Pattern Recognition Letters 63 (2015) 30–35

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

The Parzen Window method: In terms of two vectors and one matrix

Hamse Y. Mussa [a,b,*], John B.O. Mitchell [a], Avid M. Afzal [b]

[a] EaStCHEM School of Chemistry and Biomedical Sciences Research Complex, University of St Andrews, North Haugh, St Andrews KY16 9ST, Scotland, UK
[b] Centre for Molecular Informatics, Department of Chemistry, University of Cambridge, Lensfield Road, Cambridge CB2 1EW, England, UK

ARTICLE INFO

Article history:
Received 15 February 2015
Available online 18 June 2015

Keywords:
Probability density function
Kernel functions
Parzen Window

ABSTRACT

Pattern classification methods assign an object to one of several predefined classes/categories based on features extracted from observed attributes of the object (pattern). When L discriminatory features for the pattern can be accurately determined, the pattern classification problem presents no difficulty. However, precise identification of the relevant features for a classification algorithm (classifier) to be able to categorize real world patterns without errors is generally infeasible. In this case, the pattern classification problem is often cast as devising a classifier that minimizes the misclassification rate. One way of doing this is to consider both the pattern attributes and its class label as random variables, estimate the posterior class probabilities for a given pattern and then assign the pattern to the class/category for which the posterior class probability value estimated is maximum. More often than not, the form of the posterior class probabilities is unknown.

The so-called Parzen Window approach is widely employed to estimate class-conditional probability (class-specific probability) densities for a given pattern. These probability densities can then be utilized to estimate the appropriate posterior class probabilities for that pattern. However, the Parzen Window scheme can become computationally impractical when the size of the training dataset is in the tens of thousands and L is also large (a few hundred or more). Over the years, various schemes have been suggested to ameliorate the computational drawback of the Parzen Window approach, but the problem still remains outstanding and unresolved.

In this paper, we revisit the Parzen Window technique and introduce a novel approach that may circumvent the aforementioned computational bottleneck. The current paper presents the mathematical aspect of our idea. Practical realizations of the proposed scheme will be given elsewhere.

1. Introduction

In mathematical pattern recognition, the problem of pattern classification entails assigning an object – based on a number of specific features of the object – to one of a finite set of predefined classes/categories ω_j, where $j = 1, 2, \ldots, J$, with J being the number of classes/categories of interest. Typically the object (or simply the pattern) is represented by an L-dimensional vector $\mathbf{x}$ whose elements, $(x_1, x_2, \ldots, x_L)$, are values assumed to contain the appropriate information about the specific pattern features utilized to accurately classify the pattern represented by $\mathbf{x}$.

When L discriminatory features for a pattern can be determined accurately, the pattern classification problem presents no difficulty: it reduces to a simple look-up table scheme. However, identifying the relevant features to classify realistic patterns without classification errors is generally impossible. Thus, the pattern classification problem is often cast as the task of finding a classifier that minimizes the misclassification rate [1]. One popular way of achieving this objective is to treat both the pattern vector $\mathbf{x}$ and the class label ω_j as random variables. In this case, the posterior class probabilities $p(\omega_j|\mathbf{x})$ for a given pattern $\mathbf{x}$ is computed; then pattern $\mathbf{x}$ is assigned to the class, for which the $p(\omega_j|\mathbf{x})$ value is maximum [1–6]. (In the last step it is being assumed that all misclassification errors are equally bad [1,3,4].)

However, in practice, the form of the function $p(\omega_j|\mathbf{x})$ is unknown; instead, N patterns $\mathbf{x}_i$ and their corresponding correct class labels $y_i \in \{\omega_1, \omega_2, \ldots, \omega_J\}$ – i.e., $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, assumed to constitute a representative dataset of the joint probability density function $p(\omega_j, \mathbf{x})$ for ω_j and $\mathbf{x}$ – are usually available. It is from these prototype patterns $\mathbf{x}_i$ and their corresponding class labels y_i that one tries to estimate $p(\omega_j|\mathbf{x})$.

* Corresponding author at: EaStCHEM School of Chemistry and Biomedical Sciences Research Complex, University of St Andrews, North Haugh, St Andrews KY16 9ST, Scotland, UK. Tel.:+44 7951519416; fax: +44 1223763076.
E-mail address: mussax021@gmail.com (H.Y. Mussa).

http://dx.doi.org/10.1016/j.patrec.2015.06.002

According to basic probability rules [1–7],

$$p(\omega_j|\mathbf{x})p(\mathbf{x}) = p(\omega_j, \mathbf{x}) = p(\mathbf{x}|\omega_j)p(\omega_j) \tag{1}$$

These rules allow one to modularize the estimation problem and estimate $p(\omega_j|\mathbf{x})$ (and, of course, $p(\mathbf{x})$) in terms of $p(\mathbf{x}|\omega_j)$ and $p(\omega_j)$:

$$p(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_j)p(\omega_j)}{p(\mathbf{x})}, \tag{2}$$

whereby we may have a better chance of being able to estimate $p(\mathbf{x}|\omega_j)$ and $p(\omega_j)$ from $\mathcal{D}$ than estimating $p(\omega_j|\mathbf{x})$ directly from $\mathcal{D}$. In the denominator, $p(\mathbf{x}) = \sum_{j=1}^{J} p(\mathbf{x}|\omega_j)p(\omega_j)$.

In the Bayesian statistics framework, $p(\omega_j)$ is referred to as the class prior probability, which is the probability that a member of class ω_j will occur. The function $p(\mathbf{x}|\omega_j)$ is called the class-conditional probability density function, i.e. the probability density of observing pattern $\mathbf{x}$ given that $\mathbf{x}$ is a member of class ω_j. The denominator term on the right hand side of Eq. 2 is often called the "evidence" or "marginal likelihood". For the purpose of this paper we can afford to simply view this term as a normalization factor.

If there is evidence that the number of prototype patterns per class is an indication of the importance of that class, then a sensible approximation of $p(\omega_j)$ can be

$$p(\omega_j) = \frac{\sum_{i=1}^{N} \nu_j^i}{N}, \tag{3}$$

where $\nu_j^i = 1$ if the ith prototype $\mathbf{x}_i$ belongs to class ω_j, otherwise $\nu_j^i = 0$; and N is as described before. Nonetheless, $p(\omega_j)$ is typically assumed to be uniform, i.e., $p(\omega_j) = \frac{1}{J}$, where J is as defined before.

Estimating $p(\mathbf{x}|\omega_j)$ from $\mathcal{D}$ is not straightforward [1–5]. In the last half-century, a plethora of methods have been proposed for estimating $p(\mathbf{x}|\omega_j)$ based on $\mathcal{D}$, the so-called training set. There are ample excellent reviews and text books on this topic; for example, the two books – one by Hand [4] and the other by Murphy [8] – give adequate and accessible descriptions of the bulk of these approaches devised in recent (and not so recent) years.

In this paper we are concerned with one particular approach that is widely thought to be apropos to the task of estimating $p(\mathbf{x}|\omega_j)$ from a representative training dataset: the so-called Parzen Window method [1,2,4,9,10], also known as Parzen estimator, Potential function technique [10], to name but a few.

In the preceding discussion and in the rest of the paper, the terms "class", "label", "class label" and "category" are employed interchangeably. For notational simplicity we use $\mathbf{x}$, $\mathbf{x}_i$ and ω_j both as random variables and their realizations. Furthermore we follow (in line with the current trend in machine learning and statistics) the convenient – but not necessarily correct – practice of using the term "density" for both a discrete random variable's probability function and for the probability function of a continuous random variable. An implicit assumption being made throughout the paper is that all spaces, matrices, vectors, functions and variables (discrete or not), *etc.*, are real.

2. Literature review

The Parzen Window approach can approximate $p(\mathbf{x}|\omega_j)$ by a simple formula [1,2,10]:

$$\hat{p}(\mathbf{x}|\omega_j) = \frac{1}{\sum_{i=1}^{N} \nu_j^i} \sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i; \lambda)\nu_j^i, \tag{4}$$

where $\mathbf{x}_i \in \mathcal{D}$ denotes prototype patterns and ν_j^i is as defined before. $K(\mathbf{x}, \mathbf{x}_i; \lambda)$ – commonly known as the kernel function – is a two-variable function with specific properties, which are abundantly covered in the statistical pattern recognition literature [1,2,9,10]. At any rate, it might be helpful to think of $K(\mathbf{x}, \mathbf{x}_i; \lambda)$ as a measure of similarity returning how similar patterns $\mathbf{x}$ and $\mathbf{x}_i$ are, λ being a tunable (smoothing) parameter. In other words, the kernel function peaks at $\mathbf{x} = \mathbf{x}_i$ and decays away elsewhere; the λ parameter, *inter alia*, has an important role in determining the rate of the decay.

Eq. 4 indicates that $\hat{p}(\mathbf{x}|\omega_j)$ is formed from the superposition of kernel function $K(\mathbf{x}_i, \mathbf{x}; \lambda)$ values at the given prototype patterns $\mathbf{x}_i$ for class ω_j. The Parzen Window method is powerful in the sense that, with enough representative data points (prototypes/references), its estimate of the class conditional probability density converges to $p(\mathbf{x}|\omega_j)$ (see Ref. [1], Chapter 4). Although Eq. 4 is conceptually simple and capable of providing a good estimate of $p(\mathbf{x}|\omega_j)$, it can computationally suffer from the requirements that all the prototypes/references $\mathbf{x}_i$ for class ω_j must be retained in main-memory to compute $\hat{p}(\mathbf{x}|\omega_j)$, the estimate of $p(\mathbf{x}|\omega_j)$. Furthermore, considerable CPU-time may be required each time this method is used to estimate $p(\mathbf{x}|\omega_j)$ to classify a novel pattern. The fact that the size of the reference pattern vectors $\mathbf{x}_i$ can be easily in the hundreds (or more) may exacerbate the main-memory and CPU-time requirements.

Over the years, various schemes have been developed to address the computational drawback of this otherwise elegant and powerful method. For example, one of these schemes entails – see Ref. [1] (Chapter 4), Ref. [4] (Chapter 2) and Ref. [10] (Chapters 6) for detailed technical and practical discussions – expressing the kernel function as a finite series expansion

$$K(\mathbf{x}, \mathbf{x}_i; \lambda) = \sum_{m=1}^{M} \phi_m(\mathbf{x})\phi_m(\mathbf{x}_i), \tag{5}$$

which renders the right hand side of Eq. 4

$$\frac{1}{\sum_{i=1}^{N} \nu_j^i} \sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i; \lambda)\nu_j^i = \frac{1}{\sum_{i=1}^{N} \nu_j^i} \sum_{i=1}^{N} \nu_j^i \sum_{m=1}^{M} \phi_m(\mathbf{x})\phi_m(\mathbf{x}_i), \tag{6}$$

with $\{\phi_m\}_{m=1}^{M}$ being appropriate basis functions (not necessarily polynomials) defined in the feature space in which the pattern vectors $\mathbf{x}$ and $\mathbf{x}_i$ reside.

From Eqs. 4 and 6, we have

$$\hat{p}(\mathbf{x}|\omega_j) = \sum_{m=1}^{M} c_{m_{\omega_j}} \phi_m(\mathbf{x}), \tag{7}$$

where

$$c_{m_{\omega_j}} = \frac{1}{\sum_{i=1}^{N} \nu_j^i} \sum_{i=1}^{N} \nu_j^i \phi_m(\mathbf{x}_i),$$

with $\nu_j^i = 1$ if pattern $\mathbf{x}_i$ belongs to class ω_j, otherwise $\nu_j^i = 0$.

This scheme certainly removes the reference patterns' storage problem. However, it can create a computational problem of its own, in particular when both M and L are large, which is often the case in real world classification problems. Computing M basis functions of L variables to classify a new pattern $\mathbf{x}$ is not a trivial computational task [1–4,10–12].

Another approach – albeit a particular case of the scheme above – is that proposed by Specht [13]. It was based on a Taylor series expansion of $\rho(\mathbf{x}, \mathbf{x}_i; \lambda)$ (see Eq. 8), such that an rth order polynomial in L variables was required to estimate and store $\binom{L+r}{L}$ terms to approximate $\hat{p}(\mathbf{x}|\omega_j)$ [10,13,14]. For short but "insightful" descriptions of the relationship between an appropriate value of r and the smoothing parameter λ, see Ref. [1] (Chapter 4) and Ref. [14] (Chapter 4). In principle, Specht's scheme has a strong appeal of simplicity providing the number of terms required in the Taylor series can be held to a practical limit. Unfortunately, both r and L can be large in current realistic classification problems [1,4,10].

Despite these (and many other) efforts, to the best of our knowledge, the computational bottleneck that the Parzen Window method encounters, when N and L are large, remains an unresolved issue.

Thus, the motivation for this paper is to introduce yet another scheme that might be able to circumvent the aforementioned computational bottleneck problem, while retaining the estimation power and conceptual simplicity of the Parzen Window method. This work is confined to Parzen Window based approaches, in which the kernel function is – or can be expressed – in the form

$$K(\mathbf{x}, \mathbf{x}_i; \lambda) = Af(\mathbf{x}; \lambda)\rho(\mathbf{x}, \mathbf{x}_i; \lambda)f(\mathbf{x}_i; \lambda), \tag{8}$$

where $A > 0$; $f(\mathbf{x}; \lambda)$ and $f(\mathbf{x}_i; \lambda)$ are any real functions defined in the feature space; $\rho(\mathbf{x}, \mathbf{x}_i; \lambda)$ is a polynomial in $\mathbf{x}$ and $\mathbf{x}_i$; and λ is as defined before. The kernel functions that are or can be written in the form above are ubiquitous nowadays in data analysis [4,6,8]. They have most often been successfully applied to discrete data; for this reason we decided to confine attention to the discrete case. For illustrative purposes, we focus on binary data, i.e., $x_l = 0$ or 1 denoting absence or presence of feature x_l in the pattern vector, respectively. That is to say, both $\mathbf{x}$ (test pattern vector) and $\mathbf{x}_i$ (reference/prototype pattern vector) reside in a binary feature space $\mathcal{X} = \{0, 1\}^L$.

The extension of the proposed scheme to continuous data – i.e., $\mathcal{X} = \mathcal{R}^L$ – is straightforward.

One final, but important remark is that Specht's approach and our proposed scheme are arguably similar in spirit. However there are crucial differences: unlike Specht's formulation, our scheme does not estimate $\binom{L+r}{L}$ terms, it does not retain $\binom{L+r}{L}$ terms in main-memory, nor does the variable r feature in the final form of our algorithm – instead, in our case, only two L-dimensional vectors and one L-by-L matrix are required to retain in main-memory. The two vectors and the matrix can notably be highly sparse. We will briefly expound on this assertion shortly.

3. Proposed method and discrete Parzen Window approach

As a concrete example, we use the widely utilized kernel function (albeit in cheminformatics [15,16], and references therein) introduced by Aitchison and Aitken (AA–kernel) [17,18],

$$K(\mathbf{x}, \mathbf{x}_i; \lambda) = \lambda^L \left(\frac{1-\lambda}{\lambda}\right)^{d(\mathbf{x}, \mathbf{x}_i)}, \tag{9}$$

where $0.5 < \lambda < 1$ and $d(\mathbf{x}, \mathbf{x}_i)$ denotes the number of components in which $\mathbf{x}$ and $\mathbf{x}_i$ disagree. This dissimilarity measure $d(\mathbf{x}, \mathbf{x}_i)$ can be conveniently expressed as [4]

$$d(\mathbf{x}, \mathbf{x}_i) = \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{x}_i + \mathbf{x}_i^T\mathbf{x}_i \tag{10}$$

In passing, the AA–kernel is basically a discrete analogue of an isotropic Gaussian kernel [17,18].

From Eqs. 9 and 10, and the fact that $e^{\ln w} = w$, we have

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}_i; \lambda) &= \lambda^L e^{\ln\left(\frac{1-\lambda}{\lambda}\right)(\mathbf{x}^T\mathbf{x} + \mathbf{x}_i^T\mathbf{x}_i)} \times e^{-2\ln\left(\frac{1-\lambda}{\lambda}\right)\mathbf{x}^T\mathbf{x}_i} \\ &= \lambda^L e^{-\alpha(\mathbf{x}^T\mathbf{x} + \mathbf{x}_i^T\mathbf{x}_i)} \times e^{2\alpha\mathbf{x}^T\mathbf{x}_i} \\ &= \lambda^L e^{-\alpha(\mathbf{x}^T\mathbf{x})} \times e^{2\alpha\mathbf{x}^T\mathbf{x}_i} \times e^{-\alpha(\mathbf{x}_i^T\mathbf{x}_i)}, \end{aligned} \tag{11}$$

where $\alpha = \ln\left(\frac{\lambda}{1-\lambda}\right)$. The term $e^{2\alpha\mathbf{x}^T\mathbf{x}_i}$ can be written as

$$e^{2\alpha\mathbf{x}^T\mathbf{x}_i} = \sum_{r=0}^{\infty} \frac{(2\alpha)^r}{r!}(\mathbf{x}^T\mathbf{x}_i)^r = \sum_{r=0}^{\infty} \gamma_r(\mathbf{x}^T\mathbf{x}_i)^r, \tag{12}$$

where $\gamma_r = \frac{(2\alpha)^r}{r!}$.

Inserting Eq. 12 into Eq. 11 yields

$$K(\mathbf{x}, \mathbf{x}_i; \lambda) = \lambda^L e^{-\alpha(\mathbf{x}^T\mathbf{x})}\left[\sum_{r=0}^{\infty} \gamma_r(\mathbf{x}^T\mathbf{x}_i)^r\right] e^{-\alpha(\mathbf{x}_i^T\mathbf{x}_i)} \tag{13}$$

(*cf.* Eq. 8).

From Eqs. 13 and 4, we have

$$\hat{p}(\mathbf{x}|\omega_j) = \frac{\lambda^L e^{-\alpha(\mathbf{x}^T\mathbf{x})}}{\sum_{i=1}^{N} \nu_j^i} \sum_{i=1}^{N} \nu_j^i e^{-\alpha(\mathbf{x}_i^T\mathbf{x}_i)} \sum_{r=0}^{\infty} \gamma_r(\mathbf{x}^T\mathbf{x}_i)^r \tag{14}$$

Now we come to the main contribution of this paper: removing the requirement for retaining all the reference/prototype patterns for class ω_j in main-memory to compute $\hat{p}(\mathbf{x}|\omega_j)$ in order to estimate $\hat{p}(\omega_j|\mathbf{x})$ to classify a new pattern $\mathbf{x}$. However, first we simplify the notation by defining these variables N_{ω_j}, a, $\mathbf{z}$ and $\mathbf{z}'$, which will be consistently used throughout, as follows: $N_{\omega_j} = \sum_{i=1}^{N} \nu_j^i$; $\beta_i = e^{-\alpha(\mathbf{x}_i^T\mathbf{x}_i)}$; $a = \sum_{i=1}^{N_{\omega_j}} \beta_i$; $\mathbf{z} = \sum_{i=1}^{N_{\omega_j}} \mathbf{x}_i$; and $\mathbf{z}' = \sum_{i=1}^{N_{\omega_j}} \beta_i\mathbf{x}_i$, where N_{ω_j} refers to the number of patterns in the training dataset that belong to class ω_j; N, α and ν_j^i are as described before. In our new notation, Eq. 14 becomes

$$\begin{aligned} \hat{p}(\mathbf{x}|\omega_j) &= B\left[\gamma_0 \sum_{i=1}^{N_{\omega_j}} \beta_i + \sum_{r=1}^{\infty} \gamma_r \sum_{i=1}^{N_{\omega_j}} \beta_i(\mathbf{x}^T\mathbf{x}_i)^r\right] \\ &= B\left[\gamma_0 a + \sum_{r=1}^{\infty} \gamma_r \sum_{i=1}^{N_{\omega_j}} (\mathbf{x}^T(\beta_i\mathbf{x}_i))(\mathbf{x}^T\mathbf{x}_i)^{r-1}\right], \end{aligned} \tag{15}$$

where $B = \frac{\lambda^L e^{-\alpha(\mathbf{x}^T\mathbf{x})}}{N_{\omega_j}}$.

The main contribution of the paper is formulating Eq. 15 in terms of γ_r , a, $\mathbf{z}$, $\mathbf{z}'$ and an L-by-L matrix, $\mathbf{Q}$ which will be defined shortly. The task of this formulation basically amounts to expressing $\sum_{i=1}^{N_{\omega_j}} (\mathbf{x}^T(\beta_i\mathbf{x}_i))(\mathbf{x}^T\mathbf{x}_i)^{r-1}$ in terms of $\mathbf{z}$, $\mathbf{z}'$ and $\mathbf{Q}$. In doing this, we hope to ameliorate the computational drawback of the Parzen Window method based on kernel functions in the form given in Eq. 8.

4. Results

When $r = 1$, the task is trivial: $\sum_{i=1}^{N_{\omega_j}} (\mathbf{x}^T(\beta_i\mathbf{x}_i))(\mathbf{x}^T\mathbf{x}_i)^{r-1}$ reduces to

$$\sum_{i=1}^{N_{\omega_j}} \mathbf{x}^T(\beta_i\mathbf{x}_i) = \mathbf{x}^T\mathbf{z}' \tag{16}$$

where, by definition (see Section 3), $\mathbf{z}' = \sum_{i=1}^{N_{\omega_j}} (\beta_i\mathbf{x}_i)$.

However, when $r > 1$, the task can be taxing. To this end, we make use of a simple – but useful – proposition (Proposition 1, whose proof is provided in Appendices A and B) to demonstrate that $\sum_{i=1}^{N_{\omega_j}} (\mathbf{x}^T(\beta_i\mathbf{x}_i))(\mathbf{x}^T\mathbf{x}_i)^{r-1}$ for $r > 1$ can be written as (see Eq. 19 in Appendix C)

$$\sum_{i=1}^{N_{\omega_j}} (\mathbf{x}^T(\beta_i\mathbf{x}_i))(\mathbf{x}^T\mathbf{x}_i)^{r-1} = (\mathbf{x}^T\mathbf{Q}\mathbf{x})(\mathbf{x}^T\mathbf{z})^{r-2}, \tag{17}$$

where $\mathbf{Q}$ is just an L-by-L matrix, (see Eq. 19).

Inserting Eqs. 16 and 17 into Eq. 15 results in

$$\begin{aligned} \hat{p}(\mathbf{x}|\omega_j) &= B\left[\gamma_0 a + \gamma_1(\mathbf{x}^T\mathbf{z}') + (\mathbf{x}^T\mathbf{Q}\mathbf{x}) \sum_{r=2}^{\infty} \gamma_r(\mathbf{x}^T\mathbf{z})^{r-2}\right] \\ &= B\left[\gamma_0 a + \gamma_1(\mathbf{x}^T\mathbf{z}') + \left(\frac{4\alpha^2(e^{\mu} - \mu - 1)}{\mu^2}\right)(\mathbf{x}^T\mathbf{Q}\mathbf{x})\right], \end{aligned} \tag{18}$$

where $B = \frac{\lambda^L e^{-\alpha(\mathbf{x}^T\mathbf{x})}}{N_{\omega_j}}$; $\sum_{r=2}^{\infty} \gamma_r(\mathbf{x}^T\mathbf{z})^{r-2} = \frac{4\alpha^2(e^{\mu} - \mu - 1)}{\mu^2}$, with $\mu = 2\alpha(\mathbf{x}^T\mathbf{z})$, $\gamma_r = \frac{(2\alpha)^r}{r!}$, $\alpha = \ln\left(\frac{\lambda}{1-\lambda}\right)$ and $0.5 < \lambda < 1$

Evidently, Eq. 18 illustrates that it is not necessary to retain all reference/prototype patterns for a given class in main-memory to compute the value of $\hat{p}(\mathbf{x}|\omega_j)$ for a test pattern $\mathbf{x}$; instead, all that is required is an L-by-L matrix $\mathbf{Q}$ and two L size vectors ($\mathbf{z}$ and $\mathbf{z}'$), which are computed once and then retained in main-memory. This was the objective we set out to achieve in this paper.

One final, but important, remark is that $\mathbf{z}$, $\mathbf{z}'$ and $\mathbf{Q}$ can be highly sparse in real world applications when $x_l \in \{0, 1\}$, in particular if the

value of L is large. The fundamental reason for this sparsity is that in a high-dimensional reference pattern vector $\mathbf{x}_i$ there is the potential of many of its components being zero – i.e., many of the features are very likely to be absent in $\mathbf{x}_i$.

In passing, if we are dealing with continuous data, whereby $\mathbf{x}_i \in \mathcal{R}^L$, the vectors $\mathbf{z}, \mathbf{z}'$ and matrix $\mathbf{Q}$ could be dense. Nonetheless, storing $\mathbf{Q}$ can still be computationally cheaper than retaining N_{ω_j} reference patterns $\mathbf{x}_i$ per class in main-memory – providing that $N_{\omega_j} > L$.

The current paper presents the mathematical aspect of our idea. Practical realizations of the proposed scheme will be given elsewhere.

5. Conclusion

The Parzen Window method is a powerful tool for estimating class conditional probability density functions. However, it can suffer from a severe computational bottleneck when the training dataset is large. Over the years, attempts have been made to rectify this computational drawback of the method. To the best of our knowledge the issue has remained unsolved. In this paper we have proposed a novel scheme, which we hope contributes to our endeavor of alleviating the computational bottleneck from which the Parzen Window algorithm suffers when the training dataset is large.

Acknowledgments

We thank the BBSRC for funding this research through Grant BB/I00596X/1. JBOM thanks the Scottish Universities Life Sciences Alliance (SULSA) for fin ancial support.

Appendix A

Proposition 1. Let a_i and b_i be real variables $\in [0, \infty)$. Then the product of $(\sum_{i=1}^{n} b_i)(\sum_{i=1}^{n} a_i)^q$ can be given as $\sum_{i=1}^{n} b_i (a_i)^q + \sum_{j=1}^{q} (\sum_{i=1}^{n} a_i)^{j-1} \sum_{i=1}^{n} a_i \sum_{i' \neq i}^{n} b_{i'} (a_{i'})^{q-j}$, where $i' = 1, 2, \ldots, n$ with n and $q \in \mathbb{Z}^+$.

Proof. Let us suppose, with no loss of generality, that $n = 4$ and $q = 3$. In this scenario, we have the product $(b_1 + b_2 + b_3 + b_4) \times (a_1 + a_2 + a_3 + a_4) \times (a_1 + a_2 + a_3 + a_4) \times (a_1 + a_2 + a_3 + a_4)$. For the sake of clarity, we write the product as $(d_1 + d_2 + d_3 + d_4) \times (c_1 + c_2 + c_3 + c_4) \times (a_1 + a_2 + a_3 + a_4) \times (b_1 + b_2 + b_3 + b_4)$, where $a_i = c_i = d_i$, with i being 1, 2, 3, 4. To labour the obvious, we express $(d_1 + d_2 + d_3 + d_4) \times (c_1 + c_2 + c_3 + c_4) \times (a_1 + a_2 + a_3 + a_4) \times (b_1 + b_2 + b_3 + b_4)$ as follows

$$
\begin{aligned}
&(d_1 + d_2 + d_3 + d_4) \times (c_1 + c_2 + c_3 + c_4) \\
&\quad \times (a_1 + a_2 + a_3 + a_4) \times (b_1 + b_2 + b_3 + b_4) \\
&= b_1 a_1 c_1 d_1 + b_2 a_2 c_2 d_2 + b_3 a_3 c_3 d_3 + b_4 a_4 c_4 d_4 \\
&\quad + d_1(b_2 a_2 c_2 + b_3 a_3 c_3 + b_4 a_4 c_4) \\
&\quad + d_2(b_1 a_1 c_1 + b_3 a_3 c_3 + b_4 a_4 c_4) \\
&\quad + d_3(b_1 a_1 c_1 + b_2 a_2 c_2 + b_4 a_4 c_4) \\
&\quad + d_4(b_1 a_1 c_1 + b_2 a_2 c_2 + b_3 a_3 c_3) \\
&\quad + d_1(c_1(b_2 a_2 + b_3 a_3 + b_4 a_4) + c_2(b_1 a_1 + b_3 a_3 + b_4 a_4) \\
&\quad + c_3(b_1 a_1 + b_2 a_2 + b_4 a_4) + c_4(b_1 a_1 + b_2 a_2 + b_3 a_3)) \\
&\quad + d_2(c_1(b_2 a_2 + b_3 a_3 + b_4 a_4) + c_2(b_1 a_1 + b_3 a_3 + b_4 a_4) \\
&\quad + c_3(b_1 a_1 + b_2 a_2 + b_4 a_4) + c_4(b_1 a_1 + b_2 a_2 + b_3 a_3)) \\
&\quad + d_3(c_1(b_2 a_2 + b_3 a_3 + b_4 a_4) + c_2(b_1 a_1 + b_3 a_3 + b_4 a_4) \\
&\quad + c_3(b_1 a_1 + b_2 a_2 + b_4 a_4) + c_4(b_1 a_1 + b_2 a_2 + b_3 a_3)) \\
&\quad + d_4(c_1(b_2 a_2 + b_3 a_3 + b_4 a_4) + c_2(b_1 a_1 + b_3 a_3 + b_4 a_4) \\
&\quad + c_3(b_1 a_1 + b_2 a_2 + b_4 a_4) + c_4(b_1 a_1 + b_2 a_2 + b_3 a_3)) \\
&\quad + d_1[c_1(a_1(b_2 + b_3 + b_4) + a_2(b_1 + b_3 + b_4) \\
&\quad + a_3(b_1 + b_2 + b_4) + a_4(b_1 + b_2 + b_3)) + \ldots \\
&\quad + c_4(a_1(b_2 + b_3 + b_4) + a_2(b_1 + b_3 + b_4) \\
&\quad + a_3(b_1 + b_2 + b_4) + a_4(b_1 + b_2 + b_3))] \\
&\quad + d_2[c_1(a_1(b_2 + b_3 + b_4) + a_2(b_1 + b_3 + b_4) \\
&\quad + a_3(b_1 + b_2 + b_4) + a_4(b_1 + b_2 + b_3)) + \ldots \\
&\quad + c_4(a_1(b_2 + b_3 + b_4) + a_2(b_1 + b_3 + b_4) \\
&\quad + a_3(b_1 + b_2 + b_4) + a_4(b_1 + b_2 + b_3))] \\
&\quad + d_3[c_1(a_1(b_2 + b_3 + b_4) + a_2(b_1 + b_3 + b_4) \\
&\quad + a_3(b_1 + b_2 + b_4) + a_4(b_1 + b_2 + b_3)) + \ldots \\
&\quad + c_4(a_1(b_2 + b_3 + b_4) + a_2(b_1 + b_3 + b_4) \\
&\quad + a_3(b_1 + b_2 + b_4) + a_4(b_1 + b_2 + b_3))] \\
&\quad + d_4[c_1(a_1(b_2 + b_3 + b_4) + a_2(b_1 + b_3 + b_4) \\
&\quad + a_3(b_1 + b_2 + b_4) + a_4(b_1 + b_2 + b_3)) + \ldots \\
&\quad + c_4(a_1(b_2 + b_3 + b_4) + a_2(b_1 + b_3 + b_4) \\
&\quad + a_3(b_1 + b_2 + b_4) + a_4(b_1 + b_2 + b_3))], \qquad (1)
\end{aligned}
$$

which can be written more compactly as:

$$
\begin{aligned}
\left(\sum_{i=1}^{4} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^3 &= \sum_{i=1}^{4} b_i (a_i)^3 + \sum_{i=1}^{4} a_i \sum_{i' \neq i}^{4} b_{i'} (a_{i'})^2 \\
&\quad + \sum_{i=1}^{4} a_i \sum_{i=1}^{4} a_i \sum_{i' \neq i}^{4} b_{i'} a_{i'} \\
&\quad + \left(\sum_{i=1}^{4} a_i\right)^2 \sum_{i=1}^{4} a_i \sum_{i' \neq i}^{4} b_{i'} \qquad (2)
\end{aligned}
$$

where $i' = 1, 2, 3, 4$; note that in Eq. 2 we have made use of fact that, by definition, $a_i = c_i = d_i$. It readily follows that for the general case

$$
\begin{aligned}
\left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^q &= \sum_{i=1}^{n} b_i (a_i)^q + \sum_{i=1}^{n} a_i \sum_{i' \neq i}^{n} b_{i'} (a_{i'})^{q-1} \\
&\quad + \sum_{i=1}^{n} a_i \sum_{i=1}^{n} a_i \sum_{i' \neq i}^{n} b_{i'} (a_{i'})^{q-2} + \ldots \\
&\quad \ldots + \left(\sum_{i=1}^{n} a_i\right)^{q-1} \sum_{i=1}^{n} a_i \sum_{i' \neq i}^{n} b_{i'}, \qquad (3)
\end{aligned}
$$

or in a more compact form

$$
\begin{aligned}
\left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^q &= \sum_{i=1}^{n} b_i (a_i)^q + \sum_{j=1}^{q} \left(\sum_{i=1}^{n} a_i\right)^{j-1} \\
&\quad \times \sum_{i=1}^{n} a_i \sum_{i' \neq i}^{n} b_{i'} (a_{i'})^{q-j}, \qquad (4)
\end{aligned}
$$

which can be immediately proved by induction, see Appendix B.

This completes the proof of the proposition. □

Clearly Eq. 3 can be rearranged to give

$$
\begin{aligned}
\sum_{i=1}^{n} b_i (a_i)^q &= \left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^q - \sum_{i=1}^{n} a_i \sum_{i' \neq i}^{n} b_{i'} (a_{i'})^{q-1} \\
&\quad - \sum_{i=1}^{n} a_i \sum_{i=1}^{n} a_i \sum_{i' \neq i}^{n} b_{i'} (a_{i'})^{q-2} - \ldots \\
&\quad \ldots - \left(\sum_{i=1}^{n} a_i\right)^{q-1} \sum_{i=1}^{n} a_i \sum_{i' \neq i}^{n} b_{i'} \qquad (5)
\end{aligned}
$$

Remark 1. From Eqs. 1–3, one does not fail to observe that, if n is large (i.e. asymptotically),

$$\left(\sum_{i=1}^{n} a_i\right)^{q-1} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} > \left(\sum_{i=1}^{n} a_i\right)^{q-2} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} a_{i'} > \left(\sum_{i=1}^{n} a_i\right)^{q-3} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^2 > \left(\sum_{i=1}^{n} a_i\right)^{q-4} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^3 > \ldots > \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{q-1} > \sum_{i=1}^{n} b_i (a_i)^q \tag{6}$$

Furthermore, a closer look at the terms above also reveals that one might – albeit asymptotically – view

$$\left(\sum_{i=1}^{n} a_i\right)^{q-1} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'}, \left(\sum_{i=1}^{n} a_i\right)^{q-2} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} a_{i'}, \ldots, \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{q-1}, \sum_{i=1}^{n} b_i (a_i)^q$$

as a geometric progression whose common ratio is in the interval (0,1).

Remark 2. When n is large (that is, asymptotically), another important inference that one could glean from Eqs. 1–3 is that

$$\sum_{i'\neq 1}^{n} b_{i'} (a_{i'})^m = \sum_{i'\neq 2}^{n} b_{i'} (a_{i'})^m = \sum_{i'\neq 3}^{n} b_{i'} (a_{i'})^m = \ldots = \sum_{i'\neq n}^{n} b_{i'} (a_{i'})^m, \tag{7}$$

and even more so in our pattern recognition context whose basic credo is that all reference/prototype patterns $\mathbf{x}_i$ for a given class are similar. $i' = 1, 2, \ldots, n$ and $m = q - l$, where $l = 1, 2, \ldots, q-1$

Proposition 1 , Remarks 1 and 2 essentially form the theoretical basis for the algorithm proposed in this work.

Based on Remark 1, it is justifiable to assume that Eq. 5 can be well approximated as

$$\sum_{i=1}^{n} b_i (a_i)^q \approx \left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^{q} - \left(\sum_{i=1}^{n} a_i\right)^{q-1} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} - \left(\sum_{i=1}^{n} a_i\right)^{q-2} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} a_{i'} \tag{8}$$

Due to space constraints, practical realization and validation of Eq. 8 will be given in application journals on pattern recognition and cheminformatics.

Appendix B

Proof of Eq. 4 by induction.

Base case: when $q = 1$, from Eq. 4 we have

$$\left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^{1} = \sum_{i=1}^{n} b_i (a_i)^1 + \sum_{j=1}^{1} \left(\sum_{i=1}^{n} a_i\right)^{j-1} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{1-j} \tag{9}$$

Clearly the left hand side of the equation can be expressed as $\sum_{i=1}^{n} b_i a_i + \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'}$ and the right hand side of the equation is $\sum_{i=1}^{n} b_i a_i + \sum_i a_i \sum_{i'\neq i}^{n} b_{i'}$. Thus Eq. 9 holds for $q = 1$.

General case:

Assume that Eq. 4 is correct for some positive integer $k \in \mathbb{Z}^+$. From Eq. 4 we have

$$\left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^{k} = \sum_{i=1}^{n} b_i (a_i)^k + \sum_{j=1}^{k} \left(\sum_{i=1}^{n} a_i\right)^{j-1} \sum_{i} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{k-j} \tag{10}$$

We now show that the equation above holds also for $k+1$. Starting with the left hand side, we have

$$\left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^{k}\left(\sum_{i=1}^{n} a_i\right) = \left(\sum_{i=1}^{n} b_i (a_i)^k + \sum_{j=1}^{k} \left(\sum_{i=1}^{n} a_i\right)^{j-1} \times \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{k-j}\right)\left(\sum_{i=1}^{n} a_i\right), \tag{11}$$

where we have made use of the assumption that Eq. 10 is correct; hence

$$\left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^{k}\left(\sum_{i=1}^{n} a_i\right) = \sum_{i=1}^{n} b_i (a_i)^k \left(\sum_{i=1}^{n} a_i\right) + \sum_{j=1}^{k} \times \left(\sum_{i=1}^{n} a_i\right)^{j} \sum_{i} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{k-j}, \tag{12}$$

where $\sum_{i=1}^{n} b_i (a_i)^k \left(\sum_{i=1}^{n} a_i\right)$ can be expressed as

$$\sum_{i=1}^{n} b_i (a_i)^k \left(\sum_{i=1}^{n} a_i\right) = \sum_{i=1}^{n} b_i (a_i)^{k+1} + \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^k \tag{13}$$

Inserting Eq. 13 into Eq. 12 results in

$$\left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^{k}\left(\sum_{i=1}^{n} a_i\right) = \sum_{i=1}^{n} b_i (a_i)^{k+1} + \left[\sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^k + \sum_{j=1}^{k} \left(\sum_{i=1}^{n} a_i\right)^{j} \sum_{i} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{k-j}\right] \tag{14}$$

$[\sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^k + \sum_{j=1}^{k} (\sum_{i=1}^{n} a_i)^j \sum_i a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{k-j}]$ is basically $\sum_{j=0}^{k} (\sum_{i=1}^{n} a_i)^j \sum_i a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{k-j}$, which can be rewritten as $\sum_{j=1}^{k+1} (\sum_{i=1}^{n} a_i)^{j-1} \sum_{i=1} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{(k+1)-j}$. Inserting this expression in Eq. 14 and rewriting $(\sum_{i=1}^{n} a_i)^k (\sum_{i=1}^{n} a_i)$ on the left hand side of Eq. 14 as $(\sum_{i=1}^{n} a_i)^{k+1}$ yield

$$\left(\sum_{i=1}^{n} b_i\right)\left(\sum_{i=1}^{n} a_i\right)^{k+1} = \sum_{i=1}^{n} b_i (a_i)^{k+1} + \left[\sum_{j=1}^{k+1} \left(\sum_{i=1}^{n} a_i\right)^{j-1} \sum_{i=1}^{n} a_i \sum_{i'\neq i}^{n} b_{i'} (a_{i'})^{(k+1)-j}\right], \tag{15}$$

which is what we set out to prove. Thus, Eq. 4 (and for that matter Eq. 3) in Appendix A hold for all values of $q \in \mathbb{Z}^+$.

Appendix C

In this work: $b_i = \mathbf{x}^T(\beta_i \mathbf{x}_i)$; $b_{i'} = \mathbf{x}^T(\beta_{i'} \mathbf{x}_{i'})$; $a_i = \mathbf{x}^T \mathbf{x}_i$; $a_{i'} = \mathbf{x}^T \mathbf{x}_{i'}$; $q = r - 1$; and $n = N_{\omega_j}$, where $\mathbf{x}$, β_i, $\beta_{i'}$, $\mathbf{x}_i$, $\mathbf{x}_{i'}$, r and N_{ω_j} are

as described in the main text (see Section 3). Hence, from Eq. 8 we have

$$\sum_{i=1}^{N_{\omega_j}} (\mathbf{x}^T(\beta_i\mathbf{x}_i))(\mathbf{x}^T\mathbf{x}_i)^{r-1} \approx c^{r-1}\left(\mathbf{x}^T\sum_{i=1}^{N_{\omega_j}}(\beta_i\mathbf{x}_i)\right) - c^{r-2}\left(\sum_{i=1}^{N_{\omega_j}}\mathbf{x}^T\mathbf{x}_i\sum_{i\neq i'}^{N_{\omega_j}}\mathbf{x}^T(\beta_{i'}\mathbf{x}_{i'})\right) - c^{r-3}\left(\sum_{i=1}^{N_{\omega_j}}\mathbf{x}^T\mathbf{x}_i\sum_{i\neq i'}^{N_{\omega_j}}(\mathbf{x}^T(\beta_{i'}\mathbf{x}_{i'}))(\mathbf{x}^T\mathbf{x}_{i'})\right) \quad (16)$$

where $c = (\mathbf{x}^T\sum_{i=1}^{N_{\omega_j}}\mathbf{x}_i)$.

Making use of the two definitions $\mathbf{z}' = \sum_{i=1}^{N_{\omega_j}}(\beta_i\mathbf{x}_i)$ and $\mathbf{z} = \sum_{i=1}^{N_{\omega_j}}\mathbf{x}_i$ given in Section 3 in the main text and with some algebraic manipulation (such as the fact that $\mathbf{g}^T\mathbf{h} = \mathbf{h}^T\mathbf{b}$ for any vectors $\mathbf{g}$ and $\mathbf{h}$), the three lines on the right hand side of Eq. 16 can be recast in a more simple and familiar form:

- Line 1: $c^{r-1}(\mathbf{x}^T\sum_{i=1}^{N_{\omega_j}}(\beta_i\mathbf{x}_i)) = (\mathbf{x}^T\mathbf{z})^{r-1}\mathbf{x}^T\mathbf{z}'$
- Line 2: $c^{r-2} = (\mathbf{x}^T\mathbf{z})^{r-2}$, whereas

  $$\left(\sum_{i=1}^{N_{\omega_j}}\mathbf{x}^T\mathbf{x}_i\sum_{i\neq i'}^{N_{\omega_j}}\mathbf{x}^T(\beta_{i'}\mathbf{x}_{i'})\right) = \mathbf{x}^T\mathbf{Y}\mathbf{x},$$

  where $\mathbf{Y} = (\sum_{i=1}^{N_{\omega_j}}\mathbf{x}_i\sum_{i\neq i'}^{N_{\omega_j}}(\beta_{i'}\mathbf{x}_{i'})^T)$ is an L-by-L matrix.
- Line 3: $c^{r-3} = (\mathbf{x}^T\mathbf{z})^{r-3}$, whereas

  $$\left(\sum_{i=1}^{N_{\omega_j}}\mathbf{x}^T\mathbf{x}_i\sum_{i\neq i'}^{N_{\omega_j}}(\mathbf{x}^T(\beta_{i'}\mathbf{x}_{i'}))(\mathbf{x}^T\mathbf{x}_{i'})\right) = \mathbf{x}^T\sum_{i=1}^{N_{\omega_j}}\mathbf{x}_i\mathbf{x}^T\sum_{i\neq i'}^{N_{\omega_j}}\left((\beta_{i'}\mathbf{x}_{i'})\mathbf{x}_{i'}^T\right)\mathbf{x} = \mathbf{x}^T\sum_{i=1}^{N_{\omega_j}}\mathbf{x}_i(\mathbf{x}^T\mathbf{S}_i\mathbf{x}), \quad (17)$$

  where $\mathbf{S}_i = \sum_{i\neq i'}^{N_{\omega_j}}((\beta_{i'}\mathbf{x}_{i'})\mathbf{x}_{i'}^T)$ is an L-by-L matrix.
  According to Eq. 7, the matrices $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_{N_{\omega_j}}$ can be considered equivalent/similar. Hence, Eq. 17 modifies to

  $$\left(\sum_{i=1}^{N_{\omega_j}}\mathbf{x}^T\mathbf{x}_i\sum_{i\neq i'}^{N_{\omega_j}}(\mathbf{x}^T(\beta_{i'}\mathbf{x}_{i'}))(\mathbf{x}^T\mathbf{x}_{i'})\right) = (\mathbf{x}^T\mathbf{z})(\mathbf{x}^T\mathbf{S}\mathbf{x}), \quad (18)$$

  where $\mathbf{S} = \mathbf{S}_i$ and $(\mathbf{x}^T\mathbf{z}) = \mathbf{x}^T\sum_{i=1}^{N_{\omega_j}}\mathbf{x}_i$

Expressed in terms of $\mathbf{Y}$, $\mathbf{S}$, $\mathbf{z}'$ and $\mathbf{z}$, Eq. 16 becomes

$$\begin{aligned}\sum_{i=1}^{N_{\omega_j}}(\mathbf{x}^T(\beta_i\mathbf{x}_i))(\mathbf{x}^T\mathbf{x}_i)^{r-1} &\approx (\mathbf{x}^T\mathbf{z})^{r-1}(\mathbf{x}^T\mathbf{z}') - (\mathbf{x}^T\mathbf{z})^{r-2}(\mathbf{x}^T(\mathbf{Y}+\mathbf{S})\mathbf{x}) \\ &= (\mathbf{x}^T\mathbf{z})^{r-2}((\mathbf{x}^T\mathbf{z}')(\mathbf{x}^T\mathbf{z}) - \mathbf{x}^T(\mathbf{Y}+\mathbf{S})\mathbf{x}) \\ &= (\mathbf{x}^T\mathbf{z})^{r-2}((\mathbf{x}^T\mathbf{z}'\mathbf{z}^T\mathbf{x}) - \mathbf{x}^T(\mathbf{Y}+\mathbf{S})\mathbf{x}) \\ &= (\mathbf{x}^T\mathbf{z})^{r-2}(\mathbf{x}^T\mathbf{Q}\mathbf{x}), \end{aligned} \quad (19)$$

where $\mathbf{Q} = \mathbf{z}'\mathbf{z}^T - (\mathbf{Y}+\mathbf{S})$, an L-by-L matrix.

Note that $(\mathbf{x}^T\mathbf{z})^{r-2} \geq 0$, $(\mathbf{x}^T\mathbf{z}'\mathbf{z}^T\mathbf{x}) \geq \mathbf{x}^T(\mathbf{Y}+\mathbf{S})\mathbf{x}$; hence, $(\mathbf{x}^T\mathbf{z})^{r-2}(\mathbf{x}^T\mathbf{Q}\mathbf{x}) \geq 0$ as it should be – because, by definition, $\sum_{i=1}^{N_{\omega_j}}(\mathbf{x}^T(\beta_i\mathbf{x}_i))(\mathbf{x}^T\mathbf{x}_i)^{r-1} \geq 0$.

References

[1] R. Duda, P.E. Hart, D.G. Stork, Pattern Classification, Wiley, New York, NY, USA, 2000.
[2] T.Y. Young, T. Calvert, Classification, Estimation, Pattern Recognition, American Elsevier, New York, NY, USA, 1974.
[3] B.D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge, UK, 1996.
[4] D.J. Hand, Discriminant and Classification, Wiley, New York, NY, USA, 1981.
[5] A.R. Webb, Statistical Pattern Recognition, Wiley, Chicester, UK, 2002.
[6] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, New York, NY, USA, 2006.
[7] K. Koch, Introduction to Bayesian Statistics, Springer–Verlag, Berlin, Germany, 2007.
[8] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, Cambridge, MA, USA, 2012.
[9] E. Parzen, On estimation of a probability density function and mode, Ann. Math. Stat. 33 (1962) 1065–1076.
[10] W.S. Meisel, Computer-oriented Approaches to Pattern Recognition, Academic Press, New York, NY, USA, 1972.
[11] S. Fiori, Probability density function learning by unsupervised neurons, Int. J. Neural Syst. 11 (2001) 399–417.
[12] S. Fiori, Non-symmetric pdf estimation by artificial neurons: application to statistical characterization of reinforced composites, IEEE Trans. Neural Netw. 14 (2003) 959–962.
[13] D.P. Specht, Generation of polynomial discriminant functions for pattern recognition, IEEE Trans. Electron Comput. 16 (1967) 308–319.
[14] H.C. Andrews, Mathematical Techniques in Pattern Recognition, Wiley, New York, NY, USA, 1972.
[15] G. Harper, J. Bradshaw, J.C. Gittins, D.V.S. Green, A.R. Leach, Prediction of biological activity for high-throughput screening using binary kernel discrimination, J. Chem. Inf. Comp. Sci. 41 (2001) 1295–1300.
[16] R. Lowe, H.Y. Mussa, J.B.O. Mitchell, R.C. Glen, Classifying molecules using a sparse probabilistic kernel binary classifier, J. Chem. Inf. Model. 51 (2011) 1539–1544.
[17] J. Aitchison, C.G.G. Aitken, Multivariate binary discrimination by the kernel method, Biometrika 63 (1976) 413–420.
[18] H.Y. Mussa, The Aitchison and Aitken kernel function revisited, J. Math. Res. 5 (2013) 22–25.