Seismic facies classification with parametric and nonparametric statistics

Brian Russell[1]

[1] HampsonRussell, A GeoSoftware Company, Calgary, Alberta, brian.russell-contractor@cgg.com

ABSTRACT

In this study, I compare parametric and nonparametric statistical methods for seismic facies classification. I will use an example that involves two seismic attributes, two facies and ten points. These seismic attributes are the normalized V_P/V_S ratio and the acoustic impedance, I_P. This problem includes several outliers and therefore illustrates the advantages and disadvantages of each method discussed. The classification techniques covered in this study consist of linear least-squares, k-nearest-neighbours (kNN), Quadratic Discrimination Analysis (QDA), Kernel Density Estimation (KDE) and the Deep Feedforward Neural Network (DFNN).

INTRODUCTION

This study will apply several basic and advanced classification methods to a ten-point facies classification example (Bishop, 2006, Hastie et al., 2009). Before applying advanced statistical methods of classification to the ten-point problem, I first apply two simpler methods: linear least-squares, which is a parametric approach that only requires two weight parameters, and k-Nearest Neighbor, or kNN, classification, which is a nonparametric approach which requires a separate calculation at each point on the map.

.

I then move to statistical classification using two different approaches: quadratic discrimination analysis (QDA) using the bivariate normal Gaussian distribution and kernel density estimation (KDE) using a Gaussian kernel. The underlying approach to each of these methods is Bayesian, in that the Bayes' decision rule is used to determine the membership in each class.

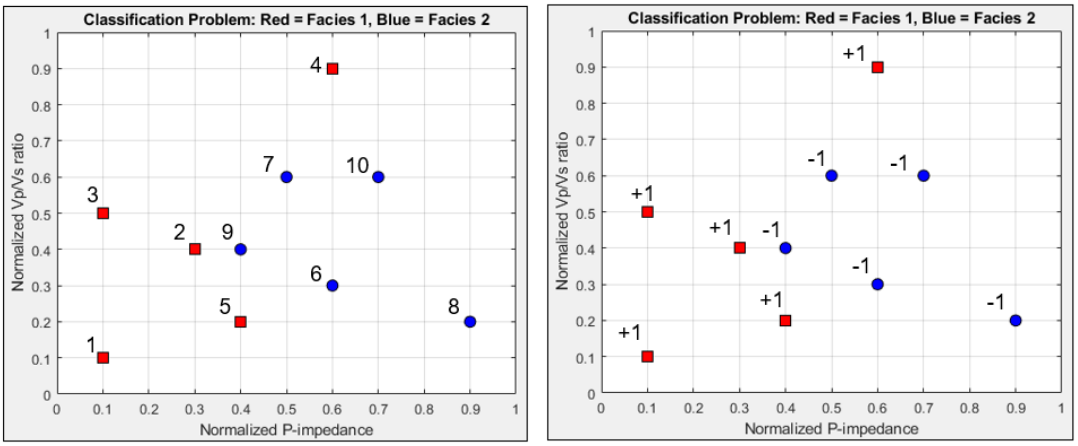
QDA classification with the normal distribution is called a parametric statistical approach since it uses a limited set of parameters to perform the classification (i.e., the mean, variance, and covariance of each class). KDE is similar except that it applies a Gaussian distribution to each point in the two classes and sums the result. Although KDE is dependent on the width of a scaling parameter, it is called a non-parametric statistical method because it is dependent on all the points in each class.

Finally, I show how a deep feedforward neural network (DFNN) can be used to solve the same ten-point classification problem. After defining the structure of the neural network, I will do a detailed analysis of how the network solves the classification problem. Although the DFNN is generally not considered to be a statistical technique, it falls more into the parametric side of classification than the nonparametric side, since the final trained network consist of a set of linear weights. The Bayes' rule is also used to compute the decision boundary for the DFNN.

SYNTHETIC EXAMPLE

Figure 1(a) shows a simple facies classification problem, where the horizontal axis is normalized P-impedance (I_P), and the vertical axis is normalized V_P/V_S ratio. The red squares represent facies 1 and the blue circles represent facies 2, ordering of the input points is shown in the figure, where the values are as follows:

$$\boldsymbol{x}_1 = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}, \boldsymbol{x}_2 = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix}, \boldsymbol{x}_3 = \begin{bmatrix} 0.1 \\ 0.5 \end{bmatrix}, \boldsymbol{x}_4 = \begin{bmatrix} 0.6 \\ 0.9 \end{bmatrix}, \boldsymbol{x}_5 = \begin{bmatrix} 0.4 \\ 0.2 \end{bmatrix},$$
$$\boldsymbol{x}_6 = \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix}, \boldsymbol{x}_7 = \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix}, \boldsymbol{x}_8 = \begin{bmatrix} 0.9 \\ 0.2 \end{bmatrix}, \boldsymbol{x}_9 = \begin{bmatrix} 0.4 \\ 0.4 \end{bmatrix}, \boldsymbol{x}_{10} = \begin{bmatrix} 0.7 \\ 0.6 \end{bmatrix}.$$



(a) *(b)*

Figure 1: The synthetic Vp/Vs ratio versus P-impedance dataset used to study the classification algorithms, where (a) shows the order of the input points, where the red squares represent one facies and the blue circles represent the second facies, and (b) shows the facies labels, +1 and -1.

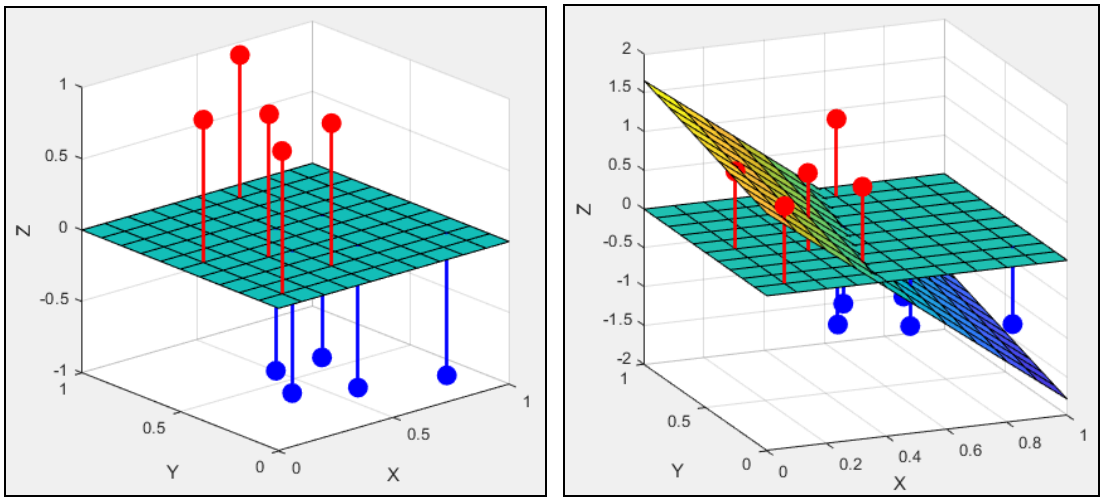
Notice that we can also arrange these points into two vectors that contain the *x* and *y* values separately, given by

$$\boldsymbol{x}^T = [0.1, 0.3, 0.1, 0.6, 0.4, 0.6, 0.5, 0.9, 0.4, 0.7], \text{ and}$$
$$\boldsymbol{y}^T = [0.1, 0.4, 0.5, 0.9, 0.2, 0.3, 0.6, 0.2, 0.4, 0.4].$$

In this case, classification is a binary problem (there are only two possibilities) so we can implement the linear approach by assigning +1 to the red points and -1 to the blue points, as shown here. We can therefore introduce a third dimension, which can be written as *z*, with points given by:

$$z_1 = +1,\ z_2 = +1,\ z_3 = +1,\ z_4 = +1,\ z_5 = +1, z_6 = -1,\ z_7 = -1,\ z_8 = -1,\ z_9 = -1,\ z_{10} = -1.$$

We can visualize this in three dimensions as shown in Figure 2(a). By looking at this plot, we see that linear classification is identical to linear regression in one higher dimension, albeit with only two sets of equal values. In three dimensions this is an easy problem to solve using the $z = 0$ plane, shown in green. In two dimensions, we want to find the projection of a dipping 3D plane onto the X-Y plane.



(a) (b)

Figure 2: The 3D classification problem, where (a) shows that the z = 0 plane performs separation in three dimensions, and (b) shows the linear separation plane projected onto the z = 0 plane.

To perform this classification, we therefore find the three weights for the best 3D separation plane, as follows:

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = (X^T X)^{-1} X^T \boldsymbol{z} = \begin{bmatrix} 1.03 \\ -1.43 \\ 0.30 \end{bmatrix}, \tag{1}$$

where $X = \begin{bmatrix} \mathbf{1} & \boldsymbol{x} & \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} 1 & 0.1 & 0.1 \\ 1 & 0.3 & 0.4 \\ 1 & 0.1 & 0.5 \\ 1 & 0.6 & 0.9 \\ 1 & 0.4 & 0.2 \\ 1 & 0.6 & 0.3 \\ 1 & 0.5 & 0.6 \\ 1 & 0.9 & 0.2 \\ 1 & 0.4 & 0.4 \\ 1 & 0.7 & 0.6 \end{bmatrix}$, $\mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ $\boldsymbol{x} = \begin{bmatrix} 0.1 \\ 0.6 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$, $\boldsymbol{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$ and $\boldsymbol{z} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$.

The equation for this plane is therefore given as

$$\hat{\boldsymbol{z}} = w_0 + w_1 \boldsymbol{x} + w_2 \boldsymbol{y} = 1.03 - 1.43 \boldsymbol{x} + 0.3 \boldsymbol{y}\,. \tag{2}$$

Figure 2(b) shows the linear separation plane between the two facies. In this 3D view it is hard to see how successful this classification line has been, so we will next go back to the 2D view.

We can find the equation for the line by setting equation 2 equal to 0, giving:

$$y = -\frac{w_0}{w_2} - \frac{w_1}{w_2}\boldsymbol{x} = -1.78 + 4.78x\,. \tag{3}$$

The separation boundary is shown in Figure 3. One of the first facies values has been miss-classified, as has one of the second facies values. Also, one of the second facies values is on the separation boundary.

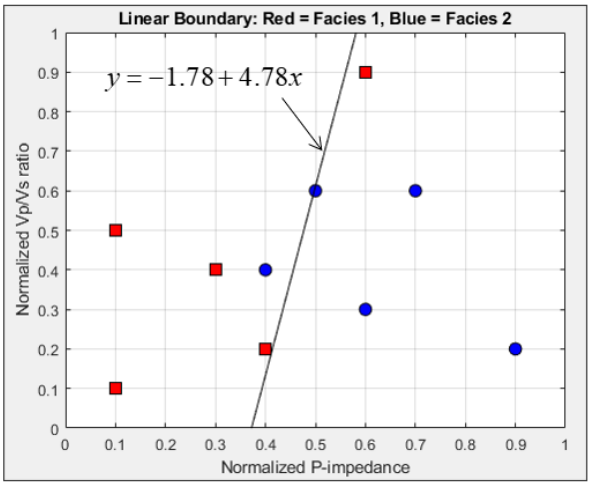


Figure 3: The linear classification solution in 2D.

Next, we will look at the simplest non-parametric method, the *k*-nearest neighbors, or kNN, approach. Hastie et al. (2009) state that kNN is at "the other end of the spectrum" from the linear model just discussed. This approach simply finds the *k* nearest input values $\boldsymbol{x}_i$ that fall in a neighborhood $N_k(x)$ around each point $\boldsymbol{x} = [x, y]^T = [I_P\ ,\ V_P/V_S]^T$ on the graph, and then averages these values. In regression, this is called a *k*-point smoother, and in the classification problem it is like a smoother, but the difference is that the output value is set to -1 if the average is less than 0 and the +1 if the average is greater than or equal to 0.

Mathematically, we can write:

$$z(\boldsymbol{x}) = \frac{1}{k}\sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} \boldsymbol{x}_i = \begin{cases} -1 \text{ if } z(\boldsymbol{x}) < 0 \\ +1 \text{ if } z(\boldsymbol{x}) \geq 0 \end{cases}. \tag{4}$$

Figure 4 shows the *k*-nearest neighbors (kNN) boundary for a value of $k = 1$. Notice that the facies have been correctly classified but that the boundary is quite "jagged". Later, we will see that we can get a similar, but much smoother, version of this boundary using the kernel density estimation approach.

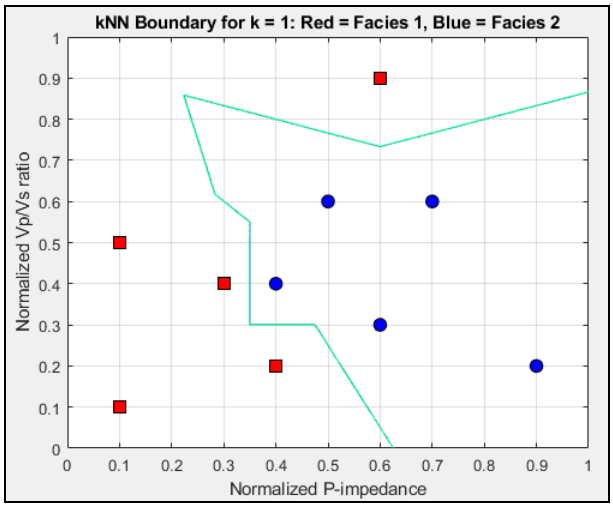
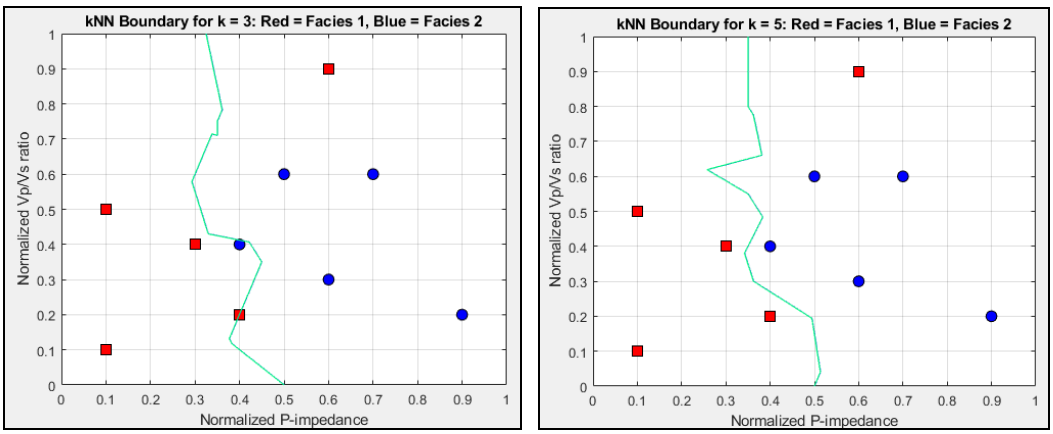


Figure 4: The kNN approach using k = 1.

Although k = 1 has done a perfect classification, figure 5 shows kNN classification for values of 3 and 5, which do not do as good a job. However, k = 5 has only miss-classified one point, better than the linear approach.



(a) *(b)*

Figure 5: The kNN approach using (a) k = 3 and (b) k = 5.

STATISTICAL CLASSIFICATION

Having discussed the parametric approach of linear least-squares classification and the nonparametric approach of classification with kNN, I will now move on to more advanced statistical classification methods using two different approaches. The first approach is Quadratic Discriminant Analysis (QDA) using the bivariate normal Gaussian distribution. QDA classification with the normal distribution is called a parametric statistical approach since it uses a limited set of parameters to perform the classification (i.e., the mean, variance, and covariance of each class). The second approach is called kernel density estimation (KDE) using the Gaussian distribution as its kernel (although other types of kernels, such as the Epanechnikov kernel, can be used). KDE is called the Probabilistic Neural Network in machine learning. KDE, or

PNN, is called a non-parametric statistical method since they cannot be defined by a finite set of parameters. Both the QDA and KDE methods are Bayesian since the Bayes' decision rule is used to determine the membership in each class.

QUADRATIC DISCRIMINATION ANALYSIS

The bivariate normal ellipse for a set of input points is written as follows, where ***m*** is the mean and S is the covariance:

$$f(x,y) = \frac{1}{2\pi|\sum|^{1/2}} \exp\left[-\left(\boldsymbol{x}-\boldsymbol{\mu}\right)^T \Sigma^{-1}\left(\boldsymbol{x}-\boldsymbol{\mu}\right)\right], \tag{5}$$

where $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \boldsymbol{\mu} = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}$, and $\Sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{bmatrix}$.

Figure 6 shows Gaussian ellipses for all the points in our example, where:

$$\boldsymbol{\mu} = \begin{bmatrix} 0.46 \\ 0.42 \end{bmatrix}, \text{ and } \Sigma = \begin{bmatrix} 0.065 & 0.012 \\ 0.012 & 0.057 \end{bmatrix}.$$

The ellipses are at three constant variances and the x shows the mean value.

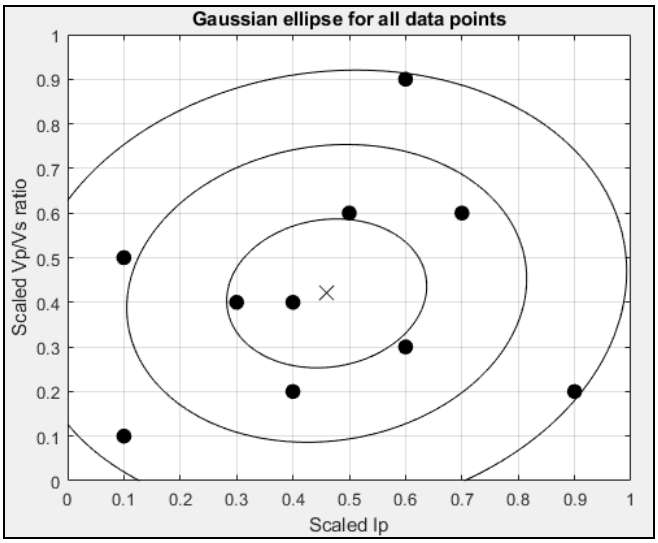


Figure 6: Bivariate Gaussian contours at three constant variances for the complete dataset shown in Figure 1, where the x shows the mean value of all the points.

Bayes' Theorem can used to calculate the probability we have a particular class c_i, given a pair of seismic attributes, $x = I_P$ and $y = V_P/V_S$, and is written:

$$p\left(c_i \mid x,y\right) = \frac{p(x,y \mid c_i)\cdot p(c_i)}{p(x,y)}, \tag{6}$$

where $p(x, y) = \sum_{i=1}^{N} p(x, y \mid c_i) \cdot p(c_i)$ for N classes.

In equation 6, $p(c_i|x,y)$ is the conditional probability of having class c_i given attributes x and y, and is called the **posterior**, $p(x,y|c_i)$ is the conditional probability of having attributes x and y given class c_i, and is called the **likelihood**, $p(x,y)$ is the joint probability of having attributes x and y , and is called the **evidence**, and $p(c_i)$ is the probability of class c_i and is called the **prior**. Note that the evidence $p(x,y)$ is the same for each class and is therefore simply a scale term, so for our two-class problem Bayes' Theorem can be written:

$$\text{If } p(x, y \mid c_1) \cdot p(c_1) \geq p(x, y \mid c_2) \cdot p(c_2), \text{ point } (x, y) \text{ is in facies 1,}$$
$$\text{but if } p(X \mid c_1) \cdot p(c_1) < p(X \mid c_2) \cdot p(c_2), \text{point } (x, y) \text{ is in facies 2.}$$

We therefore need to find both the likelihood and prior for each class. The prior is usually computed by finding the proportion of points in each class, and since we have equal numbers in this example (5 in each class) we can set the prior to ½ for each class, so that only the likelihood is needed in the calculation. The likelihood for each class is given by the modified Gaussian shown below, where each class has its own mean and covariance matrix:

$$p(x, y \mid c_i) = \frac{1}{2\pi \left|\sum_i\right|^{1/2}} \exp\left[-\frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{\mu}_i\right)^T \sum_i^{-1} \left(\boldsymbol{x} - \boldsymbol{\mu}_i\right)\right], \tag{7}$$

where $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \boldsymbol{\mu}_i = \begin{bmatrix} \mu_{ix} \\ \mu_{iy} \end{bmatrix}$, and $\sum_i = \begin{bmatrix} \sigma_{ixx} & \sigma_{ixy} \\ \sigma_{ixy} & \sigma_{ixy} \end{bmatrix}$.

Figure 7(a) shows the contoured bivariate normal ellipses of the two facies, with mean and covariance of

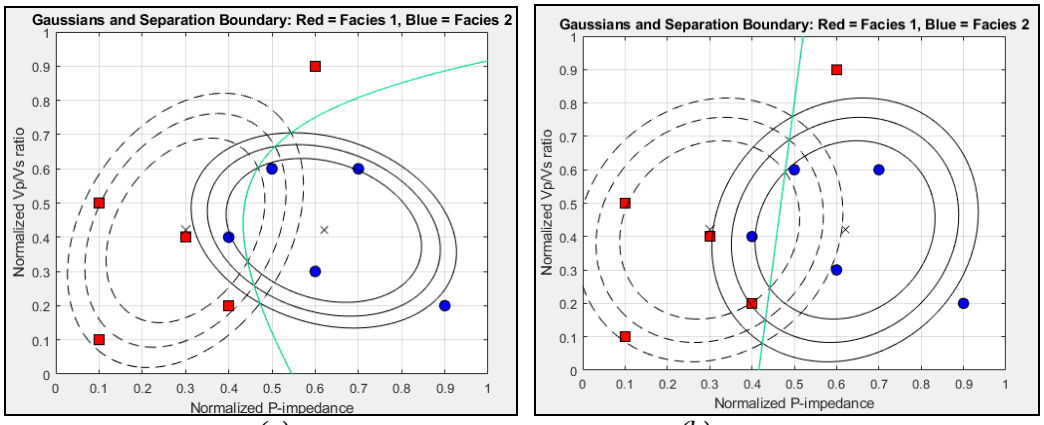
$$\boldsymbol{\mu}_1 = \begin{bmatrix} 0.30 \\ 0.42 \end{bmatrix} \text{ and } \sum_1 = \begin{bmatrix} 0.045 & 0.043 \\ 0.043 & 0.097 \end{bmatrix}$$

for the first (dashed) ellipses, and mean and covariance of

$$\boldsymbol{\mu}_2 = \begin{bmatrix} 0.62 \\ 0.42 \end{bmatrix} \text{ and } \sum_2 = \begin{bmatrix} 0.037 & -0.016 \\ -0.016 & 0.032 \end{bmatrix}$$

for the second (solid) ellipses.

In figure 7(a) the separation boundary (shown as the green line) is a quadratic function that can be computed analytically. Note that one of the second facies values has been miss-classified. In Figure 7(b) the covariance matrices of the two facies have been set to the average of the two individual covariances. The separation boundary is now linear (called linear discriminant analysis or LDA). Also, note that it is very close to the result obtained by linear classification. However, it would be rare for two arbitrary covariances to be equal, and the correct approach is to compute each independently as shown in Figure 7(a).



(a) *(b)*

Figure 7: Quadratic Discrimination Analysis (QDA) for our ten-point dataset, where (a) uses the separate covariance matrices for each facies, and (b) uses the average covariance matrix, which reduces the result to Linear Discrimination Analysis (LDA).

KERNEL DENSITY ESTIMATION

In this section, I will discuss kernel density estimation, or the probabilistic neural network (PNN), as it is called in machine learning. The kernel density estimation (KDE) method finds a continuous density function using the following equation:

$$p(\boldsymbol{x}) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right), \tag{8}$$

where $\boldsymbol{x}_i$ = the N observed values, h = a smoothing parameter, and K = a kernel function. Although there are many choices for the kernel function the most common one, and the one I will use here, is the Gaussian kernel, which is computed separately for each facies as follows:

$$p(x, y \mid c_i) = \frac{1}{2N\sigma^2}\sum_{j=1}^{N_{c_i}} \exp\left[-\frac{(x-x_{ij})^2+(y-y_{ij})^2}{2\sigma^2}\right]. \tag{9}$$

Recall Bayes' rule for our two-class problem:

$$\text{If } p(c_1)\cdot p(x, y \mid c_1) \geq p(c_2)\cdot p(x, y \mid c_2), \text{ point } (x, y) \text{ is in facies 1,}$$
$$\text{but if } p(c_1)\cdot p(X \mid c_1) < p(c_2)\cdot p(X \mid c_2), \text{point } (x, y) \text{ is in facies 2.}$$

Bayes' rule can therefore also be applied to the KDE problem as follows where, since we have the same number of points in both facies, we can drop both the scaling and the prior. The first facies is defined by the relationship:

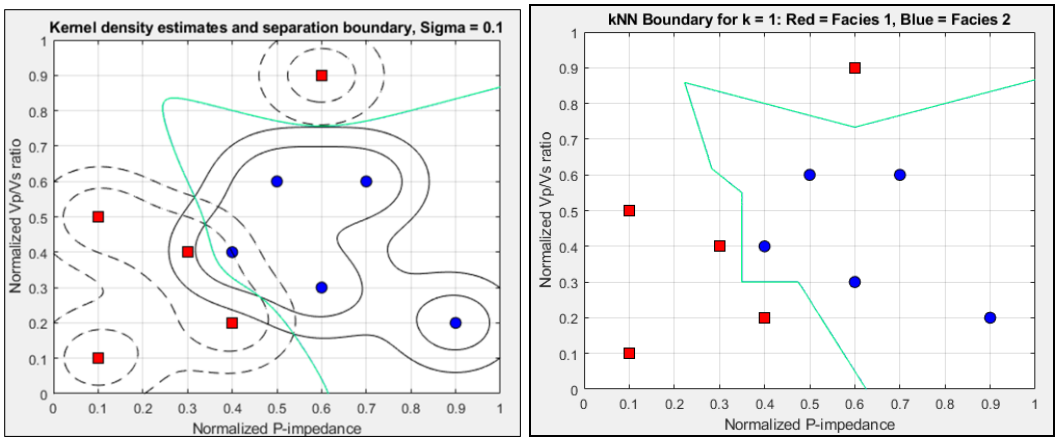
$$\sum_{j=1}^{N_{c1}} \exp\left[-\frac{(x-x_{1j})^2+(y-y_{1j})^2}{2\sigma^2}\right] \geq \sum_{j=1}^{N_{c2}} \exp\left[-\frac{(x-x_{2j})^2+(y-y_{2j})^2}{2\sigma^2}\right], \tag{10}$$

and the second facies is defined by the relationship:

$$\sum_{j=1}^{N_{c_1}} \exp\left[-\frac{(x-x_{1j})^2+(y-y_{1j})^2}{2\sigma^2}\right] < \sum_{j=1}^{N_{c_2}} \exp\left[-\frac{(x-x_{2j})^2+(y-y_{2j})^2}{2\sigma^2}\right]. \qquad (11)$$

But now there is no analytical solution, as with parametric statistics.

Figure 8(a) shows the contoured kernel density estimates of the two facies using a sigma value of 0.1. Facies 1 is shown with the dashed boundaries and facies 2 with the solid boundaries. Note the irregular shape of the contours. Also shown is the separation boundary between the kernel density estimates, which must be computed point by point. For comparison, figure 8(b) shows the kNN separation boundary using k = 1, which was shown earlier in figure 4. Note that the KDE estimate looks like a smoothed version of the kNN boundary.



(b) *(b)*

Figure 8: (a) Kernel Density Estimation (KDE) using σ = 0.1 for our ten-point dataset compared with (b) the kNN result from Figure 4, where we note that both methods have correctly classified the points, but KDE has done it in a smoother way.

Figure 9 shows the kernel density estimates and separation boundaries for sigma values of 0.3 (Figure 9(a)) and 0.05 (Figure 9(b)).

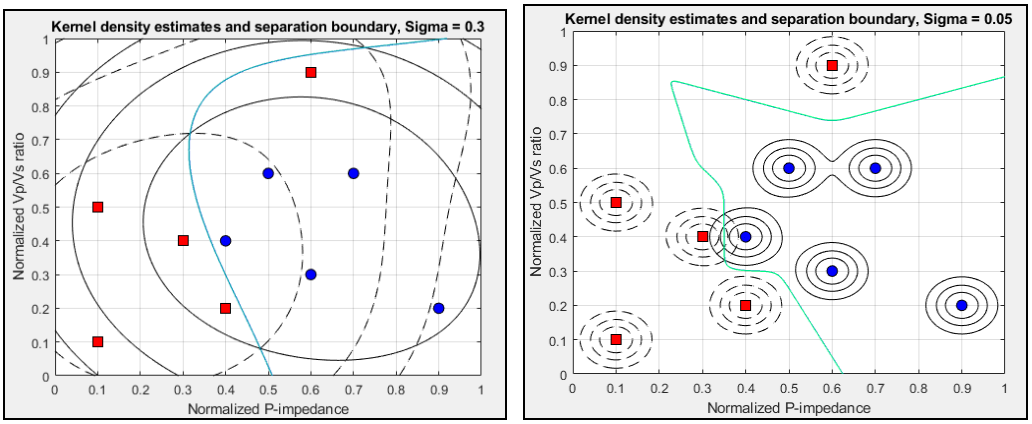


Figure 9: (a) Kernel Density Estimation (KDE) using σ = 0.5 and (b) (a) Kernel Density Estimation (KDE) using σ = 0.05.

Notice in Figure 9(a) that the sigma value is too large, and the KDE result is under-trained, resulting in miss-classified points. in Figure 9(b) that the sigma value is too small, and the KDE result is over-trained. That is, although the points have been well-classified, the resulting boundary and contours are too detailed when compared with the result shown in Figure 8 using a $\sigma = 0.1$.

CLASSIFICATION WITH A DEEP NEURAL NETWORK

Next, I will show how to perform facies classification with the four layer deep neural network shown in Figure 10. My example is taken from the paper by Higham and Higham (2019), in which they use the deep neural network shown to compute a separation boundary, but assumed the inputs were wet or dry wells. Their problem is the same one that I have adapted to the facies classification problem. Therefore, I modified the two inputs to be $x_1 = I_P$ and $x_2 = V_P/V_S$, but kept two outputs as vectors with the following binary values:

$$\boldsymbol{d} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow \text{Facies 1}, \quad \boldsymbol{d} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow \text{Facies 2}.$$

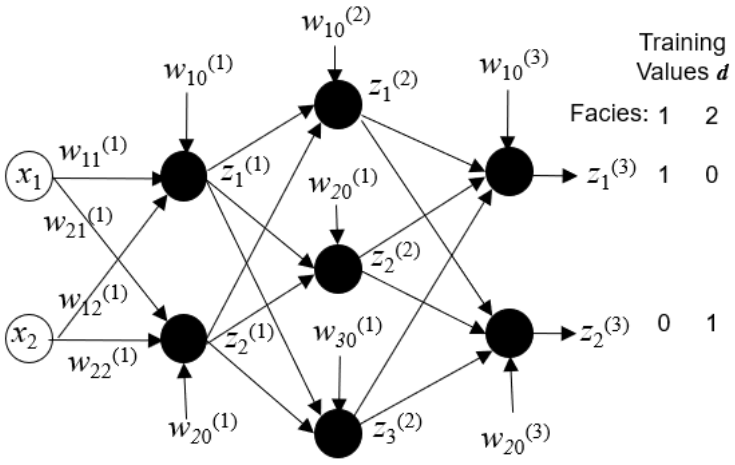


Figure 10: The four-layer Deep Feedforward Neural Network (DFNN) used to solve our ten-point classification problem, which consists of an input layer and two hidden layers with 3 and 2 neurons, respectively, and an output layer.

The neural network shown in Figure 10 is called a four-layer network since it has an input layer with two input, two "hidden" layers with two and three neurons (the intermediate black circles), respectively, and an output layer with two outputs (the final two black circles). The connections between the neurons are linear weights that are computed by a backpropagation algorithm. Although the network in Figure 10 looks intimidating at first, we can break it into it's component parts by looking at each neuron (the black circles) in more detail.

Figure 11 shows the top left neuron without the second subscript (which indicates the neuron number) or superscripts (which indicates the layer number). First, the neuron computes a weighted sum of the inputs, where w_0 is called the

bias. Second, it applies a nonlinear function to the weighted sum, where in this case we are using the logistic function (many other functions are possible).

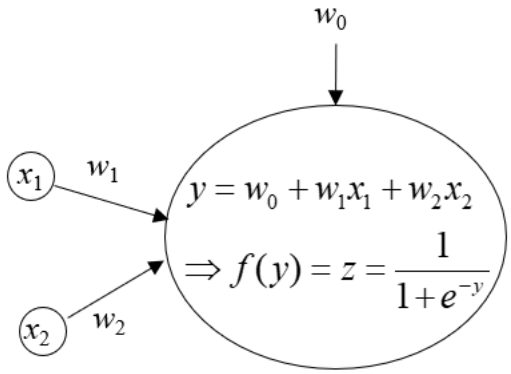


Figure 11: The neurons shown in Figure 10 by the black circles accept the weighted input and transform it using the logistic function given by f(y).

Figure 12 shows a graph of the logistic function defined in Figure 11 and in equation 12.

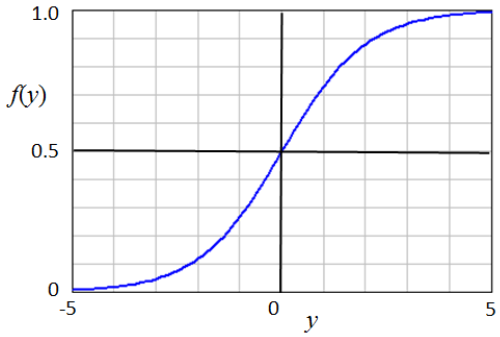


Figure 12: The logistic function defined in Figure 11 and equation 12.

The detailed mathematical formulation of this neural network is as follows:

$$\boldsymbol{z}^{(3)} = f(W^{(3)}(f(W^{(2)} f(W^{(1)} \boldsymbol{x}))), \tag{12}$$

where

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}, W^{(1)} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix}, \; f(W^{(1)}\boldsymbol{x}) = \begin{bmatrix} 1 \\ z_1^{(1)} \\ z_2^{(1)} \end{bmatrix},$$

$$W^{(2)} = \begin{bmatrix} w_{10}^{(2)} & w_{11}^{(2)} & w_{12}^{(2)} \\ w_{20}^{(2)} & w_{21}^{(2)} & w_{22}^{(2)} \\ w_{30}^{(2)} & w_{31}^{(2)} & w_{32}^{(2)} \end{bmatrix}, \; f(W^{(2)} f(W^{(1)}\boldsymbol{x})) = \begin{bmatrix} 1 \\ z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix},$$

$$W^{(3)} = \begin{bmatrix} w_{10}^{(3)} & w_{11}^{(3)} & w_{12}^{(3)} & w_{13}^{(3)} \\ w_{20}^{(3)} & w_{21}^{(3)} & w_{22}^{(3)} & w_{23}^{(3)} \end{bmatrix}, \; \boldsymbol{z}^{(3)} = \begin{bmatrix} z_1^{(3)} \\ z_2^{(3)} \end{bmatrix}, \text{ and } f(y) = \frac{1}{1+e^{-y}}.$$

Note that the function used in this neural network is called the logistic function, which is sigmoidal, or S-shaped function, and is shown in Figure 12.

Since the output has two values, the decision boundary becomes:

$$\text{If } z_1^{(3)} \geq z_2^{(3)}, \text{ the output = 1, but if } z_1^{(3)} < z_2^{(3)}, \text{ the output = 0.}$$

The weights are then computed by the backpropagation technique (Higham and Higham, 2019), which minimizes the least-squares error for the computed weights. The convergence of the error for this problem, called a cost function, is shown in Figure 13.

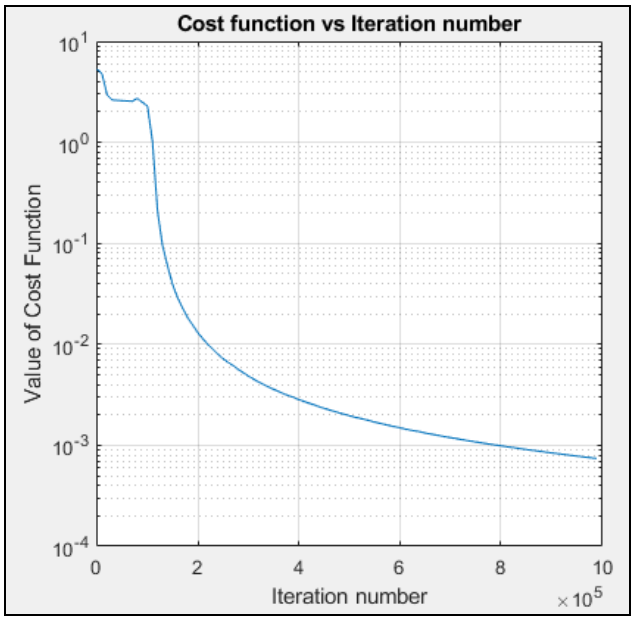


Figure 13: The cost function, or error convergence, for the neural network versus iteration number.

After 10^5 iterations, the final weights are:

$$W^{(1)} = \begin{bmatrix} -5.51 & -1.66 & 9.17 \\ 6.01 & -10.8 & -6.89 \end{bmatrix}, \; W^{(2)} = \begin{bmatrix} 2.54 & -3.99 & -6.19 \\ -4.08 & 7.00 & 9.34 \\ 2.16 & -3.27 & -5.44 \end{bmatrix}, \text{ and}$$

$$W^{(3)} = \begin{bmatrix} 0.12 & -5.53 & 9.61 & -4.85 \\ -0.46 & 5.95 & -9.34 & 4.85 \end{bmatrix}.$$

Although computing the weights is done by backpropagation, the output values are computed as the forward set of mathematical operations given by:

$$\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \end{bmatrix} = \begin{bmatrix} (1+\exp(5.51+1.16x_1 - 9.17x_2))^{-1} \\ (1+\exp(-6.01+10.81x_1 + 6.89x_2))^{-1} \end{bmatrix},$$

$$\Rightarrow \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \begin{bmatrix} (1+\exp(-2.54+3.99z_1^{(1)}+6.19z_2^{(1)}))^{-1} \\ (1+\exp(4.08-7.00z_1^{(1)}-9.34z_2^{(1)}))^{-1} \\ (1+\exp(-2.16+3.27z_1^{(1)}+5.44z_2^{(1)}))^{-1} \end{bmatrix},$$

$$\Rightarrow \begin{bmatrix} z_1^{(3)} \\ z_2^{(3)} \end{bmatrix} = \begin{bmatrix} (1+\exp(-0.12+5.53z_1^{(2)}-9.61z_2^{(2)}+4.85z_3^{(2)}))^{-1} \\ (1+\exp(0.46-5.95z_1^{(2)}+9.34z_2^{(2)}-4.85z_3^{(2)}))^{-1} \end{bmatrix}.$$

For this reason, this type of neural network can be described by two different names, which at first appear contradictory: the backpropagation neural network (based on how the weights are computed). Or the feedforward neural network (based on how the weights are applied).

Using the Bayes' rule defined earlier, we set the value on the graph to 1 or 0 depending on the output values, or:

$$z_1^{(3)} \geq z_2^{(3)} \Rightarrow 1, \text{ and } z_1^{(3)} < z_2^{(3)} \Rightarrow 0.$$

Contouring the decision surface gives the boundary shown in Figure 14(a), which is probably the most optimal boundary computed so far. However, it is very dependent on the design of the network. Instead of finding the exact cutoff between the predicted points (i.e., 0 in the case of +1 and -1 and 0.5 in the case of 0 and 1) we can also look at the actual value of $z_1^{(3)}$ or $z_2^{(3)}$, which will look like a sum of logistic functions. The contours of $z_1^{(3)}$ are shown in Figure 14(b) and we see that the boundary is steeper where the points are closer together.

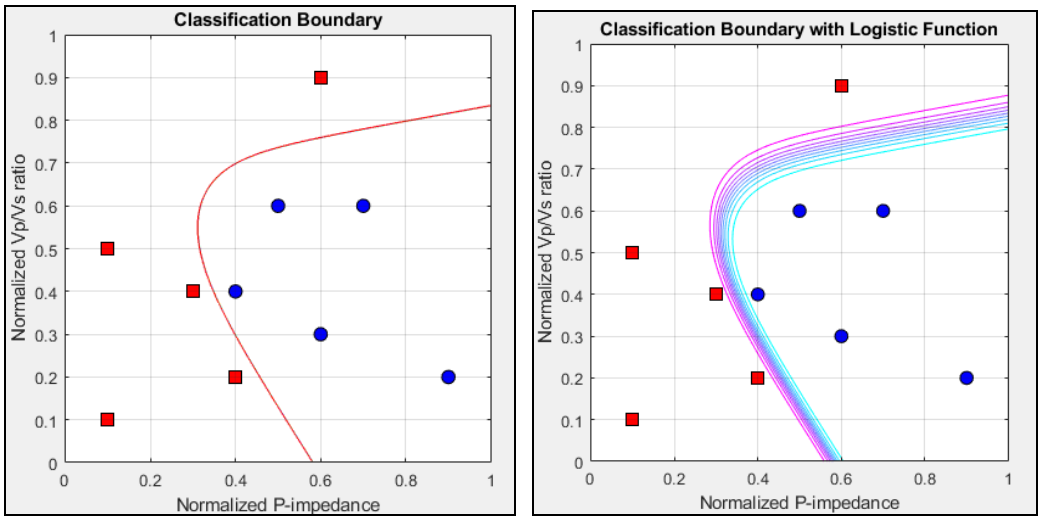


Figure 14: The separation boundary from the neural network approach, where (a) shows the exact boundary and (b) shows the logistic function contours.

The 3D contours of $z_1^{(3)}$ are shown in Figure 15 and again we see that the boundary is steeper where the points are closer together. This plot has been re-scaled to be between -1 and +1.

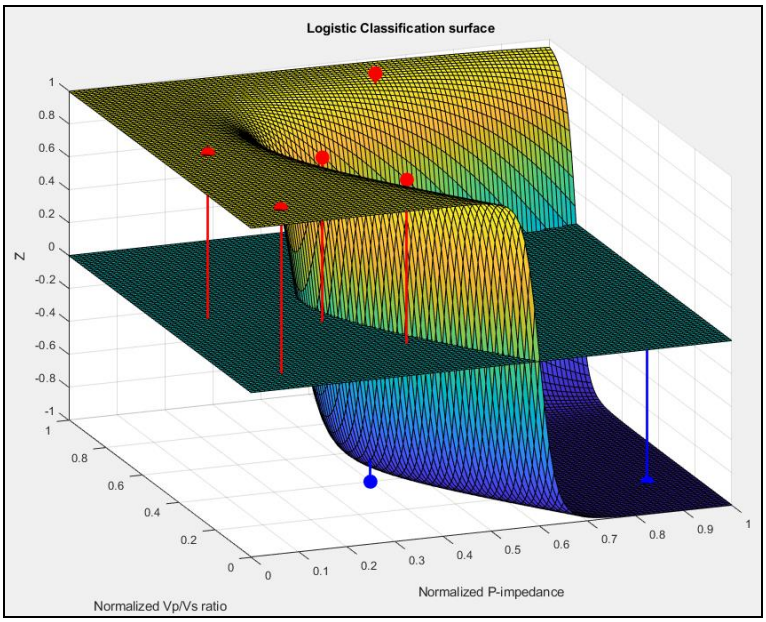


Figure 15: The 3D view of Figure 14(b), where the contours represent logistic functions.

CONCLUSIONS

In this study, I compared parametric and nonparametric statistica methods for performing facies classification, using a ten-point example. These methods consisted of the linear least-squares method, the k-Nearest-Neighbor, of kNN method, the Quadratic Discriminant Analysis (QDA) method, the Kernel Density Estimation (KDE) method and the Deep Feedforward Neural Network (DFNN) method. Both the QDA and KDE methods were Bayesian approaches, in which Bayes' rule was used to classify the facies.

I found that the least-squares linear method did a very poor job of separating the classes and, although kNN did an excellent job, it did so by using linear segments rather than a smooth boundary. The QDA method using a Bayesian decision approach gave a smooth boundary but miss-classified some of the points, whereas the KDE method with a Bayesian decision boundary and the DFNN approaches both gave smooth boundaries and were both able to perfectly classify the points in each facies. However, since the DFNN approach is highly dependent on the structure of the network and is also time consuming to run, the KDE approach is suggested as the optimum method to apply to a real data example.

REFERENCES

Bishop, C.M, 2006, Pattern Recognition and Machine Learning: Springer-Verlag.

Hastie, T., Tibshirani, R., and Friedman, J., 2009, The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd Edition: Springer Series in Statistics.

Higham, C.F and Higham, D.J., 2019, Deep Learning: An Introduction for Applied Mathematicians: SIAM Review, Vol. 61 No. 4 pp 860-891.

ACKNOWLEDGEMENTS

We wish to thank our colleagues at the CREWES Project and at HampsonRussell Software, a division of GeoSoftware for their support and ideas, as well as the sponsors of the CREWES Project.