An in-depth analysis of logarithmic data transformation and per-class normalization in machine learning: Application to unsupervised classification of a turbidite system in the Canterbury Basin, New Zealand, and supervised classification of salt in the Eugene Island minibasin, Gulf of Mexico

Thang N. Ha1, David Lubo-Robles1, Kurt J. Marfurt1, and Bradley C. Wallet2

Abstract

In a machine-learning workflow, data normalization is a crucial step that compensates for the large variation in data ranges and averages associated with different types of input measured with different units. However, most machine-learning implementations do not provide data normalization beyond the z-score algorithm, which subtracts the mean from the distribution and then scales the result by dividing by the standard deviation. Although the z-score converts data with Gaussian behavior to have the same shape and size, many of our seismic attribute volumes exhibit log-normal, or even more complicated, distributions. Because many machine-learning applications are based on Gaussian statistics, we have evaluated the impact of more sophisticated data normalization techniques on the resulting classification. To do so, we provide an in-depth analysis of data normalization in machine-learning classifications by formulating and applying a logarithmic data transformation scheme to the unsupervised classifications (including principal component analysis, independent component analysis, self-organizing maps, and generative topographic mapping) of a turbidite channel system in the Canterbury Basin, New Zealand, as well as implementing a per-class normalization scheme to the supervised probabilistic neural network (PNN) classification of salt in the Eugene Island minibasin, Gulf of Mexico. Compared to the simple z-score normalization, a single logarithmic transformation applied to each input attribute significantly increases the spread of the resulting clusters (and the corresponding color contrast), thereby enhancing subtle details in projection and unsupervised classification. However, this same uniform transformation produces less-confident results in supervised classification using PNNs. We find that more accurate supervised classifications can be found by applying class-dependent normalization for each input attribute.

Introduction

Machine learning has been applied to seismic facies classification for more than 20 years, with early successes reported for supervised classification by Meldahl et al. (1999) and West et al. (2002) and for unsupervised classification by Poupon et al. (1999) and Strecker and Uden (2002). Since that time, a wide variety of commercial and research implementation of machine learning facies classification algorithms has been adopted by the seismic interpretation community. A key assumption of most machine-learning workflows is that the input data approximate Gaussian-shaped distributions. However, if we wish to measure the distance of a given data point to a cluster center, we need to account for the differences in units. For example, the P-wave impedance may exhibit values that range from 1500 to 12,000 g/cm² m/s, whereas Poisson’s ratio may vary from 0.1 to 0.45. In this case, a simple z-score normalization of each sample j of the kth attribute $a_{jk}$ by its mean $\mu_k$ and standard deviation $\sigma_k$

$$\bar{a}_{jk} = \frac{a_{jk} - \mu_k}{\sigma_k},$$  

removes the effect of using different measurement units and assigns equal importance to each measurement. Although seismic amplitude and some seismic attributes can be well represented by a Gaussian distribution, most seismic attributes are skewed, whereas some may exhibit an approximately uniform distribution (e.g., cosine of the instantaneous phase), suggesting that we need a nonlinear transformation (Figure 1).
In general, machine-learning classifications can be categorized into two major types: (1) unsupervised classification, which automatically assigns different colors to different groups of data, and (2) supervised classification, which requires the human interpreter to explicitly label a subset of the data as belonging to a given class or seismic facies.

Unsupervised classifications

Unsupervised classifications, in turn, can be divided into two categories: (1) projection techniques, in which $N$ input attributes are projected onto a latent space of smaller dimensionality (usually two to three dimensions allowing mapping against a 2D or 3D continuous color matrix), and (2) clustering techniques, in which different data points are arranged into different clusters, with each cluster assigned a distinct color.

The most well-known projection technique is principal component analysis (PCA), in which the input attributes are projected onto two or more eigenvectors of their covariance matrix to capture the maximum variation of the input data (Guo et al., 2008; Chopra and Marfurt, 2014). Although the principal components are orthogonal and theoretically uncorrelated, seismic facies are not, such that PCAs can mix different seismic facies. Honorio et al. (2014) and Lubo-Robles and Marfurt (2019) address this issue using independent component analysis (ICA), which is a nonorthogonal projection method based on higher order statistics. Among the latest projection methods is a stochastic, nonlinear projection technique described by Wallet and Ha (2019), which uses an autoencoder deep-learning neural network to “encode” $N$ input attributes into a 3D latent space to be displayed via red-green-blue (RGB) blending.

Zhao et al. (2015) apply different clustering techniques, including $k$-means, self-organizing maps (SOMs), and generative topographic mapping (GTM) to a turbidite system in the Canterbury Basin, New Zealand. Among those clustering techniques, $k$-means is the simplest and fastest method, in which the input training data are “partitioned” into a user-defined number of clusters based on the distance from a data point to the clusters’ centers. In general, $k$-means does not take into account the size and shape of each cluster in the classification. In contrast, an extension of $k$-means results in a Gaussian mixture model (GMM), in which each cluster is assigned its own multidimensional Gaussian density functions. The size, shape, and position of each Gaussian density function are modified in an iterative expectation-maximization scheme (Hardisty and Wallet, 2017). A common drawback of $k$-means and GMM algorithms is that the classification result does not graphically show the proximity of one cluster to another. Instead, SOM directly maps the clusters to a deformed manifold, such that the relative position of each cluster is defined (Zhao et al., 2015). The manifold is in turn mapped to a latent space amenable to color mapping. While early SOM algorithms (e.g., Po-upon et al., 1999) used a 1D curve mapped against a rainbow color bar as the manifold, 2D deformed surfaces mapped against a 2D color bar are now more common (e.g., Strecker et al., 2005; Castro de Matos et al., 2010; Roden et al., 2015; Zhao et al., 2015), although 3D manifolds can be mapped against RGB. Castro de Matos et al. (2010) and Zhao et al. (2016) use Sammon mapping to better measure the distances in $n$-D attribute space to a deformed 2D manifold. Zhao and Marfurt (2017) evaluate different training data extraction schemes, constraining SOM analysis with stratigraphy (Zhao et al., 2017), and, most recently, applying a data-adaptive weighting scheme for SOM input attribute selection (Zhao et al., 2018). The original SOM algorithms perform clustering by finding the closest prototype vector (or neuron) to a data sample and do not provide a measure of the confidence of the clustering process (Chopra and Marfurt, 2014; Roy et al., 2014). More recent innovations construct a Gaussian distribution after clustering, which then provides a measure of the confidence of each clustered data point (Roden and Chen, 2017). In contrast, GTM provides a direct measure of the likelihood that a data point falls within a given class by estimating the contributions of all of the latent space grid points to a data sample using a mixture of Gaussian density functions (Zhao et al., 2015).

Supervised classifications

Supervised classifications also consist of two main types: (1) semisupervised classifications, in which a clustering algorithm is trained (or labeled) by an interpreter to associate a cluster with a specific class or seismic facies, and (2) purely supervised neural network classifications, in which an input sample is fed through many hidden neuron layers to determine to which class, or “label,” it belongs.

Qi et al. (2016) show how the otherwise unsupervised GTM can be modified to be a semisupervised classification.
First, they apply a Kuwahara median filter to the input attributes to smooth out the interior of seismic facies, while sharpening the boundaries between different facies. The authors then define training samples belonging to $N$ classes, which are mapped onto a regularly gridded GTM “manifold” and assigned with the corresponding $N$ Gaussian density functions, allowing the classification to generate a probability density function for each facies.

Many of the explicitly supervised neural network algorithms were applied to predicting the well-log response from one or more seismic attribute volumes (Verma et al., 2012; Torres et al., 2018). Neural network application to well logs occurred first because the limited computational power of interpretation workstations was able to train the network for a suite of 1D well logs, but not on seismic data volumes that are typically two to three orders of magnitude larger in size. Meldahl et al. (1999) are among the first to classify seismic facies using artificial neural networks (ANN), starting with defining a single target facies (a gas chimney) and everything else. Here, the training data were attribute vectors selected by a skilled interpreter on the computer screen. West et al. (2002) develop an ANN algorithm that classifies some seven different seismic facies. Since then, other machine-learning algorithms have been evaluated. Zhao et al. (2014) perform lithofacies classification in the Barnett Shale by applying proximal support vector machine on density, gamma-ray, and sonic logs. Qi et al. (2019) use mud logs and apply this same algorithm to predict areas of low and high rates of penetration in drilling horizontal wells in the Mississippian cherty-lime reservoirs of Oklahoma. Verma et al. (2012) map high-frackability and high total organic carbon (TOC) zones in the Barnett Shale by predicting a gamma-ray volume from seismic attributes extracted along well paths, using the probabilistic neural network (PNN). Similarly, Torres et al. (2018) apply PNN on a suite of seismic attributes and inversion results extracted along well paths to map the TOC distribution in the Woodford Shale, Oklahoma. With the rapid evolution of PC hardware, Lubo-Robles et al. (2021) are able to implement PNN in supervised seismic facies classification using 3D seismic attributes. Kim et al. (2019) also use 3D seismic attributes in their supervised classification based on the random forest (RF) algorithm. The authors carefully select relevant input attributes for their RF supervised classification by computing the linear and non-linear correlation coefficients between each pair of input attributes and rejecting redundant attributes that are highly correlated to the others.

Negligence to data normalization

The vast majority of machine learning publications, unsupervised and supervised alike, do not describe how the training data were normalized. Although some papers describe a z-score normalization step in their machine-learning workflow, most do not mention normalization at all. A rare exception is a paper by Qi et al. (2020), outlining an attribute scaling step in an automatic attribute selection workflow for a supervised classification scheme based on GTM and GMM algorithms. The authors find that most of the input attributes used in their supervised classification, such as coherence, energy, and gray-level cooccurrence matrix (GLCM) texture attributes, require a logarithmic scaling scheme to better approximate the data distributions by Gaussian curves. However, the authors do not specify the reason behind their simple formula of logarithmic scaling, such as $\log(1.0 - x)$ for coherence and $\log(5.0 - x)$ for GLCM attribute. Neither do the authors provide a comparison between logarithmically scaled and z-score normalized final classification results.

Without a comprehensive analysis and formulation of data normalization in the literature, the goal of this paper is to define a suite of useful normalization schemes using only basic elements of mathematics. To simplify our demonstration, we separate the mathematical formulation and pseudocode of our logarithmic transformation in Appendices A and B for seismic interpreters and computer scientists who want to incorporate our logarithmic transformation into their own workflows. We then implement and compare our logarithmic transformation and per-class normalization methods to the traditional z-score normalization and bulk normalization schemes in two case studies: (1) unsupervised classifications of a turbidite system in the Canterbury Basin, New Zealand, and (2) supervised classification of salt facies in the Eugene Island minibasin, Gulf of Mexico. Finally, we analyze the histograms of the input data and the classification results to explain the differences made by using different normalization schemes.

Case study 1: Unsupervised classification of a turbidite channel system in the Canterbury Basin, offshore New Zealand

Geologic settings

The Canterbury Basin, New Zealand, is well-known for the complexity of its turbidite channel systems. The study area is located in the northern Waka3D seismic survey, at the transition between the continental shelf and slope (Figure 2). The Canterbury Basin underwent three main stages of geologic deformation. A thick layer of clastic and coaly sediments was deposited during Middle to Late Cretaceous rifting and subsidence (Sutherland and Browne, 2003). Then, organic-rich black shale and widespread limestone were formed when the basin entered a transgression from the Late Cretaceous to Middle Tertiary (Cozens, 2011). Since the Late Tertiary, the basin underwent a regression due to uplifting and minor tectonic inversion, thus shifting the study area into a transition zone between the continental shelf and slope, where many paleocanyons and turbidite channels were formed (Zhao et al., 2016). These canyons and channels were filled with late Tertiary carbonate debris and thus are potential reservoirs for hydrocarbons (Wallet and Ha, 2019).
Methods

Figure 3 represents our unsupervised classification workflow. The first step is to select the input attributes that are relevant to the classification of turbidite facies. Following the work by Wallet and Ha (2019), we choose six attributes as the input for our unsupervised classification (Figure 4): (1) coherent energy: for measuring reflector strength (2) structural curvedness: for highlighting channel axes and levees (3) GLCM entropy and (4) GLCM homogeneity (texture attributes): for differentiating architectural elements that have different seismic texture appearance (5) peak frequency and (6) peak magnitude (spectral decomposition attributes): for measuring the most dominant layer thicknesses and reflector strength.

To ensure that the training data adequately represent different facies in a turbidite system, we follow the work by Zhao and Marfurt (2017) and Zhao et al. (2018) to constrain the training data extraction to three adjacent horizon slices: one at our horizon of interest, one above, and one below, amounting to a total of approximately 3 million data samples. Parameters of attribute computations and unsupervised training data extraction are listed in Appendix E. We then perform z-score normalization and logarithmic transformation on the training data. Mathematical descriptions of the z-score normalization and our logarithmic transformation are shown in Appendices A and B.

The original input data distributions are shown in Figure 5a. All six input attributes are nonnegative and have skewed distributions. Note that z-score normalization does not change the shapes of the original distributions; rather, it only shifts and stretches them (Figure 5b). In contrast, logarithmically transformed data have much more symmetrical shapes that closely resemble the “bell” curve of an ideal Gaussian distribution (Figure 5c).

We then feed the normalized training data to different unsupervised classification algorithms, including two projections techniques (PCA and ICA) and two clustering techniques (SOM and GTM), using the exact same parameter configuration for the z-score normalization and the logarithmic transformation. Depending on whether the classification generates two or three output components, we perform a 2D color crossplot or RGB blending (Figure 6) to display the results.

PCA

The purpose of PCA is to find an orthogonal coordinate system that best captures the data variation and to project the input attributes onto this new coordinate system (Figure 7). Mathematically, PCA is equivalent to first computing the covariance matrix of the input attributes, then finding the eigenvectors and eigenvalues of the covariance matrix and, finally, performing matrix multiplication between the input data and the eigenvectors to obtain the principal components. Each eigenvector represents the direction of a principal coordinate axis, whereas each eigenvalue represents

Figure 2. Seafloor bathymetry of the Canterbury Basin, New Zealand (modified from Zhao et al., 2016). The study area indicated by the red rectangle is in the northern part of the Waika3D survey images of multiple turbidites at the transition between the continental shelf and the slope.

Figure 3. Flowchart of our unsupervised classification.
the amount of data variation along an axis. We choose three output principal components corresponding to the three largest eigenvalues for our analysis to capture most of the data variation.

Figure 8 shows the 2D histograms of three output principal components. Each 2D histogram corresponds to a pair of principal components. To reduce the effect of extreme values on the histogram displays, we clip 5% of the data at the extreme negative and 5% of the data at the extreme positive of each principal component. We observe that logarithmic transformation produces 2D histograms that are more symmetric, circular, and diffuse than those generated from the z-score normalization.

The final RGB blended images of three output principal components are shown in Figure 9. Overall, the colors in the logarithmically transformed result are balanced, whereas the z-score normalization is biased toward the green (the second component). Magnifying the yellow box in the logarithmically transformed image, we can distinguish between the magenta crevasse splay (marked by a magenta arrow) and the orange crevasse splays (marked by orange arrows), which we hypothesize are composed of different types of sediment. In the z-score normalized image, those splays appear to a more uniform purple. A close-up inspection of the red box in the logarithmically transformed image reveals small, yellowish channels that make up the internal structure of a larger orange crevasse splay. In the z-score normalized image, those details are absent, and the splay appears as a homogeneous dark-red patch.

**ICA**

One assumption of PCA is that the data can be represented by a single multidimensional ellipsoidal “cloud.” However, in practice, a data distribution can be composed of different clouds with different shapes, sizes, and orientations, making orthogonal principal components unable to fully capture the variation among all the data clouds (Figure 10). By finding a nonorthogonal coordinate system in an iterative manner, using higher order statistics, ICA aims to further separate different seismic facies and capture even more data variation (Lubo-Robles and Marfurt, 2019). To make it consistent with our PCA workflow, we specify three output independent components for our ICA model. Detailed parameter configuration of our ICA implementation is listed in Appendix E.

Similar to PCA, we construct a 2D histogram per each pair of output independent components (Figure 11). ICA 2D histograms exhibit the same phenomenon that we observed in PCA 2D histograms: Logarithmic transformation produces 2D histograms that are more symmetric and diffuse than those generated from z-score normalization. Yet, there is a critical difference between ICA and PCA 2D histograms. In PCA, even though the shapes of the data clouds are different between logarithmic transformation and z-score normalization, there is still a common pattern between a 2D histogram produced by logarithmic transformation and a corresponding one produced by z-score normalization. For example, the data cloud in Figure 8c is somewhat similar to the data cloud in Figure 8f. How-
ever, in ICA, logarithmically transformed histograms show radically different data clouds than the corresponding z-score normalized histograms. For example, the data cloud in Figure 11a has a totally different shape, size, and orientation from the data cloud in Figure 11d. This is likely because the rank and polarity of the output independent components are undefined (Lubo-Robles and Marfurt, 2019). In other words, unlike PCA in which principal components are ordered by their corresponding eigenvalues, it is impossible to tell beforehand that an independent component captures more data variation than another. Using a different normalization scheme somehow changes the polarity and possibly the order of the independent components, causing the ICA algorithm to converge to completely different results.

The radical histogram difference between logarithmic transformation and z-score normalization is also reflected in the final RGB blended images of three output independent components (Figure 12). The images exhibit a completely different color gamut. The level of detail and color contrast in logarithmically transformed and z-score normalized ICA images are similar to each other and are comparable to the logarithmic PCA image in Figure 9b. It is difficult to tell which normalization scheme is better, even in the magnified sections. Therefore, for ICA, there is little difference in using a more sophisticated logarithmic transformation rather than the simpler z-score normalization.

**SOMs**

Originally used in medical research for gene pattern recognition, SOM is now widely adopted as a clustering algorithm for seismic facies thanks to its relatively fast computation and its capability to show the proximity of one cluster to another (Zhao et al., 2015). The SOM algorithm works by defining an initial grid of prototype vectors (neurons) on a plane defined by the first two eigenvectors called the manifold, which, in turn, are mapped to a 2D color table. As the process iterates, the manifold deforms to better represent the input training data (Figure 13). Clustering is done by finding the closest prototype
vector to the data vector at each voxel. In our SOM implementation, we output not only the class number, but also the coordinates of each class on the two axes of the latent space, thereby allowing us to use commercial crossplotting tools to define features of interest. The parameter configuration of our SOM model is listed in Appendix E.

Figure 14 shows the crossplot images of the two SOM components along with the corresponding 2D histograms. Figure 14 requires strong data clipping for the crossplot of the z-score normalized SOM components to have a similar color contrast with the nonclipped crossplot of the logarithmically transformed SOM components. White pixels in the z-score normalized image represent clipped extreme data points. Appendix C explains what happens if we underclip or overclip the data.

**Figure 7.** Illustration of PCA. The first eigenvector $v_1$ best represents the variation in the data cloud. The first principal component (PCA 1) is generated by projecting each data point onto the $v_1$ axis. The second eigenvector $v_2$ best represents the data not represented by $v_1$ and is orthogonal to it. The second principal component (PCA 2) is generated by projecting each data point onto the $v_2$ axis.

**Figure 9.** RGB blended images of three principal components computed using (a) z-score normalization and (b) logarithmic transformation. Overall, (b) is more evenly distributed against the 3D color bar whereas (a) is biased toward green. In the yellow box of (b), we can distinguish the crevasse splays indicated by the magenta and orange arrows from each other, whereas all three splays have the magenta purple color in (a). In the red box of (b), note the small, yellowish channels within the orange crevasse splay, whereas in (a), the splay appears as a homogeneous dark-red mass.

**Figure 8.** The 2D histograms of the first three principal components computed from (a-c) the z-score normalized and from (d-f) the logarithmically transformed input. To avoid the effect of extreme values on the histograms, 5% of the data at the extreme positive and 5% of the data at the extreme negative of each principal component are clipped and fall outside the images. The logarithmic histograms exhibit more evenly distributed, elliptical, and symmetric distributions than the z-score histograms. Note that there is still a recognizable common pattern between each pair of histograms (a-d), (b-e), and (c-f). For example, (f) looks like a magnified section near the center of (c).
for the 2D crossplot. We observe that the z-score normalized image, even after a strong clipping, is a lot greener than the logarithmically transformed image, due to the data distribution being skewed to the lower left corner of the z-score normalized 2D histogram. In contrast, the logarithmic transformation produces a much more color-balanced image and a more diffuse 2D histogram, allowing us to better delineate different facies, such as distinguishing the yellow and orange crevasse plays from the green flood plain or mud-filled channels.

GTM

Similar to SOM, GTM also aims to generate and iteratively deform a gridded manifold to best fit the input training data. However, the difference is in how the clustering is done. As noted by Zhao et al. (2015), whereas SOM “snaps” a data point to the closest grid node to generate a cluster, GTM puts a Gaussian probability distribution function at each and every grid node on the manifold to estimate the contribution of all of the grid nodes to a data point (Figure 15). The parameter configuration of our GTM model is listed in Appendix E.

Our GTM implementation results in two GTM components, which we display using crossplots (Figure 16). Unlike SOM, no clipping is needed for z-score normalized and logarithmically transformed GTM components. Again, we observe a similar phenomenon in the 2D histograms: The data are more evenly distributed in the logarithmic 2D histogram, whereas the z-score normalization exhibits a very high data concentration within a single cell just below the center of the 2D histogram, causing the z-score crossplot image to have more green and grayish yellow pixels than the logarithmic crossplot image. Although the z-score crossplot image appears to enhance small details, these “details” are not geologic, but rather random noise, thus causing the actual geologic features (such as crevasse splays and channels) to appear more segmented than those in the logarithmic crossplot image.

Case study 2: Supervised classification of salt in the Eugene Island Minibasin, Gulf of Mexico

Geologic setting

The Eugene Island minibasin (Figure 17) contains one of the largest oil and gas fields in the northern part of the Gulf of Mexico, offshore Louisiana. The development of the Eugene Island minibasin occurs in relatively recent geologic time, during the Pliocene-Pleistocene (Joshi and Appold, 2016), and it consists of three phases: (1) prodelta, (2) proximal deltaic, and (3) fluvial (Alexander and Flemings, 1995). During the prodelta phase, rapid deposition of shales and turbidite caused the underlying Miocene salt to withdraw from the basin and buoy upward, leading to the creation of salt diapirs (Joshi and Appold, 2016). Salt continued to mobilize...
during the proximal deltaic phase, creating more accommodation space for low-stand shelf-margin deposition of alternating sand-shale sequences (Alexander and Flemings, 1995). At the final fluvial stage, salt withdrawal ended, leading to the formation of erosional unconformities during lowstands and the deposition of shallow-water deltaic and fluvial sand and shale during highstands (Joshi and Appold, 2016). Most of the major reservoirs are laterally extensive sand sheets deposited during the proximal deltaic phase, which are sealed by laterally extensive shale layers on the top and by salt diapirs on the sides (Alexander and Flemings, 1995).

**Methods**

Figure 18 shows our supervised classification workflow. As with unsupervised classification, the first step is to select input attributes relevant to the supervised classification of salt. Lubo-Robles et al. (2021) perform an exhaustive search to find the best combination of input attributes and the best corresponding parameters for a PNN supervised classification. The authors find that a combination of seismic attributes given by coherence, GLCM contrast, total energy, and dip deviation yields the lowest global validation error. However, as we implemented the exhaustive search to different normalization schemes, we found that each normalization scheme leads to a different optimal combination of input attributes (see Appendix D for more details). Because our research focuses on data normalization, we want our supervised classification workflow to be consistent with different normalization schemes. This requires us to use the same set of input attributes for all normalization schemes. We notice that the coherence attribute is found in all four normalization schemes’ optimal input combinations. Therefore, to simplify our demonstration, we decide to use coherence attributes as the sole input for all of our supervised classifications.

In the next step, we follow the work by Qi et al. (2016) and apply a Kuwahara median filter to the input coherence attribute to smooth the internal noise of the salt diapirs, while sharpening the boundaries between the salt diapirs and the surrounding sedimentary layers (Figure 19). Parameters of the Kuwahara median filter are listed in Appendix E.

We pick different polygons inside and outside a salt diapir to denote salt and not-salt training data (Figure 20). After that, we extract the training samples within those polygons from the Kuwahara-filtered co-
herence attribute volume. The parameters of our supervised training data extraction are listed in Appendix E.

Figure 21 illustrates the difference between bulk normalization scheme (in which we compute and apply the same normalization to the training data of all classes) and per-class normalization scheme (in which we normalize the training data of each class separately). Together with logarithmic transformation and z-score normalization, we perform a total of four different normalization schemes on the training data: (a) logarithmic bulk normalization, (b) logarithmic per-facies normalization, (c) z-score bulk normalization, and (d) z-score per-facies normalization. We then feed the four normalized training data sets into the PNN algorithm to generate four models, and we finally classify the entire volume using the generated models.

PNN

Essentially, PNN algorithm performs supervised classification by estimating the probability density functions of different classes or facies (hence the name probabilistic neural network), most commonly by approximating a Gaussian distribution to the input training data of each class (Specht, 1990; Masters, 1995; Lubo-Robles et al., 2021). We summarize the mathematical description of the PNN algorithm in Appendix D. Again, to make sure our workflow is consistent, we set the PNN smoothing parameter $r = 1.0$ in all four normalization schemes.

Figure 14. A 2D color crossplot of two output SOM components with (a) z-score normalization and (b) logarithmic transformation. We clip the extreme values of the z-score SOM components to allow the z-score image in (a) to have approximately the same level of color contrast as the logarithm scaled image (b), which does not require any data clipping at all. The white pixels in (a) represent clipped data. Even after such strong clipping, the z-score image is biased toward green representing the data distribution skewed to the lower left of the 2D histogram. Many of the data points are highly concentrated within one cell (the white arrow). The logarithm transformation image in (b) is more diffuse and better spans the 2D color bar, thereby better delineating the yellow and orange crevasse splays from the green flood plain.

Figure 15. Illustration of GTM (modified from Zhao et al., 2015). Similar to SOM, GTM also generates a 2D gridded manifold that best fits the input training data. However, instead of snapping a data sample (the green cube) to the closest grid node, each node (the blue spheres) is given a Gaussian distribution function (the big translucent grayish spheres) to measure the probability of a data sample to a grid node. All Gaussian distributions are of the same size. This size and the positions of the Gaussian distributions change with each iteration in the clustering process. At the end, the “responsibility” of each node to a given data vector is computed. In unsupervised classification, the most “responsible” grid node of a data sample is plotted against a 2D color table. In semi-supervised classification (e.g., Roy et al., 2014; Qi et al., 2016), a probability density function is computed and output as a separate volume for each class, resulting in a measure of probability that a data sample belongs to a given class.
Figure 22 shows the prediction results of PNN supervised classification using four normalization schemes. Green corresponds to salt facies, red corresponds to not-salt facies, and black corresponds to muted and no-permit zones. We observe several green patches (yellow ellipses) outside the salt diapir in the prediction result associated with the logarithmic bulk normalization scheme (Figure 22a), which do not appear in the prediction result associated with the z-score bulk normalization scheme (Figure 22c). The faulted region above the salt diapir (the blue ellipse) is correctly classified as the not-salt facies in the prediction results associated with the per-facies normalization (Figure 22b and 22d), but it is misclassified as salt facies in the prediction results associated with bulk normalization (Figure 22a and 22c). Within the per-facies normalization scheme, logarithmic transformation and z-score normalization appear to yield very similar prediction results (Figure 22b and 22d). However, as we display the salt probability results (Figure 23) corresponding to the prediction results in Figure 22, using the same probability scale from 0.0 to 1.0, we notice that logarithmic per-facies normalization scheme (Figure 23b) yields a dimmer, lower contrast image of salt probability than z-score per-facies normalization scheme (Figure 23d). This means that the PNN algorithm is less confident in classifying salt facies with the logarithmic per-facies normalization scheme than with the z-score per-facies normalization scheme.

Discussion

Given such radical differences in machine learning results using different data normalization schemes, there are several questions to be addressed. Why did logarithmic transformation improve unsupervised clas-

Figure 16. A 2D color crossplot of two output GTM components with (a) z-score normalization and (b) logarithmic transformation. No data clipping is needed in (a and b). Again, data are highly concentrated at a cell just below the center of the z-score 2D histogram, whereas the logarithmic 2D histogram shows a more spread out distribution. The two crossplots have a very similar color gamut, and, at first glance, the z-score crossplot appears to better enhance smaller “features.” However, a close inspection reveals that these features are not geologically meaningful but, rather, represent random noise, causing the real geologic features such as channels and crevasse splays to be less continuous than those in the logarithmic crossplot.
sification but not supervised classification? What is the reason behind the enhancement made by the per-class normalization scheme to supervised classification? To answer these questions, we need to understand a fundamental difference between unsupervised and supervised classifications (other than the obvious need for human interaction).

**Unsupervised essence: Color mapping**

In unsupervised classification, the manner in which colors are mapped to the results is critical. In fact, the whole idea behind unsupervised classification is to "paint" each and every single data point in such a way that points having similar values (i.e., belonging to the same cluster) have similar colors. However, if a group of data points is assigned with one single color, then we lose the differences among that group of points when displaying the classification results. The loss of color differences is illustrated in the SOM and GTM crossplots using z-score normalization scheme (Figures 14a and 16a). The 2D histograms of z-score SOM and GTM show that the data are highly concentrated within a cell, thus "painting" a large chunk of data points with the same color in the crossplots. In contrast, 2D histograms of logarithmic SOM and GTM (Figures 14b and 16b) show a more diffuse distribution, highlighting subtle variations of data points near the center of the data distribution, thus allowing more colors and fine details to appear in the crossplots.

Another illustration of color mapping in unsupervised classification is the RGB blending process of PCA results. Examine the first principal component projection plotting against the red color channel (Figure 24). The histograms are clipped in the same way as in RGB blending of the PCA projections: Only 90% of the data distribution is shown, whereas the 5% of the data at the extreme left and 5% of the data at the extreme right of the distribution are clipped. Even with such a strong clipping, we can still observe that z-score data distribution is skewed to the right and has a narrower shape compared to the logarithmic data distribution, meaning a large portion of the data is concentrated near the peak of the distribution. Thus, fewer colors are used to map the area around the peak of the distribu-

Figure 18. Flowchart of our PNN supervised classification workflow.

Figure 19. AA’ vertical slices through the (a) original and (b) Kuwahara-filtered coherence volumes. The Kuwahara filter smooths the internal detail of a salt diapir and sharpens the edges. Because the low-coherence faults are thinner than the $3 \times 3 \times 3$ Kuwahara window, they are also attenuated, avoiding their misclassification as a seismic facies.
In contrast, the logarithmic data distribution exhibits a much wider and more symmetric shape than the z-score normalization. Thus, more colors are mapped near the peak of the distribution. Essentially, the logarithmic transformation “squeezed” the extreme positive data points to the center and “stretched” the left side of the data distribution, thereby increasing the resolution near the peak of the distribution while reducing the resolution of the extrema. Fortunately for us, the subtle geologic detail of the turbidite channel system resides near the peak of the distribution and was thus better resolved using the logarithmic transformation. We anticipate that logarithmic transformation would have produced worse unsupervised classification results than z-score normalization if our analysis had instead focused on resolving the extreme data points.

**Supervised essence: Shapes of clusters and distances among clusters**

Unlike unsupervised classification’s dependence on color mapping, supervised classification relies exclusively on the ability of the internal algorithm to distinguish among different clusters. Whether the clusters are distinguishable or not, in turn, heavily depends on the shapes of the clusters and the distances from one cluster to another. To illustrate the differences in the shapes of the clusters and the distances among the clusters for different normalization schemes, we plot the histograms of PNN’s input coherence attribute in four cases (corresponding to the four PNN salt probability images shown in Figure 23): (a) logarithmic bulk transformation, (b) logarithmic transformation based on training data of the salt facies, (c) z-score bulk normalization, and (d) z-score normalization based on training data of the salt facies (Figure 25). These histograms display 100% of the data distribution and are resized to the same horizontal scale. Because salt is incoherent, the salt cluster has lower coherence values than the not-salt cluster; thus, the salt cluster falls to the left of the histograms, whereas the not-salt cluster falls to the right of the histograms.

**Figure 20.** Vertical slice BB’ and time slice at \( t = 2.0 \) s through the Kuwahara-filtered coherence volume, showing the picked polygons of the salt and not-salt facies. We carefully define the polygons in such a way that the area covered by the salt polygons is approximately equal to the area covered by the not-salt polygons. We then extract our supervised training data from these polygons.

**Figure 21.** Difference between bulk normalization (a), in which we apply the same normalization to the training data of all classes, and per-class normalization (b and c), in which we normalize the training data of each class separately. The histograms were aligned at zero and resized to have the same horizontal scale. The magenta dashed lines represent the ideal normal distribution curves — the ultimate desired outcome of the normalization. With bulk normalization, the ideal curve is in the middle of the two clusters, whereas with per-class normalization, the ideal curve is at each cluster’s distribution, although not perfectly aligned.
We observe that with the bulk normalization scheme, the distance between the salt and not-salt clusters is approximately the same in the $z$-score and logarithmic histograms, but the shape of the salt cluster associated with logarithmic transformation (Figure 25a) is significantly wider than that associated with $z$-score normalization (Figure 25c). This is likely due to an inherent characteristic of the logarithmic function, which

**Figure 22.** Vertical slices AA′ through the PNN prediction volumes associated with (a) logarithmic bulk normalization, (b) logarithmic per-class normalization, (c) $z$-score bulk normalization, and (d) $z$-score per-class normalization. The yellow ellipses highlight some misclassified salt patches within the not-salt region of (a). The blue ellipses highlight the faulted region above the salt diapir that are misclassified as salt in (a and c). (b) The logarithmic per-class normalization and (d) $z$-score per-class normalization produce very similar predictions.

**Figure 23.** Vertical slices AA′ through the PNN salt probability volumes associated with (a) logarithmic bulk normalization, (b) logarithmic per-class normalization, (c) $z$-score bulk normalization, and (d) $z$-score per-class normalization. All displays have the same fixed scaling of salt probability from 0.0 to 1.0. Note the dimmed image produced by (b) logarithmic per-facies normalization, whereas (d) $z$-score per-facies normalization image has a much higher contrast.

**Figure 24.** Histograms of the first principal component projection associated with (a) $z$-score normalization and (b) logarithmic transformation, blended with black-to-red color gradient to illustrate the process of color mapping against the red color channel. The histograms are clipped in such a way that only 90% of the data distribution is shown, whereas 5% in the extreme left and 5% in the extreme right of the distribution are discarded. Note the wider and more symmetric histogram curve associated with logarithmic transformation, allowing more colors to be mapped near the peak of the data distribution, making the final RGB blended image of principal component projections capable of showing the subtle, fine details of the turbidite channel system in Canterbury Basin, New Zealand.
clusters. By normalizing the training data per each seismic facies, the distance between the two clusters becomes much larger, allowing the supervised algorithm to distinguish between different classes with a much higher confidence, hence with a significantly higher contrast in the PNN probability result (Figure 23b and 23d).

**Conclusion**

Our verdict is that data normalization, despite being a hidden step that is most often set to the default in a machine-learning workflow, deserves more attention. Different data normalization schemes can significantly generate different classification results. Logarithmic transformation produces unsupervised classification results with more subtle details and higher color contrast than z-score normalization because the logarithmic function tends to collapse the extreme positive data points while simultaneously expanding the near-zero values, effectively allowing more colors to be mapped around the peak of a distribution — where the data concentrate the most. However, for the same reason, the logarithmic transformation tends to shift the clusters of different classes closer together, thereby reducing the confidence of a supervised algorithm to distinguish between different classes, compared to z-score normalization. The per-class normalization scheme yields significantly greater distances among different clusters than the typical bulk normalization scheme, thus producing much higher contrast in the probability results. Therefore, it is critically important to try different normalization schemes on the input training data, carefully analyzing the distributions of the training data and the classification results and comparing the final displays side by side to obtain the most optimal normalization scheme for a specific machine-learning workflow.

**Acknowledgments**

We highly appreciate New Zealand Petroleum and Minerals for providing the Canterbury Basin data set as well as the U.S. National Archive of Marine Seismic Surveys and the U.S. Geological Survey for providing the Eugene Island data set. We use the Attribute-Assisted Seismic Processing and Interpretation software in our research to compute attributes, perform machine-learning classifications, and display results. Finally, we would like to express our gratitude to I. Yasutaka at the JX Nippon Oil and Gas Exploration Corporation in Japan for his advice and suggestions.

<table>
<thead>
<tr>
<th>Bulk normalization</th>
<th>Per-class normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Histograms" /></td>
<td><img src="image" alt="Histograms" /></td>
</tr>
</tbody>
</table>

**Figure 25.** Histograms of the PNN’s input coherence attribute, showing different shapes of clusters and distances between clusters via different normalization schemes: (a) logarithmic bulk transformation, (b) logarithmic transformation based on training data of salt facies, (c) z-score bulk normalization, and (d) z-score normalization based on training data of salt facies. Among the histograms, the z-score per-class normalization distribution (d) has the largest distance between salt and not-salt clusters, which corresponds to the PNN salt probability image with the highest contrast (Figure 23d). The general trend is that z-score histograms have greater distances between clusters than logarithmic histograms, and the per-class normalization scheme yields significantly greater distances between clusters than the bulk normalization scheme. Note that in (a and c), the distances between the two clusters are approximately the same, but the salt cluster in logarithmic histogram (a) has a wider distribution than the z-score histogram (c), causing many coherent data points to be misclassified as salt (the yellow ellipses in Figure 22a).
Data and materials availability
Data associated with this research are available and can be obtained by contacting the corresponding author.

Appendix A
Derivation of logarithmic transformation
Expectation versus reality
Mathematically, a logarithmic transformation refers to the application of the logarithmic function \( \log_b(x) \) to the input data. Our goal is to use a logarithmic transformation to reshape a nonnegative, right-skewed data distribution to be more symmetric and thus closer to an ideal bell-shaped Gaussian distribution (Figure A-1). However, if we directly apply \( \log_b(x) \) to the input data, we would instead end up with an even more asymmetric data distribution, only this time it is left-skewed, with a very long left “tail” stretching to negative infinity. This undesired result occurs because \( \log_b(x) \) approaches negative infinity as \( x \) goes to zero (Figure A-2). Furthermore, \( \log_b(x) \) is undefined for negative values of \( x \), the results of which in a computer are commonly represented by NaN (the abbreviation for not-a-number), which is detrimental to all subsequent computations. To find a better way to apply the logarithmic transformation, we need to go back to the basics, where we will first generalize linear transformations, including the well-known z-score normalization.

Z-score normalization revisited: Generalizing a linear transformation
The z-score, or standard score, is by far the most widely used data normalization scheme. The process involves subtracting the mean from the data, and then it divides the result by the data’s standard deviation:

\[
y_n = \frac{x_n - \mu}{\sigma}, \quad (A-1)
\]

where \( x \) is the input data vector of length \( N: x = [x_1, x_2, \ldots, x_N] \), and \( \mu \) is the mean (or the arithmetic average) of \( x \):

\[
\mu = \frac{\sum_{n=1}^{N} x_n}{N}, \quad (A-2)
\]

\( \sigma \) is the standard deviation of \( x \):

\[
\sigma = \sqrt{\frac{\sum_{n=1}^{N} (x_n - \mu)^2}{N}}, \quad (A-3)
\]

and \( y \) is the normalized data vector: \( y = [y_1, y_2, \ldots, y_N] \).

Equation A-1 can be rewritten into a different form, in which the subtraction is equivalent to an addition of negative mean, and the division is equivalent to a multiplication of the reciprocal of the standard deviation:

\[
y_n = \frac{1}{\sigma} [x_n + (-\mu)], \quad (A-4)
\]

where \( a = -\mu \) is the shift factor, and \( b = 1/\sigma \) is the scale factor of the z-score normalization. There are only two parameters (shifting and scaling) associated with a linear transformation, and any normalization scheme that has the form of equation A-5 is a linear transformation.

The goal of z-score normalization is to transform the input data into one with a mean of zero and a standard deviation of one, similar to a normal distribution (hence the term “normalization”). Z-score normalization assumes that the input data distribution has approximately the same bell shape of a normal distribution. However, in practice, most seismic attributes are nonnegative, exhibiting skewed and asymmetrical distribution that are poorly represented by bell shape assumption. The linear z-score normalization neither changes the skewness nor the asymmetry of a distribution to more closely resemble a normal distribution.

Equation A-4 can be further generalized as

\[
y_n = b(x_n + a), \quad (A-5)
\]

Figure A-1. Illustrations of (a) the original distribution of a typical nonnegative, right-skewed attribute, (b) the expected symmetric, bell-shaped distribution after logarithmic transformation, and (c) the bitter reality of directly applying a logarithmic function to the original data: a left-skewed, even more asymmetric than original distribution with a left tail stretching to negative infinity.

Figure A-2. Graph of the logarithmic function. As \( x \) goes to zero, the logarithmic function approaches negative infinity, thus causing the long left tail of the actual distribution in Figure A-1c. Also, note that the logarithmic function is not defined where \( x < 0 \).
**Generalizing the logarithmic transformation**

Based on the generalized formula of linear transformation in equation A-5, a fully generalized logarithmic transformation is equivalent to first linearly transforming the input, followed by the application of the logarithmic function, and then followed by yet another linear transformation in the logarithmic domain:

\[ y_n = c \{ \log_k [b(x_n + a)] + d \}, \quad (A-6) \]

where \( \log_k(\cdot) \) is the logarithmic function of base \( k \), \( d \) is the shift factor in the logarithmic domain, and \( c \) is the scale factor in the logarithmic domain.

Recall from high-school algebra that there are three important properties of the logarithmic functions. First, the logarithmic function of base \( k \) can be rewritten using the natural logarithmic function:

\[ \log_k(u) = \frac{\ln(u)}{\ln(k)}. \quad (A-7) \]

Second, the power rule of the logarithm states that

\[ \ln(u^m) = m \ln(u). \quad (A-8) \]

Third, and most importantly, the logarithm of a product is the sum of the individual logarithms:

\[ \ln(uv) = \ln(u) + \ln(v). \quad (A-9) \]

Using equations A-7, A-8, and A-9, we can rewrite equation A-6 as

\[
\begin{align*}
y_n &= c \left\{ \frac{\ln[b(x_n + a)]}{\ln(k)} + \ln(k^d) \right\} \\
&= \frac{c}{\ln(k)} \{ \ln[b(x_n + a)] + \ln(k^d) \}; \text{ thus,} \\
y_n &= \frac{c}{\ln(k)} \{ \ln[k^d b(x_n + a)] \}. \quad (A-10)
\end{align*}
\]

Note that, in equation A-10, \( c/\ln(k) \) can be viewed as a single factor of scaling in the logarithmic domain, whereas \( k^d b \) can be viewed as a single factor of scaling in the original data domain. This simplification occurs because shifting in the logarithmic domain can be expressed as scaling in the original data domain, as implied by equation A-9. Thus, the generalized formula for logarithmic transformation can be rewritten as follows:

\[ y_n = c \{ \ln[b(x_n + a)] \}. \quad (A-11) \]

Essentially, we have reduced the number of parameters in the logarithmic transformation from five \( (a, b, c, d, \text{ and } k) \) in equation A-6 to three \( (a, b, \text{ and } c) \) in equation A-11. Even at three parameters, there are still unlimited possibilities for a logarithmic transformation. We need to define some constraints for logarithmic transformations to reshape the input data distribution as close to a Gaussian distribution as possible.

**Finding optimal logarithmic transformation parameters**

One way to reframe the Gaussian reshaping problem is to simplify the shape of a data distribution into three anchor points: the left (L), the right (R), and the peak (P) (Figure A-3a). In our implementation, to avoid the effects of the extreme values, we retain only 95% of the data distribution by defining \( x_L \) as the 2.5% percentile instead of the minimum value and \( x_R \) as the 97.5% percentile instead of the maximum value. The goal of the Gaussian reshaping process is that after the logarithmic transformation, the transformed left and right anchor points are symmetric at approximately zero, whereas the peak of the transformed distribution is located exactly at zero:

\[
\begin{align*}
y_L &= -y_R \\
y_P &= 0
\end{align*}
\]

(A-12)

Substituting equation A-11 into equations A-12, we have

\[
\begin{align*}
c \{ \ln[b(x_L + a)] \} &= -c \{ \ln[b(x_R + a)] \} \\
c \{ \ln[b(x_P + a)] \} &= 0
\end{align*}
\]

(A-13)

Using the derived parameters in equation bracket A-13, we expect the transformed data distribution to be a nice symmetrical one (Figure A-3b). Instead, we end up with a slightly skewed distribution (Figure A-3c). As logarithmic transformation reshapes the data distribution, it also moves the relative position of the peak! In other words, the peak of the transformed distribution is
not at the same position as the peak of the original distribution. Therefore, we need to compute the transformation parameters iteratively by recomputing the peak anchor point of the original distribution \(x_P\) from the peak of the transformed distribution \(y_P\) at every iteration \(j\):

\[
x^{(j)}_P = \frac{\exp(y^{(j)}_P)}{b^{(j)}} - a^{(j)}. \tag{A-14}
\]

As we soon find out, reality is far from expectation. Using equation A-14 in an iterative manner, the peak of the transformed distribution does not converge in some cases, but it instead “jumps” left and right alternatively, and it may diverge in some extreme cases. Our final remedy to this peak-jumping issue is to compute the average peak of all iterations up to the current \(j\)th iteration:

\[
x^{(j)}_P = \frac{1}{j+1} \left\{ x^{(0)}_P + \sum_{m=1}^{j} \exp(y^{(m)}_P) - a^{(m)} \right\}. \tag{A-15}
\]

where \(x^{(0)}_P\) is the actual peak of the original data distribution. Although equation A-15 does not provide a mathematically accurate convergence of the peak anchor point, the computation is relatively fast, is guaranteed to converge, and results in a good approximation to a Gaussian distribution. In all of our tests, with only \(M = 100\) iterations, the peak was able to converge within at least four to five significant digits. With approximately 3 million training samples in our project, it took only 5 s to finish 100 iterations, without parallelization, using Fortran code.

Even though the peak of the reshaped distribution is centered at approximately zero, the mean of the transformed distribution is not necessarily at the same position as the peak. Therefore, at the end of the iterative computation, we need to add another term to the linear scale factor \(b\) in equation bracket A-13 to shift the mean of the reshaped distribution to zero:

\[
\tilde{b} = b^{(M)} \exp(-\mu^{(M)}_{\log}) = \frac{\exp(-\mu^{(M)}_{\log})}{x^{(M)}_P + a^{(M)}}, \tag{A-16}
\]

where \(M\) is the last iteration, and \(\mu^{(M)}_{\log}\) is the mean of the last iteration’s result:

\[
\mu^{(M)}_{\log} = \frac{\sum_{n=1}^{N} \ln(b^{(M)}(x_n + a^{(M)}))}{N}. \tag{A-17}
\]

Note that in equation bracket A-13, there is no solution for the logarithmic scale factor \(c\). Instead, \(c\) is defined as the reciprocal of the standard deviation of the reshaped data, to ensure that the final transformed data have a standard deviation of one. In addition, if the distribution is flipped (i.e., the linear scale factor \(\tilde{b}\) is negative), then the sign of the logarithmic scale factor \(c\) needs to be reversed as well:

\[
c = \begin{cases} \frac{1}{\sigma^{(M)}_{\log}} & \text{if } \tilde{b} > 0 \smallskip \\ \text{undefined} & \text{if } \tilde{b} = 0 \smallskip \\ -\frac{1}{\sigma^{(M)}_{\log}} & \text{if } \tilde{b} < 0 \end{cases}. \tag{A-18}
\]

where \(\sigma^{(M)}_{\log}\) is the standard deviation of the last iteration’s result:

\[
\sigma^{(M)}_{\log} = \sqrt{\frac{\sum_{n=1}^{N} \{ \ln(b^{(M)}(x_n + a^{(M)})) - \mu^{(M)}_{\log} \}^2}{N}}. \tag{A-19}
\]

Using equations A-11, A-13, A-16, and A-18, we summarize the logarithmic transformation in the following pseudocode:

Function \(\text{log}\_\text{transform}(x)\)

• sort(x)
• \(x_P = \text{peak}(x)\)
• \(x_L = \text{percentile}_{2.5}(x)\)
• \(x_R = \text{percentile}_{97.5}(x)\)
• \(\text{sum} = x_P\)
• Loop \(j = 1\) to \(100\):
  • \(a = (x_P - 2\times x_L)/(x_L + x_R - 2\times x_P)\)
  • \(b = 1.0/\text{mean}(x)\)
  • If \(b = 0\) or \(b = \pm\infty\) or \(a = \pm\infty\):
    • Fall back to z-score:
      • \(a = -\text{mean}(x)\)
      • \(b = 1.0/\text{standard\_deviation}(x)\)
      • \(y = b^a(x + a)\)
      • Return \(y, a, b\)
  • End If
  • \(y = \ln(b^a(x + a))\)
  • \(y_P = \text{peak}(y)\)
  • \(\text{sum} = \text{sum} + \exp(y_P)/b - a\)
  • \(x_P = \text{sum}/(j+1)\)
• End loop \(j\)
• \(b = b^a \exp(-\text{mean}(y))\)
• \(c = 1.0/\text{standard\_deviation}(y)\)
• If \((b < 0)\):
  • \(c = c\)
  • \(y = c \times \ln(b^a(x + a))\)
  • Return \(y, a, b, c\)
End function \(\text{log}\_\text{transform}\)

Appendix B

Finding the peak (or the mode) of a data distribution

As described in Appendix A, one of the critical tasks in the computation of logarithmic transformation parameters is to find the peak of a data distribution. The peak (in statistics, more precisely denoted as the mode) of a data distribution is the location of greatest data density. Usually, the process of determining the peak of a data distribution involves segregating the di-
distribution into multiple histogram columns and then locating the highest-valued column (Figure B-1). The precision of the peak depends on the width of each histogram column. Furthermore, if there is a spike in a data distribution (where multiple data points have exactly the same value, such as zero samples belonging to a dead trace or a muted zone), this method will most likely pick the spike instead of the desired peak.

In this appendix, we present an alternative method to find the peak of a data distribution based on the binary search algorithm described by Lin (2019). The process starts by limiting the search between the 15% percentile and the 85% percentile of a distribution with the assumption that, in most cases, the peak falls within such a range around the distribution center (Figure B-2). We then divide this range into two halves that have exactly the same width. Next, we choose the denser half (i.e., the half representing the greater number of data points) and start dividing it into another two halves, and repeat. The assumption is that the peak of a distribution will always reside in the denser half; thus, by successively choosing and dividing the denser half, eventually we will come down to two halves with exactly one data point in each. At the end, we take the average of these two points to represent the peak of the distribution.

As with every other algorithm, there are pros and cons to this binary-search method to find the peak of a data distribution. One of the algorithm's positive features is that it is highly precise, on the order of one half the distance between two adjacent data points. It is inherently fast because binary search algorithms have a worst-case-scenario efficiency of $O(\log N)$ (Lin, 2019). And because it is limiting the search to the center part of the distribution, it is robust against zero-value spikes caused by dead traces and muted zones, which are usually located at the extreme left of the distributions of nonnegative seismic attributes, provided that the spikes do not constitute a significant portion of the data (i.e., less than 15%).

In terms of negative features, the algorithm requires the data to be sorted in ascending order (although in logarithmic transformation, we need to sort the data anyway to estimate the 2.5% percentile and 97.5% percentile for the left and right anchor points). A more serious issue arises when, after dividing a denser half into two, the resulting two halves have exactly the same data count of a large quantity (Figure B-3). In this case, the midpoint between the two halves is set as the peak of the distribu-

---

**Figure B-1.** (a) Illustration of the peak (or statistical mode) of a distribution. The traditional procedure to find the peak is to construct histogram columns and locate the column with the greatest number of points (b). The precision of the peak is determined by the width of a histogram column. However, if there is a spike in the distribution, such as zero-value samples belonging to dead traces and muted zones, the highest histogram column could represent the spike instead of the actual peak (c).

**Figure B-2.** Illustration of our method to find the peak of a data distribution using a binary search algorithm. First, we limit the search between the 15th and 85th percentiles of the distribution to avoid the effect of spikes at the extreme left and right, assuming that the peak is most likely located within this range near the distribution center. We divide this range into two halves of the same width and count the data points residing in each half. We then choose the half with a greater data count, divide it into another two halves, and start counting data points within these new halves again. The process is repeated until there is exactly one data point in each half, at which time we define the peak of the distribution as the average of the last two data points.

**Figure B-3.** In some rare occurrences, when a data distribution has more than one cluster and the total number of data points is relatively small, it is possible that after a division, the two halves have exactly the same data count of a large quantity. If this happens, the binary search would stop, and the midpoint of the two halves is considered to be the peak, while the actual peak may reside in one of the two halves. A partial solution to this issue is to further divide two halves into four quarters and find the quarter with the maximum data count, assuming that the peak belongs to the densest quarter.
tion and the binary search would stop, even though the actual peak might still be located within either of the two halves. This is rare, but not impossible, and it becomes more likely when the total number of data points is relatively small (on the order of thousands or fewer) and the data distribution contains more than one cluster of similar sizes. To fix this issue, we need to further divide the two halves into four quarters and choose the quarter with the maximum data count, assuming that the four quarters have different data counts and the actual peak of the distribution is in the biggest quarter.

We summarize our peak binary search method in the following pseudocode:

Function peak_binary_search(x)
• n = length(x)
• If not_sorted(x): sort(x)
• j_mid = index_percentile_15(x)
• j_right = index_percentile_85(x)
• left_count = 0
• right_count = j_right - j_mid + 1
• Loop while right_count > 1:
  • Loop while left_count ≠ right_count:
    ➢ If left_count < right_count:
      ✓ j_left = j_mid
      ➢ Else
      ✓ j_right = j_mid - 1
      ➢ End If
    ➢ mid_val = 0.5 * (x(j_left) + x(j_right))
    ➢ Loop j_mid = j_left + 1 to j_right:
      ✓ If x(j_mid) >= mid_val:
      ✓ Exit loop j_mid
    ➢ End loop j_mid
    ➢ left_count = j_mid - j_left
    ➢ right_count = j_right - j_mid + 1
  • End loop while left_count ≠ right_count
• If right_count > 1:
  ➢ mid_left_val = 0.5 * (x(j_left) + x(j_mid - 1))
  ➢ mid_right_val = 0.5 * (x(j_mid) + x(j_right))
  ➢ Loop j_mid_left = j_left + 1 to j_mid - 1:
    ✓ If x(j_mid_left) >= mid_left_val:
    ✓ Exit loop j_mid_left
  ➢ End loop j_mid_left
  ➢ Loop j_mid_right = j_mid + 1 to j_right:
    ✓ If x(j_mid_right) >= mid_right_val:
    ✓ Exit loop j_mid_right
  ➢ End loop j_mid_right
  ➢ left_1_count = j_mid_left - j_left
  ➢ left_2_count = j_mid - j_mid_left
  ➢ right_1_count = j_mid_right - j_mid
  ➢ right_2_count = j_right - j_mid_right + 1
  ➢ max_left = max(left_1_count, left_2_count)
  ➢ max_right = max(right_1_count, right_2_count)
• If max_left < max_right:
  ✓ j_left = j_mid
  ✓ j_mid = j_mid_right
  ✓ mid_val = mid_right_val
  ✓ left_count = right_1_count
  ✓ right_count = right_2_count
  ➢ Else:
    ✓ j_right = j_mid - 1
    ✓ j_mid = j_mid_left
    ✓ mid_val = mid_left_val
    ✓ left_count = left_1_count
    ✓ right_count = left_2_count
  ➢ End If max_left < max_right
• End if right_count > 1
• End loop while right_count > 1
• Return mid_val
End function peak_binary_search

Appendix C
Data clipping in crossplot

This appendix shows the effect of different data clipping strategies on the 2D crossplot between two components where we will use the crossplot of two z-score SOM output components as an example. Figure C-1 shows the SOM crossplot image without any data clipping. The entire range from the minimum value to the maximum value of each component is used in the construction of the crossplot. We observe that the image is very pale (i.e., it has a very low color contrast) and is

![Figure C-1. A 2D color crossplot of two output SOM components using z-score normalization, without any data clipping. The crossplot is mostly cyan because the data distribution is heavily skewed to the upper left corner of the 2D histogram, which corresponds to a variety of cyan colors in the 2D color table. We can barely see any geologic detail due to the low color contrast.](image-url)
mostly cyan. We can barely see any geologic detail. This lack of contrast is because the data are mapped mostly to the upper left corner of the 2D histogram, which corresponds to a variety of cyan colors.

More careful inspection of the 2D histogram of the full-range SOM crossplot in Figure C-1 reveals that most of the data distribution falls within −2.6 to 3.0 for SOM component 1 and −3.0 to 3.0 for SOM component 2. By clipping the data using these ranges, we end up with Figure C-2. The color contrast is good, but there are too many white pixels representing clipped data values (or “outliers”).

Therefore, we slightly increase the ranges of SOM components for the crossplot, from −2.6 to 4.0 for SOM component 1 and −3.5 to 3.5 for SOM component 2 (as indicated in Appendix E), to reduce the number of white pixels while maintaining most of the color contrast. The result is Figure C-3, which is exactly the same with Figure 14a. This strategy does require some trial-and-error attempts to configure data clipping to obtain the (subjectively) best display of the 2D crossplot.

Appendix D

Mathematical description of pnn exhaustive search

In this appendix, we summarize the mathematical equations of the PNN algorithm and the exhaustive search to determine the best combination of input attributes and the corresponding PNN parameters. Following Specht (1990), Masters (1995), and Lubo-Robles et al. (2021), the PNN algorithm estimates the probability density function of each class in the input supervised training data, most commonly by using the Gaussian function. The estimated density function of the \(M\)-element attribute data vector \(x_j\) at voxel \(j\) with respect to class \(k\) is

\[
g_k(x_j) = \frac{1}{N_k} \sum_{n=1}^{N_k} \exp \left( -\frac{1}{2r^2} \sum_{m=1}^{M} (x_{jm} - u_{nmk})^2 \right). \quad (D-1)\]

where \(N_k\) is the number of training samples assigned to class \(k\), \(M\) is the number of input attributes, \(x_{jm}\) is the value of the \(m\)th attribute of the normalized input data vector at voxel \(j\), \(u_{nmk}\) is the value of the \(m\)th attribute of the \(n\)th normalized training sample belonging to class \(k\), and \(r\) is the smoothing parameter, which is also the only PNN parameter that requires further optimization. The term \(g_k(x_j)\) is then normalized by the sum of the estimated density functions for the input data vector \(x_j\) with respect to all \(K\) classes, resulting in the probability of sample \(x_j\) belonging to class \(k\) as

\[
P_k(x_j) = \frac{g_k(x_j)}{\sum_{q=1}^{K} g_q(x_j)}. \quad (D-2)\]

The probability of an input sample \(x_j\) belonging to class \(k\) depends on the proximity of \(x_j\) to all of the training samples \(u_{nk}\) belonging to class \(k\). We define the class exhibiting the highest probability as the predicted class of the input sample \(x_j\). A PNN model is just a collection of the smoothing parameters \(r\), the normalized training samples \(u_{nk}\), and the normalization parameters. Note that for a per-class normalization scheme, we calculate the normalization parameters for class \(k\) using only the training samples belonging to class \(k\). Then, we normalize the input-to-be-classified samples \(x\) and the training samples \(u_{nk}\) using class \(k\’s\) unique normalization parameters before the probability computation.

To perform an exhaustive search using the PNN algorithm, Lubo-Robles et al. (2021) iterate through all possible combinations of input attributes. For each

Figure C-2. A 2D color crossplot of two output SOM components using z-score normalization, with strong data clipping. The number of white pixels (i.e., clipped data points) is greatly reduced, and the color contrast increased over Figure C-1. This figure is the one displayed as Figure 14a.

Figure C-3. A 2D color crossplot of two output SOM components using z-score normalization, with moderate data clipping. The number of white pixels (i.e., clipped data points) is greatly reduced, and the color contrast increased over Figure C-1. This figure is the one displayed as Figure 14a.
combination of input attributes, the optimal smoothing parameter $r$ is the one that yields the lowest global validation error:

$$E = \frac{1}{N_v} \sum_{j=1}^{N_v} e_k(x_j),$$

(D-3)

where $N_v$ is the number of validation samples (i.e., blind-test samples), and $e_k(x)$ is a continuous error function (Masters, 1995) defined as

$$e_k(x) = [1 - P_k(x)]^2 + \sum_{h \neq k} [P_h(x)]^2.$$  \hspace{1cm} (D-4)

Lubo-Robles et al. (2021) find that a combination of coherence, GLCM contrast, total energy, and dip deviation attributes, with the smoothing parameter $r = 0.1$, yields the lowest global validation error of 0.01689. We follow Lubo-Robles et al. (2021) and apply their PNN exhaustive search for each of our normalization schemes and find that for different normalization schemes, there are different validation errors and hence a different best combination of input attributes with different smoothing parameters (Table D-1).

### Appendix E

#### Lists of computing parameters

For case study 1, Tables E-1 to E-6 list parameters of attribute computations, including dip attributes (which are required as input to define the orientation of the structure for almost of the other attributes), dip filtering, coherence, structural curvature, GLCM attributes, and spectral decomposition. Table E-7 lists parameters of training data extraction. Tables E-8 to E-11 list parameters of individual algorithm’s modeling and plotting (including PCA, ICA, SOM, and GTM).

For case study 2, Tables E-12 to E-14 list parameters of attribute computation, including dip, dip filtering, and coherence attributes. Tables E-15 and E-16 list parameters of Kuwahara median filter. Table E-17 lists parameters of training data extraction and PNN modeling.

### Table D-1. Optimal combination of input attributes of each normalization scheme, together with the corresponding PNN smoothing parameter $r$ and validation error.

<table>
<thead>
<tr>
<th>Normalization scheme</th>
<th>Optimal combination of input attributes</th>
<th>PNN smoothing parameter $r$</th>
<th>Validation error (lower is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logarithmic bulk</td>
<td>Coherence, GLCM contrast, GLCM entropy</td>
<td>2.1</td>
<td>0.047</td>
</tr>
<tr>
<td>Logarithmic per-facies</td>
<td>Coherence, GLCM contrast</td>
<td>0.6</td>
<td>0.031</td>
</tr>
<tr>
<td>Z-score bulk</td>
<td>Coherence, GLCM contrast</td>
<td>1.3</td>
<td>0.037</td>
</tr>
<tr>
<td>Z-score per-facies</td>
<td>Coherence, Most-positive curvature</td>
<td>1.9</td>
<td>0.024</td>
</tr>
</tbody>
</table>

Note: The coherence attribute is found in all four normalization schemes’ optimal combination of input attributes.

### Table E-1. Parameters of dip computation in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Semblance search</td>
</tr>
<tr>
<td>Maximum angle searched (degree)</td>
<td>15</td>
</tr>
<tr>
<td>Search angle increment (degree)</td>
<td>3</td>
</tr>
<tr>
<td>Time-to-depth conversion velocity (m/s)</td>
<td>4000</td>
</tr>
<tr>
<td>Vertical window half height (s)</td>
<td>0.02</td>
</tr>
<tr>
<td>Inline window radius (m)</td>
<td>12.5</td>
</tr>
<tr>
<td>Crossline window radius (m)</td>
<td>25.0</td>
</tr>
</tbody>
</table>

### Table E-2. Parameters of dip filtering in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Lower upper middle (LUM)</td>
</tr>
<tr>
<td>Lower-upper-median percentile</td>
<td>20</td>
</tr>
<tr>
<td>Vertical window half height (s)</td>
<td>0.02</td>
</tr>
<tr>
<td>Inline window radius (m)</td>
<td>12.5</td>
</tr>
<tr>
<td>Crossline window radius (m)</td>
<td>25.0</td>
</tr>
</tbody>
</table>

### Table E-3. Parameters of coherence computation in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical window half height (s)</td>
<td>0.02</td>
</tr>
<tr>
<td>Inline window radius (m)</td>
<td>12.5</td>
</tr>
<tr>
<td>Crossline window radius (m)</td>
<td>25.0</td>
</tr>
<tr>
<td>Similarity power</td>
<td>2.0</td>
</tr>
<tr>
<td>Low-cut filter roll-off $f_{\text{low}}$ (Hz)</td>
<td>5</td>
</tr>
<tr>
<td>High-cut filter roll-off $f_{\text{high}}$ (Hz)</td>
<td>100</td>
</tr>
</tbody>
</table>
### Table E-4. Parameters of curvature computation in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curvature type</td>
<td>Structural</td>
</tr>
<tr>
<td>Filter corner point ( \lambda_1 ) (m)</td>
<td>22016.2</td>
</tr>
<tr>
<td>Filter corner point ( \lambda_2 ) (m)</td>
<td>800</td>
</tr>
<tr>
<td>Filter corner point ( \lambda_3 ) (m)</td>
<td>400</td>
</tr>
<tr>
<td>Filter corner point ( \lambda_4 ) (m)</td>
<td>200</td>
</tr>
<tr>
<td>Filter weight ( w_1 )</td>
<td>1</td>
</tr>
<tr>
<td>Filter weight ( w_2 )</td>
<td>0.666</td>
</tr>
<tr>
<td>Filter weight ( w_3 )</td>
<td>0.333</td>
</tr>
<tr>
<td>Filter weight ( w_4 )</td>
<td>0</td>
</tr>
<tr>
<td>Constant multiplier of curvature</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum operator radius (m)</td>
<td>1000</td>
</tr>
<tr>
<td>Vertical compression factor</td>
<td>0.25</td>
</tr>
<tr>
<td>Operator truncation value</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### Table E-5. Parameters of GLCM computation in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical window half height (s)</td>
<td>0.004</td>
</tr>
<tr>
<td>Inline window radius (m)</td>
<td>25.0</td>
</tr>
<tr>
<td>Crossline window radius (m)</td>
<td>50.0</td>
</tr>
<tr>
<td>Number of gray levels</td>
<td>33</td>
</tr>
</tbody>
</table>

### Table E-6. Parameters of spectral decomposition in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectral balancing factor (%)</td>
<td>4</td>
</tr>
<tr>
<td>Bluing exponent</td>
<td>0</td>
</tr>
<tr>
<td>Line and common-depth-point decimation</td>
<td>5</td>
</tr>
<tr>
<td>Ormsby filter corner point ( f_1 ) (Hz)</td>
<td>5</td>
</tr>
<tr>
<td>Ormsby filter corner point ( f_2 ) (Hz)</td>
<td>10</td>
</tr>
<tr>
<td>Ormsby filter corner point ( f_3 ) (Hz)</td>
<td>100</td>
</tr>
<tr>
<td>Ormsby filter corner point ( f_4 ) (Hz)</td>
<td>120</td>
</tr>
<tr>
<td>Continuous wavelet transform (CWT)</td>
<td>0.26051</td>
</tr>
<tr>
<td>mother wavelet bandwidth (Hz)</td>
<td></td>
</tr>
<tr>
<td>Temporal taper (s)</td>
<td>0.1</td>
</tr>
<tr>
<td>Percentile excluded in spectral shape</td>
<td>0.15</td>
</tr>
<tr>
<td>Lowest output frequency ( f_{low} ) (Hz)</td>
<td>10</td>
</tr>
<tr>
<td>Highest output frequency ( f_{high} ) (Hz)</td>
<td>100</td>
</tr>
<tr>
<td>Output frequency increment ( \Delta f ) (Hz)</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table E-7. Parameters of training data extraction in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline start</td>
<td>1005</td>
</tr>
<tr>
<td>Inline end</td>
<td>1570</td>
</tr>
<tr>
<td>Inline increment</td>
<td>1</td>
</tr>
<tr>
<td>Crossline start</td>
<td>4645</td>
</tr>
<tr>
<td>Crossline end</td>
<td>6395</td>
</tr>
<tr>
<td>Crossline increment</td>
<td>1</td>
</tr>
<tr>
<td>Vertical boundary type</td>
<td>About a horizon</td>
</tr>
<tr>
<td>Horizon</td>
<td>Late Tertiary</td>
</tr>
<tr>
<td>Window above horizon (s)</td>
<td>0.004</td>
</tr>
<tr>
<td>Window below horizon (s)</td>
<td>0.004</td>
</tr>
<tr>
<td>Vertical increment (s)</td>
<td>0.004</td>
</tr>
</tbody>
</table>

### Table E-8. Parameters of PCA in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of principal components</td>
<td>3</td>
</tr>
<tr>
<td>Minimum display percentile (%)</td>
<td>5</td>
</tr>
<tr>
<td>Maximum display percentile (%)</td>
<td>95</td>
</tr>
</tbody>
</table>

### Table E-9. Parameters of ICA in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of independent components</td>
<td>3</td>
</tr>
<tr>
<td>Error tolerance</td>
<td>1.0E-6</td>
</tr>
<tr>
<td>Maximum number of iterations</td>
<td>500</td>
</tr>
<tr>
<td>Minimum display percentile (%)</td>
<td>5</td>
</tr>
<tr>
<td>Maximum display percentile (%)</td>
<td>95</td>
</tr>
</tbody>
</table>
### Table E-10. Parameters of SOM in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of prototype vectors (maximum number of classes)</td>
<td>256</td>
</tr>
<tr>
<td>Number of standard deviations along the first two eigenvector directions to define the initial $16 \times 16 = 256$ prototype vectors on the manifold</td>
<td>±4</td>
</tr>
<tr>
<td>Initial neighborhood scale</td>
<td>1.2</td>
</tr>
<tr>
<td>Maximum number of training iterations</td>
<td>50</td>
</tr>
<tr>
<td>Grid spacing</td>
<td>150</td>
</tr>
<tr>
<td>Z-score clipping range of SOM axis 1</td>
<td>−2.6 to 4.0</td>
</tr>
<tr>
<td>Z-score clipping range of SOM axis 2</td>
<td>−3.5 to 3.5</td>
</tr>
</tbody>
</table>

### Table E-11. Parameters of GTM in case study 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of samples in 2D latent space</td>
<td>256</td>
</tr>
<tr>
<td>Number of basis functions</td>
<td>144</td>
</tr>
<tr>
<td>Relative width of basis functions</td>
<td>0.5</td>
</tr>
<tr>
<td>Weight regularization factor</td>
<td>0.05</td>
</tr>
<tr>
<td>Number of training iterations</td>
<td>50</td>
</tr>
</tbody>
</table>

### Table E-12. Parameters of dip computation in case study 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>GST</td>
</tr>
<tr>
<td>Time-to-depth conversion velocity (ft/s)</td>
<td>10000</td>
</tr>
<tr>
<td>Vertical window half height (s)</td>
<td>0.02</td>
</tr>
<tr>
<td>Inline window radius (ft)</td>
<td>82.5</td>
</tr>
<tr>
<td>Crossline window radius (ft)</td>
<td>82.5</td>
</tr>
</tbody>
</table>

### Table E-13. Parameters of dip filtering in case study 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>LUM</td>
</tr>
<tr>
<td>Lower-upper-median percentile</td>
<td>20</td>
</tr>
<tr>
<td>Vertical window half height (s)</td>
<td>0.02</td>
</tr>
<tr>
<td>Inline window radius (ft)</td>
<td>82.5</td>
</tr>
<tr>
<td>Crossline window radius (ft)</td>
<td>82.5</td>
</tr>
</tbody>
</table>

### Table E-14. Parameters of coherence computation in case study 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical window half height (s)</td>
<td>0.02</td>
</tr>
<tr>
<td>Inline window radius (ft)</td>
<td>82.5</td>
</tr>
<tr>
<td>Crossline window radius (ft)</td>
<td>82.5</td>
</tr>
<tr>
<td>Similarity power</td>
<td>2.0</td>
</tr>
<tr>
<td>Low-cut filter roll-off $f_{\text{low}}$ (Hz)</td>
<td>5</td>
</tr>
<tr>
<td>High-cut filter roll-off $f_{\text{high}}$ (Hz)</td>
<td>100</td>
</tr>
</tbody>
</table>

### Table E-15. Parameters of median smoothing.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of iterations</td>
<td>5</td>
</tr>
<tr>
<td>Vertical window half height (s)</td>
<td>0.002</td>
</tr>
<tr>
<td>Inline window radius (ft)</td>
<td>165.0</td>
</tr>
<tr>
<td>Crossline window radius (ft)</td>
<td>165.0</td>
</tr>
</tbody>
</table>

### Table E-16. Parameters of Kuwahara filtering.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical window taper (%)</td>
<td>20</td>
</tr>
<tr>
<td>Vertical window half height (s)</td>
<td>0.004</td>
</tr>
<tr>
<td>Inline window radius (ft)</td>
<td>82.5</td>
</tr>
<tr>
<td>Crossline window radius (ft)</td>
<td>82.5</td>
</tr>
</tbody>
</table>

### Table E-17. Parameters of PNN training data extraction and modeling.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline increment</td>
<td>5</td>
</tr>
<tr>
<td>Crossline increment</td>
<td>5</td>
</tr>
<tr>
<td>Vertical increment (s)</td>
<td>0.02</td>
</tr>
<tr>
<td>Number of salt samples</td>
<td>5480</td>
</tr>
<tr>
<td>Number of not-salt samples</td>
<td>6368</td>
</tr>
<tr>
<td>Smoothing parameter $r$</td>
<td>1.0</td>
</tr>
</tbody>
</table>
References


Torres, E., R. Slatt, K. J. Marfurt, L. Infante, and L. Castillo, 2018, Identification of potential lacustrine stratigraphic intervals in the Woodford Shale, Oklahoma, using multiattribute 3-D seismic displays and a supervised neural

Downloaded 08/12/21 to 68.97.118.233. Redistribution subject to SEG license or copyright; see Terms of Use at http://library.seg.org/page/policies/terms


Wallet, B., and T. Ha, 2019, Deep learning method for latent space analysis: Middle East Oil and Gas Show and Conference, SPE, Extended Abstracts.


Thang Ha received a B.S. and an M.S. in geophysics from the University of Oklahoma. He is currently a Ph.D. student in geophysics, advised by Kurt Marfurt, at the University of Oklahoma. He is a member of SEG, AAPG, EAGE, and the Geophysical Society of Oklahoma City. His research interests include seismic processing, 3D seismic attributes, data conditioning, and automatic classification of geologic facies via machine learning.

David Lubo-Robles received a B.S. in geophysical engineering from Simon Bolivar University, Venezuela, and an M.S. in geophysics from the University of Oklahoma under Kurt J. Marfurt. He is a student member of SEG and currently is pursuing a Ph.D. in geophysics at the University of Oklahoma, studying under Kurt J. Marfurt and Matthew Pranter. His research interests include the development and application of modern machine learning and pattern recognition techniques, together with quantitative interpretation skills, including prestack inversion and seismic attribute analysis to delineate geologic features amenable to hydrocarbon accumulation.

Kurt J. Marfurt began his geophysical career teaching geophysics and contributing to an industry-supported consortium on migration, inversion, and scattering (project MIDAS) at Columbia University’s Henry Krumb School of Mines in New York City. In 1981, he joined Amoco’s Tulsa Research Center and spent the next 18 years leading research efforts in modeling, migration, signal analysis, basin analysis, seismic attribute analysis, reflection tomography, seismic inversion, and multicomponent data analysis. In 1999, he joined the University of Houston as a professor in the Department of Geosciences and as a director of the Allied Geophysics Laboratories. He is currently a member of the Geophysical Societies of Tulsa and Houston, SEG, EAGE, AAPG, AGU, and SIAM, and he serves as deputy editor-in-chief of Interpretation. His current research activity includes prestack imaging, velocity analysis and inversion of converted waves, computer-assisted pattern recognition of geologic features on 3D seismic data, and interpreter-driven seismic processing. His research interests include seismic signal analysis, 3D seismic attributes, seismic velocity analysis, subsurface imaging, and multicomponent data analysis.

Bradley Wallet’s expertise is in quantitative reservoir analysis and characterization using seismic attributes and statistical methods. He has a strong history of developing cutting-edge solutions and transitioning them to fielded applications for seismic interpretation, remote sensing, and automatic target recognition, with 17 years working on the cutting edge of statistical machine learning, data mining, and pattern recognition.