifier.

A probabilistic compression method was also introduced (CDPW) to decrease the memory required by the DPW. As expected, because of some information loss, CDPW classifier performed %1-2 worse than the DPW. Overall, the algorithms introduced in this article are not as successful as the most successful classifiers in the literature, but they are much faster. They might be still useful for applications that has restrictions on processing power, such as the mobile robots and the drones.

Conflicts of Interest

The author declares that there are no conflicts of interest regarding the publication of this article.

Authors Contribution Statement

The author has read and agreed the published version of the manuscript.

References

[1] Guney, S., Kilinc, I., Hameed, A.A., Jamil, A., *Abalone age prediction using machine learning*, Mediterranean Conference on Pattern Recognition and Artificial Intelligence, Springer International Publishing, (2021), 879–883.

[2] https://en.wikipedia.org/wiki/Kernel_(statistics)

[3] https://www.kaggle.com

[4] Jain, V., Singh, R., Gupta, A., *Exploring binary classification models for Parkinson's disease detection,* Procedia Computer Science **235**(2024), 2332–2341.

[5] Kaur, S., Chaudhary, S., Thakur, A., Bajaj, R., Gupta, A. et al. *Abalone age prediction using optimized ensembel model*, IEEE 11th International Conference on System Modeling & Advancement in Research Trends, (2022), 1023–1027.

[6] Kawamura, T., Roberts, R.D., Takami, H., *Importance of periphyton in Abalone culture*, Periphyton: Ecology, Exploitation and Management, (2005), 269–883.

[7] Kecman, V., Huang, T.M., Vogt., M., *Iterative single data algorithm for training kernel machines from huge data sets: Theory and Performance*, In Support Vector Machines: Theory and Applications. Edited by Lipo Wang, 255–274. Berlin, Springer-Verlag, 2005.

[8] Levenberg, K., *A method for the solution of certain non-linear problems in least squares*, Quarterly of Applied Mathematics, **2**(2)(1944), 164–168.

[9] Marquardt, D., *An algorithm for least-squares estimation of nonlinear parameters*, SIAM Journal on Applied Mathematics, **11**(2)(1963), 431–441.

[10] Misman, M.F., Samah, A.A., Ab Aziz, N.A., Majid, H.A., Shah, Z.A. et al. *Prediction of abalone age using regression-based neural network*, IEEE 1st International Conference on Artificial Intelligence and Data Sciences (AiDAS), (2019), 23–28.

[11] Sahin, E., Saul, C.J., Ozsarfati, E., Yilmaz, A., *Abalone life phase classification with deep learning*, IEEE 5th International Conference on Soft Computing & Machine Intelligence, (ISCMI), (2018), 163–167.

[12] Shaikh, M.S., Alftieh, A.M., *Evaluation of four classification algorithms for P300 based brain computer interface*, Life Science Journal, **10**(3)(2013), 879–883.

[13] Wang, X., Tiiio, P., Fardal, M.A., Raychaudhury, S., Babul, A., *Fast parzen window density estimator*, IEEE International Joint Conference on Neural Networks, **60**(2009), 3267–3274.