SEMISUPERVISED MACHINE LEARNING ALGORITHMS AND THEIR APPLICATION TO GEOSCIENCE CLASSIFICATION PROBLEMS

by

©Michael W. Dunham

B.Sc. Colorado School of Mines, 2014
M.Sc. Memorial University of Newfoundland, 2017

A thesis submitted to the School of Graduate Studies
in partial fulfillment of the requirements for the
degree of Doctor of Philosophy

Department of Earth Sciences
Memorial University of Newfoundland

October 2022

St. John's, Newfoundland and Labrador, Canada

"...doing more with less"
-R. Buckminster Fuller-

Abstract

In recent years, many disciplines have been challenged with trying to extract valuable information from large datasets efficiently. Technological advances have improved data storage capabilities and how data can be obtained (e.g., real-time data). Manually interpreting data that are exponentially growing in volume has obvious management and analysis challenges. Machine learning algorithms recognize patterns in data and assign repetitive patterns to similar categories. This process automates pattern recognition in data and allows meaningful information to be extracted in an efficient manner.

For many machine learning problems, there are sufficient labeled data to train a wide range of algorithms. Some applications, such as image classification and speech recognition, have large labeled datasets readily available. However, in several geoscience-related problems, labeled data are generally obtained by sampling the Earth in some manner (e.g., drilling wells, field sampling, etc.), which is not trivial due to cost and logistical factors. As such, many earth science-related machine learning problems have limited labeled data. Supervised machine learning algorithms are prone to overfitting when labeled data are scarce, but semisupervised approaches are designed for these problems because unlabeled data are also used to inform the learning process.

Three geoscience applications inherently challenged with limited training data are well-log classification, seismic classification, and bedrock-lithology mapping. I apply various semisupervised algorithms to these three geoscience problems and determine if semisupervised algorithms can perform better than supervised methods and under what conditions. The semisupervised methods that I consider are self-training, label propagation, and semisupervised Gaussian mixture models. I consider several supervised methods in my work, but the most prevalent are the gradient boosting decision tree methods. The results demonstrate that semisupervised methods can outperform their supervised counterparts for each of the geoscience applications, but not in all situations. Nonetheless, semisupervised methods are rarely considered for many geoscience disciplines, which is demonstrated by the lack of published examples in the literature. The outcomes of this work are raising the awareness of semisupervised methods by showing their applicability to different geoscience problems and making recommendations on how and when to use these tools.

General Summary

The objective of many earth science disciplines is to better understand how geologic information is distributed beneath the ground or across the surface of the Earth. Earth scientists can gain insights into this geologic information by collecting data (e.g., geophysics, geochemistry, geologic observations, etc.) and using these data to generate models. Traditional modeling techniques use physical relationships, or governing rules (e.g., Maxwell's equations, Newton's laws, etc.) to generate models from our observed data. Performing traditional modeling is not trivial if the physical relationships are complex, and in some situations, it may not even be possible because the *explicit* physical relationships are unknown. In these scenarios where the governing rules are unknown, we may still have constraints on what the model should be in certain locations due to observations made by a geoscientist (e.g., information gathered from drilled wells, geologic field sampling, etc.). Machine learning techniques, namely supervised learning (SL) methods, can make use of these constraints by learning an *implicit* relationship between the data and the model; this learned relationship can then be used to make model predictions where there are no constraints. However, if the number of constraints is minimal, then the relationship learned by SL methods may be unstable. Another type of machine learning, semisupervised learning (SSL), is specifically designed for problems with limited constraints and is shown to be more robust than SL in these situations.

Many geoscience applications have no explicitly known relationships to leverage traditional modeling techniques and are inherently challenged with limited constraints. These situations are those that SSL algorithms are specifically designed for, but SSL remains largely unexplored for many geoscience applications. In this thesis, I apply SSL techniques to three different geoscience problems (well-log classification, seismic classification, and bedrock-lithology mapping) and determine if SSL is more effective than SL. The results indicate that SSL generally produces better earth models than SL as hypothesized, but there are exceptions. The impact of this work is showing the usefulness of SSL methods across different problems and providing recommendations on how and when to use these tools.

Acknowledgements

I would like to begin by thanking my supervisors, Alison Malcolm and Kim Welford, for supporting me on this journey. This PhD project was unique in the sense that its specific scope was not established beforehand. Rather than my supervisors formulating the project for me, it was quite the opposite; I identified a research topic that seemed interesting to me and proposed it to them. I was given the opportunity to make this project my own and take it in the direction I did, and I sincerely value the trust they put in me to do so.

My gratitude is also extended to many individuals and entities that made various parts of this PhD possible. First and foremost, funding resources enable research projects to happen, and I am thankful to Chevron, the Natural Sciences and Engineering Research Council of Canada (NSERC), and InnovateNL for financially supporting this work. Various commercial software packages were used to conduct this work, and I would like to acknowledge Petrosys (GLOBEClaritas), Ikon Science (RokDoc), and Schlumberger (Petrel) for providing academic licenses for their software. Computing was an important component of this thesis, and I am thankful to the Digital Research Alliance of Canada (formerly Compute Canada) for their resources. Mike Fehler (Senior researcher, MIT) was instrumental in securing the additional model properties I needed for my seismic project. In regards to this seismic project, seismic modeling and processing is not a strength of mine, so I am grateful to Chris Williams, Pei Yang, Elena Jaimes-Osorio, and my supervisors for assisting me in that process. Matthew Cracknell (Faculty, University of Tasmania) was a tremendous help in my final project for pointing me towards the appropriate online resources for locating the publicly available data in Australia. I would also like to thank Goldspot Discoveries Corporation for assisting me with the unsupervised clustering in the final project as well. On a less technical note, I would like to thank the MUN Community Garden and my friend Steve Garland for allowing me to be a part of their gardens. It is important to have a healthy work-life balance, even for a PhD student, and I personally found gardening to be an excellent activity to distance myself from work when needed.

Contents

Co-Authorship Statement

The broad scope for this PhD project, applying machine learning to geoscience problems, was first established by my supervisors, Alison Malcolm and Kim Welford. I refined the scope to using semisupervised learning methods and personally determined the semisupervised methods that were used. I performed all the research, algorithm testing and development, project formulation, and numerical calculations in this thesis, and wrote the first drafts of all the papers. The contributions made to each chapter are summarized below.

Chapter 3: Improved well-log classification using semisupervised label propagation and self-training, with comparisons to popular supervised algorithms

This chapter applies two semisupervised methods (label propagation and self-training) and multiple supervised techniques to a real well-log dataset made publicly available by the Kansas Geological Survey. I located the well-log dataset for this study and formulated the research questions I wanted to pursue. The choices for all machine learning methods used were made by myself, and I also developed the new self-training technique. The involvement of my supervisors was providing guidance and feedback throughout the process. All manuscript contents were written by myself, and the co-authors (Alison Malcolm and Kim Welford) provided editorial guidance. This chapter is derived from the publication:

- Dunham, M.W., Malcolm, A. and Welford, J.K., 2020. Improved well-log classification using semisupervised label propagation and self-training, with comparisons to popular supervised algorithms, Geophysics, 85(1), O1–O15. https://doi.org/10.1190/geo2019-0238.1

This work was also presented at the 2019 Society of Exploration Geophysicists (SEG) Annual Meeting in San Antonio, Texas, USA.

Chapter 4: Improved well-log classification using semisupervised Gaussian mixture models and a new hyper-parameter selection strategy

In this chapter, I apply semisupervised Gaussian mixture models (ssGMM) and multiple supervised techniques to the same well-log dataset from Chapter 3. I developed the Python code that implements the

ssGMM technique, and I also optimized the code using profiling tools. The new hyper-parameter selection strategy was also a concept that I developed to try and improve the ssGMM results. All manuscript contents were written by myself, and the co-authors (Alison Malcolm and Kim Welford) provided editorial guidance. This chapter is derived from the publication:

- Dunham, M.W., Malcolm, A. and Welford, J.K., 2020. Improved well-log classification using semisupervised Gaussian mixture models and a new hyper-parameter selection strategy, Computers and Geosciences, 140, 1–12. https://doi.org/10.1016/j.cageo.2020.104501

This work was also presented at the 2020 Geological Association of Canada Newfoundland and Labrador (GAC-NL) Annual Meeting in St. John's, NL, Canada.

Chapter 5: A seismic petrophysical classification study of the 2-D SEAM model using semisupervised techniques and detrended attributes

This chapter applies the same semisupervised methods from Chapter 3 (label propagation and self-training) to a synthetic seismic dataset derived from a model made publicly available by the Society of Exploration Geophysicists. Alison Malcolm helped initiate the dialogue with one of her former colleagues (Mike Fehler), who provided me with additional properties that I needed for the model. Alison Malcolm and Elena Jaimes-Osorio supported me with the seismic modeling software Devito. Kim Welford, Chris Williams, and Pei Yang also offered assistance with the subsequent pre-stack time migration performed in GLOBEClaritas. I performed the pre-stack seismic inversion in RokDoc, and I formulated the machine learning problem and research questions for this study. All manuscript contents were written by myself, and the co-authors (Alison Malcolm and Kim Welford) provided editorial guidance. This chapter is derived from the publication:

- Dunham, M.W., Malcolm, A. and Welford, J.K., 2021. A seismic petrophysical classification study of the 2-D SEAM model using semisupervised techniques and detrended attributes, Geophysical Journal International, 227(2), 1123–1142. https://doi.org/10.1093/gji/ggab258

This work was presented at the 82nd European Association of Geoscientists & Engineers (EAGE) Annual Conference & Exhibition (virtual) and also at the 2021 GAC-NL Annual Meeting (virtual).

Chapter 6: Are seismic attributes still helpful for deep learning?

This chapter is a continuation of Chapter 5 using the same datasets, but it approaches the problem from a deep learning perspective. I formulated the research questions surrounding this study, and I also developed

the understanding needed to implement the recurrent neural networks (RNNs) used. I am the sole author of this work, but Alison Malcolm and Kim Welford provided editorial guidance. This chapter is derived from the conference proceeding:

- Dunham, M.W., 2021. Are seismic attributes still helpful for deep learning?, in SEG Technical Program Expanded Abstracts 2021, pp. 1561-1565, Society of Exploration Geophysicists.
- https://doi.org/10.1190/segam2021-3579734.1

This work was presented virtually at the 2021 International Meeting for Applied Geoscience and Energy (IMAGE).

Chapter 7: Predictive lithology mapping using semisupervised learning: practical insights using a case study from New South Wales, Australia

In this chapter, I apply label propagation, but also supervised and unsupervised methods to a geophysical dataset from Australia. I located all of the datasets for this study, and I also devised each of the machine learning scenarios. Julien Mailloux from Goldspot Discoveries Corporation provided coding assistance for the unsupervised clustering approach, but the supervised and semisupervised methods were implemented on my own. The involvement of my supervisors was providing guidance and feedback throughout the process. All manuscript contents were written by myself, and the co-authors (Alison Malcolm and Kim Welford) provided editorial guidance. This manuscript is currently under review in the journal *Geophysics*. This work was also presented at the 2022 GAC-NL Annual Meeting (virtual), and will be presented at the Prospectors and Developers Association of Canada (PDAC) 2022 Convention in Toronto, ON, Canada on June 13-15, 2022.

Chapter 1

Introduction

1.1 Motivation

In recent years, many disciplines have been challenged with trying to efficiently extract meaning, or value, out of large datasets. Technological advances have improved data storage capabilities as well as how data can be obtained (e.g., real-time data). Manually interpreting data that are exponentially growing in volume has obvious management and analysis challenges. Another challenge facing many scientists is derived from a shortcoming in standard modeling techniques. In many scientific problems, we try to explain the phenomena we observe in our data using a model. The simplistic representation of the forward modeling problem is expressed as $d = \mathbf{G}m$ where d is the observed dataset (a vector of length n), m is the model (a vector of length r), and $\mathbf{G}$ is the forward operator that contains the *explicit* physical relationships or mathematics needed to relate the observed data to the model (a matrix of size $n \times r$). Here, the model can be described by taking the inverse of the forward operator, $m = \mathbf{G}^{-1}d$. Performing this operation is not trivial if $\mathbf{G}$ is complicated, or non-linear; this operation may not even be possible if $\mathbf{G}$ is unknown for the problem of interest.

Machine learning (ML) is a solution to both of these challenges. First, machine learning algorithms

recognize patterns in data and assign data with similar patterns to the same category. This process automates pattern recognition within datasets and allows meaningful information to be extracted in an efficient manner. Secondly, machine learning algorithms learn an *implicit* mapping to go from d to m without the need for explicit relationships to be programmed.

Unsupervised learning (UL) focuses on learning a particular representation of the data, such as dimensionality reduction and clustering techniques. Clustering is concerned with segregating the data into *clusters* where the data in each cluster are considered similar. A benefit of clustering methods is that the predicted clusters are thought to be *unbiased* (i.e., the clusters are determined directly from the data), and valuable information can be extracted from the data without a priori constraints. However, suppose some data have apparent targets, or labels, which can exist through interpretations or other means (e.g., assigning a collection of pixels to a 'cat' label). In this case, it is possible to leverage this information through a different machine learning approach. The labels can be thought of as the model (m) in the inversion analogy above, and supervised learning (SL) methods are designed to learn a mapping to go from the data to the known labels. This mapping is performed using what is called the training dataset, a set of data points with known corresponding classes or labels. Ultimately this mapping is used to infer the labels for data points without labels. However, the training dataset needs to be large enough to ensure a robust mapping for supervised methods. Some applications, such as image classification and speech recognition, have large training datasets readily available. However, in several geoscience-related problems, labeled data are generally obtained by sampling the Earth in some manner (e.g., drilling wells, field sampling, etc.), which is not trivial due to cost and logistical factors. As such, many earth science-related machine learning problems have limited training data.

In these situations with limited training data, SL methods are prone to a phenomenon called *overfitting*. Overfitting occurs when a machine learning algorithm learns the detail and noise in the training dataset, which negatively impacts the performance of the model on unseen data; this leads to the algorithm perform-

ing suspiciously well on the training data, but generalizing poorly in making predictions for the testing data (Bishop, 2006). An analogous situation occurs in inverse problems. When there are more model parameters than data in an inverse problem, the system is underdetermined, and the predicted data tend to overfit the observed data; this leads to erroneous model parameter predictions. One way to mitigate the issues posed by an underdetermined inverse problem is to stabilize the objective function by adding an additional term that involves the model parameters, otherwise known as *regularization* (Aster et al., 2005). Regularization smooths the objective function, which helps prevent the model from overfitting the observed data. This same concept of regularization can be applied to machine learning to mitigate overfitting. Perhaps the most straightforward form of regularization is merely trying to include more data. Generating more training data is trivial for applications such as image classification, where data augmentation strategies (e.g., flipping, rotating, distorting images, etc.) are used to artificially enlarge the training dataset (Krizhevsky et al., 2012). However, collecting additional training data for geoscience applications is not always as simple as a click of a button. Generally, obtaining additional training data requires more sampling of the Earth, which costs money and time.

An alternative form of data augmentation is to instead incorporate the readily available unlabeled data into the training process using a different type of machine learning, *semisupervised learning* (SSL). SSL is essentially a hybrid of UL and SL, where the SSL algorithms train using both the labeled and unlabeled data. As such, SSL can also be thought of as SL where a regularization term, including the unlabeled data, is added to the objective function (Zhu & Goldberg, 2009). In the context of limited training data sets, it has been shown that SSL can make improved predictions for the unlabeled data compared to SL methods when the labeled data are scarce (Chapelle et al., 2006; Zhu & Goldberg, 2009; van Engelen & Hoos, 2020). Many machine learning applications are challenged with scarce training data that have utilized SSL methods, such as hyperspectral image classification (Camps-Valls et al., 2007; Gomez-Chova et al., 2008a; Tuia & Camps-Valls, 2009; Liu et al., 2013; Meng et al., 2017), protein classification (Weston et al., 2005; Kireeva

et al., 2012; Sigdel et al., 2014), and medical imaging (Livieris et al., 2018). However, in many geoscience problems, semisupervised methods have been relatively unexplored.

1.2 Thesis outline

Three geoscience applications inherently challenged with limited training data are well-log classification, seismic classification, and bedrock-lithology mapping. Throughout this thesis, I apply various semisupervised algorithms to these three geoscience problems and determine if semisupervised algorithms can perform better than supervised methods and under what conditions. One of the challenges associated with implementing semisupervised algorithms is the determination of their hyper-parameters. The literature lacks consensus on rules of thumb, or systematic approaches for determining semisupervised hyper-parameters. As such, throughout the body chapters in this thesis, there are additional treatments for figuring out how to best determine the hyper-parameters for the SSL algorithms that I consider. One culmination of this work is recommendations in the Conclusion on how to approach hyper-parameter estimation for the SSL methods considered in this thesis.

Chapters 3-7 in this thesis represent work that is already published, or is under review. Chapters 3, 4, and 5 are based on manuscripts that are already peer-reviewed and published in the journals *Geophysics*, *Computers and Geosciences*, and *Geophysical Journal International*, respectively (Dunham et al., 2020a,b, 2021). Chapter 6 is an exception as it is based on a published conference proceeding (Dunham, 2021). Chapter 7 is work from a manuscript that is under review. However, each of these body chapters has redundancies because similar methods or the same data are used (see below). As such, the introduction and methods sections of each chapter are consolidated to reduce repetition in the thesis. It is important to note that doing this consolidation means that these chapters and their published versions are not identical.

The outline of the thesis is as follows. The literature review is not provided here in *Chapter 1*, but focused literature reviews for the three geoscience applications are instead discussed within their pertinent

chapters. The introductory sections in Chapters 3, 5, and 7 contain the literature review of machine learning applications to well-log classification, seismic classification, and bedrock-lithology mapping, respectively. Chapters 4 and 6 do not contain proper literature reviews because these chapters are continuations of previous chapters, and literature reviews would be redundant (see below). *Chapter 2* introduces the machine learning methods, supervised and semisupervised, and the concept of hyper-parameter tuning. The methods discussed in Chapter 2 are those that are repeatedly used in the subsequent body chapters. However, each of the body chapters still contains its own methods section specific to that chapter (i.e., some methods are only used in one chapter). *Chapter 3* applies label propagation and self-training SSL techniques to a well-log dataset made publicly available by the Kansas Geological Survey, and compares the SSL performance to SL methods. *Chapter 4* is a continuation of Chapter 3 using the same well-log dataset, with the main distinction being that semisupervised Gaussian mixture models are used as the SSL method. Both of these chapters spend some time exploring the hyper-parameter space of the SSL methods and determining if default values or cross-validation strategies are preferred for hyper-parameter selection. The SSL results presented in Chapters 3 and 4 are the first SSL applications to well-log classification, to my knowledge, providing convincing alternatives to the mainstream utilization of supervised methods for that application. *Chapter 5* takes the same label propagation and self-training SSL techniques from Chapter 3 and applies them to a synthetic seismic classification problem derived from a model published by the Society of Exploration Geophysicists (SEG). *Chapter 6* is a continuation of Chapter 5 using the same datasets, but Chapter 6 approaches the problem from a supervised deep learning perspective and provides results to compare with the Chapter 5 SSL results. Other semisupervised approaches to seismic classification do exist in the literature, but the methods I provide are arguably more conceptually straightforward with a shallower learning curve. In *Chapter 7*, I use a data suite from a region in New South Wales, Australia, to simulate bedrock-lithology classification problems with different training data scenarios reflective of mineral exploration situations. What is unique about this chapter is that I use all three types of machine learning (UL, SL, and SSL) and determine which

methods are preferred under which conditions. This work is also the first, to my knowledge, that uses SSL in a bedrock-lithology classification context. *Chapter 8* concludes this thesis with summarizing remarks and recommendations for future work.

Chapter 2

Methodology

2.1 Supervised learning

The goal of supervised learning (SL) algorithms is to learn a mapping, otherwise known as a classifying function (f), to go from input samples (x_i) to their corresponding targets, or labels (y_i). This mapping function (f) is determined using all the available sample-label pairs, or what is also called the training data set,

$$L = (x_1, y_1), ..., (x_l, y_l), \tag{2.1}$$

where l is the number of labeled data. This mapping is learned and corrected through the differences between the predicted and true labels of the training data. What distinguishes different SL algorithms from each other are the assumptions on f and how it can be learned. Ultimately, the mapping learned from this process is used to make predictions for data where the labels are unknown, i.e. the unlabeled data

$$U = \{x_{l+1}, ..., x_{l+u}\} \tag{2.2}$$

where the actual predictions are given by,

$$H = \{y_{l+1}, ..., y_{l+u}\} \tag{2.3}$$

and u represents the number of unlabeled data. This mapping (f) from supervised learning is analogous to the inverse of the forward operator ($\mathbf{G}^{-1}$) from inversion, $m = \mathbf{G}^{-1}d$. However, with inversion, this mapping is known and the labels (m) are unknown, whereas the labels (y_i) are known with supervised learning and the mapping is unknown.

The purpose of using SL in this thesis is for SL methods to serve as a baseline to compare against the SSL methods. All of the SL methods used in this thesis, with the exception of Chapter 6, are discussed below. However, given that SL methods are not the focus of this thesis, they are discussed from a conceptual rather than mathematical perspective. What is essential in the context of this work is understanding what each algorithms' hyper-parameters represent and how they can impact performance, not the mathematical intricacies.

2.1.1 Gaussian Naïve Bayes (GNB)

Perhaps the simplest supervised method considered is the Gaussian Naïve Bayes (GNB) classifier. GNB is an example of what is called a parametric machine learning algorithm. Parametric methods simplify the learning process by assuming that f has a certain form over the whole input space (Chapter 4, Alpaydin, 2010). Similarly, the number of parameters that control f is fixed regardless of how much training data exists. GNB assumes that each class can be represented as a multivariate normal distribution $\mathcal{N}(\mu_k, \Sigma_k)$, where μ_k and Σ_k represent the Gaussian mean vectors and covariance matrices, respectively, for a particular class k. Training a parametric ML algorithm amounts to learning the coefficients, or parameters, that describe the form assumed for f, which are the mean vectors and covariance matrices for each class in the case of GNB (Chapter 7.4, Theodoridis, 2015). Assigning a class label to a new data point amounts to deter-

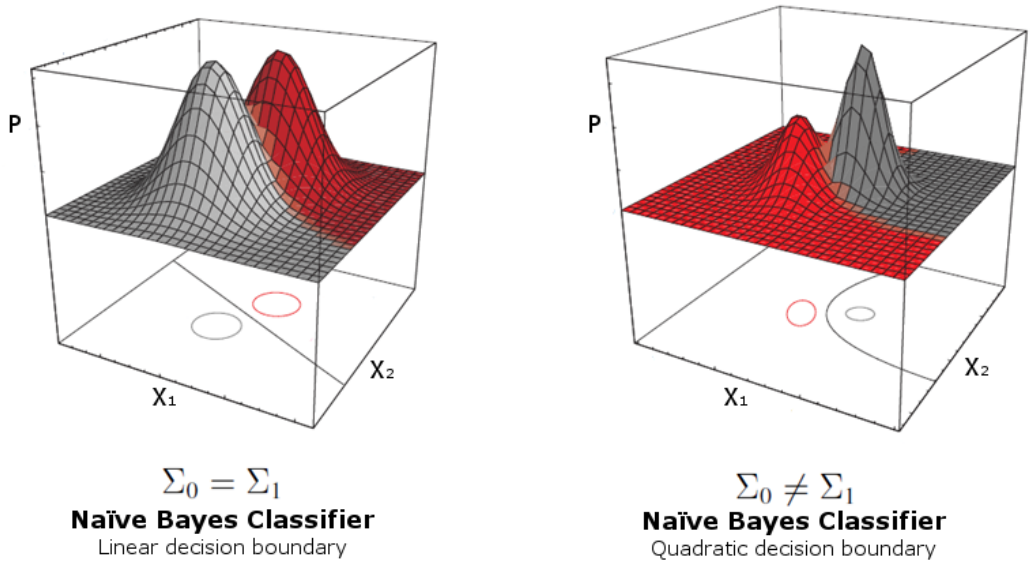

Figure 2.1: A binary classification problem showing the behavior of a GNB classifier. P is the probability associated with $\mathcal{N}(\mu_k, \Sigma_k)$ and X_1 and X_2 represent the input space in $\mathbb{R}^2$. If the covariance matrices are equal, then the decision boundary is linear (left). If the covariance matrices are not equal, then the decision boundary is quadratic (right). Modified from Figures 2.10 and 2.14 from Duda et al. (2001).

mining which class has the highest posterior probability. Figure 2.1 shows a binary classification problem using GNB where the covariance matrices are equal versus distinct (left and right, respectively).

A benefit of GNB is that it has no *hyper-parameters*. Hyper-parameters are external variables that the user adjusts before training and are often tuned to obtain machine learning models with optimal performance. However, a distinction needs to be made between hyper-parameters and *parameters*. Parameters are variables internal to the algorithm and can only be learned through training (i.e., these variables cannot be set by the user). For instance, the parameters for GNB are the mean vectors and covariance matrices describing each class, which are learned by training the algorithm. A potential disadvantage of GNB is indeed its assumption that a multivariate Gaussian distribution can characterize each class. The data may not be Gaussian distributed, in which case, this particular SL method may not perform well. An analogous situation occurs in linear regression. One can assume that the trends in their data are linear (i.e., making a model assumption), but fitting a line to data that are actually quadratic will not describe the data well. Nonetheless, GNB can still be appropriate for some problems, and it is implemented using the *GaussianNB*

class in `scikit-learn`.

2.1.2 Support Vector Machines (SVMs)

An algorithm with more flexibility is support vector machines (SVMs). The SVM tries to find the optimal hyperplane (i.e., decision boundary) that maximizes the margin between two linearly separable classes (Chapter 7.1, Bishop, 2006) as illustrated in Figures 2.2(a) and 2.2(b). SVMs do not make any distribution assumptions like GNB, but SVMs do assume that the classes are linearly separable in some multi-dimensional feature space. However, it is common for classes in the training data to overlap in the input space, which quickly violates this linearly separable assumption. An alternate formulation for SVM allows points to be on the wrong side of the margin, i.e., a soft margin, but with a penalty that increases with distance away from the boundary (Bishop, 2006). These penalties are called slack variables, $\xi_n \geq 0$ for $n = 1, .., l$, with one slack variable for every training point (Cortes & Vapnik, 1995). Training points on the correct side of the boundary have $\xi_n = 0$, but points on the wrong side have $\xi_n > 1$ (see Figure 2.2c). A hyper-parameter C is applied to the sum of all the slack variables, and the goal is to minimize the contribution of the soft margin in the overall optimization of the algorithm.

Even with the soft margin, SVMs are still limited to linear decision boundaries in the input space. However, SVMs use what is called a kernel, which implicitly transforms the data into vector spaces with higher dimensions (through inner products) where we can assume that the classes are linearly separable (Cortes & Vapnik, 1995; Hastie et al., 2009). In this higher-dimensional space, the hyperplane can be learned, and projecting this hyperplane back into the original input space can define a non-linear decision boundary (see Figure 2.3 for an example). There are several kernel choices, such as polynomial or sigmoid, but arguably the most popular is the radial basis function (RBF) kernel, where the kernel width (σ) is a hyper-parameter that requires setting (Chapter 12.3, Hastie et al., 2009). Assigning a class label to a new data point amounts to determining which side of the hyperplane the data point is situated. It is worth noting

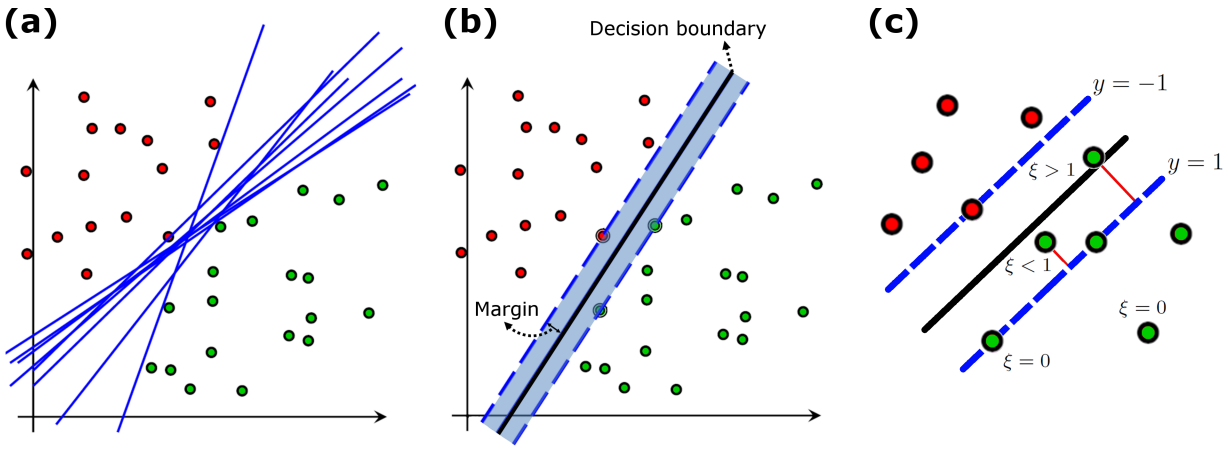

Figure 2.2: Concepts of the support vector machine (SVM) classifier. (a) There is an infinite number of hyperplanes that can separate two linearly-separable classes, but (b) SVMs determine the optimal hyperplane that maximizes the margin between two classes. (c) The soft margin allows some points to be misclassified within the margin. Slack variables (ξ) are penalty parameters assigned to each data point that are 0 if correctly classified and $>$ 1 if misclassified. The optimal decision boundary balances maximizing the margin and minimizing the number of misclassified training points. (a,b) Modified from Figure 1 from Wang et al. (2013) and (c) modified from Figure 7.3 from Bishop (2006).

that an SVM classifier is inherently binary, i.e., it can only distinguish two classes from each other. However, SVMs can be extended to multiclass by decomposing the multiclass problem into a series of one-versus-all or one-versus-one binary problems.

I implement SVM using the *SVC* class in `scikit-learn`, and I choose to use the RBF kernel. As such, the two hyper-parameters that must be set are C and σ. If the slack variables are to be minimized, then larger values for C will discourage any positive ξ_n values, which can lead to complex decision boundaries and possibly overfitting. Smaller values for C can allow positive ξ_n values to exist without much penalty, which causes the decision boundaries to be much smoother. The hyper-parameter C is analogous to a regularization coefficient because it can control the tradeoff between minimizing training error and model complexity (Chapter 7.1, Bishop, 2006). The σ hyper-parameter of the RBF kernel determines the reach of a single training instance. Smaller values of σ mean every training instance will have a far reach, which results in a decision boundary that is influenced by training points further away from it. This often results in smoother, less complex decision boundaries. In comparison, larger values of σ mean training points will

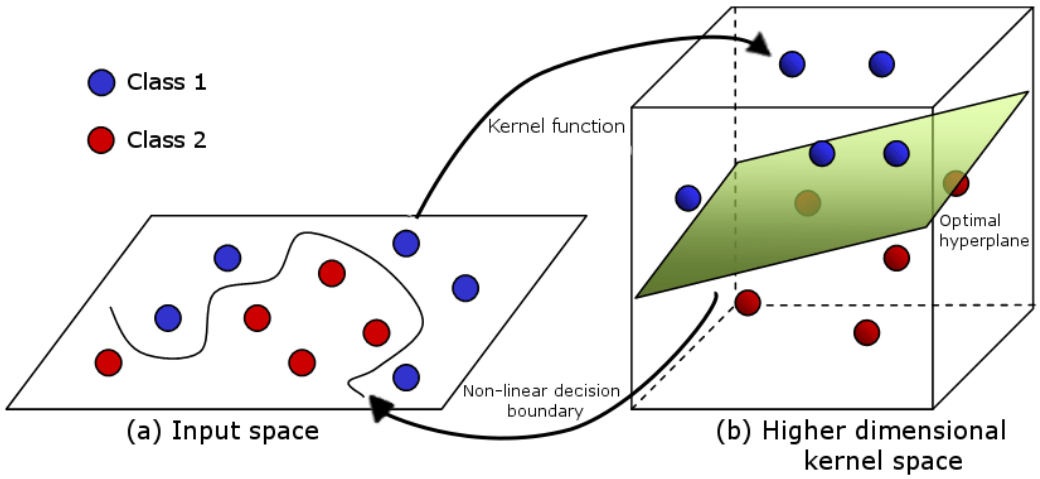

Figure 2.3: The kernel transformation concept for SVM. (a) The data are not linearly separable in the original input space. The kernel function takes the inputs in the original lower dimensional space and returns the transformed data in the higher dimensional space. (b) The optimal separating hyperplane is determined in the higher dimensional space, which appears as a non-linear decision boundary in the original input space. Modified from Figure 2 from Vocaturo et al. (2019).

have a close reach, which causes the decision boundary to be more dependent on the training points closest to it. This tends to make the decision boundaries more complex. It is important to recognize how these hyper-parameters can impact the parameters of the SVM (i.e., the hyperplane) because adjusting these two hyper-parameters can mitigate under or overfitting if either is observed.

2.1.3 Gradient boosting decision trees (GBDTs)

The most frequently used SL methods in this thesis are gradient boosting decision tree (GBDT) methods. These methods are complex, but they can be distilled down to three main components as indicated in their name: decision trees, boosting, and gradient optimization. Decision trees are hierarchical methods that use binary decision rules to segment the input space into regions based on the training data (see Figure 2.4). As shown in Figure 2.4(a), a decision tree is composed of decision nodes, branches, and terminal leaves (Chapter 9, Alpaydin, 2010). Each decision node contains a binary condition operating on one of the input dimensions, with both outcomes labeling the branches. A decision tree is grown until splitting is no longer needed, which is when leaf nodes are created and given a label. Classifying a new data point is done by

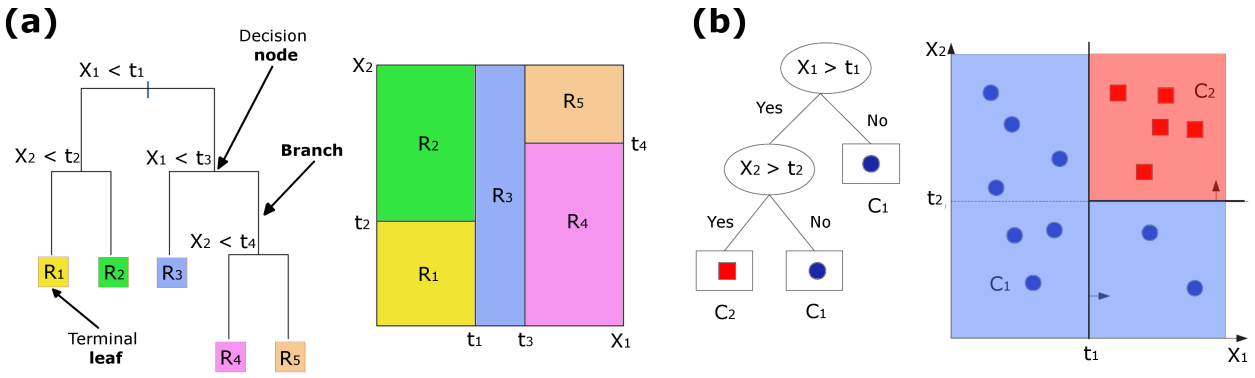

Figure 2.4: Two decision tree examples. Each region is assigned a label according to a majority vote of the training data assigned to it. Therefore, the label assigned to a new point will be the label of the region it falls within. (a) Modified from Figure 9.2 from Hastie et al. (2009) and (b) modified from Figure 9.1 from Alpaydin (2010).

simply following the decisions through the tree until a terminal leaf node is reached, at which point the value assigned to the leaf is given as the predicted output. An aspect of decision trees that makes them favorable is their interpretability; a tree can be conceptualized as a set of IF-THEN flowchart rules that are understandable to the user (Chapter 9, Alpaydin, 2010). A decision tree is also an example of a *non-parametric* method because no class distributions are assumed, and the model parameters are not fixed (the number of decision tree nodes and leaves can change depending on the nature of the data).

One challenge with decision trees is that a single tree can be unstable. Theoretically, one tree could be grown large enough to partition the input space to perfectly classify all training points (i.e., no training error). However, this would surely overfit the training data and generalize poorly to new data. The established techniques in the literature for mitigating the overfitting of decision trees are to use an ensemble of decision trees (Dietterich, 2000). One such method is called bootstrap aggregation (i.e., bagging, Breiman, 1996). A bootstrap sample is a resampling of the original training dataset with replacement. Then, a decision tree is trained on B bootstrap datasets (where B is an integer), and a new point is classified as the majority vote from the B classifiers. Combining an ensemble of individual classifiers introduces variability not present in a single classifier and improves the overall predictability by averaging out any instability (Dietterich, 2000). However, improvement using bootstrapped datasets will reach a threshold because each dataset is

still highly correlated. Random forests address this problem by constructing decision trees in such a way as to reduce correlation (Chapter 15, Hastie et al., 2009). This algorithm works by building a decision tree for each bootstrap dataset, but when splitting a given node, only a random subset of the input data dimensions is used (Breiman, 2001). The type of ensemble learning most relevant for SL methods used in this thesis is *boosting*, which sequentially applies weak classifiers (i.e., shallow decision trees) to the training data and their combination forms a powerful voting committee (Freund & Schapire, 1996). Boosting does not resample the training data and instead assigns weights to each training point; these weights are modified at each iteration based on whether they were correctly or incorrectly classified (Chapter 10, Hastie et al., 2009). The weak classifiers from each iteration are combined using a weighted majority vote to produce the ensemble classifier (see Figure 2.5 for an example).

The final building block of GBDTs is incorporating gradient descent. The original boosting algorithms, such as AdaBoost (Freund & Schapire, 1996), have a statistical formulation, but Friedman (2001) recasts the boosting framework as a numerical optimization problem. In other machine learning algorithms, such as neural networks, gradient descent is used to minimize the loss, and that gradient is used to update the neural network parameters (i.e., the weights and biases). The idea here is much the same, except the parameters being updated by the gradient are those pertaining to a decision tree, and each boosting iteration is interpreted as a step in the direction of the steepest descent. The learning rate constrains how far each boosting iteration steps in the direction of steepest descent, which is a hyper-parameter for these algorithms (see below).

XGBoost

The GBDT algorithm used in all three geoscience applications in this thesis is eXtreme Gradient Boosting (XGBoost, Chen & Guestrin, 2016). XGBoost is chosen as the primary SL method in this thesis because it is shown to be robust across disciplines (Tamayo et al., 2016; Torlay et al., 2017; Zhang et al., 2018a; Kumar et al., 2022), and is a popular method in data science competitions. As such, XGBoost is perhaps one of the

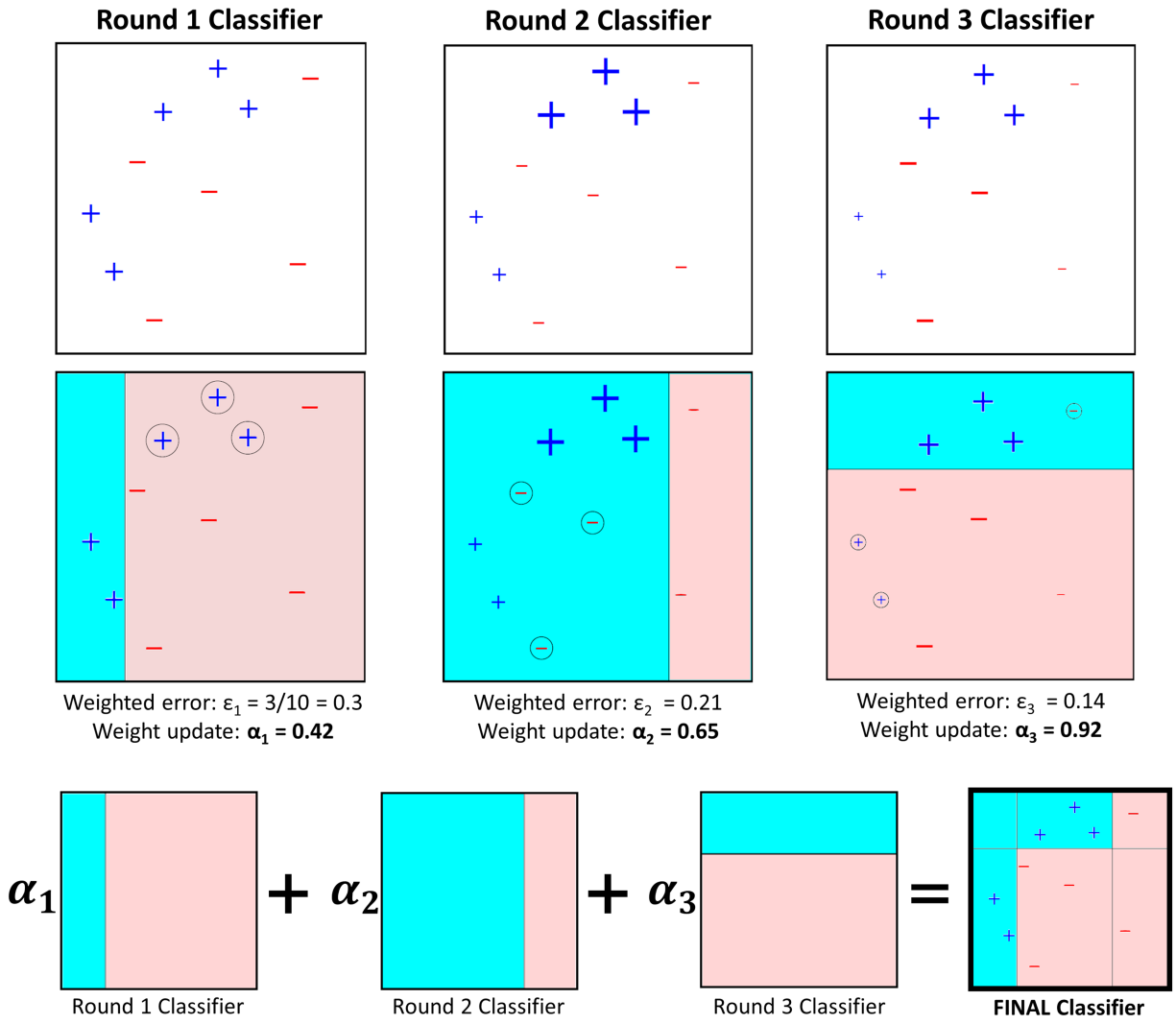

Figure 2.5: A binary classification example illustrating the boosting procedure using decision stumps (a decision tree with only one decision, i.e., depth = 1) as the classifier, $f(x)$. The points belonging to the positive and negative classes have blue '+' symbols and red '-' symbols, respectively. Each training data point is given a weight, and all the weights are initially set to $1/N$. The weight of each point is denoted by the *size* of the '+' or '-' symbols (notice how the sizes of all the symbols in Round 1 are equal). At a given boosting step m, the observations that were misclassified by classifier $f_{m-1}(x)$ have their weights increased, whereas the weights are decreased for those that were classified correctly. For example, the decision stump misclassifies three positive training points in Round 1 (these are the circled points); in Round 2, these three misclassified positive points from Round 1 have their weights increased (denoted by the larger '+' symbols), and the seven points that were classified correctly in Round 1 have their weights decreased (denoted by smaller symbols). This process encourages each successive classifier to concentrate on correcting the previous classifiers' mistakes. The weighted error (ε_m) is the sum of all the misclassified weights and is used to calculate the weight update, α_m. The final classifier is a weighted vote of each of the individual m classifiers, as indicated at the bottom of the figure. Modified from Paisley (2017).

best SL methods to compare against the SSL methods. XGBoost has several hyper-parameters, but those that I choose to focus on setting values for are listed below:

- Number of estimators (default = 100)
- Max depth (default = 6)
- Minimum child weight (default = 1)
- Learning rate (default = 0.30)
- Lambda (L2 regularization, default = 1)

The number of estimators is the number of trees (i.e., boosting iterations) in the ensemble. Boosting is commonly a process that operates on weak learners, so the max depth (i.e., how many levels a decision tree can have) tends to be a small integer. The decision stumps used to illustrate boosting in Figure 2.5 have a depth of 2. Another hyper-parameter that limits how the trees can grow is the minimum child weight. If at a given decision node, a partition results in a leaf node with a weight that is less than the defined value, the tree-building process gives up further partitioning on that decision node. The learning rate, as mentioned above, is a parameter introduced from using gradient descent, which controls how fast (or slow) the boosting algorithm learns. Larger values for the learning rate may allow the algorithm to converge more quickly, but it may be too quick to learn and risk overfitting the training data. Smaller values can prevent overfitting, but it may take larger to converge (i.e., a higher computational cost). Lastly, the lambda hyper-parameter is a regularization term that can make the model more conservative. Generally, higher values for minimum child weight and lambda can help reduce overfitting, but lower values are needed for the max depth and learning rate to achieve reduced overfitting (Chapter 5.3.6 gives an example of the ranges of values considered for these hyper-parameters).

LightGBM

Another GBDT method that I use later in this thesis (Chapter 7) is called Light Gradient Boosting Machine (LightGBM, Ke et al., 2017). Like XGBoost, LightGBM has been robust across disciplines such as medical, biological, and financial fields (Wang et al., 2017; Chen et al., 2019; Sun et al., 2020). As LightGBM and XGBoost are both GBDT methods, they are quite similar, but one fundamental difference between the two algorithms is how the decision trees are grown. XGBoost grows its trees *level-wise*, whereas LightGBM grows trees *leaf-wise* (see Figure 2.6). Ke et al. (2017) show that leaf-wise growth can reduce computation times significantly and improve accuracy compared to XGBoost. Leaf-wise trees can grow deeper much more quickly than level-wise trees, which can result in overfitting of the training data; however, this can be addressed with the max depth hyper-parameter. LightGBM has several of the same hyper-parameters as XGBoost (e.g., learning rate, number of estimators, max depth, and lambda), but two hyper-parameters are more specific to LightGBM:

- Number of leaves (default = 31)
- Minimum data per leaf (default = 20)

The minimum data per leaf is analogous in its function to the minimum child weight from XGBoost. The number of leaves is the main parameter to control the complexity of the trees in LightGBM. Generally, the number of leaves is set to be less than $2^{\text{max depth}}$ to avoid overfitting (Ke et al., 2017). A functionality that XGBoost and LightGBM both possess that has not yet been discussed is the capability of randomly selecting only a fraction of the features to construct each tree (similar to Random Forests). However, both algorithms have this fraction set to 1 as their default (i.e., all input features are used to construct each tree), and I do not explore changing this hyper-parameter.

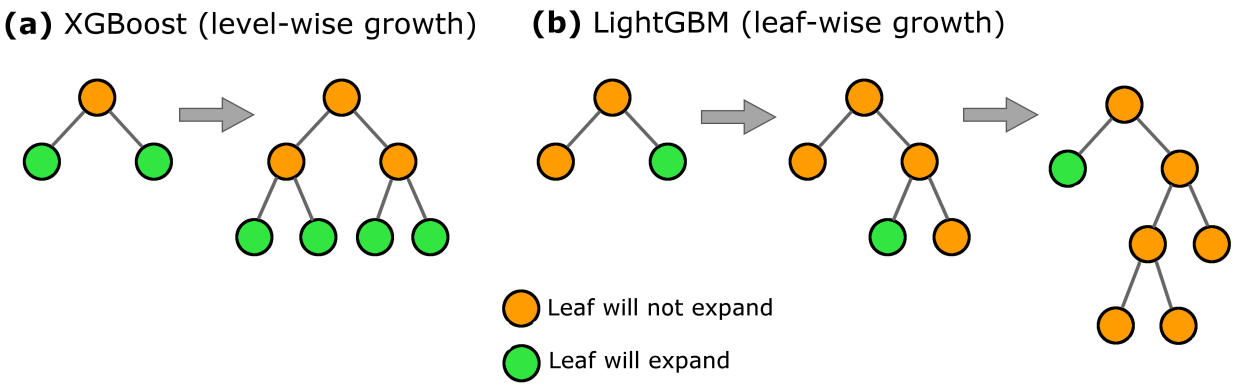

Figure 2.6: The differences between (a) level-wise decision tree growth and (b) leaf-wise decision tree growth. LightGBM uses leaf-wise growth that is shown to require less computation time and achieve higher accuracy compared to XGBoost (Ke et al., 2017).

2.2 Semisupervised methods

SSL algorithms are similar to SL algorithms in that their ultimate goal is to make predictions for the unlabeled data (U). However, the distinction with SSL methods is that the unlabeled data (U) are incorporated into the training process. As a result, SSL algorithms train with

$$D = \{(x_1, y_1), ..., (x_l, y_l), x_{l+1}, ..., x_{l+u}\} = L \cup U \tag{2.4}$$

where D is the union of L (Eq. 2.1) and U (Eq. 2.2), l and u are the amounts of labeled and unlabeled data, respectively, and the objective is to still make predictions (H) for the unlabeled data. From a theoretical perspective, including the unlabeled data in the training process can help SSL algorithms achieve better generalized predictions for U than SL algorithms in the context of low training data (Chapelle et al., 2006; Zhu & Goldberg, 2009; van Engelen & Hoos, 2020).

The problems that SSL algorithms are designed for are those with far more unlabeled than labeled data ($u >> l$) and many geoscience problems reflect these scenarios. Figure 2.7 illustrates why and how SSL is useful using a toy dataset that is popular for SSL testing, the "two moons" dataset. Figure 2.7(a) shows the binary dataset with four labeled data points per class. Training a supervised classifier, such as a

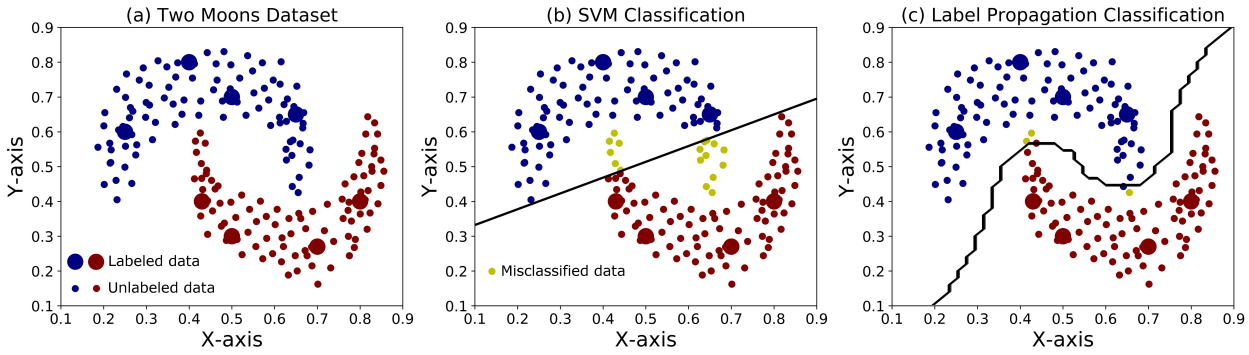

Figure 2.7: (a) The "two moons" dataset representing a binary classification problem where both classes have 91 unlabeled and 4 labeled points respectively (dataset provided by Gieseke et al., 2014). (b) The support vector machine (SVM) classification and decision boundary (black line). The accuracy is 90.66% with 17 misclassified points. (c) The semisupervised classification and decision boundary (black line) using label propagation. The accuracy is 98.35% with only 3 misclassified points. This demonstrates the benefit of including the unlabeled data (semisupervised learning) in the training process.

support vector machine (SVM), on these data gives the decision boundary indicated by the solid black line in Figure 2.7(b). Unfortunately, the SVM classifier misclassifies seventeen unlabeled data points. However, if the unlabeled data are included in the training process using an SSL algorithm, such as label propagation, then the new semisupervised decision boundary (solid black line in Figure 2.7c) achieves an improved classification of the unlabeled data (only three misclassifications). The following sub-sections introduce the SSL methods used in this work in more detail.

2.2.1 Label propagation (LP)

One well-established semisupervised technique is label propagation (LP), otherwise known as graph-based learning. From a conceptual perspective, LP is designed to let the data points iteratively spread their label information to their unlabeled neighbors until a global stable state is reached. The algorithm that I use for LP is publicly available (*LabelSpreading* class from `scikit-learn` in Python) and is based on the methodology from Zhou et al. (2004). The algorithm is summarized here, but see Zhou et al. (2004) and Chapter 5 from Zhu & Goldberg (2009) for more details.

An essential component of the LP algorithm is constructing the adjacency matrix, edge-weight matrix,

or graph, which measures a pairwise similarity between each data point. There are two common methods for constructing this matrix. The first option uses a radial basis function (RBF) kernel to generate the graph,

$$W_{ij} = \exp(-\sigma||x_i - x_j||^2), \tag{2.5}$$

where $i, j = 1, ..., l + u = n$ (i.e., the indices for both the labeled and unlabeled data), $||x_i - x_j||^2$ is the L2 distance between two data points, and σ controls the kernel width (similar to the σ hyper-parameter for SVM). For a fixed L2 distance, a larger sigma results in a smaller weight W_{ij} between points x_i and x_j (i.e., the *reach* of each data point is smaller). Conversely, a smaller sigma results in a larger weight W_{ij} between points x_i and x_j (i.e., the *reach* of each data point is farther). In essence, if W_{ij} is large, then the points x_i and x_j likely share the same label ($y_i = y_j$). This adjacency matrix produces a dense matrix ($n \times n$), which becomes problematic with larger datasets. The alternative option is to build the adjacency matrix using a nearest-neighbors (NNs) approach,

$$W_{ij} = \begin{cases} 1 & \text{if } x_j \in NN_p(x_i) \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

where p is the number of nearest neighbors. So, if x_j is one of the p neighbors of x_i (i.e., $W_{ij} = 1$) then x_i and x_j likely share the same label. Technically, Eq. 2.6 is still an $n \times n$ matrix, but the benefit here is it can be saved in the computer as an ($n \times p$) sparse matrix by ignoring the zeros. This sparse representation makes Eq. 2.6 more manageable for larger datasets compared to Eq. 2.5.

The next step of LP is calculating the spreading function, which utilizes the graph (**W**) established previously. The graph (**W**) is first used to calculate what is called a normalized graph Laplacian matrix,

$$\mathcal{L} = \mathbf{I} - \mathbf{A}^{-1/2}\mathbf{W}\mathbf{A}^{-1/2}, \tag{2.7}$$

where $\mathbf{I}$ is the identity matrix, and $\mathbf{A}$ is a diagonal matrix with its (i, i) elements equal to the sum of the ith row of $\mathbf{W}$. This matrix ($\mathcal{L}$) is then used to define the spreading function,

$$\mathbf{F}(t+1) = \alpha \mathcal{L} \mathbf{F}(t) + (1-\alpha)\mathbf{Y}, \tag{2.8}$$

where $\mathbf{Y}$ is an $n \times K$ matrix containing the initial labels ($Y_{ik} = 1$ if x_i is labeled as $y_i = k$, and $Y_{ik} = 0$ for all unlabeled data points), K is the number of classes, $\mathbf{F}$ is initialized by $\mathbf{F}(t = 0) = \mathbf{Y}$, and α is a hyper-parameter defined on the interval [0,1].

The spreading function $\mathbf{F}$ is solved for iteratively, and at each iteration, each point receives information from neighboring points (first term in Eq. 2.8) and the initial labels (second term in Eq. 2.8), where α controls the weight between these two terms in Eq. 2.8. The α hyper-parameter can also be interpreted as a clamping factor where α defines the proportion of initial labels that are allowed to change their labels (e.g., setting $\alpha = 0.20$ means that the algorithm will maintain 80% of the initial label distribution, but it has the flexibility to change its confidence in 20% of the label distribution). Once $\mathbf{F}$ converges, each row is normalized to provide a soft classification on all n data points (i.e. each row of $\mathbf{F}$ sums to 1). Therefore, one can assign labels to the unlabeled data (U) by,

$$y_i^{pred} = \arg\max_k F_{ik} \quad for \quad i = l+1, ..., l+u. \tag{2.9}$$

Figure 2.8 shows an example of using LP on a toy example, which is also helpful for understanding how the LP classification in Figure 2.7(c) is produced.

The two assumptions of the LP method are the *smoothness* (i.e. two points that are close to each other in a high-density region will likely share the same label) and *manifold* assumptions (i.e. two points on the same global structure are likely to have the same label). Conceptually, it is intuitive that LP implements the smoothness assumption because of the way that the algorithm spreads its label information to neighboring

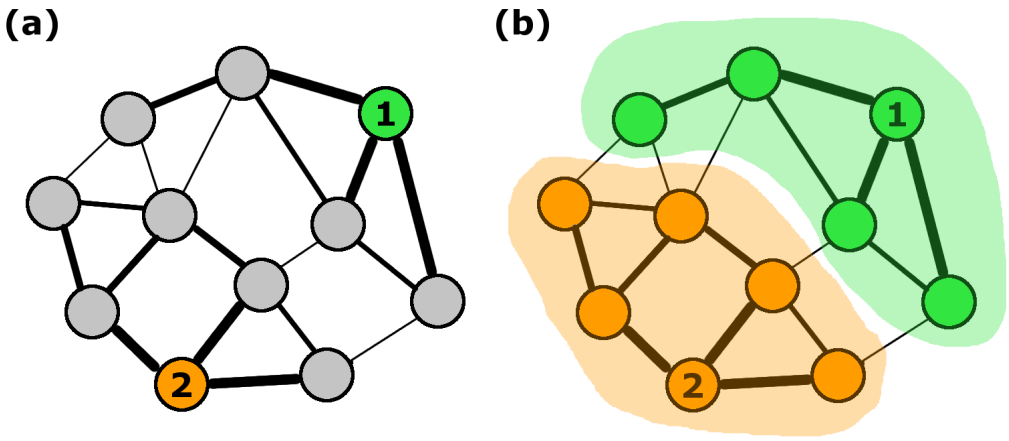

Figure 2.8: Label propagation demonstrated using a small example. (a) The two colored nodes are initial labeled data points (Classes 1 and 2), while the gray nodes represent unlabeled data. The thickness of the lines between nodes represents the strength of similarity between points. (b) Each unlabeled data point is assigned the class which receives the most information during the label propagation process (Eq. 2.8). Inspired by Fig. 2 from Camps-Valls et al. (2007).

points. The manifold assumption is harder to conceptualize, but if you imagine that each moon in Figure 2.7 belongs on its own manifold, then all this assumption is stating is that points in one moon should share the same label. Implementing this algorithm requires setting two hyper-parameters: one pertaining to the adjacency matrix (σ or p) and α. Since these two parameters are the only two factors controlling the behavior of the LP algorithm, choosing optimal settings for them is the only method for ensuring that the smoothness and manifold assumptions are met. Similarly, incorrect values for these two hyper-parameters could violate one or both of these assumptions, and this could lead to incorrect classifications (see Chapters 3.4 and 3.5 for examples of assumption violations). However, the data themselves can also violate these assumptions, and an example is shown in Chapter 5.4.1.

Label propagation is also considered to be a *transductive* algorithm. Transductive algorithms only have the ability to learn an internal mapping of the existing labeled and unlabeled data points in order to label the unlabeled data. A significant disadvantage of these algorithms is that they are not classifiers and they cannot make 'out-of-box' predictions. This is because the adjacency matrix is fixed for the set of data (D) used to train the algorithm; if labels are sought for additional unlabeled data, then the edge-weight matrix and

subsequent internal mapping must be recomputed and this may not be computationally feasible. This is not the case for inductive methods, or classifiers, because they are designed to make predictions for new data without needing to retrain the entire algorithm. Therefore, large datasets can be problematic for transductive methods. For instance, if there are 10 million unlabeled points and 1 million labeled points, computing the adjacency matrix (using RBF or nearest neighbors) for all 11 million points could be computationally limiting. Semi-supervised inductive algorithms have the advantage of training a classifier on the 1 million labeled and perhaps 5 million unlabeled points, and then that classifier could be used to predict the labels for the remaining unlabeled 5 million points.

Nonetheless, there are some advantages specific to label propagation and to transductive learning in general. The α parameter in Equation 2.8 can help the LP method overcome noisy points or outliers. Outliers may be more characteristic of other classes and they could be mistakes in the labeling process, but if their classes are *fixed* (i.e. $\alpha = 0$), then this could cause instabilities. However, setting $\alpha > 0$, for instance 0.4, implies that 60% of the original label distribution will be retained, but the algorithm has the flexibility to change its confidence in 40% of the label distribution. This could allow misclassified outliers to take on the information from their surroundings rather than propagating their false label information. A similar argument can be made for datasets with poor class separability where allowing labeled points to have some flexibility in overlapping zones may be beneficial. New data are problematic for transductive algorithms, but in many cases, datasets are finite and transductive learning is still appropriate in these situations. Furthermore, since the mapping for transductive learning is only defined on the existing data, these algorithms tend to be simpler than inductive algorithms (Zhu & Goldberg, 2009).

2.2.2 Self-training

One of the earliest forms of semisupervised learning is self-training (Yarowsky, 1995), which is the second SSL method that I consider. This algorithm operates just as it sounds; a supervised (or semisupervised)

method uses its own predictions to teach itself. A benefit of self-training is that it has the flexibility to wrap around any algorithm that makes predictions for unlabeled data. However, I find that self-training seems to work the best with the LP method, which is the combination that I use in Chapters 3 and 5. While there are a few different flavors of self-training (Rosenberg et al., 2005; Liu et al., 2013; Asghar et al., 2020), the approach that I take is slightly different. This implementation of self-training is an iterative process described below.

- **Step 1**: Assign values for the two hyper-parameters, S and T. The subset size (S) controls how many points will be removed from U and added to L at each iteration (this can be an integer, or it can be defined as a percentage value multiplied by the initial amount of unlabeled points, u). The stopping threshold (T) is a decimal value on [0, 1] that controls how much of the unlabeled data is to be removed before the algorithm stops.

- **Step 2**: Train a predictor f on L for SL, or $L + U$ for SSL. Any predictor f can be used for self-training (supervised or semisupervised) as long as f has a measure of confidence in its predictions for the unlabeled instances, U.

- **Step 3**: Apply f to the unlabeled instances, U.

- **Step 4**: Calculate the class probability (π_k, see Eq. 2.11 below), and then compute $S_k = S\pi_k$. Here, S_k constrains precisely how many points per class to remove from U and add to L. If S_k is already defined from a previous iteration, skip to Step 5.

- **Step 5**: Remove the subset of points (defined by S_k) from U and add them to L; this does assume that the predicted classes of the points being added to L are correct. As an example, if $S_k = S\pi_k = 20[0.25, 0.65, 0.10] = [5, 13, 2]$, then the 5 points with the highest confidence of belonging to Class 1 would be removed from U and added to L (similarly 13 and 2 points, respectively, for Classes 2 and

3). This defines one self-training iteration, and the new sizes of L and U are $(l + Si)$ and $(u - Si)$, respectively, where i is the number of self-training iterations.

- **Step 6**: Repeat steps 2-5 until the defined threshold is exceeded (i.e., $size(L)/n \geq T$). For example, if $T = 0.50$, the self-training process will continue to grow the labeled dataset (L) until it exceeds 50% of the total data.

The selection criteria of any self-training technique define how the method selects which points to remove from U and add to L. The two parameters in Step 1 define the selection criteria specific for my implementation. The class probability (π_k) and subset (S) are critical because together (S_k) they constrain how much of each class is to be removed from U at each iteration. If different classes have different numbers of points associated with them, then we want to extract an appropriate relative number of points from each class in U at each iteration; this is the information that should be encoded in π_k. One way to determine π_k is by computing the class prior probability directly from the initial labeled set L via,

$$\pi_k = \frac{1}{l} \sum_{i=1}^{l} \mathbb{1}(y_i = k) \tag{2.10}$$

where $\mathbb{1}$ is an indicator function that returns 1 if $y_i = k$ and 0 otherwise. However, a problem with this approach arises if the distribution of classes in L is not representative of the class distributions in U. Imagine a binary classification scenario where L has 20 data points, 15 belonging to Class A and 5 belonging to Class B, and U has 80 data points, 20 belonging to Class A and 60 belonging to Class B. Therefore, the class prior probability via Equation 2.10 is $\pi_k = [0.75, 0.25]$ and the unlabeled data has a class distribution of $\pi = [0.25, 0.75]$. In this case, if the subset size $S = 12$, then the amount of points removed from each class in U at each iteration is $S_k = [9, 3]$. After two iterations, 18 points from Class A and 6 points from Class B are removed from U. At the third iteration, 9 points with Class A would be removed from U, but only 2 points with Class A remain in U. Therefore, the remaining 7 points that the algorithm assumes belong to

Class A actually belong to Class B; for all subsequent iterations, Class B points in U will be misclassified as Class A. In summary, if an incorrect π_k is used, then self-training could force an incorrect class distribution on U and this would violate the assumption mentioned in Step 5. It is common for the training data class distribution to not necessarily match the testing data class distribution, so perhaps the self-training selection criteria can be modified to accommodate this.

Perhaps a more robust approach is to derive the class probability from U instead of L. A realistic approach that I use to calculate π_k is from an algorithm's predicted class membership probabilities on U, such as $\mathbf{F}$ from LP (Equation 2.8),

$$\pi_k = \frac{1}{u} \sum_{i=l+1}^{l+u} F_{ik} \tag{2.11}$$

where k is the class index, and I refer to this as the *predicted* class probability. This approach helps alleviate the problem of potentially different class probabilities between L and U. However, in the context of applying self-training to LP, obtaining an accurate π_k from U using Equation 2.11 may be reliant on the LP method's ability to properly validate the smoothness and manifold assumptions. Nonetheless, once the predicted class probability is established, the only hyper-parameters that must be set are the subset size (S) and stopping threshold (T). Figure 2.9 uses a simple example to help conceptualize this self-training process.

What distinguishes different self-training techniques are the selection criteria, i.e., deciding which points to remove from U and add to L. The selection criteria for my implementation are constrained by S_k and this is to accommodate class imbalanced datasets. Other approaches use a probability threshold τ as their selection criteria (Liu et al., 2013; Livieris et al., 2018; Asghar et al., 2020) where all points with a confidence higher than τ (e.g., $\tau > 0.90$) are removed from U and added to L at each iteration. A potential issue that I see with these approaches relates to when classes in U are unbalanced. The points in U associated with the most-populous class(es) may have the highest prediction confidence. So, the threshold-based criteria may always choose points to remove from U that are associated with the most-frequent class(es), thereby ignoring any classes with fewer points. As such, the self-training selection criteria based on probability

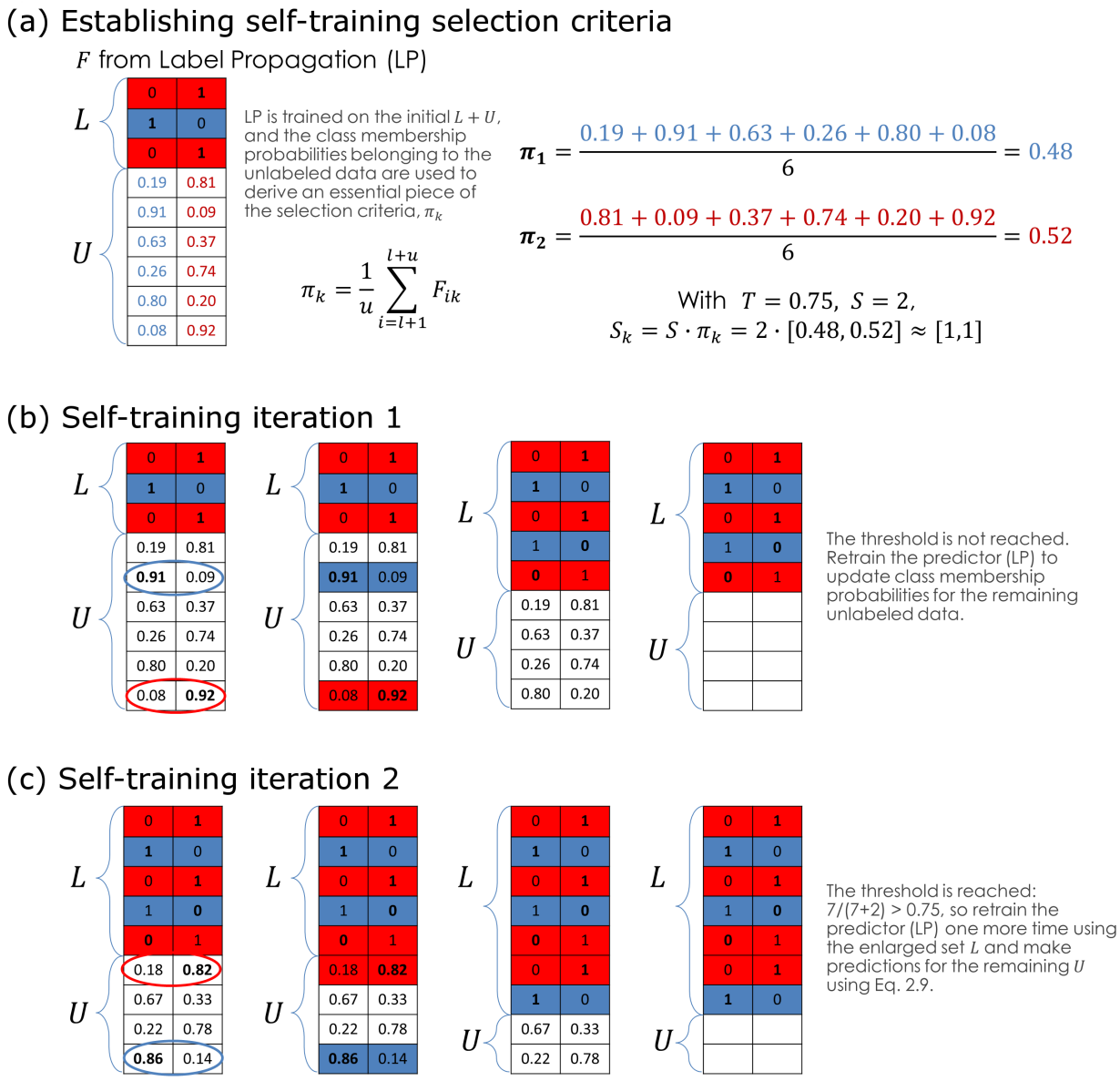

Figure 2.9: A demonstrative binary classification example showing how the self-training process works. (a) A particular machine learning method, LP in this case, is trained on the labeled and unlabeled data (D). The class membership probabilities on the unlabeled data are used to determine π_k. With $S = 2$, this constrains the selection criteria to removing one point from each class for each iteration, $S_k = [1, 1]$. (b) The first self-training iteration. One unlabeled point from each class with the highest probability is selected and added to the labeled dataset. With the two points removed from U and added to L, the threshold of 0.75 is not surpassed, so the self-training process continues. The algorithm is retrained to update the class membership probabilities. (c) The second self-training iteration. After this iteration, the threshold is surpassed, so self-training stops. The algorithm is re-trained one final time and predictions for the remaining unlabeled data are obtained as normal using thresholds on the class membership probabilities.

thresholds may only be appropriate for balanced datasets. The way that I constrain the selection criteria forces the algorithm to remove from U what I approximate to be the appropriate amounts of data from each class at each iteration, thereby better accommodating class imbalance.

While I design the selection criteria to accommodate class imbalance, self-training, in general, has the potential to still be rather unstable. If the base algorithm is performing poorly (LP in this context), then errors could be perpetuated with each self-training iteration. What could cause this instability is if the underlying adjacency matrix (**W**) is poorly defined, and Chapter 5.5.1 describes an example where this occurs. Nonetheless, self-training is a simple technique to help improve the performance of classification problems when training data are limited, and this approach is easy to implement with only two hyper-parameters (S, T).

2.2.3 Semisupervised Gaussian mixture models (ssGMM)

Many semisupervised techniques are simply extensions of existing unsupervised or supervised methods to include additional information. For instance, semisupervised Gaussian mixture models (ssGMM) use an algorithm that essentially combines a Naïve Bayes classifier (supervised) and Gaussian mixture models (unsupervised). A few different implementations of ssGMM do exist (Nigam et al., 2000; Zhu & Goldberg, 2009; Xing et al., 2013), but I have chosen to follow the approach outlined in Yan et al. (2017). However, none of these implementations provide open-source code for their methods (including Yan et al., 2017), but I have elected to do so (see Computer Code Availability below). The algorithm is summarized below.

In this semisupervised framework, the training data consist of both labeled and unlabeled data (D) and the goal for ssGMM is to employ a probabilistic approach that seeks the labels that maximize the conditional probability $p(D|\theta)$. Training amounts to finding good model parameters (θ), and the maximum likelihood

estimate (MLE),

$$\hat{\theta} = \arg\max_{\theta}[p(D|\theta)] = \arg\max_{\theta}[\log p(D|\theta)] \tag{2.12}$$

gives the parameters under which the data likelihood is the largest. The log-likelihood yields the same maxima as the straight likelihood because $\log()$ is monotonic, and using a log-likelihood simplifies the next step considerably. Simplifying log-products into sum-logs and substituting Bayes rule into Equation 2.12 gives,

$$\begin{aligned} \log p(D|\theta) &= \log\left(\prod_{i=1}^{l} p(x_i, y_i|\theta)^{\beta} \prod_{i=l+1}^{l+u} p(x_i|\theta)^{1-\beta}\right) \\ &= \beta\sum_{i=1}^{l} \log\left[p(y_i|\theta)p(x_i|y_i,\theta)\right] + (1-\beta)\sum_{i=l+1}^{l+u} \log p(x_i|\theta) \end{aligned} \tag{2.13}$$

where the first term is the supervised likelihood, the second term is the marginal (unsupervised) likelihood, and β establishes a weight between these two terms. Equation 2.13 is the objective function in general, but if the model parameters are those that describe a multivariate Gaussian distribution for each class, then Equation 2.13 becomes,

$$\log p(D|\theta) = \beta\sum_{i=1}^{l} \log\left[\pi_{y_i}\mathcal{N}(x_i;\mu_{y_i},\Sigma_{y_i})\right] + (1-\beta)\sum_{i=l+1}^{l+u} \log\left(\sum_{k=1}^{K}\mathcal{N}(x_i;\mu_k,\Sigma_k)\right) \tag{2.14}$$

where π, μ, and Σ represent the Gaussian priors, means, and covariances, respectively, and $\mathcal{N}$ represents a multivariate normal distribution. The standard implementations of ssGMM (Nigam et al., 2000; Zhu & Goldberg, 2009, Chapter 3) do not include β, but this hyper-parameter is introduced by Yan et al. (2017) to give a relative weighting ($0 < \beta < 1$) between the labeled and unlabeled portions of the log-likelihood.

To find the MLE, Equation 2.14 needs to be optimized. Unfortunately, the unobserved labels (H) make the log-likelihood non-concave and hard to optimize. The standard remedy is to use the Expectation-

Maximization (EM) algorithm (Dempster et al., 1977), which provides an iterative solution to obtaining the MLE in the context of hidden data. For this implementation of ssGMM, the EM algorithm contains several steps:

- **Step 1**: Set the iteration step to $t = 0$ and initialize the model parameters for each class (there are K classes) for a multivariate Gaussian distribution. I achieve this by training a Gaussian Naïve Bayes classifier (Chapter 2.1.1) on the labeled data and obtaining,

$$\theta^{(0)} = \{\pi_k^{(0)}, \mu_k^{(0)}, \Sigma_k^{(0)}\}_{\forall k}. \tag{2.15}$$

- **Step 2**: The E-step. Create a matrix γ that is size $n \times K$ where $n = l + u$. For labeled instances ($i = 1, ..., l$), define $\gamma_{ik} = 1$ if $y_i = k$, and 0 otherwise. For the unlabeled instances ($i = l + 1, ..., l + u$), calculate γ_{ik} via,

$$\gamma_{ik} = \frac{\pi_k^{(t)} \mathcal{N}(x_i; \mu_k^{(t)}, \Sigma_k^{(t)})}{\sum_j \pi_j^{(t)} \mathcal{N}(x_i; \mu_j^{(t)}, \Sigma_j^{(t)})}. \tag{2.16}$$

- **Step 3**: The M-step. Determine $\theta^{(t+1)}$ using the current γ_{ik}. For $k = 1, ..., K$ compute:

$$NL_k = \sum_{i=1}^{l} \gamma_{ik} \qquad NU_k = \sum_{i=l+1}^{l+u} \gamma_{ik} \qquad C = \beta NL_k + (1 - \beta) NU_k$$

$$\begin{aligned} \pi_k^{(t+1)} &= \frac{C}{\beta l + (1 - \beta) u} \\ \mu_k^{(t+1)} &= \frac{\beta \sum_{i=1}^{l} \gamma_{ik} x_i}{C} + \frac{(1 - \beta) \sum_{i=l+1}^{l+u} \gamma_{ik} x_i}{C} \\ \Sigma_k^{(t+1)} &= \frac{\beta \sum_{i=1}^{l} \gamma_{ik} x_i (x_i - \mu_k^{(t+1)})(x_i - \mu_k^{(t+1)})^T}{C} + \frac{(1 - \beta) \sum_{i=l+1}^{l+u} \gamma_{ik} x_i (x_i - \mu_k^{(t+1)})(x_i - \mu_k^{(t+1)})^T}{C} \end{aligned}$$

- **Step 4**: Increment the iteration step, $t = t + 1$
- **Step 5**: Repeat Steps 2-4 until Equation 2.14 converges to a tolerance defined as the percent change in the log-likelihood (e.g., tolerance = 0.1 is one-tenth of a percent)

The matrix γ represents the *soft labels*, or class membership probabilities, for the unlabeled data. Once the algorithm converges, a threshold can be applied to assign a hard classification to the unlabeled data points via,

$$y_i = \arg\max_k \gamma_{ik} \quad \text{for} \quad i = l+1, ..., l+u. \tag{2.17}$$

Implementing this algorithm requires setting the hyper-parameters a priori, which are the tolerance and β. Yan et al. (2017) considers β as the only hyper-parameter, but I find that ssGMM is quite sensitive to the tolerance and choose to optimize it rather than set it to a fixed value.

In ssGMM, there is a critical assumption that the labeled data have been generated by multivariate Gaussian distributions and the unlabeled data use the same parametric model for classification. This cluster assumption states that if two points are in the same cluster, then they likely belong to the same class. However, it does not imply that each class forms an isolated cluster, but rather that we should not observe data of two distinct classes in the same cluster (Chapelle et al., 2006, Chapter 1). For my implementation, this assumption also implies that each class can only be described by one Gaussian cluster (i.e. it cannot be multinomial). If this cluster assumption is violated in any way (e.g., multinomial, non-Gaussian distributions), then semisupervised learning could actually degrade performance (see Figures 3.2 and 3.3 in Zhu & Goldberg, 2009). Therefore, it is important to ensure that the data features for each class can be described by Gaussian distributions a priori.

Computer Code Availability

- *Name of code*: `ssGMM`
- *Developer*: Michael W. Dunham
- *Contact*: Department of Earth Sciences, Memorial University of Newfoundland, St. John's NL A1B 3X5, Canada; mwdunham@mun.ca
- *First release*: 2019
- *Hardware required*: tests were performed on a standard workstation with the following specifications: Intel i5-4750 (4 CPUs) 3.20 GHz processor + 32 GB RAM
- *Software required*: core Python 3 modules: `scikit-learn`, `joblib`, `numpy`, `pandas`, `matplotlib`.
- *Programming language*: the code is written in Python 3
- *How to access the source code*: the source file for the `ssGMM` program, data files, and accompanying Jupyter notebooks that reproduce results from Chapter 4 are provided here: `https://github.com/mwdunham/ssGMM`

2.3 Hyper-parameter tuning

Many of the supervised and semisupervised algorithms discussed in Chapters 2.1 and 2.2 have hyper-parameters that need their values set. Trial-and-error is possible, but manually tuning many parameters (e.g., those for XGBoost) can be cumbersome and inefficient. There are various established techniques in the literature for supervised methods to use what information is available to *automate* the determination of hyper-parameters, and one popular approach is called grid search cross-validation. The first step is considering a range of values for each hyper-parameter; this requires some intuition about the hyper-parameter

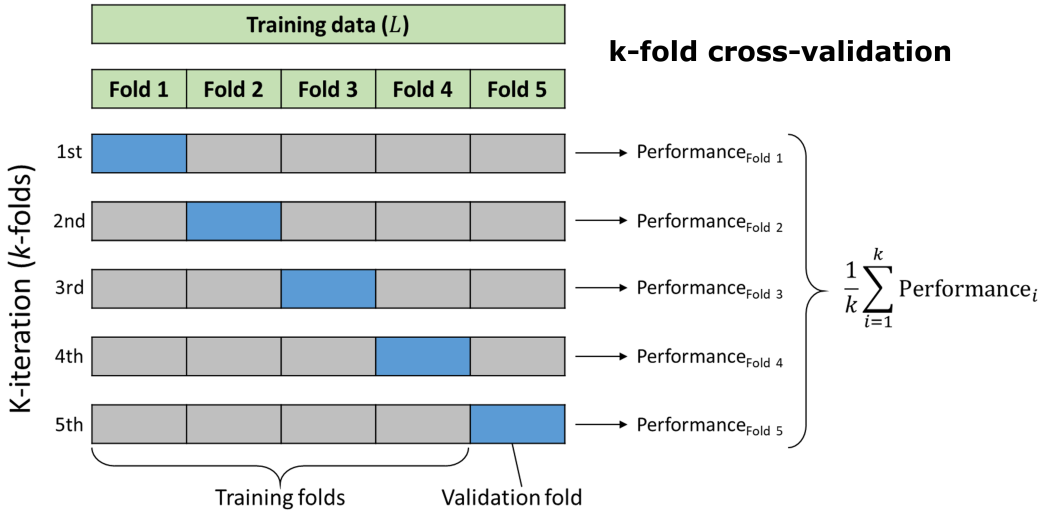

Figure 2.10: The standard k-fold cross-validation method to determine hyper-parameter settings for supervised methods.

and whether or not the goal is to make the algorithm more conservative. The information that is used to determine the hyper-parameters is the training dataset (L) because we know what the labels are for these data. To use the training dataset (L) for cross-validation, we split it into two pieces: a larger training set and a smaller validation set. For each hyper-parameter setting, the algorithm is trained on the training set and then evaluated on the validation set using classification metrics such as accuracy, precision, recall, or F1 (Lever et al., 2016). However, if only one validation set is used, the hyper-parameter setting with the best score may only be optimized for that portion of data, i.e., it may not be a setting that helps the machine learning model generalize well to other data. One remedy is to break the training data into k pieces, or folds, and train the machine learning algorithm with a given hyper-parameter setting on $k-1$ folds, evaluate the performance on the k^{th} fold, and then repeat this process for each fold; this is k-fold cross-validation (CV). For five-fold CV, each hyper-parameter setting has a classification score on each of the five validation sets (Figure 2.10 illustrates this concept). Conventionally, all of the classification scores are averaged and the standard hyper-parameter selection tactic is to select the hyper-parameter combination with the largest mean CV score (Bishop, 2006, Chapter 1.3; Hastie et al., 2009, Chapter 7.10; Krstajic et al., 2014).

Hyper-parameter tuning for semisupervised methods is more challenging. As stated in Chapter 1.2, there is a lack of consensus on systematic approaches for determining hyper-parameters for SSL algorithms because most of the SSL literature that I have encountered neglects to address this subject. This is in stark contrast to SL, where it is common to use k-fold cross-validation. Nonetheless, I pursue three different strategies in this thesis. The first is simply a trial-and-error, heuristic-based approach to trying different hyper-parameter settings and seeing which works best. The second strategy is to simply use the cross-validation approach designed for supervised methods (i.e., using the labeled data only). For semisupervised methods, the validation fold in cross-validation can be treated as unlabeled data to include during training, and the goal is to still predict labels for the validation fold and evaluate the performance. There are some examples of using this approach in the literature (e.g., Gieseke et al., 2014). What is potentially problematic with this approach is that some SSL hyper-parameters relate to the density of the data as a whole (e.g., the number of nearest-neighbors for LP) or give a relative weight between the labeled and unlabeled data (e.g., β for ssGMM and α for LP), and using only the labeled data to determine these hyper-parameters may suggest inappropriate values.

In Chapters 3 and 4, I explore using k-fold cross-validation on L to determine hyper-parameters for LP and ssGMM, respectively, and the results are indeed mixed. In Chapter 5, I find that the heuristic approach and default parameter settings work well, but I explore using a new approach in Chapter 7. Zhang & Lee (2006) use a different hyper-parameter learning technique for a graph-based SSL algorithm similar to the LP method used in this thesis. They use a *leave-one-out* (LOO) cross-validation approach, which assumes that there are as many folds as labeled data (i.e., l folds). What they do differently is, for each held out training sample (i.e., validation fold containing a single data point), they predict its label using the remaining $l - 1$ labeled data *and* the unlabeled data. The predictions for the unlabeled data cannot be used to evaluate the performance of any hyper-parameter setting because the true labels for these data are assumed to be unknown. However, this does not preclude the inclusion of the unlabeled data. The benefit of

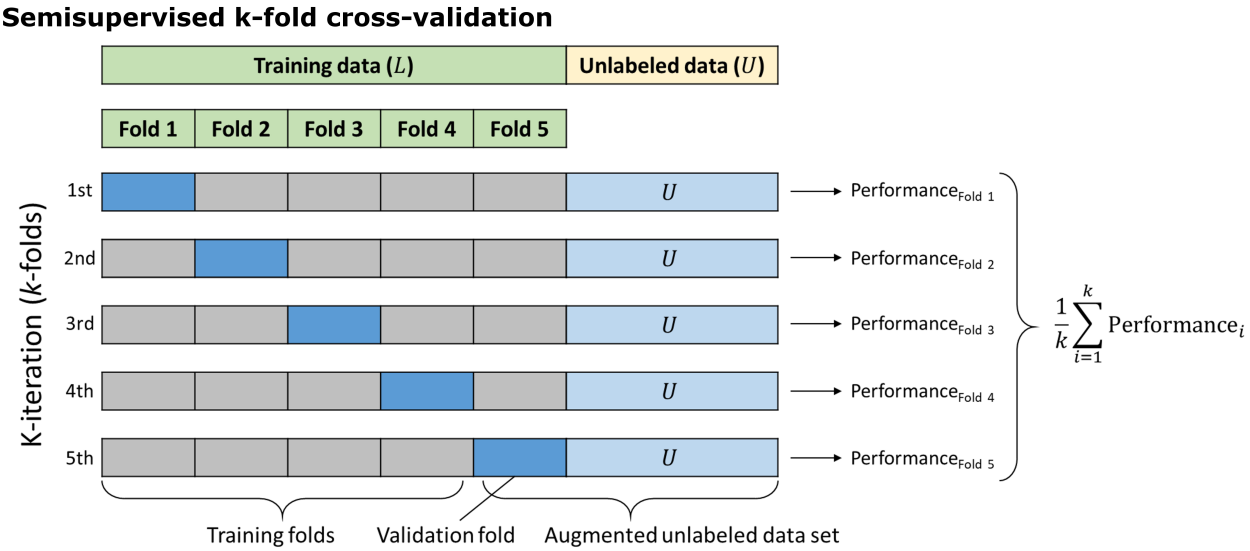

Figure 2.11: The standard cross-validation technique for supervised methods (Figure 2.10) can be modified for semisupervised methods by augmenting the validation fold with the unlabeled data. When predictions are made on the augmented unlabeled data set, only those corresponding to the validation fold can actually be evaluated, but including the unlabeled data can still help inform the learning for semisupervised algorithms.

including the unlabeled data is that they can help better inform the determination of some hyper-parameters. The disadvantage of this LOO approach is that the SSL algorithm must be trained l times for each hyper-parameter setting. This is a significant computational demand. So, rather than pursue the LOO approach, I develop a new technique that builds on this idea, and this is the third approach that I use toward the end of this thesis. I use the same k-fold cross-validation approach used for supervised methods (Figure 2.10), but I augment the validation fold with all (or a fraction) of the unlabeled data from the problem (see Figure 2.11). When predictions are made on this augmented unlabeled data set, only those corresponding to the validation fold can actually be evaluated, but including the unlabeled data can still help inform the learning for semisupervised algorithms. I have labeled this as semisupervised k-fold cross-validation, and the benefit of this approach over the LOO approach is that the SLL algorithm only has to be trained k-times compared to l-times for each hyper-parameter setting.

Chapter 3

Improved well-log classification using semisupervised label propagation and self-training, with comparisons to popular supervised algorithms[1]

3.1 Introduction

Lithofacies classification, which assigns a rock-type or class to specific rock samples on the basis of measured properties, is fundamental in geologic investigations (Dubois et al., 2007). The best source of lithofacies information is core samples acquired from drilled wells; however, cores are not always recovered because of high costs. Due to the limited availability of core samples, a method for *indirectly* estimating lithofacies in wells without core is necessary (Dubois et al., 2007). Wire-line log measurements (i.e. gamma

[1] Dunham, M.W., Malcolm, A. and Welford, J.K., 2020. Improved well-log classification using semisupervised label propagation and self-training, with comparisons to popular supervised algorithms, Geophysics, 85(1), O1–O15.

ray, density, neutron porosity, sonic, resistivity, etc.) have long been recognized as being indirectly related to lithofacies. However, manually assigning lithofacies, using wire-line log measurements, at each depth sample in every well would be impractical; a more practical and automated approach is needed (Dubois et al., 2007).

Over the past few decades, this problem has been widely addressed using both supervised learning (SL) and unsupervised learning (UL). Neural networks were the first supervised machine learning algorithms applied to well-log classification (Baldwin et al., 1990; McCormack, 1991; Rogers et al., 1992), and they continued to be popular for the next two decades (Benaouda et al., 1999; Saggaf & Nebrija, 2000; Maiti et al., 2007; Al-Bulushi et al., 2009; Malvić et al., 2010; Al-Bulushi et al., 2012). Other supervised implementations have included k-nearest neighbors (Dubois et al., 2007), support vector machines (Wang et al., 2013; Hall, 2016) and ensemble methods (Bestagini et al., 2017; Keynejad et al., 2019). Bayesian-based techniques have also been popular for facies classification problems, and these methods encompass both supervised and unsupervised learning. A common supervised Bayesian technique is a Naïve Bayes classifier (Li & Anderson-Sprecher, 2006; Dubois et al., 2007; Grana et al., 2017), and Gaussian mixture models (GMMs) are popular for unsupervised implementations (Gallop, 2006; Grana et al., 2017; Hardisty & Wallet, 2017; Wallet & Hardisty, 2019). More recently, unsupervised GMMs have been extended to include transition probabilities from hidden Markov models (Lindberg & Grana, 2015; Feng et al., 2018a,b).

A challenge with well-log classification is the availability of ground truth, labeled data for supervised learning. It is relatively cheap to collect unlabeled data (wireline data), but obtaining ground truth labels for the unlabeled data can be difficult because extracting, preserving, and storing core samples is costly. Therefore, these classification problems commonly have a paucity of labeled data. For instance, Dubois et al. (2006) train a neural network using 14 wells with core and they use that mapping to predict the lithofacies for 1364 wells without core samples. In this situation, the supervised classifier is trained using roughly 1% of the data and this assumes that the classifier can generalize in classifying the remaining 99%.

This may be a poor assumption, in general, because it is known that training a supervised classifier on a small training set can lead to overfitting.

One solution to this overfitting problem is to use *semisupervised learning* (SSL) techniques because they utilize both the labeled *and* unlabeled data in the training process, which is predicted to achieve more accurate labels for the unlabeled data than SL techniques in the context of limited training data (Chapelle et al., 2006; Zhu & Goldberg, 2009). The semisupervised methods utilized in this paper are (1) label propagation and (2) self-training. Both methods share a few benefits: they are well-established, conceptually simple, and easy to implement. Here I present a case study that applies these two semisupervised methods to lithology classification of wire-line data from well logs made publicly available through an SEG competition held in 2016 (Hall, 2016; Hall & Hall, 2017). However, I also use three supervised algorithms for comparison purposes. My research question of interest is if SSL techniques achieve better predictions for the unlabeled data (in the context of minimal training/labeled data) compared to SL techniques. By comparing the performance of SL and SSL algorithms, I are able to test this hypothesis. The particular geoscience application that I are applying these SSL methods to has already been explored for two decades (i.e. lithofacies classification of well-log data). However, the merit of this work is that it is the first, to my knowledge, that uses semisupervised methods for this application, and to determine their efficacy, I compare their performance to commonly-used supervised techniques. Furthermore, I constrain my self-training technique differently than existing implementations through the use of improved selection criteria.

The outline of this chapter is as follows. First, I discuss SSL and introduce the mathematical concepts behind the two semisupervised methods considered. Second, I give an overview of the well-log dataset used for this study and how I pose it as a semisupervised problem. Next, I provide the results of this study, divided into two parts. The first results section provides the results on the well-log dataset as is (i.e. the global data) for each of the algorithms considered. For the second section, I split the dataset into two separate pieces (i.e. the split data) and investigate how this impacts the performance of all of the algorithms. Lastly, I discuss

the implications of these results and conclude with the outcomes.

3.2 Machine learning methods

Performing the machine learning analysis for this problem requires semisupervised and supervised methods in order to assess which type of technique can achieve better performance, in the context of a small set of training data (L). The semisupervised methods used are label propagation (Chapter 2.2.1) and self-training (Chapter 2.2.2). Recall that LP has a choice between using either the k-NN or RBF kernel. Preliminary tests indicated that performing cross-validation on the training data using the RBF kernel recovered parameters that caused the LP method to perform much better on the testing data compared to the k-NN kernel. The dense edge-weight matrix using the RBF kernel is problematic for larger datasets, but this is not an issue for this study as the dataset is relatively small. Therefore, I proceeded with using the RBF kernel. Self-training also has flexibility for the choice of algorithm it wraps around, but for this study, I perform self-training on LP.

For supervised methods, I choose three different types of algorithms to serve as the basis for comparison: Gaussian Naïve Bayes (GNB), support vector machines (SVMs), and XGBoost. Naïve Bayes (Chapter 2.1.1) is quite simple to implement because it contains no hyper-parameters, but it does assume that the data can be represented with multivariate Gaussian distributions. For SVM (Chapter 2.1.2), I utilize the RBF kernel to allow the algorithm to define non-linear decision boundaries (I found this was necessary given the nature of the data). Consequently, the two hyper-parameters that must be set for SVM are the kernel width, σ, and the regularization constant, C. XGBoost (Chapter 2.1.3) was the algorithm of choice for the winners of the 2016 SEG competition (Hall & Hall, 2017); it is noteworthy that this algorithm outperformed SVMs and even deep neural networks on this dataset. Not surprisingly, this algorithm is slightly more complicated to implement as it has multiple hyper-parameters that must be set, but it appears to be very robust. Since I am using the same dataset for this study as the SEG competition, I thought XGBoost would be the best

Table 3.1: The nine lithofacies for the well-log dataset and their corresponding descriptions. The Label column serves as a key for the colors associated with each facies in Figures 3.1, 3.2, and 3.11. The classes listed in the Adjacent facies column are used to compute the adjacent accuracy metric given by Hall (2016).

Facies	Description	Label	Adjacent facies
Non-marine classes			
1	Sandstone	SS	2
2	Coarse siltstone	CSiS	1,3
3	Fine siltstone	FSiS	2
Marine classes			
4	Siltstone and shale	SiSh	5
5	Mudstone	MS	4,6
6	Wackestone	WS	5,7,8
7	Dolomite	D	6,8
8	Packstone-grainstone	PS	6,7,9
9	Bafflestone	BS	7,8

comparison for the two SSL methods considered.

3.3 The well-log dataset

The well-log data for this study, and from the SEG competition, are from ten wells drilled in the Hugoton and Panoma fields of southwest Kansas and northwest Oklahoma which were made public by the Kansas Geological Survey. These fields lie on the west side of the Anadarko Basin, an asymmetric foreland basin associated with the early Pennsylvanian Ouchita-Marathon Orogeny, and they constitute one of the largest gas-producing areas in North America (Dubois et al., 2006). Fourth-order marine-continental sedimentary cycles reflecting rapid glacioeustatic sea-level fluctuations are responsible for the laterally continuous reservoir (carbonates and dolomites) and seal (marine and non-marine silts/muds) lithofacies of the Council Grove Group. For more details on the geology of the region, see Dubois et al. (2006).

Each of the ten wells in this dataset is complete with wire-line log data and core samples, each recorded at half-foot (0.15 m) increments for a total of 4137 data points. The sedimentary rocks in the core are subdivided into nine lithofacies (or classes) on the basis of rock type and texture, and their descriptions are

given in Table 3.1. The first three classes (1-3) are non-marine facies and the six remaining classes (4-9) are marine facies. The data for each well consist of five wire-line logs (gamma ray, resistivity, photoelectric effect, neutron-density porosity difference, and average neutron-density porosity) and two geologic variables (non-marine/marine (NM_M) indicator and relative formation position). The wire-line logs obviously come from digital measurements of the subsurface, but the two geologic variables are derived from interpretations of these data. Formation tops were determined from the wire-line data that separate the successions into alternating NM and M half-cycles and an NM_M indicator was assigned to intervals based on the depth to the top and base of these formations. The NM_M indicator generally separates the non-marine from the marine facies, and conceptually I interpret the marine and non-marine facies to each exist on a separate *manifold*. Similarly, imagine that the non-marine and marine facies each belong on a separate 'moon' in the context of Figure 2.7 (this is important for the subsequent results section). The relative position variable is assigned to each sample within a formation and varies from 1 (the top) to 0 (the base). These two geologic feature variables were added to the dataset because the facies in the Council Grove continental-marine cycles have predictable vertical stacking patterns (Dubois et al., 2006).

These data involve fuzziness at many levels and this is evident from the cross-plot in Figure 3.1. The NM_M indicator is effective at separating the non-marine (1-3) from the marine (4-9) facies, but notice that there is still considerable overlap within the marine and non-marine facies. This overlap can be attributed to the fact that the facies are not discrete (i.e. they gradually blend from one to another, e.g., non-marine coarse and fine siltstone) and it is well-known that the physical properties of different lithologies are not always unique (Avseth et al., 2005). Since the data are quite fuzzy, one could argue that if a predicted facies is *close* to the truth, then that could still be considered correct. The facies that are deemed close to the truth are indicated in the *adjacent facies* column in Table 3.1. This is used to compute the *adjacent accuracy* metric presented in the results. The absolute accuracy of any classifier is expected to be poor due to the fuzzy nature of this dataset, but the adjacent facies accuracy can be interpreted as an 'in-the-ballpark' accuracy.

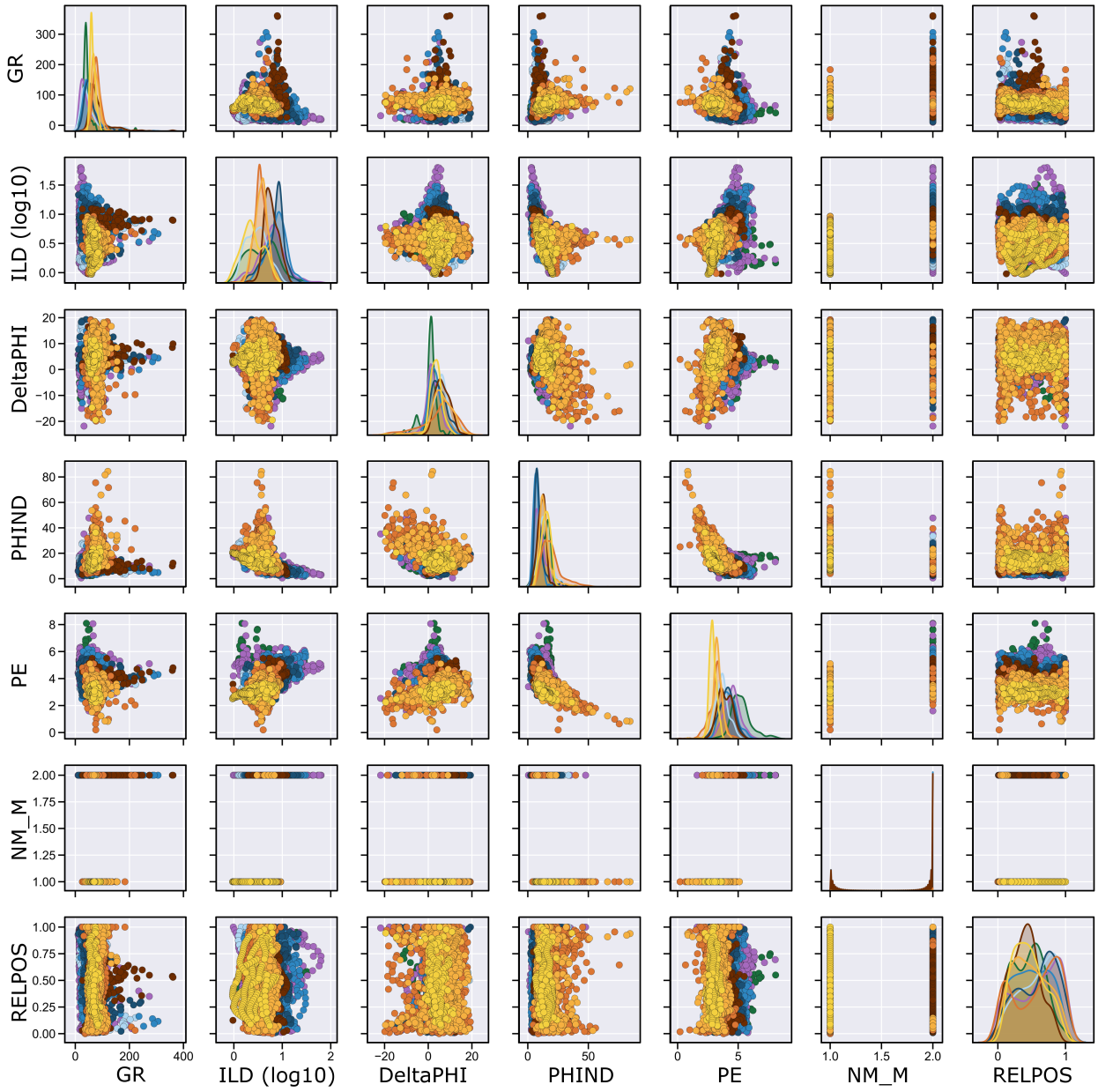

Figure 3.1: A cross-plot matrix showing the behavior for all seven data variables with respect to each other for the entire well-log dataset. The variable abbreviations represent the following: GR = gamma ray, ILD (log_{10}) = resistivity, DeltaPHI = neutron-density porosity difference, PHIND = average neutron-density porosity, PE = photoelectric effect, NM_M = non-marine/marine indicator, and RELPOS = relative position. The first five features are log variables and the last two features are user interpreted geologic variables. For the classes pertaining to each color, see Table 3.1. The distributions of each class for each variable are given by the diagonal elements in the matrix, and these distributions indicate that the classes have a high degree of overlap.

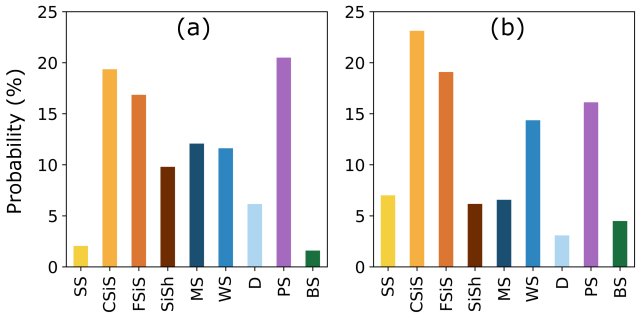

Figure 3.2: The facies distributions for the (a) single labeled well and the (b) nine unlabeled wells. Overall, the single well (a) appears to be representative of the nine unlabeled wells (b), but there are some notable differences for some facies.

The original exercise associated with this dataset was to train a classifier using the data from the ten wells (L) to predict the facies for two unknown wells (U) whose true labels were only known by the competition organizers (this study only has access to the ten wells). In order to simulate a semisupervised scenario for the context of this study, only one well is considered as the labeled data (L) and the goal is to predict the facies for the remaining nine wells (U), which are assumed to be unknown. The challenge then becomes which well to use as L because not all of the wells contain data for each facies. A given classifier cannot make predictions for classes not seen in the training data and this is a limitation that is not unique to semisupervised, but applies to supervised algorithms as well.

Only two of the ten wells contain data for each of the nine facies and I have chosen KIMZEY as the well to represent L. The probability distributions for the labeled well (KIMZEY) and the nine remaining unlabeled wells are shown in Figure 3.2. Notice that the facies distribution for KIMZEY (Figure 3.2a) is relatively similar to the facies distribution of the other nine wells (Figure 3.2b), but there are some noticeable differences. These facies distributions also relate to the class probabilities from self-training; Figure 3.2(a) represents the class prior from Equation 2.10 and Figure 3.2(b) represents what I *hope* Equation 2.11 will find. The scenario that I am simulating here where only one (l = 439 points) out of the total ten wells ($l + u$ = 4137) has core samples (approximately 10% of the entire dataset) could indeed reflect a realistic situation.

Prior to classifying the unlabeled data, the entire dataset must be scaled and normalized in order for

certain algorithms to perform properly (e.g., SVM). The data in Figure 3.1 suggest that there are outliers and the traditional approach of standardizing the data based on the mean could improperly scale the data. Instead, I consider an alternative method (using the *RobustScaler* class from `scikit-learn`) that removes the median and scales the data according to the interquartile range (IQR). The IQR measures the statistical dispersion of the data by taking the difference of the 75th and 25th percentiles (Navidi, 2010, Chapter 1). The next section shows the results of the three supervised and two semisupervised algorithms on the scaled dataset when only the KIMZEY well is used for training.

3.4 Results I - Global data

The three SL and two SSL algorithms all have hyper-parameters that require setting prior to classifying U, and realistically obtaining values for these parameters can only be achieved with the known information (L). As is standard in such problems, cross-validation (CV) is used on L to determine these hyper-parameters (see Figure 2.10). I consistently use 3-fold CV for all algorithms because testing showed that using higher folds gave diminished performance (higher folds made the validation sets too small). CV is not needed for GNB, but SVM has two hyper-parameters (C, σ) and I chose to optimize them over a 20×20 logarithmic grid. There are many hyper-parameters for XGB, but I set the grid choices similar to those from Bestagini et al. (2017) - one of the winners of the SEG competition - and modify them slightly to achieve better performance for this scenario. The challenge for these methods is determining parameter settings that can lead to good predictions for U when the amount of information to learn from in L is limited.

SSL methods still face the same challenge as SL methods of determining optimal settings for hyper-parameters. It is not uncommon to tune SSL algorithm hyper-parameters via CV on L alone (e.g., Gieseke et al., 2014) and I do consider this approach using a 41×41 grid for α and σ. However, using CV on L to determine SSL hyper-parameters may suggest inappropriate values. The α hyper-parameter gives a relative weight between the labeled and unlabeled data (Eq. 2.8), and σ controls the reach of each data point to each

Table 3.2: Results for the supervised and semisupervised algorithms on the global well-log dataset. For the self-training results, the subset size and threshold are fixed to default values of $S = 10\%$ and $T = 50\%$. However, an additional self-training setting is used ($S = 15\%$ and $T = 50\%$) which is marked by the asterisk (*). For the elapsed time, all computations are conducted on a desktop machine (3.2 GHz Intel Core i5 processor) with 16GB RAM and all cross-validations are performed in parallel on four cores. The colored cells correspond to discrete points in Figures 3.4 and 3.5.

	GNB	**SVM** with CV	**XGBoost** with CV	**LP** (default)	**Self-train LP** (default)	**LP** (with CV)	**Self-train LP** (with CV)	**Self-train* LP** (with CV)
Total accuracy (%)	40.64	41.62	44.19	43.54	43.19	45.00	**49.54**	48.76
F1 score (%)	35.91	39.43	42.50	34.19	39.29	38.70	**47.30**	46.58
Total adjacent accuracy (%)	82.59	82.04	83.53	78.66	81.50	83.02	**87.43**	86.78
Number of CV fits	0	400×3	1728×3	0	0	1681×3	1681×3	1681×3
Elapsed time	0.02 s	6.65 s	567.3 s	0.78 s	5.56 s	9.22 s	14.45 s	12.43 s

other (Eq. 2.5); both of these hyper-parameters have some relation to the unlabeled data, and so determining their values from only the labeled dataset may cause LP to be unstable and/or violate its assumptions. For instance, CV on a sparse L may suggest a σ with small reach (i.e., overestimating a value for σ). Therefore, I also test the performance of LP by simply using the default parameter settings ($\alpha = 0.5$, $\sigma = 1.0$). To simplify the problem, I also only consider default parameter values for self-training (S = 10%, and T = 50%) and do not use CV as this would add an additional level of complexity.

The results of the supervised and semisupervised algorithms are summarized in Table 3.2. XGBoost performs the best for the SL methods, but surprisingly not by a large margin. For LP, clearly choosing default parameters is not optimal and the confusion matrix (Figure 3.3a) shows that some marine facies are being classified as non-marine. The NM_M indicator variable should easily split these two groups of classes from each other, but the LP model with default parameters must be causing the non-marine and marine classes to bleed into one another, thus violating the manifold assumption. Performing self-training on the default LP model shows little-to-no improvement and the confusion matrix (Figure 3.3b) indicates that self-training is unable to fix the issues of the underlying default LP model.

Using CV to determine the hyper-parameters for LP ($\alpha = 0.68$, $\sigma = 3.16$) achieves a better accuracy than using the default LP parameters. Figure 3.4(a) shows a grid search over the LP hyper-parameter grid and we see the red dot (LP default) moving to the blue dot (LP with CV) which is closer to the maximum

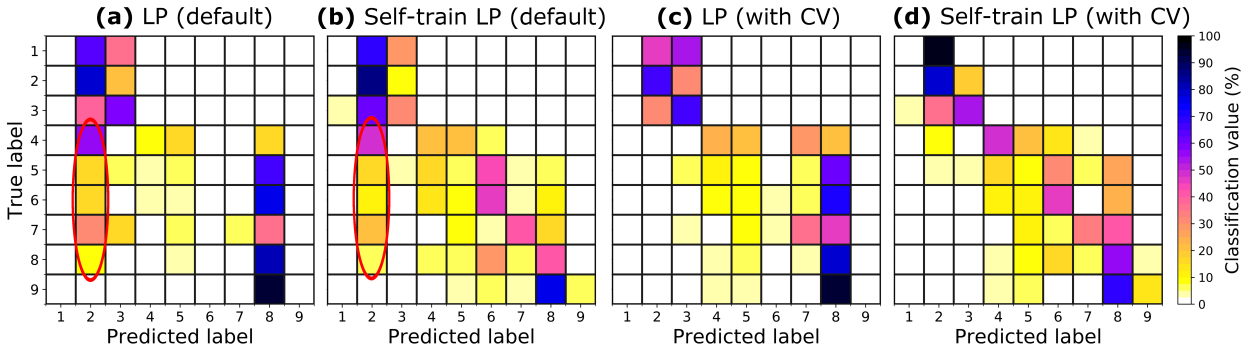

Figure 3.3: The normalized confusion matrices for the (a) LP default, (b) self-trained LP default, (c) LP with CV, and (d) self-trained LP with CV models. The predictions shown by these matrices are for the nine unlabeled wells. Diagonal cells = correct classification, off-diagonal cells = misclassification. The models in (a) and (b) violate the manifold assumption by classifying marine facies as non-marine (indicated by the red ellipses). Performing CV to determine parameter choices for LP helps correct this issue.

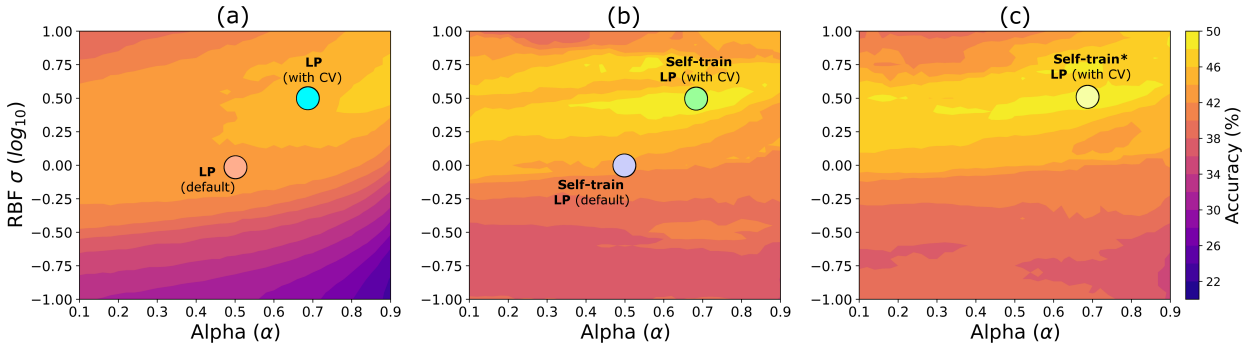

Figure 3.4: A grid search over the LP hyper-parameter grid used for CV. Self-training is performed on the entire LP parameter grid in panel (a) where the self-training parameters are fixed to (b) $S = 10\%$, $T = 50\%$ and (c) $S = 15\%$, $T = 50\%$. The color bar is shared between all panels and the colored circles correspond to the colored entries in Table 3.2.

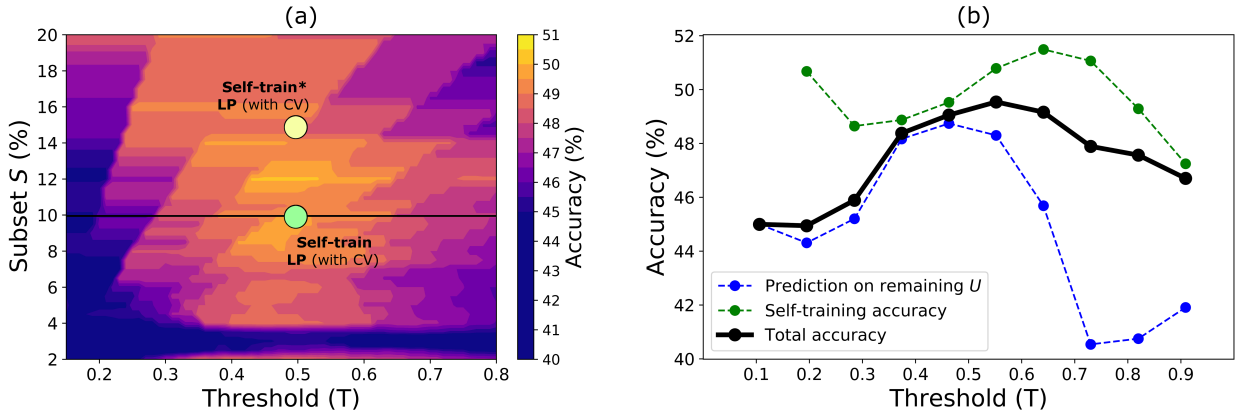

Figure 3.5: (a) A self-training grid search where the LP parameters are fixed to those determined via CV (α = 0.68, σ = 3.16) and the self-training parameters are varied. The color bar is set to show dark colors for all accuracies below 45.0%, the accuracy of LP with CV *prior* to self-training (cyan-colored cell in Table 3.2). (b) The total accuracy of self-training at a given iteration (black line) is a *weighted average* of the accuracy on the points added to W_L (green line) and the accuracy of predictions on the remaining unlabeled points (blue line). The horizontal black line in (a) corresponds to the solid black line in (b).

achievable accuracy on the unlabeled data for this method. Figure 3.3(c) shows that the parameters determined via CV no longer cause the LP model to violate the manifold assumption. However, this model is arguably not outperforming XGBoost and this is likely because the model is biased by training classes with the most points, as seen in Figure 3.3(c) where most of the predictions fall within Classes 2, 3, and 8. Performing self-training on the LP (with CV) model gives a noticeable improvement in accuracy and outperforms XGBoost by a notable margin (i.e. 49.54% vs. 44.19% accuracy). Figure 3.4(b) shows self-training (S = 10%, and T = 50%) applied to the entire LP parameter grid, and note the 6% difference in accuracy between the purple (LP default) and green dots (LP with CV). The improved performance of using self-training on the LP (with CV) model is marked by the predictions clustering closer to the diagonal in Figure 3.3(d).

The self-training parameters thus far have been fixed to default values (S = 10%, and T = 50%) for simplicity. I now investigate how the LP performance can change if the self-training parameters are modified. First, I just consider changing the subset size S to 15% instead of 10%. Applying this new self-training setting to the LP (with CV) model (last column in Table 3.2) achieves a comparable accuracy to using the

default self-training settings on the LP (with CV) model. Similar to Figure 3.4(b), I perform self training with this new set of parameters (S = 15%, and T = 50%) to the entire LP parameter grid and this is shown in Figure 3.4(c). We see overall that there is not much of a difference in accuracy between using these two different self-training parameter settings. Next, I fix the LP parameters to be those obtained via CV ($\alpha = 0.68$, $\sigma = 3.16$) and wI vary the self-training parameters and the result is shown in Figure 3.5(a). I extract the accuracy along S = 10% in Figure 3.5(a) and decompose this total accuracy into its two constituents in Figure 3.5(b) to help illustrate how the performance of self-training varies with threshold. It appears from both panels in Figure 3.5 that the subset size (S) is not as critical as the threshold (T); it seems if self-training runs too long or not long enough that the performance may not improve. Nonetheless, we see that self-training improves the base LP (with CV) model for most of the parameter choices indicating that self-training is relatively robust for this model.

3.5 Results II - Split data

The results discussed in the previous section showed an interesting phenomenon regarding LP violating the manifold assumption when default parameters are used ($\sigma = 1.0$, $\alpha = 0.5$). However, this issue of marine facies being classified as non-marine also occurred for the supervised methods, albeit to a lesser degree (the confusion matrices showing this phenomenon for the SL methods are not shown for brevity). It appears that these algorithms are not able to properly train with the NM_M indicator variable to fully separate the marine and non-marine facies. My remedy to this problem is to decompose the well-log data into two separate datasets *based on the NM_M indicator*. The first dataset consists of 1997 data points associated with NM_M = 1, which correlate to non-marine facies 98.45% of the time, and the second data set consists of 2140 data points associated with NM_M = 2, which correlate to marine facies 98.97% of the time (there is not a 100% correlation due to human error). Essentially, this breaks the dataset into non-marine (NM_M = 1) and marine (NM_M = 2) facies datasets that correspond to classes 1-3 and 4-9, respectively (see Table 3.1). The

Table 3.3: Results for the supervised and semisupervised algorithms when the well-log dataset is split into non-marine and marine datasets. The total accuracy is a weighted average of the NM and M facies accuracies. For the self-training results, the subset size and threshold are fixed to default values of $S = 10\%$ and $T = 50\%$. However, an additional self-training setting is used ($S = 15\%$ and $T = 50\%$) which is marked by the asterisk (*). The colored cells correspond to discrete points in Figures 3.7 and 3.8.

	GNB	**SVM** with CV	**XGBoost** with CV	**LP** (default)	**Self-train LP** (default)	**LP** (with CV)	**Self-train LP** (with CV)	**Self-train* LP** (with CV)
NM facies accuracy (%)	49.21	50.41	54.40	51.07	**59.81**	54.78	61.07	**62.33**
M facies accuracy (%)	32.26	36.60	34.67	34.46	**42.22**	32.64	35.04	**39.17**
Total accuracy (%)	40.64	43.43	44.43	42.67	**50.92**	43.59	47.92	**50.62**
F1 score (%)	35.91	39.50	42.33	35.56	**49.35**	37.99	46.49	**49.50**
Total adjacent accuracy (%)	82.59	84.13	83.88	82.34	**88.67**	83.23	86.94	**88.02**
Number of CV fits	0	$400 \times 3 \times 2$	$1728 \times 3 \times 2$	0	0	$1681 \times 3 \times 2$	$1681 \times 3 \times 2$	$1681 \times 3 \times 2$
Elapsed time	0.02 s	13.29 s	583.5 s	0.36 s	2.77 s	7.56 s	10.32 s	9.59 s

dimension (i.e., number of input features) of the data prior to splitting is seven, but splitting the data based on the NM_M indicator essentially removes this variable and both the data subsets now have six dimensions. Each dataset is treated independently with a separate median scaler and the training data for the non-marine ($l_M = 168$) and marine ($l_{NM} = 271$) sets are still coming from the single well, KIMZEY.

It is worth noting that I am only able to decompose this dataset into two pieces because of the information provided, i.e. the NM_M indicators. Not all well-log datasets will likely have this information, and so performing a similar decomposition on other well-log datasets is perhaps unlikely. However, the potential benefits of splitting *this* classification problem into two pieces are twofold. One, I predict that it will remove the possibility for LP to violate its manifold assumption and perhaps improve the accuracy for all algorithms. Secondly, I can get a better sense of how sensitive the hyper-parameters are for both the SSL and SL algorithms using two related, yet different datasets. If I show that an algorithm works by applying it to only one dataset, this does not necessarily imply that the algorithm is robust. However, if I can show that an algorithm can perform on two datasets, this makes a better case that the algorithm is robust.

The same parameter grids for each algorithm and the same procedures used in the previous section are repeated here, and a summary of the results are given in Table 3.3. Overall, we see higher accuracies obtained for the non-marine facies than the marine facies. The fact that both facies groups have significant overlap within them (see Figure 3.1) leads me to conclude that the higher accuracy for the non-marine facies

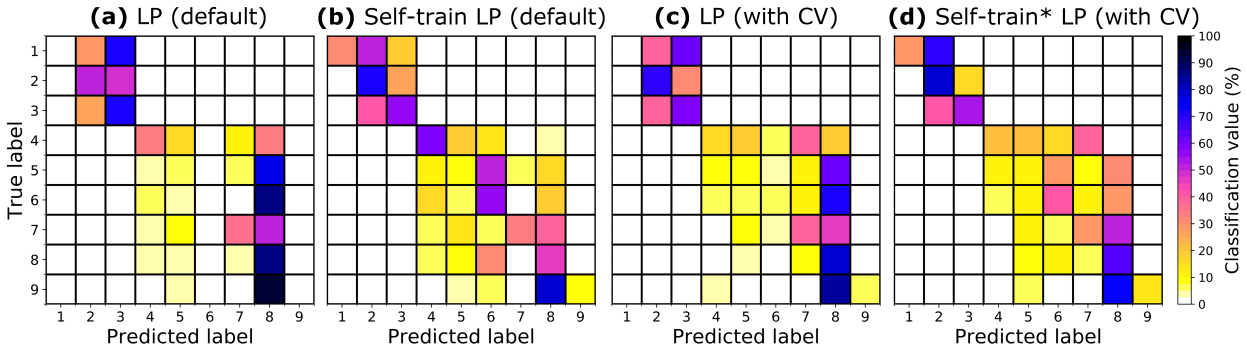

Figure 3.6: The normalized confusion matrices for the (a) LP default, (b) self-trained LP default, (c) LP with CV, and (d) self-trained* LP with CV models when the well-log dataset is split into non-marine and marine sets. The predictions shown by these matrices are for the nine unlabeled wells. The asterisk marks the self-training settings of $S = 15\%$ and $T = 50\%$. The default LP model (a) and its self-trained counterpart (b) no longer violate the manifold assumption. The LP (with CV) model (c) and self-training* LP (with CV) model (d) give comparable matrices and performance to the default models (panels a and b).

comes from having a smaller number of classes compared to the marine facies. For the SL methods, splitting the data into two sets appears to benefit SVM the most while GNB and XGBoost perform similarly. The LP model with default parameters improves slightly after splitting the data into two sets. The confusion matrix in Figure 3.6(a) also shows that the manifold assumption is no longer being violated (compare to Figure 3.3a). However, this model is still performing poorly compared to the supervised baselines due to the predictions being biased by the training classes with the most points. Performing self-training on the default LP model gives a drastic improvement in performance (over 6% higher in total accuracy than XGB) which is also marked by the predictions clustering closer to the diagonal in Figure 3.6(b). Here we see the self-trained LP (default) model performing much better when the data are split compared to not, and this must be attributed to decomposing the data into two pieces as that is the only factor that has changed.

Performing CV to determine the LP hyper-parameters for the non-marine and marine datasets gives mixed results. The LP parameter grid plot in Figure 3.7(a) indicates that CV chooses an optimal parameter combination for the non-marine dataset ($\alpha = 0.90$, $\sigma = 3.55$), but Figure 3.7(d) indicates that this is perhaps not true for the marine dataset ($\alpha = 0.10$, $\sigma = 2.00$). Figure 3.7(d) shows that the CV parameters for the marine dataset lie outside the high-accuracy region. Ideally for an $\alpha = 0.10$, the σ clearly needs to be smaller

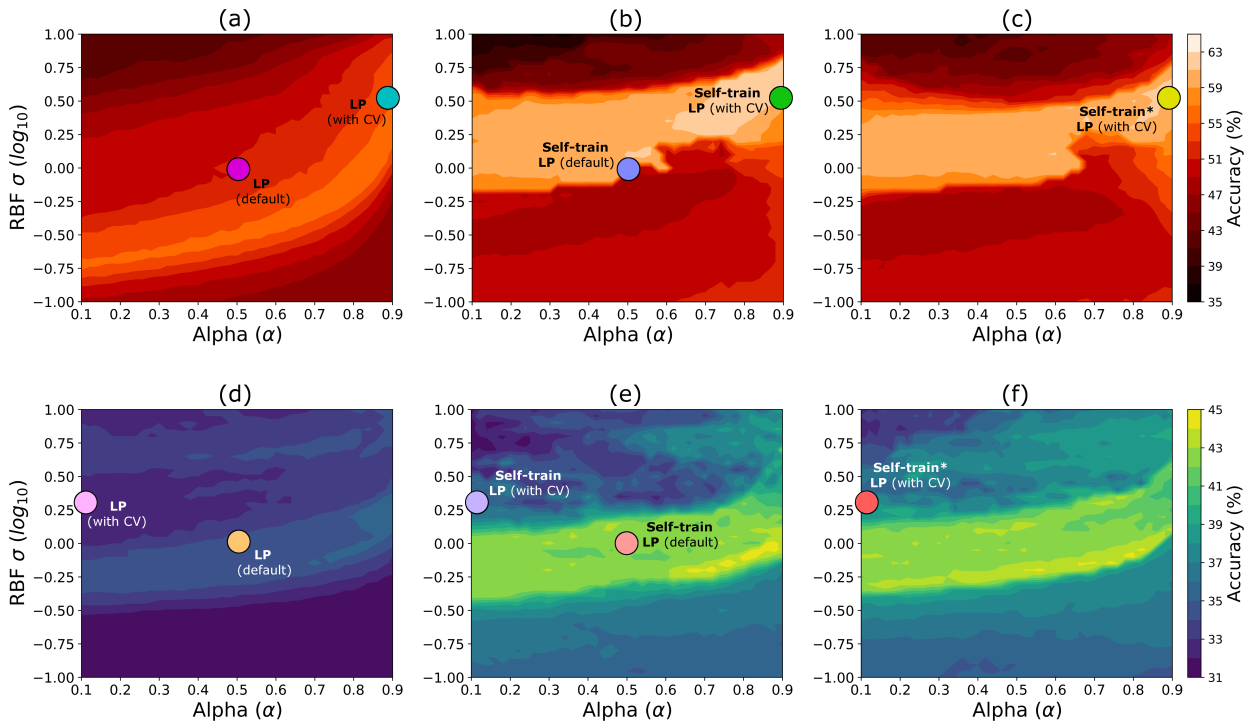

Figure 3.7: Grid searches over the LP hyper-parameter grids for the (a) non-marine and (d) marine datasets. Self-training ($S = 10\%$, $T = 50\%$) performed on the entire (b) non-marine and (e) marine LP parameter grids. Self-training* ($S = 15\%$, $T = 50\%$) performed on the entire (c) non-marine and (f) marine LP parameter grids. Note that the colorbar is shared for all non-marine panels (a-c) and marine panels (d-f). The colored circles correspond to the colored entries in Table 3.3.

(i.e., the reach of each data point needs to be larger). Here, we are seeing an example of what was mentioned in the previous section where there is a risk of performing CV on a sparse L because it can overestimate the value for σ; I interpret this as violating the smoothness assumption. Similar to before, the LP (with CV) model is still not outperforming XGBoost and the predictions are biased by the training classes with the most points (Figure 3.6c). The performance of the LP models (with CV) for both the non-marine and marine datasets translates to the performance of the subsequent self-training for each of these models. Self-training the LP (with CV) model on the non-marine data gives excellent performance (see Figure 3.7b), but self-training the LP (with CV) model on the marine data only results in minimal improvement (see Figure 3.7e). However, when combined, this model still manages to outperform the XGBoost model by 3-4% in all metrics, but does not perform as well as the self-trained LP (default) model.

Once again, I vary the self-training parameters and explore how the performance of self-training can change if the parameters themselves change. Similarly to what was done before on the global dataset, I consider changing the default subset size S from 10% to 15%. Applying this new self-training setting to the LP (with CV) model improves both the non-marine and marine accuracies slightly as indicated by Table 3.3 and the grid plots in Figures 3.7(c) and 3.7(f). The corresponding confusion matrix for this model is given in Figure 3.6(d) and it shows a clear improvement over the base model with no self-training (Figure 3.6c).

I also fix the LP parameters to their default settings and to those determined via CV and I vary the self-training parameters for each case. In Figures 3.8(a) and 3.8(b) I show the self-training variations for the default LP parameters ($\sigma = 1.0$, $\alpha = 0.5$). Figure 3.8(b) clearly shows that self-training the marine data is very robust, but Figure 3.8(a) shows that self-training the non-marine data *requires* the subset size to be above 10%. In a sense, the opposite phenomenon is observed for the self-training variations in Figures 3.8(c) and 3.8(d) when the LP parameters are fixed to those determined via CV (non-marine: $\alpha = 0.90$, $\sigma = 3.55$, marine: $\alpha = 0.10$, $\sigma = 2.00$). We observe in Figure 3.8(c) that self-training the non-marine data is fairly robust, but we see in Figure 3.8(d) that self-training the marine data is more sensitive to the hyper-parameter choices. I hypothesize that the self-training variations shown in Figures 3.8(a) and 3.8(d) are not robust because the underlying LP models are not optimal. However, the underlying LP models for Figures 3.8(b) and 3.8(c) are much closer to optimal and the subsequent self-training variations are much more robust (for the underlying LP models refer to Figures 3.7a and 3.7d)

Recall the original purpose for decomposing this dataset into two pieces was to remove the possibility for LP to violate the manifold assumption and to perhaps observe an improvement in performance. The confusion matrices in Figure 3.6 clearly show that splitting the data into marine and non-marine subsets prevents the manifold assumption from being violated. However, using CV to determine the LP parameters on the marine dataset appears to have overestimated σ, thus violating the smoothness assumption. Despite these difficulties, splitting the data does provide a marginal increase in accuracy for both the SL and SSL

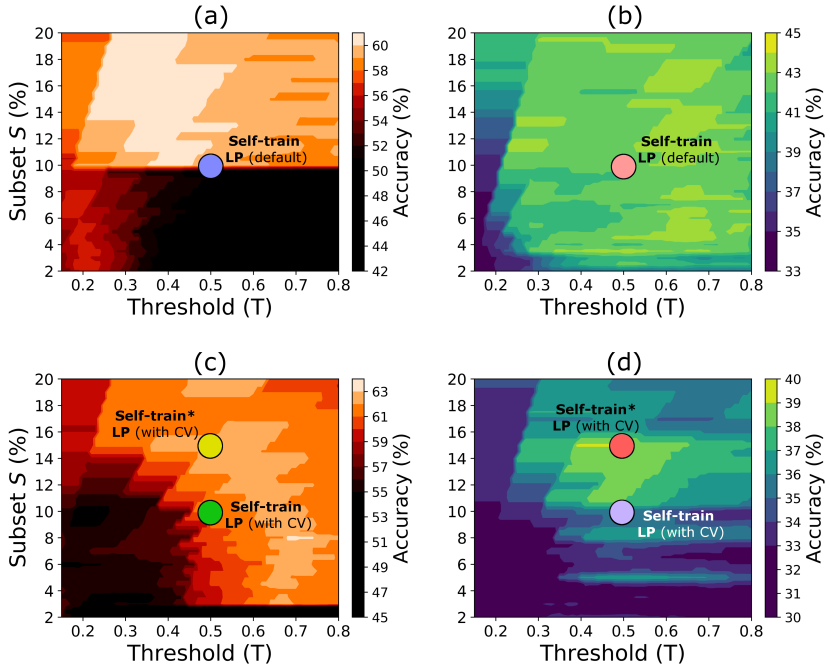

Figure 3.8: Fixing the LP parameters to default values ($\alpha = 0.50$, $\sigma = 1.00$) for the (a) non-marine and (b) marine data and varying the self-training parameters. (c) Fixing the LP parameters to those determined via CV ($\alpha = 0.90$, $\sigma = 3.55$) for the non-marine data and varying the self-training parameters. (d) Fixing the LP parameters to those determined via CV ($\alpha = 0.10$, $\sigma = 2.00$) for the marine data and varying the self-training parameters. Note that the color bar is different for each panel and dark colors in each color bar correspond to the accuracy of the underlying LP model *prior* to self-training (or worse). The colored circles correspond to the colored entries in Table 3.3.

algorithms. When comparing Tables 3.2 and 3.3, the highest accuracies for self-training are a few percent higher in Table 3.3 for all metrics. Consequently, the benefits shown here provide a justification for decomposing a classification problem into smaller subsets if possible.

3.6 Discussion

3.6.1 Label propagation and self-training implications

The results presented here have three outcomes that I wish to elaborate on, and the first focuses on implications of the label propagation and self-training techniques. It appears that the supervised methods are always able to match or outperform the label propagation technique (without self-training). The confusion matrices for LP (e.g., Figures 3.3a, 3.3c, 3.6a, and 3.6c) all show that the predictions are being biased by the training classes with the largest number of points (see Figure 3.2a). However, my new self-training technique helps distribute the predictions to their appropriate classes as shown in Figures 3.3(d), 3.6(b), and 3.6(d).

At this stage, I now have supporting evidence that suggests why my self-training technique would perform better than those proposed by others. Recall from Chapter 2.2.2 that the problem with other self-training approaches (Rosenberg et al., 2005; Liu et al., 2013) is that they always select the points in U with the highest probability to add to L and these may always be points associated with the most-frequent classes. These forms of self-training would accentuate the unbalanced predictions of LP (e.g., Classes 2, 3, and 8 for this dataset) and ignore classes with fewer points. In other words, these methods only care about the class with the maximum probability for each data point and ignore the probabilities from the other classes. Figure 3.9 shows the soft classification matrix $\mathbf{F}$ for the LP (with CV) model (when trained on the global data) for one of the wells included in U. While I did not generate results using the other self-training approaches, it is easy to imagine that their predictions would be dominated by Classes 2, 3, and 8 and the resulting confusion matrices would likely look very similar to those of the base LP models (e.g. Figure 3.6a). However, my

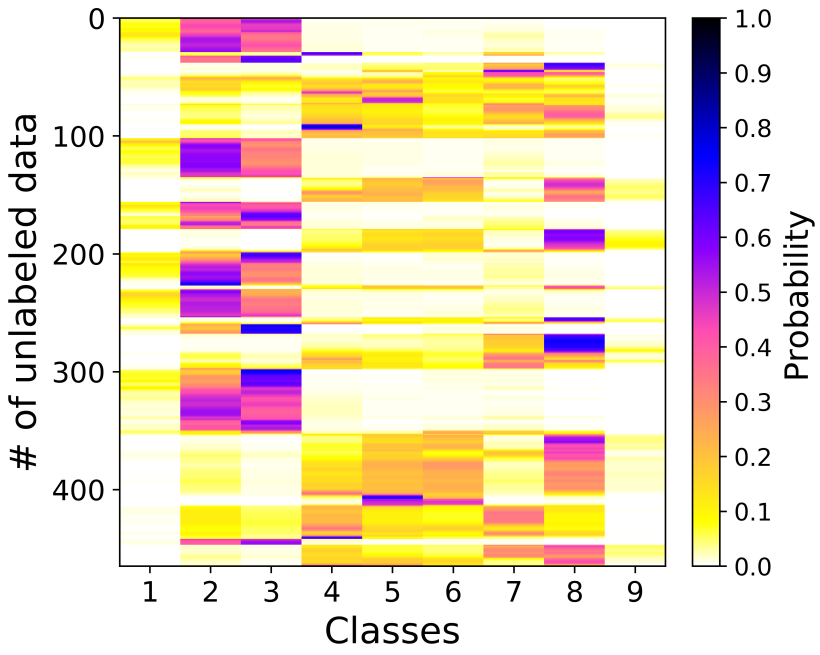

Figure 3.9: The soft classification matrix (**F**) for the LP (with CV) model (when trained on the global data) for one of the nine unlabeled wells (ALEXANDER). The highest probabilities are most prominent for Classes 2, 3, and 8 which not surprisingly also have the most points (see Figure 3.2). The predicted class probability derived from the entire matrix **F** on all nine wells is $\pi_k = [4, 22, 23, 7, 10, 9, 6, 17, 2]\%$.

self-training method takes into account *all* the probabilities for each data point. Figure 3.9 shows that most data points do have a measurable, yet smaller, probability of belonging to other classes even though there is a clear class with the highest probability for those points. The predicted class probability (Equation 2.11) captures this information by summing all the probabilities for each class (e.g. columns in Figure 3.9) and dividing by the total number of points. These results clearly show that my self-training approach improves the classification accuracy, which is confirmed by the predictions moving closer to the diagonal in the confusion matrices (e.g. Figure 3.6b).

However, these results suggest that there are certain situations when my self-training method performs poorly (assumption violations shown in Chapters 3.4 and 3.5). I suggested previously that the performance of this self-training method can diminish if the underlying LP model is not optimal. Although, it is possible that the self-training process could also be causing problems. Here, I investigate these claims with further analysis. Figures 3.9 and 3.10 show two soft classification matrices (**F**), one for the LP (with CV) and

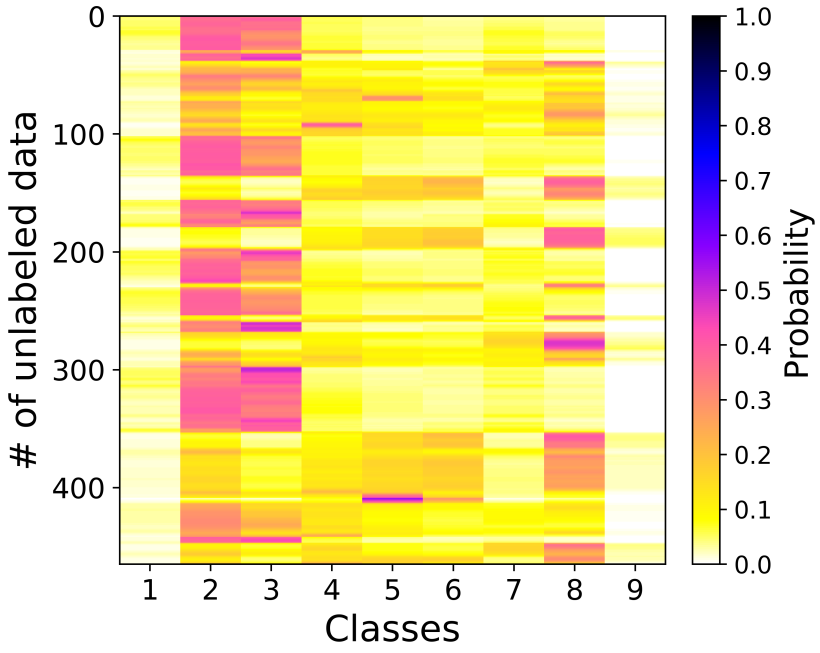

Figure 3.10: The soft classification matrix (**F**) for the LP (default) model (when trained on the global data) for one of the nine unlabeled wells (ALEXANDER). The probabilities bleeding between marine and non-marine classes is a clear indication that the manifold assumption is being violated. The predicted class probability derived from the entire matrix **F** on all nine wells is $\pi_k = [4, 24, 21, 9, 9, 9, 6, 16, 2]\%$.

another for the LP (default) models on the global data. It is evident that these matrices are different and it is obvious why the **F** shown in Figure 3.10 violates the manifold assumption because of the probabilities bleeding between marine and non-marine classes; the bleeding is signified by the profusion of lower probabilities (yellow colors) that is present in Figure 3.10, but not in Figure 3.9. Recall that the predicted class probability (π_k) is derived from the matrix **F**, and even though these two matrices look vastly different, they surprisingly give very similar probabilities. If the predicted class probabilities for these two models are nearly the same, then the self-training selection criteria for these two models must also be the same. This implies that the difference in performance between these two models *must* be coming from the underlying LP model (where the poor model, default LP, is allocating the highest probabilities to the wrong classes), and *not* the self-training process. Intuitively, this seems reasonable because if parameters are chosen for the underlying LP model that do not satisfy the smoothness and manifold assumptions, then we cannot expect self-training to improve the classification (i.e. garbage in, garbage out). The grid search plots in Figures

3.4 and 3.7 suggest that this is not always the case because some choices of α and σ that give poor LP performance actually give improved performance upon subsequent self-training. However, if self-training does improve a poor LP model (which is not always the case), the improvement is generally minor. The highest achievable accuracies with self-training occur in hyper-parameter regions close to where the base LP model is performing the best (all the results shown in Figures 3.4 and 3.7 support this claim). Luckily, in some instances, I am able to realistically recover these optimum LP parameters via CV which perform very well with subsequent self-training.

3.6.2 Results interpretation

In both results sections, the performance of the supervised and semisupervised methods is presented on the unlabeled data as a whole, but how the algorithms perform on each of the nine unlabeled wells is also worth considering. Table 3.4 shows the performance of the XGBoost and self-train LP (with CV) models trained on the global data for each of the unlabeled wells. For seven of the nine wells, the self-train LP model is outperforming the XGBoost model by a notable margin. For two of the wells, the XGBoost model does outperform the self-train LP model, but only marginally so. The same conclusions can be drawn from the split data (table not shown). In summary, even on a per-well basis, the coupled LP and self-training method is achieving better accuracies overall than XGB.

Up until this point, the performance of the semisupervised methods has been evaluated through numerical comparisons with supervised methods, but additional comparisons can be made *visually*. Figure 3.11 shows a predicted facies comparison between the self-trained LP (with CV) and XGBoost models on the global data for one of the nine unlabeled wells. I choose to show SHANKLE (first row in Table 3.4) because the performance of the two methods on this well is characteristic of their average performance on all the unlabeled wells (additional wells are not shown for brevity). The XGBoost prediction is visibly noisy and chaotic in appearance and there is a reasonable explanation as to why. An argument is made in Chapter 1.1

Table 3.4: The global data prediction accuracies for the XGBoost and self-train LP models decomposed into individual accuracies for each unlabeled well. The RECRUIT F9 well is a synthetic well created by the competition organizers that only contains Class 9 with its purpose being to help better constrain this class (Hall, 2016). The training data have very few points for Class 9 (Figure 3.2a) and so the classification accuracies for this synthetic well are quite poor. The total accuracies correspond to those in Table 3.2 and are computed by taking the weighted averages for all nine wells.

Unlabeled well name	# points	XGBoost accuracy (%)	Self-train LP accuracy (%)
SHANKLE	449	46.33	**51.45**
CROSS H CATTLE	501	29.94	**32.53**
NEWBY	463	40.60	**45.36**
LUKE	461	**60.52**	58.57
CHURCHMAN BIBLE	404	**45.79**	44.80
ALEXANDER	466	49.14	**59.44**
SHRIMPLIN	471	45.86	**54.99**
NOLAN	415	41.69	**54.70**
RECRUIT F9	68	8.82	**20.59**
Total	3698	44.19	**49.54**

that claims supervised methods are prone to over-training when the amount of training data is small, and this is analogous to over-fitting in an under-determined inverse problem; this is what is being observed here with the XGBoost prediction. However, a common solution to stabilize under-determined inverse problems is to add an additional term to the objective function involving the model parameters, otherwise known as *regularization* (Aster et al., 2005). Regularization typically smooths the objective function and prevents the predicted data from fitting noise. Similarly, one can think of semisupervised learning as supervised learning where a regularization term is added that includes the unlabeled data (Zhu & Goldberg, 2009). This explains why the self-training facies prediction in Figure 3.11 is much less chaotic and visibly matches the true facies better than the XGBoost prediction.

While the focus of this paper is not on computational efficiency, some interesting observations can be made regarding the elapsed time of both the SSL and SL methods (see Tables 3.2 and 3.3). GNB does not perform the best for the SL methods, but it is extremely fast to run because there are no hyper-parameters to optimize via CV. To achieve a quick classification of the data, perhaps this method still has merit. While XGBoost performed the best for the SL algorithms, it is computationally demanding due to the number

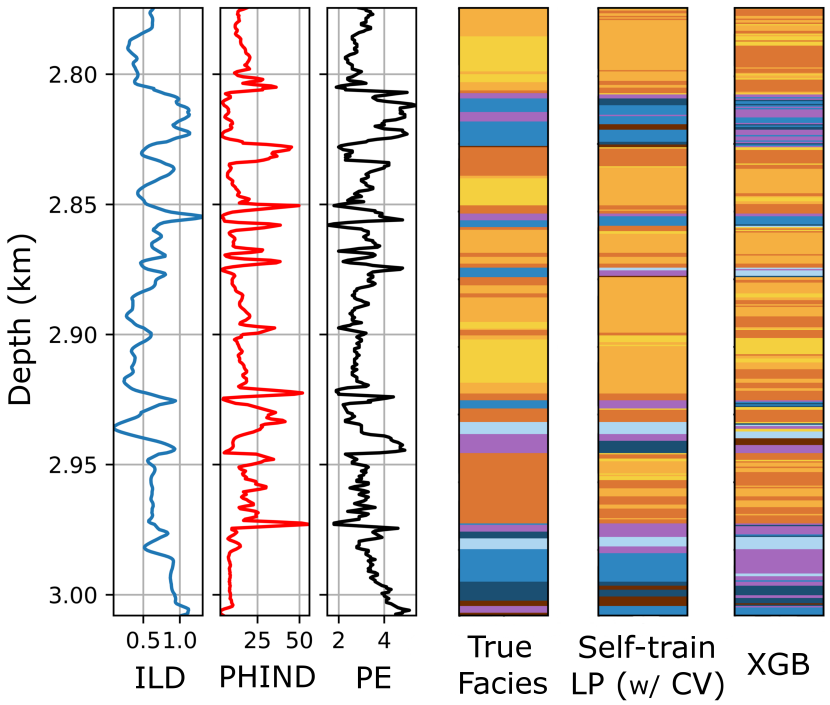

Figure 3.11: A comparison of the self-trained LP (2nd to last column of Table 3.2) and XGBoost facies predictions to the true facies for one of the nine unlabeled wells (SHANKLE). Three of the five logs shown for reference (ILD = resistivity, PHIND = average neutron-density porosity, PE = photoelectric effect). The facies predictions using the LP model visibly match the true facies better (e.g., less chaotic and oscillatory) compared to the XGBoost prediction. The associated labels for the facies colors are indicated in Table 3.1.

of parameters that must be optimized via CV. Comparatively, using CV on LP requires far less time than XGBoost for the same number of CV fits, and the self-training process appears to be computationally cheap. Therefore, not only is self-training LP outperforming XGBoost in terms of classification accuracy, it is also doing so in terms of computational requirements.

3.6.3 Overall classification performance

The final subject I wish to discuss is the overall performance of the machine learning algorithms on this dataset. In my scenario, the overall accuracies and F1-scores for all the algorithms are quite low (e.g. the best F1-score that I recover for XGBoost is 42.5%) and I think that this can be attributed to low training data and poor class separation. The winners of the SEG machine learning competition attempted to address these issues by using higher numbers of training data (all ten wells) and employing a feature expansion scheme, and they also apply a post-classification median filter on the predictions. Feature expansion is a process of generating new features from existing features that can sometimes help separate classes from each other when there is considerable overlap between classes in the original data. The larger training dataset combined with the median filtering improved the F1-score for their XGBoost implementation to roughly 55% on two withheld wells, and then incorporating feature expansion improved the F1-score to 62% (Bestagini et al., 2017; Hall & Hall, 2017). Despite the improvement that could be gained by using these schemes, I do not apply them to my study for obvious reasons. Using more training data would undermine the theme of this paper and feature expansion is not considered for simplicity reasons. Furthermore, I do not want to filter the classification outputs because I feel that the performance of an algorithm is best represented by its raw outputs.

Nonetheless, even after all these changes by the competition participants, the classification accuracy is still only roughly 60%, which is still quite poor. Different datasets with a similar number of classes are able to achieve much higher accuracies. Take the digit recognition dataset MNIST for instance that contains

ten classes, 60,000 training images, and 10,000 testing images (LeCun et al., 1998). The classes for this dataset are relatively separable and a simple linear classifier can achieve an accuracy of 92.4% (LeCun et al., 1998). Class separation is likely what distinguishes the performance of the MNIST and well-log datasets because geological facies are not as discrete as numbers in the feature space. Therefore, it appears that the underlying issue for the well-log dataset is still poor class separation, despite the feature expansion efforts of the competition participants. In order to improve the accuracy of well-log classification problems, perhaps more discriminatory logs are needed because it is from these logs that the expanded features are made, but including even more training data could also be beneficial for those classes with fewer points (e.g. Classes 1 and 9).

3.7 Conclusion

The purpose of this research is to investigate if label propagation and my version of self-training could outperform supervised algorithms in the context of scarce labeled data. To test these algorithms, I simulate a semisupervised scenario with a well-log dataset where only one out of the ten wells is assumed to have labels (i.e. core samples) and the task is to predict the labels for the remaining nine wells. I generate results from the global data and also from the dataset when it is decomposed into two subsets, and the latter shows marginal improvements in accuracy. However, similar phenomena are observed in both results sections and so the take-away messages from both sections are complementary. Label propagation by itself appears to be biased by classes with more data points in this unbalanced dataset. However, coupling label propagation with my self-training method appears to be quite powerful and outperforms the supervised algorithms. Other self-training approaches are likely to be biased by unbalanced datasets, but my self-training method incorporates a predicted class probability to alleviate this issue. The results also indicate that my self-training method is relatively robust and it only performs poorly (occasionally) when the underlying label propagation model is not optimal. Cross-validation is generally able to find optimum hyper-parameter

choices for label propagation, but not always.

In summary, my self-training method, when applied to label propagation, is able to outperform the supervised methods if the assumptions (smoothness and manifold) for the underlying label propagation model are met. These findings support the hypothesis that, in the context of low amounts of labeled data, incorporating the unlabeled data into the training process (i.e. semisupervised learning) can give better predictions than standard supervised methods. Based on these results, and those from other disciplines, I propose semisupervised methods should be considered for geophysical applications where labeled data are scarce.

Chapter 4

Improved well-log classification using semisupervised Gaussian mixture models and a new hyper-parameter selection strategy[1]

4.1 Introduction

In the previous chapter, I show that the self-training process of incrementally adding unlabeled points with the highest label propagation (LP) confidence to the labeled dataset can be effective, but a disadvantage of LP is that it is a transductive method. Transductive algorithms operate by learning an internal mapping of the existing labeled and unlabeled instances to classify the unlabeled data. As a result, transductive algorithms must be retrained using any additional unlabeled data that labels are sought for (i.e., it is not a classifier). This

[1]Dunham, M.W., Malcolm, A. and Welford, J.K., 2020. Improved well-log classification using semisupervised Gaussian mixture models and a new hyperparameter selection strategy, Computers and Geosciences, 140, 1–12.

chapter is a continuation of Chapter 3 using the same well-log dataset, but here, I utilize a different technique called semisupervised Gaussian mixture models (ssGMM, Chapter 2.2.3). This method has similar benefits to the self-training LP technique from Chapter 3 of being well established and easy to implement, but the unique benefit of ssGMM is that it is an inductive algorithm. Inductive algorithms are designed to make predictions for new unlabeled data without needing to retrain the algorithm. In larger problems, the amount of unlabeled data could number in the millions, and including all of the unlabeled data in training, which would be required of transductive algorithms, may be computationally demanding. Inductive algorithms are advantageous in these situations because they can instead use a subset of the unlabeled data during training and then the learned model can be used to classify the remaining unlabeled data. While a few different implementations of ssGMM do exist in the literature (Nigam et al., 2000; Zhu & Goldberg, 2009; Xing et al., 2013; Yan et al., 2017), none of which provide open-source code. One source of merit for this work is that my ssGMM code is open to the public (see Chapter 2.2.3), which is the first open-source ssGMM code to my knowledge.

I also try to improve the machine learning algorithm predictions through the use of a new hyper-parameter selection strategy. The conventional hyper-parameter selection approach utilizes a cross-validation scheme on the training data and the hyper-parameter combination with the largest mean cross-validation score is used to train the machine learning model (Bishop et al., 1998; Hastie et al., 2009; Krstajic et al., 2014). However, the cross-validation process also produces standard deviations that, to the best of my knowledge, have yet to be directly utilized in standard hyper-parameter selection schemes. The strategy that I introduce here selects a hyper-parameter combination based on simultaneously using the mean and standard deviation scores coming from cross-validation, rather than the default procedure that only relies on the mean cross-validation scores.

These ideas are applied to the same lithofacies classification problem from Chapter 3 using the same setup to facilitate comparisons. The first goal of this work is to determine if ssGMM can outperform a

robust supervised algorithm, XGBoost, in the context of this well-log classification problem with limited training data. The results from ssGMM are also compared to the self-training LP results from Chapter 3 to see how the two semisupervised algorithms compare. The second goal is to evaluate the efficacy of this new hyper-parameter selection strategy by showing how algorithm performance (both SL and SSL methods) varies using the new selection strategy versus the conventional approach. I also hope this new hyper-parameter selection strategy will perhaps convince readers to re-evaluate how hyper-parameters are conventionally chosen during training and recognize the applicability of this strategy to other classification problems.

4.2 Methods

4.2.1 Machine learning methods

To assess if the ssGMM method (Chapter 2.2.3) can outperform supervised methods in the context of small training sets on my well-log classification scenario, I need supervised methods to serve as a basis of comparison. The first supervised method I consider is a Gaussian Naïve Bayes (GNB) classifier because it represents the fully-supervised version of ssGMM and the output of GNB is the starting condition for ssGMM (see Equation 2.15). Comparing the performance of ssGMM to GNB will also indicate if including the unlabeled data into the training process is beneficial. Implementing GNB is trivial because this algorithm contains no hyper-parameters and no cross-validation is required. For the second supervised method, I use XGBoost, similarly to Chapter 3. XGBoost is a natural choice given that it was the winning algorithm for the 2016 Society of Exploration Geophysicists (SEG) machine learning competition (Hall & Hall, 2017). Since I use the exact same dataset as the competition for this study, XGBoost is likely the best supervised method to compare against ssGMM in terms of performance.

4.2.2 Hyper-parameter selection strategies

Supervised and semisupervised algorithms commonly have hyper-parameters that need to be tuned, and the process of choosing a set of hyper-parameters is called hyper-parameter selection. The selection process first consists of splitting the training data (L) into training and validation sets and then each hyper-parameter setting is evaluated on the validation data using classification metrics such as accuracy, precision, recall, F1, or the area under the curve (AUC) for ROC (receiver operating characteristic) curves (Lever et al., 2016). If only one validation set is used, it is well-known that there is a larger risk of overfitting the training data, which leads to poor prediction accuracies. One remedy is to break the training data into k pieces, or folds, and train the machine learning algorithm with a given hyper-parameter setting on $k-1$ folds, evaluate the performance on the k^{th} fold, and then repeat this process for each fold; this is grid search k-fold cross-validation (CV). This concept is discussed in Chapter 2.3. If the training data are shuffled and this process is repeated 5 times (i.e. 5-repeated 5-fold CV), then there are 25 classification scores for each hyper-parameter (Krstajic et al., 2014). Conventionally, all of the classification scores are averaged and the standard hyper-parameter selection tactic is to select the hyper-parameter combination with the largest mean CV score (Bishop, 2006, Chapter 1.3; Hastie et al., 2009, Chapter 7.10; Krstajic et al., 2014), and the same can be said for averaging AUC values in multi-class situations (Hand & Till, 2001; Fawcett, 2006). This hyper-parameter combination is then used for training a machine learning model. The hyper-parameter selection functions in `scikit-learn` in Python use the same selection approach, and when there is a conflict (i.e. multiple hyper-parameter combinations with the same mean CV score), the least-complicated combination is selected.

The standard hyper-parameter selection strategy only uses the mean CV scores, but there are also accompanying *standard deviation* CV scores that, to the best of my knowledge, have yet to have been directly leveraged in any selection process. What I propose here is a new selection strategy that *simultaneously* uses the mean and standard deviation CV scores to select a hyper-parameter combination, i.e. a simultaneous

mean and standard deviation (SMSD) score,

$$SMSD_j = \alpha \left(\frac{\bar{\boldsymbol{x}}_{\boldsymbol{j}} - \bar{x}_{CVmin}}{\bar{x}_{CVmax} - \bar{x}_{CVmin}} \right) + (1 - \alpha) \left(\frac{\sigma_{CVmax} - \boldsymbol{\sigma}_{\boldsymbol{j}}}{\sigma_{CVmax} - \sigma_{CVmin}} \right) \tag{4.1}$$

where $\bar{\boldsymbol{x}}_{\boldsymbol{j}}$ and $\boldsymbol{\sigma}_{\boldsymbol{j}}$ are the mean and standard CV scores for the j^{th} hyper-parameter combination, $\bar{x}_{CVmin}$ and $\bar{x}_{CVmax}$ are the minimum and maximum mean CV scores on the hyper-parameter grid, σ_{CVmin} and σ_{CVmax} are the minimum and maximum standard deviation CV scores on the hyper-parameter grid, and α gives a relative weight between the mean and standard deviation CV scores. Generally, α can vary on the interval $[0, 1]$, but allowing it to change adds a level of complexity. For the context of this paper, I fix $\alpha = 0.5$ to let the mean and standard deviation scores *equally* contribute to the decision. The hyper-parameter selection scheme using SMSD is simply choosing the hyper-parameter combination with the highest SMSD score, and if there is a conflict, then the least-complicated combination is chosen. For XGBoost (XGB), the least-complicated hyper-parameters are those with smaller values, and for ssGMM, this is the hyper-parameter combination with the smallest complexity value as defined in Figure 4.1. This scheme is analogous to regularized inverse problems where we not only care about data misfit, but we also care about some measure of the model norm, and we weight the contribution of both these factors in the objective function.

What I am trying to address with the SMSD method is that hyper-parameter combinations with the highest mean CV score may not always be optimal. For instance, if a given hyper-parameter combination has the highest mean CV score, but also has the highest standard deviation CV score, is this the optimal choice? Arguably, the optimal choice is one that is both accurate (i.e. high mean) and precise (i.e. low standard deviation) and my SMSD method will select a hyper-parameter combination that has a balance between the mean and standard deviation scores. It is worth mentioning that if the highest mean CV score and the lowest standard deviation CV score align, then the SMSD hyper-parameter choice is equivalent to the default method's choice. Where the hyper-parameter choices from the SMSD and the traditional methods

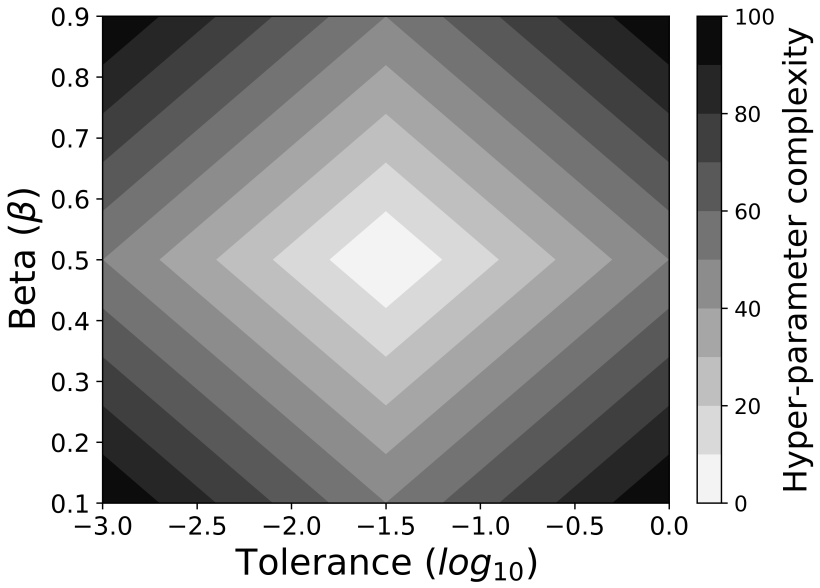

Figure 4.1: The assigned hyper-parameter complexity values for the ssGMM method. If a given hyper-parameter selection technique has a conflict, the hyper-parameter combination with the lowest complexity value is chosen. This choice of complexity for ssGMM is made to penalize extreme values for β and the $tolerance$, and favor those that are closer to what I deem to be default values: $(\beta, tolerance) = (0.5, log_{10}[-1.5])$.

will differ is when the highest mean and the lowest standard deviation CV scores do not align, and what I try to investigate in this paper is if the hyper-parameter combination chosen using the SMSD method trains models that perform better on testing data in these situations.

4.3 Well log dataset

The well-log dataset used for this study is the same dataset that was used for an SEG machine learning competition held in 2016 (Hall, 2016; Hall & Hall, 2017), but the data were ultimately made public by the Kansas Geological Survey. The data consist of ten wells (nine are real, and one is synthetic) drilled in the Hugoton and Panoma fields of southwest Kansas and northwest Oklahoma, and I refer the interested reader to Dubois et al. (2006) for a discussion of the geology of this region. All ten wells contain wire-line log data (i.e. the instances x_i) and core samples (i.e. the associated labels y_i) recorded at half-foot increments for 4137 total data points. Dubois et al. (2006) determine that there are nine lithofacies, or classes, in this dataset

and the first three (1-3) are non-marine and the remaining six classes (4-9) are marine lithofacies (see Table 3.1). The features, or dimensions, for each instance consist of five wire-line logs and two geologic variables interpreted by users. Figure 3.1 shows a cross-plot of each of the seven features plotted against each other for the entire dataset. Notice that the NM_M indicator is effective at distinguishing the non-marine from the marine classes, but the classes *within* the non-marine and marine categories are not linearly separable with considerable overlap. It is well-understood that rock units are not always discrete and their physical properties are not unique, and this can lead to poor class separability (Avseth et al., 2005). Misclassifications of this dataset are expected to occur as a result of this, but one could argue that if a predicted lithofacies is *close* to the true facies, then this could still be classified as correct. For the machine learning competition associated with this dataset, Hall (2016) introduces an *adjacent accuracy* metric that deems lithofacies that occur close to each other depositionally (i.e. via Walther's Law) are also considered correct. These adjacent facies are indicated in Table 3.1 and I also include this metric.

The machine learning competition using this dataset was structured for the competitors to train a classifier using all ten wells (L) and the competition organizers would test the competitors' classifiers on two additional withheld wells (U), but these extra two wells are not available to the public. As in Chapter 3, to properly simulate a semisupervised situation with this dataset, I restructure the classification problem so that only one well is used as the labeled data (L) and the objective is to predict the lithofacies for the remaining nine wells (U). However, this one well for training has to be chosen carefully because not all the wells contain every class (i.e. if a class is not present in the training data, then predictions cannot be made for it). For continuity purposes, I choose the same labeled well done in Chapter 3, KIMZEY, and for the original justification of this choice, see Chapter 3.3. Figure 3.2 depicts the distribution of points for both the labeled and unlabeled data in this situation. The distributions between the labeled and unlabeled data are similar with minor differences, but what is noteworthy in Figure 3.2(a) is that some classes have very few points; the consequences of this are discussed in Chapter 4.5.2 below. Prior to any classification, these data are

scaled and normalized using the *RobustScaler* class from `scikit-learn`.

4.4 Results

4.4.1 Initial ssGMM test

I begin with a testing stage to investigate how ssGMM behaves when applied to this dataset. No cross-validation procedures are used and I simply train the algorithm on the one labeled well (KIMZEY) using default hyper-parameter values ($\beta = 0.50, tolerance = log_{10}[-1.5]$). This initial test shows that the objective function for ssGMM does not converge (dashed black line in Figure 4.2). Recall from Chapter 2.2.3 that the inherent cluster assumption of ssGMM is subject to violation if the data cannot be represented by multivariate Gaussians. Notice in Figure 3.1 that most of the variables exhibit Gaussian-like distributions, but the exception is the NM_M indicator which is a binary, bimodal variable. Algorithmically, this makes the covariance matrices for each class poorly defined and taking the inverse of these matrices (required to compute $\mathcal{N}$ in Equation 2.16) causes an instability.

My remedy for this problem, as is done in Chapter 3, is to remove the NM_M indicator variable by decomposing the well-log data into two separate datasets based on the NM_M indicator. This decomposes the original dataset into non-marine (NM_M = 1) and marine (NM_M = 2) facies datasets [2] that correspond to classes 1-3 and 4-9, respectively (see Table 3.1). The ssGMM algorithm is trained again (using default hyper-parameters) on the non-marine and marine subsets of KIMZEY, and Figure 4.2 shows that the objective function easily converges for both. This is evidence that the NM_M variable is the underlying cause for the ssGMM algorithm not converging on the global data. Moving forward, I apply all algorithms to the separate non-marine and marine datasets.

[2] It is noteworthy that the NM_M indicator is a variable that is created from interpretations of the wireline log variables for the purpose of this dataset (Dubois et al., 2006). Therefore, performing this data splitting procedure on *other* datasets is unlikely, but it is advantageous for ensuring the cluster assumption of ssGMM is not violated in the context of this dataset.

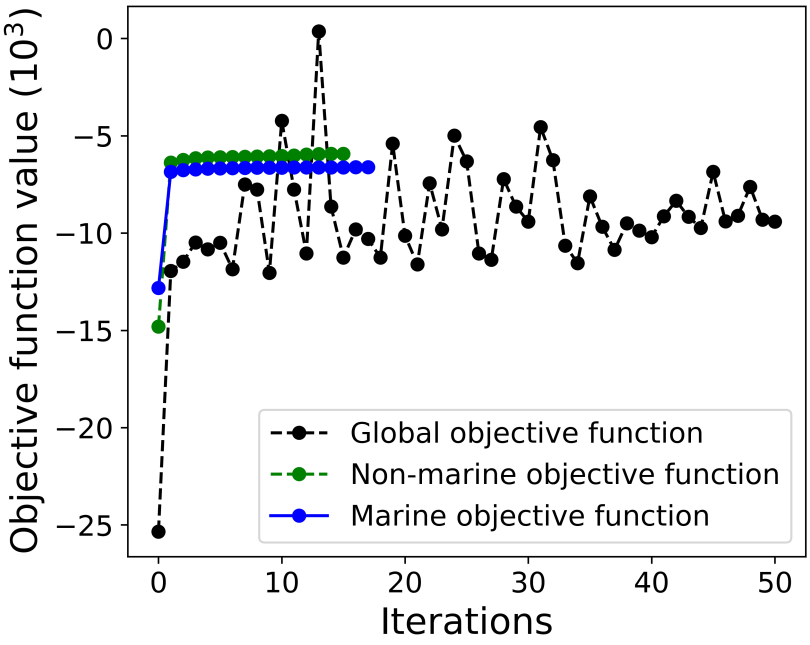

Figure 4.2: The behavior of the ssGMM objective function (Equation 2.14) with respect to the number of iterations when trained on the single labeled well using default hyper-parameter settings ($\beta = 0.50, tolerance = log_{10}[-1.5]$). The ssGMM method does not converge (dashed black line) because the NM_M indicator variable is not Gaussian distributed, but splitting the data into two pieces (non-marine and marine) allows the algorithm to converge.

4.4.2 Comparison to Chapter 3

Based on my necessity to decompose the well-log data into non-marine and marine datasets for ssGMM, I can make direct comparisons here to the previous self-training LP results from Chapter 3. In this previous work, I considered 3-fold CV with total accuracy as the classification metric to train LP, and I use the same scheme here for ssGMM to make a direct comparison. I mirror the choice of the XGB hyper-parameter grid from Bestagini et al. (2017), but I modify it slightly to achieve better performance for this situation. The hyper-parameter grid that I consider for ssGMM is indicated in Figure 4.1 with 31 logarithmically spaced values for the $tolerance$ and 17 linearly spaced values for β. I first evaluate the total accuracy of each hyper-parameter choice on the testing data for both non-marine and marine datasets; this will indicate how close future ssGMM hyper-parameter selections are to the maximum achievable accuracies (see Figure 4.3). A summary of the results is given in Table 4.1 where the last row is the self-training label propagation result taken from Table 3.3 in Chapter 3.5. The mean and standard deviation 3-fold CV scores for ssGMM are given in Figure 4.4, and the different hyper-parameter selections are indicated by circles. Using the SMSD score (Equation 4.1) as the hyper-parameter selection strategy is not the focus of this particular section, but Figures 4.4(c) and 4.4(f) indicate that using the SMSD score gives the same hyper-parameter combinations as the default approach shown in Figures 4.4(a) and 4.4(d). In Table 4.1, we see that ssGMM and self-training label propagation are performing better than GNB and XGB. However, the rows indicating the best possible performance for XGB and ssGMM on the unlabeled data suggest that there is room for improvement for both of these methods.

4.4.3 Improving XGB and ssGMM performance through hyper-parameter selection

Only 3-fold CV is considered in Chapter 3, but I consider a higher fold here to see if improvement can be gained for XGB and ssGMM. I choose to have at minimum one data point from each class in each fold, so the highest fold I can test is 7-fold because Class 9 only has seven points (see Figure 3.2). However,

Table 4.1: Testing data performance for the supervised methods (GNB, XGB) and the semisupervised methods (ssGMM, self-training label propagation) where 3-fold CV with total accuracy as the classification metric is used to train all algorithms. The F1 and adjacent accuracy metrics are not used for hyper-parameter selection, but are merely presented for comparison. The self-train LP row is taken directly from Table 3.3. All computations are conducted on a desktop machine (3.2 GHz Intel Core i5 processor) with 16GB RAM and all cross-validations are performed in parallel on four cores. The XGB (best) and ssGMM (best) rows give the highest achievable total accuracy on the non-marine and marine testing data (indicated by the black bulls-eyes in Figure 4.3). The models representing the salmon colored cells correspond to the same-colored circles in Figures 4.3 and 4.4.

Machine learning algorithm	Non-marine accuracy (%)	Marine accuracy (%)	Total accuracy (%)	F1 score (%)	Total adjacent accuracy (%)	Total # of CV fits	Elapsed time
GNB	49.21	32.26	40.64	35.91	82.59	0	0.023 s
XGB	54.40	34.67	44.43	42.33	83.88	$700 \times 3 \times 2$	269.3 s
XGB (best)	56.92	38.15	47.43	44.27	84.23	N/A	N/A
ssGMM	55.17	38.09	46.54	43.07	88.37	$527 \times 3 \times 2$	136.9 s
ssGMM (best)	58.39	39.59	48.89	45.24	89.21	N/A	N/A
Self-train LP	62.33	39.17	50.62	49.50	88.02	$1681 \times 3 \times 2$	9.59 s

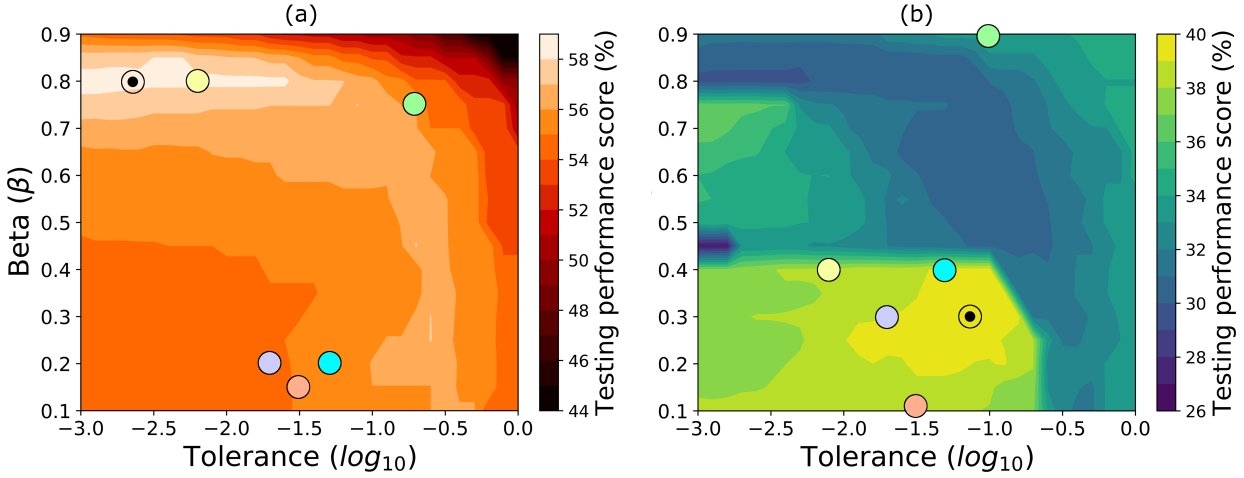

Figure 4.3: The total accuracy of each ssGMM hyper-parameter choice on the testing data for the (a) non-marine and (b) marine datasets. This does assume that H (Equation 2.3) is known, which is unrealistic, but these panels help indicate how close the ssGMM models are to the highest accuracy zones. The black bulls-eyes indicate hyper-parameter choices that give the maximum achievable accuracy. The colored circles represent various ssGMM models, and their detailed numerical performances on the nine unlabeled wells are indicated in Tables 4.1 and 4.2.

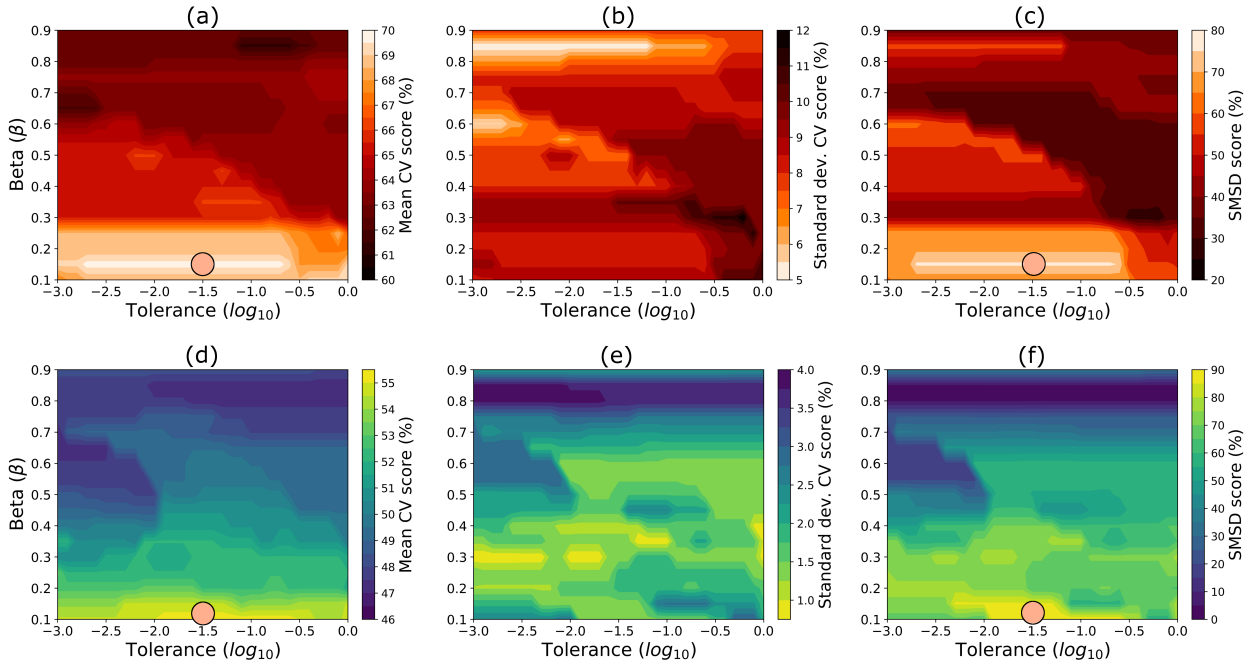

Figure 4.4: The mean, standard deviation, and SMSD 3-fold cross-validation scores using ssGMM for the non-marine (a, b, c) and marine (d, e, f) training datasets, respectively. Total accuracy is used as the classification metric. The standard hyper-parameter selection approach selects hyper-parameter combinations indicated by the circles in panels (a) and (d). The SMSD hyper-parameter selection approach selects the hyper-parameter combinations indicated by the circles in panels (c) and (f). For this situation, the chosen hyper-parameter combinations for each approach are identical. See Figure 4.3 and Table 4.1 for the testing performance of the models indicated by circles.

Table 4.2: Testing data performance for XGB and ssGMM using 5-fold and 5-repeated 5-fold cross-validation with total accuracy as the classification metric. Hyper-parameter selection for both algorithms in both situations is achieved using the standard and SMSD (Equation 4.1) approaches. GNB is not included because it contains no hyper-parameters and its performance is the same as shown in Table 4.1. The models representing the colored cells correspond to the same-colored circles in Figures 4.3, 4.5, and 4.6. The best performing XGB and ssGMM models are denoted by the *.

Machine learning algorithm	Non-marine accuracy (%)	Marine accuracy (%)	Total accuracy (%)	F1 score (%)	Total adjacent accuracy (%)	Total # of CV fits	Elapsed time
5-fold cross-validation							
ssGMM	54.84	39.11	46.89	43.09	89.21	$527 \times 5 \times 2$	151.46 s
ssGMM (SMSD)	55.27	39.38	47.24	43.39	89.40	$527 \times 5 \times 2$	152.31 s
XGB	50.30	34.56	42.34	40.06	82.86	$700 \times 5 \times 2$	460.10 s
XGB (SMSD)	50.30	36.33	43.24	40.41	82.88	$700 \times 5 \times 2$	458.66 s
5-repeated 5-fold cross-validation							
ssGMM	55.99	33.87	44.81	38.35	84.86	$527 \times 5 \times 5 \times 2$	520.19 s
ssGMM (SMSD)*	58.28	38.20	48.13	44.53	89.29	$527 \times 5 \times 5 \times 2$	550.41 s
XGB*	55.27	34.78	44.92	42.63	83.64	$700 \times 5 \times 5 \times 2$	2188.6 s
XGB (SMSD)	53.75	34.72	44.13	42.35	83.69	$700 \times 5 \times 5 \times 2$	2190.2 s

5-fold CV is standard and that is what I test here. Table 4.2 (top) summarizes the results, and Figure 4.5 gives the mean, standard deviation, and SMSD 5-fold CV scores for ssGMM. We see a minor performance improvement for ssGMM using 5-fold rather than 3-fold CV (compare to Table 4.1), and using the SMSD score gives a marginal improvement over the default hyper-parameter selection for 5-fold CV. Using 5-fold CV for XGB deteriorates the performance slightly compared to using 3-fold CV (see Table 4.1), but using the SMSD score selects hyper-parameters for XGB that perform slightly better in the 5-fold CV case.

Using 5-fold CV produces five classification metric scores for each hyper-parameter combination to compute the mean and standard deviation CV scores from. However, five values are arguably not enough for the computed mean and standard deviation scores to be statistically significant. A solution to this problem is N-repeated k-fold CV (discussed in Chapter 4.2.2) and I consider 5-repeated 5-fold CV so there are 25 classification scores for each hyper-parameter combination. Table 4.2 (bottom) summarizes these results, and Figure 4.6 gives the mean, standard deviation, and SMSD 5-repeated 5-fold CV scores for ssGMM. While the default hyper-parameter selection for ssGMM on the non-marine data performs well, the default for the marine data is quite poor and hinders the overall performance. However, the hyper-parameter choices obtained using the SMSD score improve the performance on both datasets. The default hyper-parameter

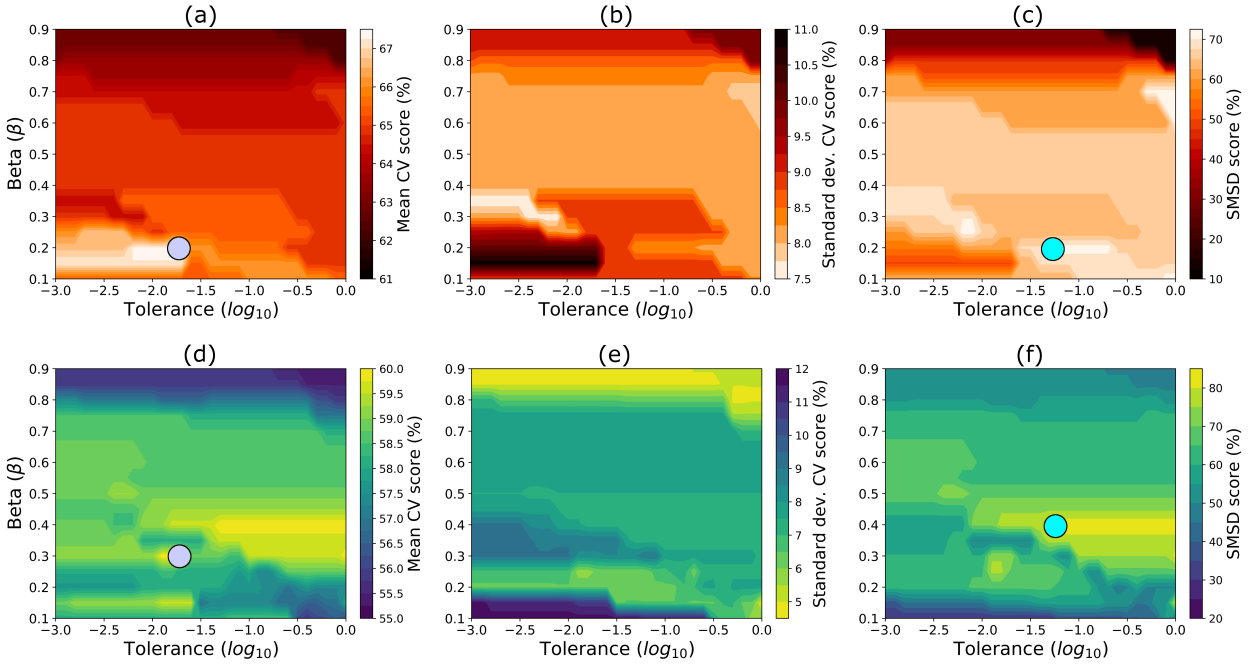

Figure 4.5: The mean, standard deviation, and SMSD 5-fold cross-validation scores using ssGMM for the non-marine (a, b, c) and marine (d, e, f) training datasets, respectively. Total accuracy is used as the classification metric. The standard hyper-parameter choices are indicated by the purple circles in panels (a) and (d), and the SMSD hyper-parameter choices are indicated by the cyan circles in panels (c) and (f). For this situation, the SMSD hyper-parameters give a slightly better performance on the testing data (see Figure 4.3 and Table 4.2).

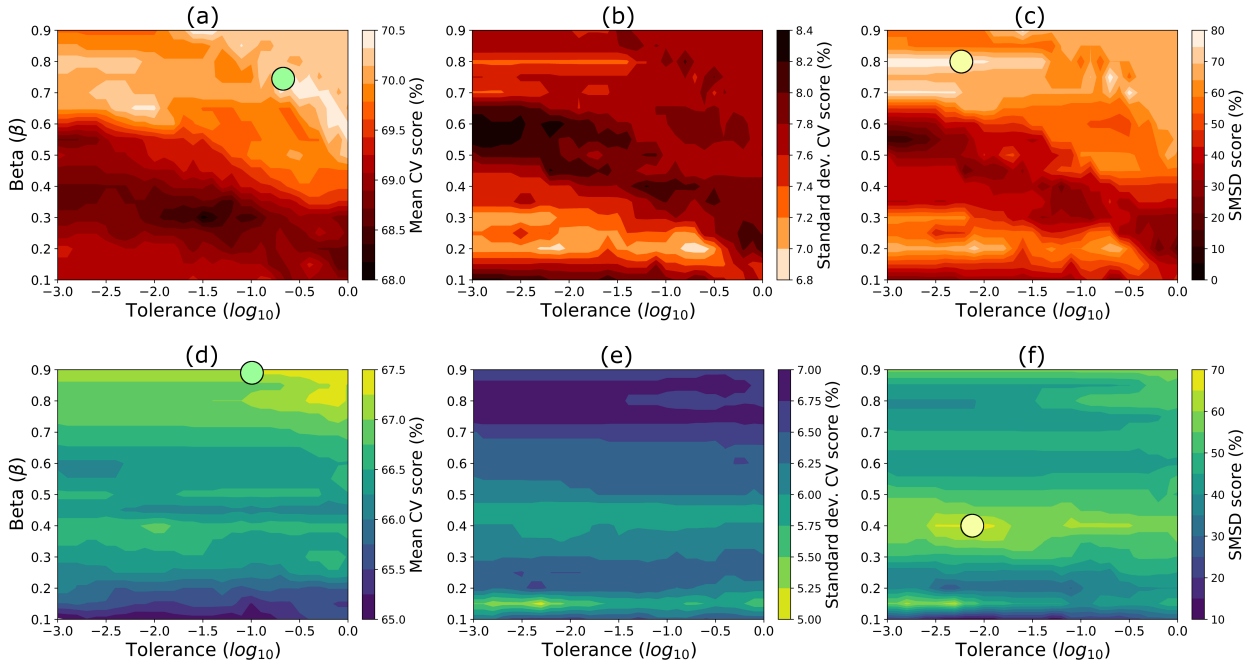

Figure 4.6: The mean, standard deviation, and SMSD 5-repeated 5-fold cross-validation scores using ssGMM for the non-marine (a, b, c) and marine (d, e, f) training datasets, respectively. Total accuracy is used as the classification metric. The standard hyper-parameter choices are indicated by the green circles in panels (a) and (d), and the SMSD hyper-parameter choices are indicated by the yellow circles in panels (c) and (f). For this situation, the standard hyper-parameter choice for the non-marine data (a) performs well on the testing data, but the standard hyper-parameter choice for the marine data (d) does not. However, the SMSD score chooses standard hyper-parameters that significantly improve the testing performance of both the non-marine and marine datasets. See Figure 4.3 and Table 4.2 for the testing performance of the models indicated by circles.

choices for XGB perform well, but the choices obtained using the SMSD score diminish the performance slightly.

4.5 Discussion

4.5.1 Overall performance comparison

One objective of this study is to assess if ssGMM can outperform the considered supervised methods (GNB and XGB) in the context of minimal training data. Recall from Chapter 4.2.1 that I consider GNB because it represents the fully-supervised version of ssGMM. Tables 4.1 and 4.2 show that ssGMM outperforms GNB in every circumstance and this indicates, for this algorithm, that including the unlabeled data into the

training process substantially improves its performance. The results also demonstrate that ssGMM is able to outperform XGB in terms of accuracy (1-5% for all classification metrics) and computation time (2-4× faster) in this context. However, the comparison to my previous self-training label propagation method in Chapter 4.4.2 indicates that even if I am able to recover the best-possible ssGMM hyper-parameters, this would still not outperform the self-training LP result in either performance or computation time (compare the ssGMM (best) and Self-train LP rows in Table 4.1). The number of data for this study is quite small, so the benefit of ssGMM being an inductive algorithm is not properly demonstrated here. However, in problems where the number of data are much greater, the inductive capabilities of ssGMM may prove to be more computationally feasible than transductive approaches, such as label propagation.

The second objective of this study is to determine if simultaneously using the mean and standard deviation CV scores (i.e. the SMSD score) to select hyper-parameters is preferred in comparison to the default approach of only using the mean CV scores. For the ssGMM method, Tables 4.1 and 4.2 indicate that the SMSD score selects hyper-parameters that perform the same or better than the default selections, and the fact that ssGMM only has two hyper-parameters made the visualization of this quite clear (i.e. Figures 4.3-4.6). Using the SMSD score to select hyper-parameters for XGB gives mixed results. It seems that when the standard hyper-parameter selection for XGB performs poorly, then the SMSD score can select hyper-parameters that perform better (Table 4.2 for 5-fold CV). However, if the standard hyper-parameter selection for XGB is already performing well, then using the SMSD score appears to make a poorer selection (Table 4.2 for 5-repeated 5-fold CV). XGB has many hyper-parameters, and so it is difficult to ascertain the cause of this phenomenon without being able to visualize its performance in the way I do for ssGMM.

4.5.2 Interpretation

There are a few methods for interpreting these classification results, and the first is inspecting the performance on a per-well basis rather than globally. Table 4.3 shows the accuracy scores on each of the nine unla-

Table 4.3: The prediction accuracies for the best XGB and ssGMM models decomposed into individual accuracies for each of the unlabeled wells. Table 4.2 denotes that the best models for XGB and ssGMM come from 5-repeated 5-fold CV. The total accuracies provided in the last row correspond to those given in Table 4.2 and are calculated by taking the weighted average of the individual accuracies for each well.

Unlabeled well name	# points	XGB accuracy (%)	ssGMM accuracy (%)
SHANKLE	449	44.77	52.78
CROSS H CATTLE	501	28.94	32.34
NEWBY	463	41.04	44.28
LUKE	461	60.52	64.43
CHURCHMAN BIBLE	404	42.08	40.84
ALEXANDER	466	52.36	53.65
SHIMPLIN	471	51.59	54.99
NOLAN	415	44.34	49.40
RECRUIT F9	68	7.35	0.00
Total	3698	44.92	48.13

beled wells for the best recovered XGB and ssGMM models. The ssGMM model is able to outperform XGB on seven of the nine unlabeled wells by a notable margin. We can also interpret the classification results by physically observing the facies predictions. The performance of XGB and ssGMM on the SHANKLE and NOLAN wells is characteristic of their overall performance, and so I choose to show the facies predictions for these two wells in Figures 4.7 and 4.8. Notice how in both wells, the XGB prediction is visibly chaotic, which is evidence of overfitting. However, the ssGMM predictions are less chaotic and fit the true facies better, which supports the claim in Chapter 1.1 that including unlabeled data in the training process is akin to regularizing underdetermined inverse problems to help prevent overfitting. The maximum probabilities used to classify the unlabeled data for ssGMM (Equation 2.17) are given as the probability logs in Figures 4.7 and 4.8. These probabilities are a benefit to ssGMM and they can be useful for interpretation. For instance, the probability tends to drop at predicted lithofacies interfaces, which supports the well-understood notion that rock unit boundaries are not always discrete.

Another technique for visualizing and interpreting the classification performance is through confusion matrices. Figure 4.9 shows the confusion matrices for the best recovered XGB and ssGMM models. Notice how the predictions cluster closer to the diagonal for ssGMM compared to XGB; this is reflected by the higher accuracy and adjacent accuracy as indicated in Table 4.2. The classification ability of XGB for

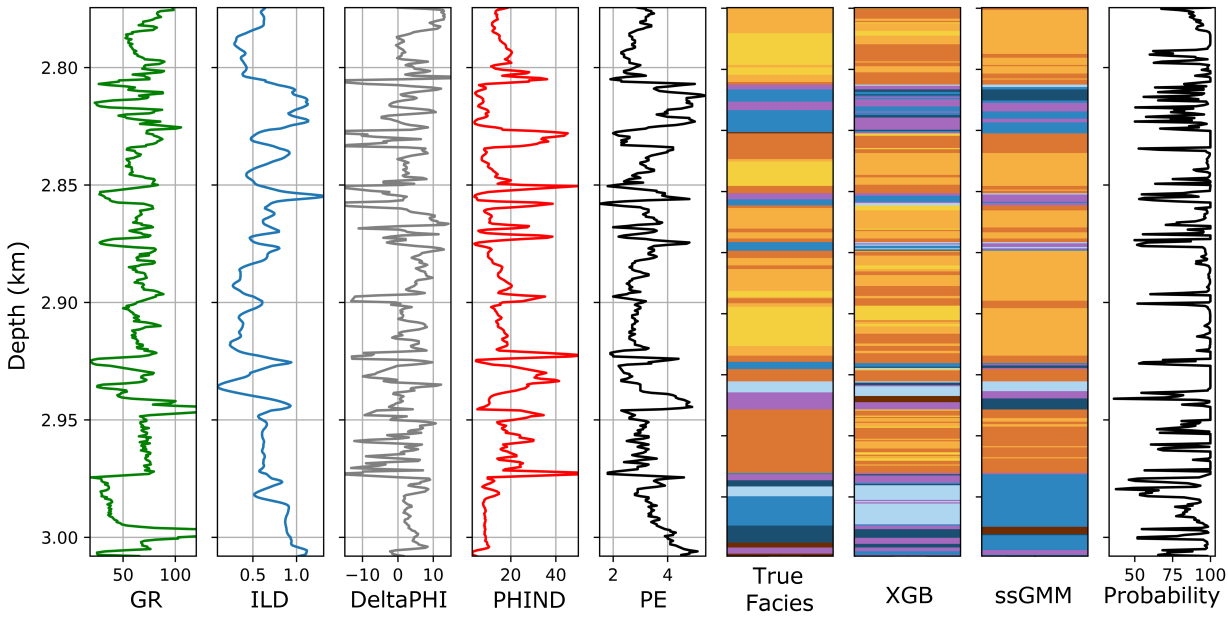

Figure 4.7: A comparison of the XGB and ssGMM facies predictions to the true facies for the unlabeled well, SHANKLE. For the performance of these models on SHANKLE, see Table 4.3. The five log variables are shown for reference. The final column gives the probabilities for the ssGMM predicted facies. See Table 3.1 for the facies colors key.

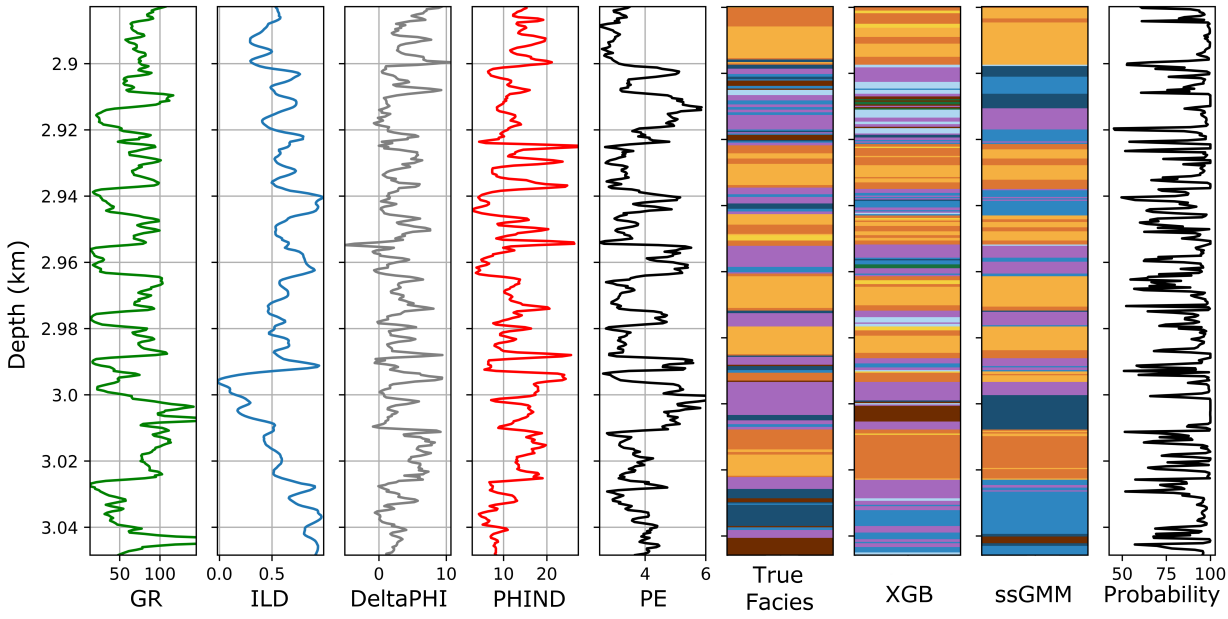

Figure 4.8: A comparison of the XGB and ssGMM facies predictions to the true facies for the unlabeled well, NOLAN. For the performance of these models on NOLAN, see Table 4.3. The five log variables are shown for reference. The final column gives the probabilities for the ssGMM predicted facies. See Table 3.1 for the facies colors key..

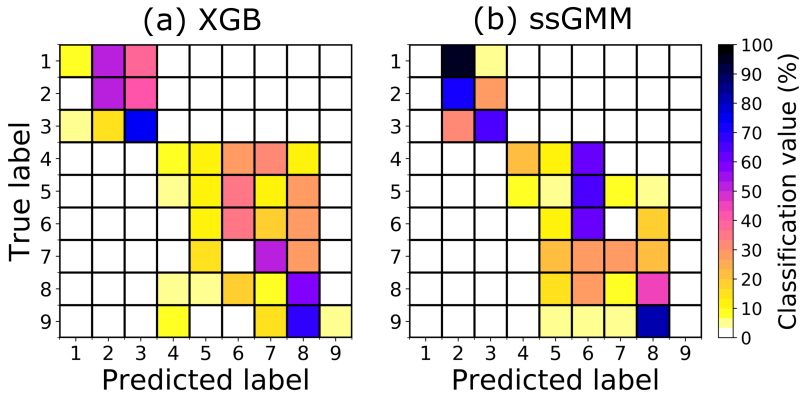

Figure 4.9: The normalized confusion matrices for the (a) default XGB 5-repeated 5-fold CV model and the (b) ssGMM 5-repeated 5-fold CV model selected using the SMSD score (i.e. the best models for XGB and ssGMM indicated by the * in Table 4.2). The predictions shown by these matrices are for the nine unlabeled wells. Diagonal cells represent correct classification, off-diagonal cells represent misclassification.

Classes 1 and 9 is poor ($<$10%), but ssGMM is unable to predict for these two classes at all. Classes 1 and 9 only have nine and seven points, respectively, in the training data (Figure 3.2a) and the initial covariance matrices describing each class, which are 6-dimensional, are poorly constrained and ill-conditioned because of this. In Table 4.3, the RECRUIT F9 well is a synthetic well that only contains Class 9 and this is why ssGMM has an accuracy of 0% on that well. Table 4.3 also shows that the prediction performance for the CROSS H CATTLE well is quite poor, and this is because CROSS H CATTLE has many data points associated with Class 1. While the training data for other classes are sufficient to define their corresponding covariance matrices, more data are certainly needed to constrain the covariance matrices for Classes 1 and 9.

4.5.3 Comparison to machine learning competition

Upon inspecting the performance of all the algorithms in Chapter 4.4, it is apparent that the prediction accuracies on the unlabeled data are quite low ($< 50\%$). My explanation for the absolute accuracies being so low is because the classes are poorly separated (see Figure 3.1), and I come to this conclusion in my previous work (Chapter 3.6.3). In comparison, however, all the winners of the SEG machine learning

competition were able to achieve F1-scores of 62-64% on the two withheld wells using XGB (Hall & Hall, 2017), whereas my implementation of XGB only roughly achieves an F1-score of 42% on the nine unlabeled wells. This discrepancy can be attributed to three factors: the amount of training data, filtered outputs, and feature expansion. For my semisupervised scenario, I am only training with roughly one-tenth of the data that was used for the competition. The competition winners also apply a median filter to their XGB predictions (Bestagini et al., 2017); I do not do this because I feel that an algorithm's prediction is best represented by its raw output. Lastly, all the winners use feature expansion, a process of artificially generating more features to help improve class separability, to improve their results. For the competition problem, feature expansion was quite effective and improved F1-scores by 5-6% (Bestagini et al., 2017). However, feature expansion does not work for this semisupervised scenario because some classes are poorly constrained (e.g., seven and nine points, respectively, for Classes 1 and 9) and representing these classes in higher dimensions makes them even less constrained; this is a consequence of the curse of dimensionality. I do not show these results for brevity, but the performance for both ssGMM and XGB diminishes when I use similar feature expansion techniques as those from the competition. Together, these three factors are responsible for the disparity between my results and those from the competition.

4.6 Conclusion

The two objectives of this paper are to investigate (1) if semisupervised Gaussian mixture models (ssGMM) can outperform a widely-used supervised algorithm, XGBoost (XGB), in the context of a well-log classification example with limited training data, and (2) if my new hyper-parameter selection strategy that simultaneously uses the mean and standard deviation (SMSD) cross-validation scores can make better selections than the default approach. The results first show that one of the well-log data features violates the Gaussian assumption, which causes ssGMM to not converge. The remedy for this situation is decomposing the dataset into two pieces to remove this feature. This demonstrates how important it is to perform

tests prior to classification to ensure that the data are not violating any underlying assumptions. Once this procedure is performed, the results demonstrate that the ssGMM method is able to outperform XGB, but not by a significant margin. However, a benefit of ssGMM is that it is a simple semisupervised algorithm (i.e. only two hyper-parameters) that can still achieve a better performance than a complex supervised algorithm, XGB. Using my new proposed SMSD score for hyper-parameter selection gives promising results for ssGMM, but mixed results for XGB. However, I only compare the performance of the default and SMSD hyper-parameter selection strategies using one dataset and two algorithms, and so future tests using different algorithms and/or datasets would help determine the true efficacy of the SMSD hyper-parameter selection strategy. Nonetheless, a visualization of the lithofacies predictions supports the well-known claim that supervised methods are prone to overfitting when the training data are minimal, but including the unlabeled data into the training process (i.e. semisupervised learning) mitigates this phenomenon.

Chapter 5

A seismic petrophysical classification study of the 2-D SEAM model using semisupervised techniques and detrended attributes[1]

5.1 Introduction

In recent years, many disciplines have been challenged with trying to efficiently extract meaning, or value, out of large datasets. Manually identifying patterns from data has become increasingly difficult due to improvements in data storage and data acquisition capabilities exponentially growing our data volumes. This is a challenge for many disciplines, and is certainly the case for hydrocarbon exploration in the domain of geophysics. Relating seismic data to geologic targets has always been a pattern recognition task. In

[1]Dunham, M.W., Malcolm, A. and Welford, J.K., 2021. A seismic petrophysical classification study of the 2-D SEAM model using semisupervised techniques and detrended attributes, Geophysical Journal International, 227(2), 1123–1142.

the early days of seismic attributes, it was recognized that instantaneous attributes exhibited anomalous behavior in certain hydrocarbon-bearing geologic units (Taner & Sheriff, 1977; Taner et al., 1979). Relating these certain properties, or attributes, of the seismic data to facies is a form of pattern recognition, but until more recently, these connections were performed manually by geoscientists. Technological advancements have provided us with tools such as machine learning to extract meaningful information from large datasets efficiently. The purpose of seismic pattern recognition using machine learning, otherwise known as seismic classification, is to map seismic data to desired classes (e.g., seismic facies, lithofacies, or petrofacies), and this has been achieved using many approaches in the literature.

One approach is unsupervised learning that essentially maps the seismic data into unlabeled clusters, and the user must assign meaning, or class labels, to these clusters. Some of the earliest unsupervised learning applications to seismic classification use self-organizing maps (SOMs, Berge et al., 2002; West et al., 2002; Strecker & Uden, 2002; Coléou et al., 2003; de Matos et al., 2007; Roy et al., 2013; Roden et al., 2015; Zhao et al., 2015). Bayesian-based techniques have also been popular, such as generative topographic mapping (GTM, Wallet et al., 2009; Roy et al., 2014; Zhao et al., 2015; Wang & Wu, 2017) and Gaussian mixture models (GMMs, Hardisty & Wallet, 2017; Wallet & Hardisty, 2019). In recent years, GMMs have been improved to include transition probabilities from hidden Markov models (Feng et al., 2018a,b).

Another approach is supervised learning, which aims to learn a direct mapping from the seismic data to the known classes. Various techniques have been employed in the context of seismic classification, such as neural networks (Saggaf et al., 2003; Raeesi et al., 2012; Aleardi & Ciabarri, 2017; Ross & Cole, 2017), and support vector machines (Li & Castagna, 2004; Bagheri & Riahi, 2015; Zhao et al., 2015). However, the trend in recent years is deep learning, particularly convolutional neural networks (CNNs). CNNs treat the seismic data as an image, and the trained CNNs extract the necessary features/textures from the seismic images through filters. The use of CNNs has led to the successful classification of faults (Araya-Polo et al., 2017; Huang et al., 2017; Xiong et al., 2018; Cunha et al., 2020), salt bodies (Waldeland et al., 2018; Shi

et al., 2019), and the delineation of seismic facies (Zhang et al., 2018b; Zhao, 2018; Liu et al., 2019; Souza et al., 2019). Another form of deep learning is a recurrent neural network (RNN), which is designed to capture the long-term temporal dependencies of sequential data. RNNs have been widely successful for processing audio signals (e.g., speech recognition & translation), and some have extended the application of RNNs to seismic classification (Grana et al., 2020).

The particular problem of interest for this work is seismic *petrophysical* classification, where the data are seismic attributes and the labels for the training dataset are derived from well data. Wells are characteristically sparse, so these problems are inherently challenged with low amounts of training data. In these situations, supervised methods, especially deep learning methods, are prone to overfitting (Goodfellow et al., 2016). These overfitting complications can be alleviated by regularization, which has many forms. Several supervised machine learning algorithms include a hyper-parameter that directly or indirectly imposes a penalty on the model complexity, e.g., the C hyper-parameter for support vector machines (Cortes & Vapnik, 1995) or dropout regularization for deep learning (Goodfellow et al., 2016). However, if the amount of training data is drastically limited, this form of regularization still may not be sufficient. One way to address this problem for seismic petrophysical classification is to use geostatistical techniques to simulate additional 1D Earth models from known wells to enlarge the training dataset (Das et al., 2019). Another method to overcome the paucity of labeled data is to generate additional data by performing Monte Carlo simulation from a petro-elastic model (Choi et al., 2017; Lee et al., 2018). While these geostatistical augmentation techniques certainly have merit, they require domain knowledge and their utilization may be limited to experts.

An alternative form of data augmentation is to incorporate the readily available unlabeled data into the learning process. This is semisupervised learning (SSL), where the labeled and unlabeled data are both utilized to build a machine learning model (see Chapelle et al., 2006; Zhu & Goldberg, 2009; van Engelen & Hoos, 2020). SSL methods have been relatively unexplored in exploration geophysics applications, and

it has not been until recently that they have begun to gain traction. Some seismic-related examples include combining RNNs and CNNs for a semisupervised approach to seismic impedance inversion (Alfarraj & AlRegib, 2019) and using a transductive regression technique to estimate porosity from impedance volumes (Lima et al., 2017; Görnitz et al., 2018). With specific regard to seismic classification problems, there are a few noteworthy applications. Some researchers have manipulated the traditional unsupervised GTM into a semisupervised probability classifier using the Bhattacharyya distance (Roy et al., 2014; Qi et al., 2016). Another application performs a petrophysical classification of seismic data by incrementally labeling the unlabeled data in a spatial manner using the most confident predictions from a deep neural network (Asghar et al., 2020). Other researchers have extended a generative adversarial network (GAN), another type of deep learning, to operate in a semisupervised context for seismic facies classification (Li et al., 2019a; Liu et al., 2020).

Two semisupervised methods, label propagation and self-training, improve the classification results of well-log data in a previous study (Chapter 3). In this paper, I extend these same semisupervised techniques to the petrophysical classification of seismic data. My self-training approach is similar in concept to the pseudo-labeling strategy from Asghar et al. (2020), but when coupled with label propagation, my approach incorporates the unlabeled data in training where their supervised deep neural network does not. While a semisupervised GAN approach looks promising (Li et al., 2019a; Liu et al., 2020), these deep learning methods have a significant learning curve, can be difficult to conceptualize for non-experts, and they contain many hyper-parameters that require tuning. my goal is to provide a simple solution for classification problems challenged with insufficient training data; the semisupervised techniques that I consider are designed for this task by being well established, easy to implement with very few hyper-parameters, and conceptually straightforward.

To investigate if these semisupervised techniques can achieve better performance than supervised methods in a low training data context, I formulate a seismic petrophysical classification scenario using a subset

of the Phase 1 2D SEAM model (Fehler & Keliher, 2011). The advantages of this study being a synthetic classification problem are that I have full control and can make quantitative comparisons. To perform a petrophysical classification of this SEAM model, I consider rock properties-based attributes as the inputs (e.g., acoustic impedance, shear impedance, density, etc.). While there are many other types of attributes that I could also consider (e.g., instantaneous, texture, etc.), it is established that rock properties-based attributes are well suited for discriminating lithology and fluid variations of rocks (Avseth et al., 2005), and many studies have used these types of attributes for seismic lithological/petrophysical classification (Mukerji et al., 2001; Roy et al., 2014; Aleardi & Ciabarri, 2017; Grana et al., 2017; Kim et al., 2018; Asghar et al., 2020). What is unique, and perhaps unfortunate, about these attributes is that they are depth-dependent (due to compaction, cementation, etc.). If one focuses on a narrow interval, such as in a production setting, these depth variations may be negligible. However, if the focus is on a much larger interval, e.g., in an exploration context, these depth dependencies can become problematic. Keynejad et al. (2019) try to address this problem in a well-log classification scenario by including depth as an attribute, but I have found this to not be as effective in this seismic classification context. What I explore in this paper is directly removing the depth trend from the attributes via filtering. Filtering seismic attributes to improve seismic class definition is not a new concept, e.g., Qi et al. (2016) use Kuwahara filters on seismic texture attributes to remove noise and preserve boundaries. However, to my knowledge I am the first to demonstrate the benefit, and necessity, of filtering rock properties-related attributes for seismic classification.

The contents of this paper are as follows. First, I introduce label propagation, self-training, and the supervised method that I use for comparison. Then, I formulate a seismic petrophysical classification situation from the 2D SEAM model. This process requires synthesizing seismic data from the model using an elastic solver, performing prestack time migration, and conducting simultaneous prestack seismic inversion to recover the attributes necessary for this machine learning problem. I simulate a low-training data scenario for the classification task by assuming that there is only one well location. In an attempt to improve perfor-

mance, I consider feature expansion, including the two-way traveltime as an input attribute, and de-trending the rock-properties related attributes. I also consider a different location for the training well to investigate the robustness of the machine learning methods to changes in the training data. The results indicate that de-trending the seismic attributes is necessary for all algorithms (supervised and semisupervised) to perform well; when the attributes are de-trended, the semisupervised techniques still perform better than the supervised baseline.

5.2 Methods

5.2.1 Machine learning methods

As mentioned above, the semisupervised methods used for this study are label propagation and self-training. While I assess the results of label propagation independently here, the same self-training label propagation approach from Chapter 3 is also considered. Recall that LP has two choices for the kernel of the edge-weight matrix, k-NN or RBF. The approach in Chapter 3 uses the RBF kernel (a dense matrix), which is not problematic given that the well-log dataset in that problem is relatively small. However, the size of the dataset for this problem is over $100\times$ larger, and using the RBF approach for LP would be computationally intractable (e.g., requiring $>$ 1000 GB of RAM). As such, the k-NN kernel is used to compute the edge-weight matrix for LP.

The self-training method is the same technique presented in Chapter 2.2.2 with selection criteria to better accommodate class imbalance. Although, it is worth re-iterating that self-training has the potential to be rather unstable. If the base algorithm performs poorly (LP in this context), then errors could be perpetuated with each self-training iteration; an example of such a situation is provided in Chapter 5.5.1. Nonetheless, self-training is a simple technique to help improve the performance of classification problems when training data are limited, and this approach is easy to implement with only two hyper-parameters (S, T).

To assess if LP and self-training LP can perform better than supervised methods in a limited training data scenario for seismic petrofacies classification, I must compare against a supervised method. I have chosen the extreme gradient boosting (XGBoost) algorithm (Chapter 2.1.3). XGBoost was the winning algorithm for the 2016 SEG machine learning competition, and it even outperformed deep learning methods (Hall & Hall, 2017). This competition was designed so that the entire dataset (≈ 4000 data points) could be used for training the participants' algorithms, but even so, this relatively small dataset made it especially difficult to train deep learning methods according to the organizer (Hall & Hall, 2017). Given that the amount of training data used in this study is comparatively far less (≈ 1000 data points), this convinced me that XGBoost might be more appropriate in this circumstance compared to deep learning methods. Therefore, XGBoost seems like a natural choice to challenge the performance of the semisupervised techniques. Unlike LP and self-training, XGBoost does have many hyper-parameters that require setting. Most of the hyper-parameters are concerned with aspects of the decision trees, however, I have found that leaving many of the hyper-parameters assigned to their defaults is sufficient. The hyper-parameters that I do choose to optimize are the learning rate, max depth, min child weight, number of estimators (trees), and lambda (L2 regularization term). The ranges of values considered for each of these hyper-parameters is discussed in Chapter 5.3.6 below.

5.2.2 De-trending attributes

As stated in the Chapter 5.1, I use rock properties-based seismic attributes as the features for my machine learning problem (I discuss this in more detail in Chapter 5.3.4 below). However, a challenge with these attributes is that they are depth-dependent due to compaction, cementation, etc. This depth dependency can have a detrimental impact on the classification performance for many algorithms because the rock properties for a given lithology are not unique. I demonstrate this with an example in Figure 5.1(a). Here, I denote a shale unit at 3.2 km depth, and at a slightly deeper depth (4.5 km), we have sand units with the same

measured density value (2.20 g/cc). If these density data are used to train a machine learning algorithm, this results in two different classes with the same input value (i.e., the two classes are overlapping in the input, or density, space). With these data, it is unclear whether the machine learning algorithm should predict sand or shale for a new data point with a density of 2.20 g/cc. Consequently, this phenomenon introduces ambiguity into the machine learning process that can misclassify unseen data.

I address this problem by trying to remove the depth trends from these types of seismic attributes. The concept that I use here is analogous to regional-residual separation of potential field data where a filter, or function, is used to determine the background field, which is then subtracted from the measured signal to recover a residual signal (Gupta & Ramani, 1980). Here, I use a simple median filter to extract the background trend from the seismic attributes (e.g., Figure 5.1b); in this example case, the filter is 1D because the trace is 1D. Then, similar to regional-residual separation, I subtract this background trend from the measured attribute to obtain the de-trended attribute. After de-trending the density trace, Figure 5.1(c) shows that the shale and sand units discussed above no longer have the same density value; the shale has a value of 0.0 g/cc, and the sand has a value of -0.1 g/cc. Transforming the rock properties-based seismic attributes in this way improves the separation of the classes in the input space, which can help improve classification performance.

5.3 Machine learning preparation

5.3.1 2D SEAM model

The model that forms the basis for this study is the Phase 1 2D SEAM model made publicly available by the Society of Exploration Geophysicists (Fehler & Keliher, 2011). This model intends to address subsalt imaging challenges in Tertiary basins, emphasizing deepwater environments in the Gulf of Mexico (see Figure 5.2a). I want to use this model to provide the basis for a seismic petrophysical classification problem,

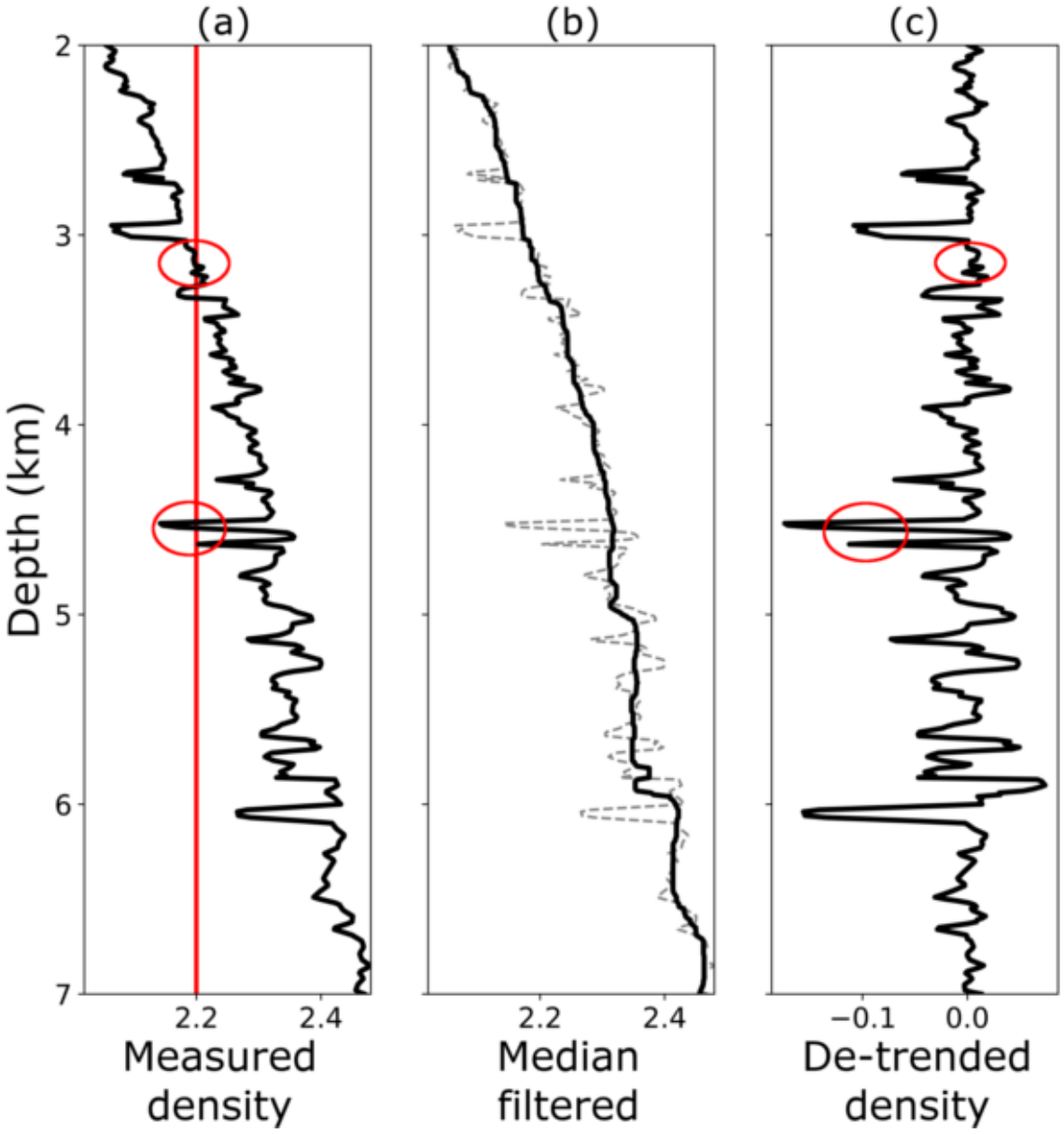

Figure 5.1: The de-trending process illustrated with an example density trace. (a) The measured density. A red line is marked at 2.2 g/cc to depict its intersection with two units, a shale and a sand interval, denoted by red circles. (b) A median filter is applied to the measured density to recover the background trend (bold). The measured density is also shown for reference (dashed). (c) The de-trended density. The same shale and sand units circled in (a) are circled again here to indicate that they have different de-trended density values. All densities shown are in g/cc.

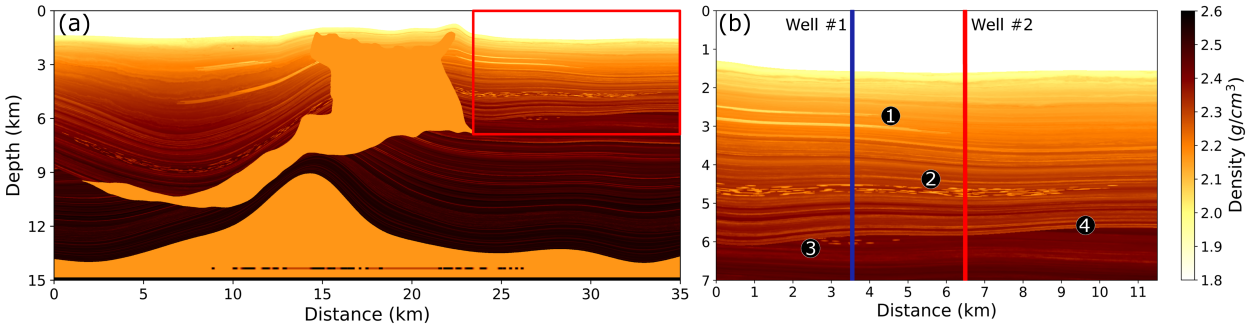

Figure 5.2: (a) The density attribute of the publicly available Phase 1 2D SEAM model. (b) The 2D SEAM model subset that I consider for this study, indicated by the red box in panel (a). The numbered annotations highlight regions of interest in this model: (1) leaf turbidites, (2, 3) channel valley fills, and (4) an unconformity. The two well locations used for training (individually) are also denoted in panel (b).

but subsalt imaging is not the focus of this study and the inclusion of salt may overcomplicate the problem. Therefore, I consider an 11.5×7.0 km subset of the entire 2D SEAM model that only contains sedimentary sequences (Figure 5.2b). Some features of interest in this model are leaf turbidites from 2.5-3.0 km depth, channel valley fills at 4.5 and 6.0 km depth, and an unconformity at 5.5 km depth (see annotations on Figure 5.2b). To explore the challenges of limited training data, I assume that there is only one well location (this well will serve as the training data once it is assigned labels, see below). However, I do consider two different locations for this well to investigate the robustness of the machine learning methods to changes in the training data. The first well is located at X = 3.5 km and intersects all three reservoir zones (Well #1 in Figure 5.2b), and the second well is located at X = 6.5 km and only intersects one of the reservoir zones (Well #2 in Figure 5.2b).

5.3.2 Clustering

A benefit of this model is that many properties are provided that I can leverage, such as elastic properties (Vp, Vs, and density), shale volume (Vshale), porosity, and resistivity. However, this dataset does not provide any class labels (e.g., sand, shale, etc.), and I must determine these from the data themselves. In these situations where labels are unknown, a common procedure is to use a data-driven, unsupervised learning approach

to obtain clusters, and then assign meaning to the clusters to define classes. I adopt this approach here by applying it to the model properties (depth, Vp, Vs, density, porosity, Vshale, and resistivity) at the Well #1 location (see Figure 5.2b). I only apply the clustering to the data at Well #1 because in a realistic setting, only the well data would contain these properties (e.g., Vshale, porosity, resistivity, etc.). The method I choose to perform the unsupervised analysis is the *SpectralClustering* class from `scikit-learn`. This clustering method is built using a normalized graph Laplacian, the same as label propagation. At its core, spectral clustering applies k-means clustering to the eigenvectors of the Laplacian matrix which can improve clustering when the structure of the individual clusters is complex. See Ng et al. (2002) and von Luxburg (2007) for more details.

A challenge with any unsupervised method is determining the optimal number of clusters to use. I use two factors to assess each cluster setting. Most importantly, I qualitatively assess the clusters for their geologic feasibility (examples below). I also use the silhouette score (Rousseeuw, 1987), which is a metric for measuring the tightness of a cluster and its separation from others. Once clusters are determined, a silhouette score can be computed for each data point where the values range from +1 to -1; data points associated with a tight and well-separated cluster achieve the best value of +1, values near 0 indicate overlapping clusters, and negative values suggest the data point may be assigned to the wrong cluster. The silhouette scores can then be averaged for all data points to quantify how well the clustering is performing for the whole dataset.

Figure 5.3 shows my analysis for determining the optimal number of clusters. I consider the number of clusters (n_clusters) to range from 2 to 6 and I perform a silhouette analysis on each setting (five total). The left-hand panels show the silhouette values for each data point color coded to each cluster and the dashed red line indicates the average silhouette score for all the data. The right-hand panels show the clusters themselves in Vshale-resistivity space. Recall that I am ultimately trying to perform petrophysical classifications with these data, so the clusters need to distinguish fluid saturations in sand units from shale. For n_clusters = 2, we observe one large cluster that spans all Vshale values for low resistivity values. I deem this setting

geologically infeasible because this cluster is essentially combining brine sands and shales into one cluster. The same argument can be made for the n_clusters = 3 setting because the only change is separating what appear to be the hydrocarbon-bearing units into two clusters. At n_clusters = 4, we see the large cluster defined in n_clusters = 2 and 3 being split into two clusters. This setting has the poorest average silhouette coefficient, and I therefore discard it. The n_clusters = 5 setting is one that looks promising. There are two different clusters representing shales at different depths (red and green), one cluster representing brine sands (orange), and two clusters representing hydrocarbon-bearing sands. The n_clusters = 6 setting is nearly identical to the n_clusters = 5 with the difference of breaking the brine sand cluster into two. Ultimately, I determine n_clusters = 5 to be the optimal setting because its clusters align with geologic understanding, and it has a higher silhouette coefficient compared to using 6 clusters.

5.3.3 Determining ground truth model

I further visualize the n_clusters = 5 result in Figure 5.4 which shows a cross-plot and a well-view of the clustering results. From the well view (Figure 5.4b), it becomes clear that there is a depth dependency with the clusters, with the exception of Cluster 3. The reason for this may be because many of the input properties are indeed themselves depth-dependent, e.g., Vp, Vs, density, and resistivity all increase (and porosity decreases) with depth due to compaction and other processes. The depth itself is also included as a feature, but this phenomenon still occurs even if depth is removed as an input feature. However, it does not seem reasonable to me to define multiple clusters for a given geologic facies if the only difference is that one is slightly more compacted than the other. So,Ie decide to combine clusters. It is clear that Clusters 1 and 5 are both representing shale, and so I combine these clusters. Clusters 3 and 4 appear to distinguish the hydrocarbon-bearing sand at 6 km depth (Feature 3 in Figure 5.2b) from the shallower hydrocarbon-bearing units (Features 1 and 2 in Figure 5.2b), but they are all hydrocarbon units and I decide to combine these two clusters as well. The result is three clusters, with one cluster each representing shale, brine sand, and

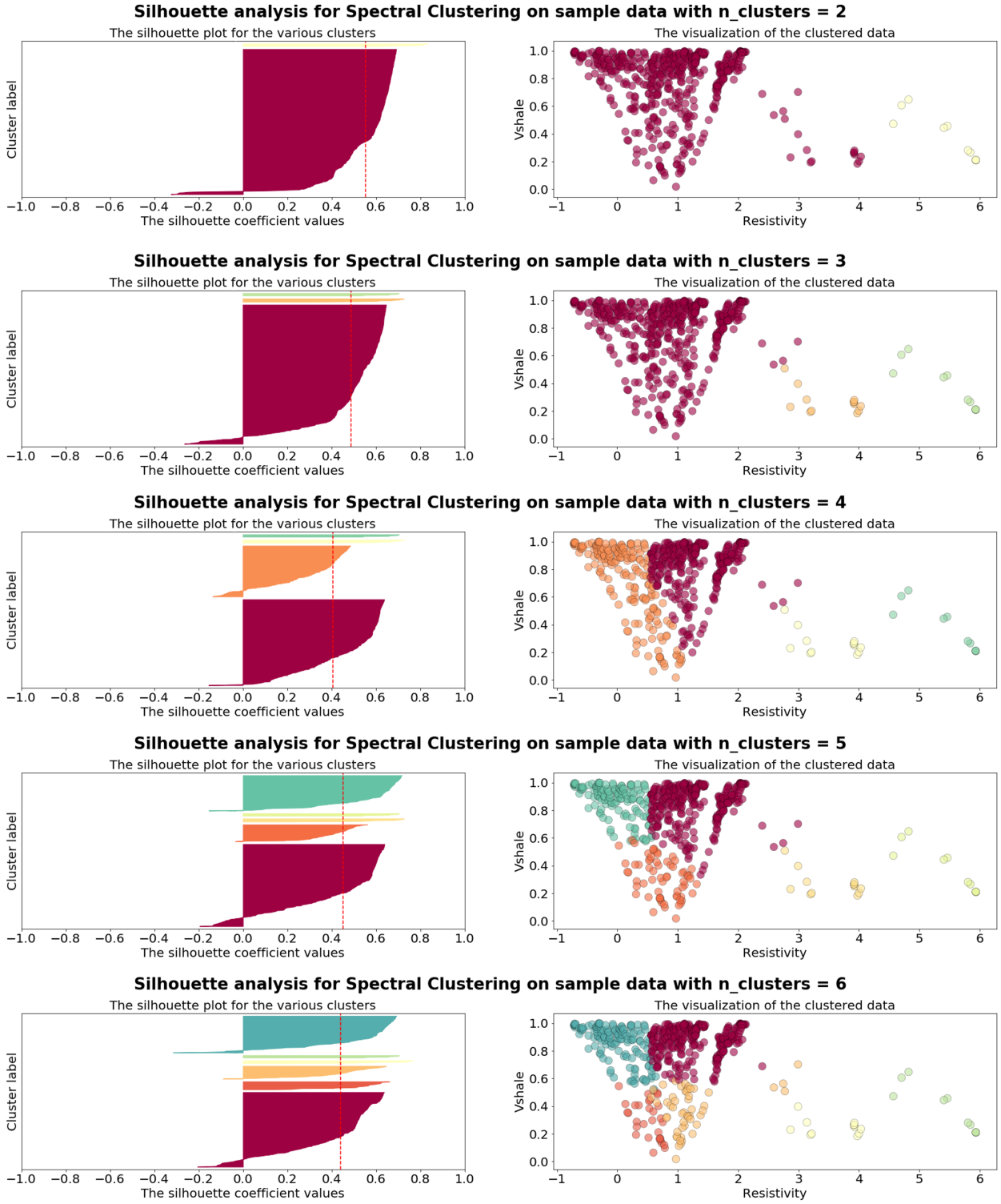

Figure 5.3: Silhouette analysis of various cluster settings, where the Well #1 physical property data are used as inputs. The average silhouette scores for n_clusters = 2, 3, 4, 5, 6 are 0.553, 0.489, 0.406, 0.452, 0.439, respectively (indicated by the dashed red lines on the left-hand panels).

hydrocarbon sand units (see Figure 5.5). Interestingly, nearly the same cluster definition can be achieved by applying the following thresholds to the Vshale and resistivity properties:

- Shale: Vshale $>= 0.55$,
- Wet sand: Vshale < 0.55, Resistivity $< 100\ \Omega m$,
- Hydrocarbon sand: Vshale < 0.55, Resistivity $>= 100\ \Omega m$.

Figure 5.6 shows the distribution of these three classes determined via thresholds, and notice the stark similarities with the clusters in Figure 5.5. It is a common industry practice to apply thresholds to Vshale and resistivity data to distinguish fluid saturations in sand from shales, and what is interesting here is that thresholding these data roughly aligns with the natural cluster structure. Moving forward, I decide to use the class definitions defined by these thresholds above because this approach helps correct some of the inconsistencies in the clusters (e.g., shales being defined down to Vshale = 0.30, see Cluster 1 in Figure 5.5) and it is much quicker to apply these thresholds to the entire model compared to a clustering algorithm.

The result of this analysis is a three-class petrophysical model, as shown in Figure 5.7(a). The resistivity property clearly illuminates the presence of the three zones containing hydrocarbon-bearing sands (Figure 5.7b), and the Vshale property is effective at distinguishing sand from shale (Figure 5.7c). This facies model is significant to this study for two reasons. First, I can sample this model at the well locations shown in Figure 5.7(a) to provide the labels for the training data. Second, since the class labels for the entire model are also known, this allows me to also evaluate the performance of the machine learning predictions on the unlabeled, or testing data. The next stage concerns generating the seismic attributes (i.e., the features) for the machine learning problem.

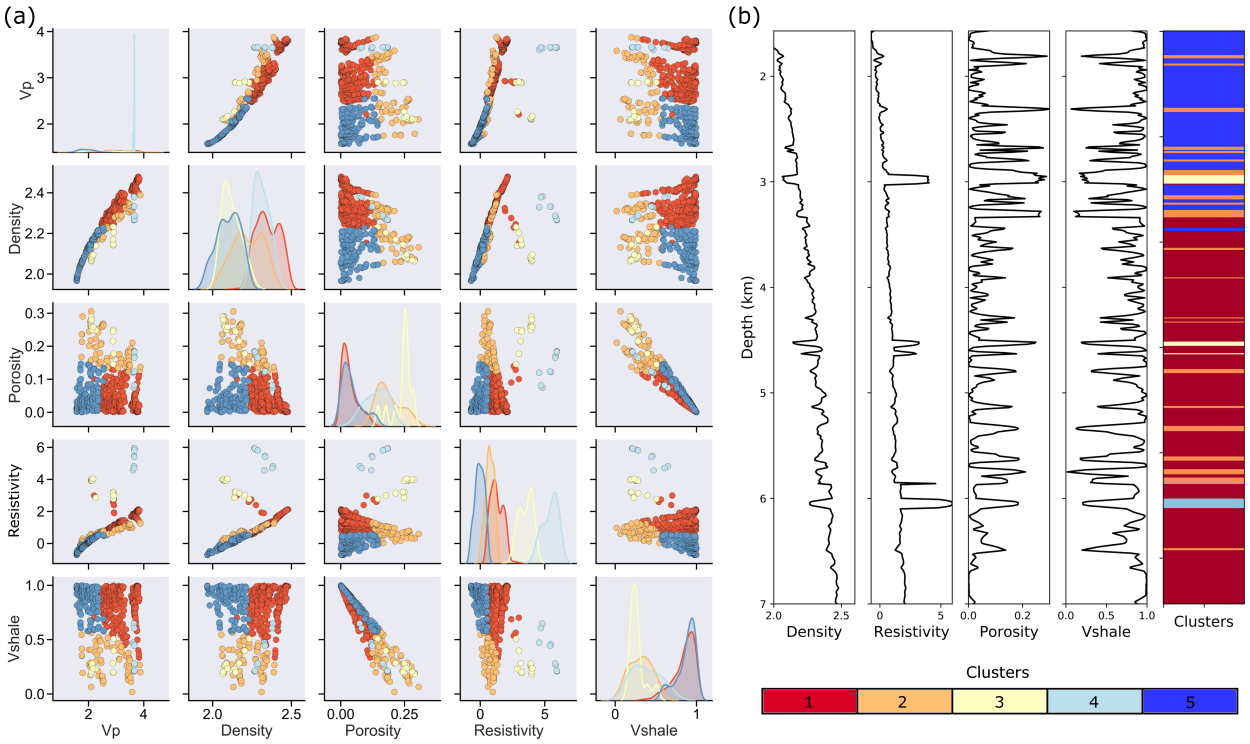

Figure 5.4: Clustering analysis using the depth, Vp, Vs, density, porosity, resistivity, and Vshale data at Well #1 as inputs. The analysis from Figure 5.3 suggests that five clusters are optimal for these data. (a) A cross-plot of the clusters using five of the inputs (depth and Vs not shown). (b) A well-view of the clusters with four logs shown for reference. I combine clusters 5 and 1, and 3 and 4 to define three units: shale, wet sand, and hydrocarbon sand. The units for Vp and resistivity are km/s and $\log_e \Omega$m, respectively.

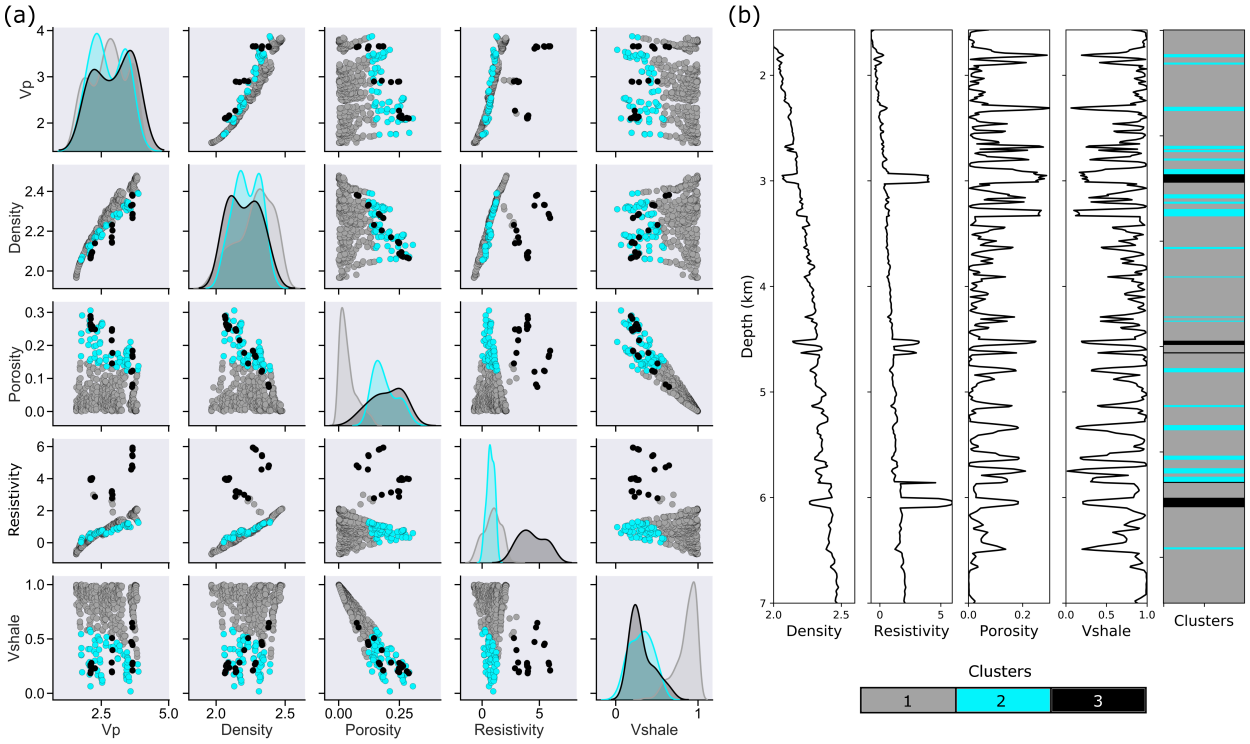

Figure 5.5: The same clustering results shown in Figure 5.4, except I combine clusters 5 and 1, and 3 and 4 to define three units: shale, wet sand, and hydrocarbon sand. (a) A cross-plot of the clusters using five of the inputs (depth and Vs not shown). (b) A well-view of the clusters with four logs shown for reference. The units for Vp and resistivity are km/s and $\log_e \Omega$m, respectively.

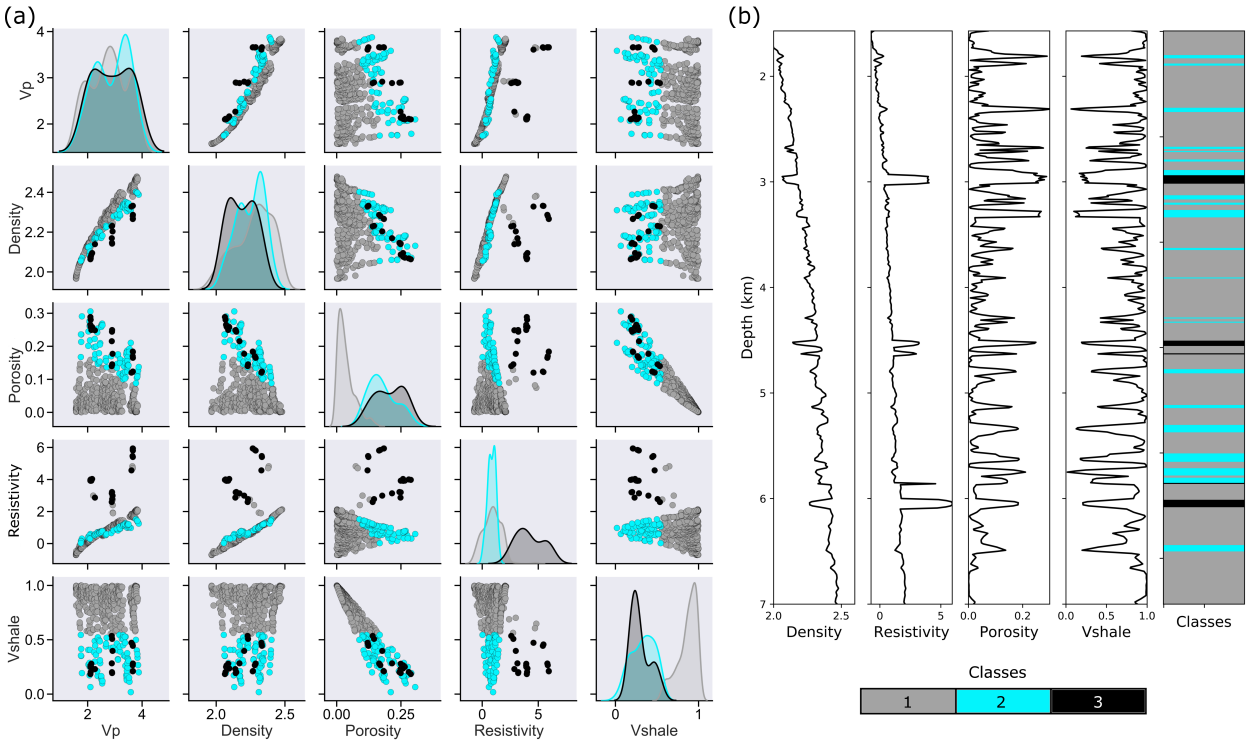

Figure 5.6: Defining the shale, brine sand, and hydrocarbon-bearing sand classes using thresholds on the Vshale and resistivity properties. (a) A cross-plot of the classes using five of the inputs (depth and Vs not shown). (b) A well-view of the classes with four logs shown for reference. These three classes determined via thresholds align strongly with the three clusters shown in Figure 5.5. The units for Vp and resistivity are km/s and $\log_e \Omega$m, respectively.

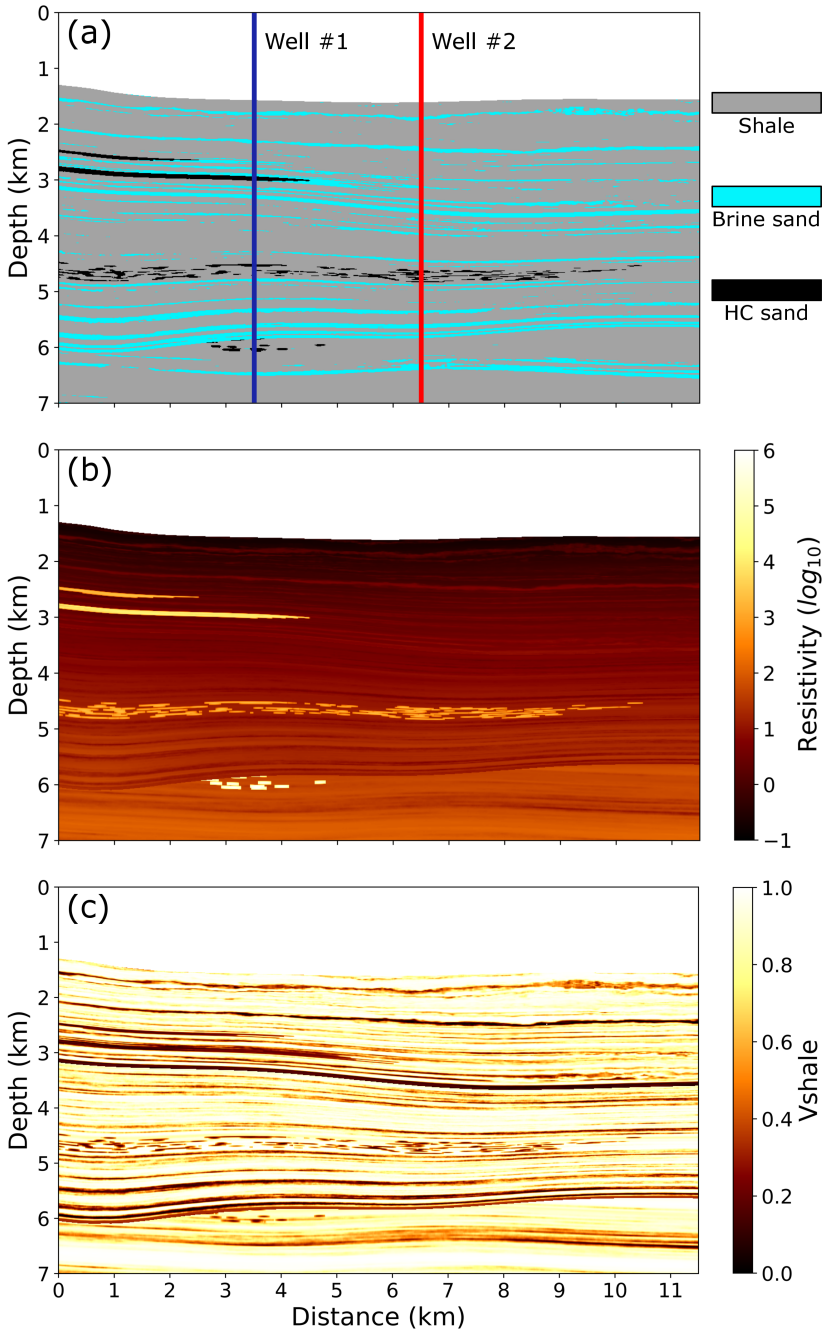

Figure 5.7: (a) The three-class facies model determined using thresholds on the (b) resistivity and (c) Vshale properties. These thresholds honor the natural cluster structure of the data. The two well locations used for training (individually) are also indicated.

5.3.4 Seismic attribute generation

To perform seismic classification on this model, I need to synthesize seismic data before computing the seismic attributes. I use an elastic solver from an open-source software called Devito (Louboutin et al., 2019) to generate the shot records. In an effort to reflect realistic source-receiver geometries, I establish 20 m spacing for the receivers and 100 m spacing for the shots. The peak frequency of the Ricker source wavelet is also set to 17.5 Hz, which is a practical choice given the size of the model. The next stage is performing prestack time migration on the shot records, which is accomplished using GLOBE*Claritas*. Upon analyzing the migrated angle gathers, I determine the critical angle to be roughly 40-45°, so I mute the data at 40°. I then stack the data into six angle stacks to provide a good balance of signal to noise (0-6°, 6-12°, 12-18°, 18-24°, 24-30°, 30-36°). Figure 5.8(a) shows one of the angle stacks (6-12°), and many of the features are recovered quite well. However, the channel valley fills at 4.8 and 5.8 s must be sub-seismic resolution because the reflections are not well defined.

The seismic attributes that I consider for this study are recovered through simultaneous inversion of the angle stack data. Specifically, I invert for acoustic impedance (AI), shear impedance (SI), and density (Smith & Gidlow, 1987; Fatti et al., 1994) using the inversion tools in RokDoc. Figure 5.8(b) shows the inverted density (Rho) attribute on the same color scale as Figure 5.2(b), and the loss of resolution inherent in these steps is apparent.

5.3.5 Problem setup

At this stage, all the necessary information is in place to set up the machine learning problem for a seismic petrophysical classification of the SEAM model subset. I choose to perform the classification in the time-domain (the original domain of the seismic attributes), however, the labels for the training data $(y_1, ..., y_l)$ are still in the depth domain. I convert the labels to two-way traveltime (TWT) using the known P-wave velocities at the well locations. The alternative approach is to perform the classification in the depth domain,

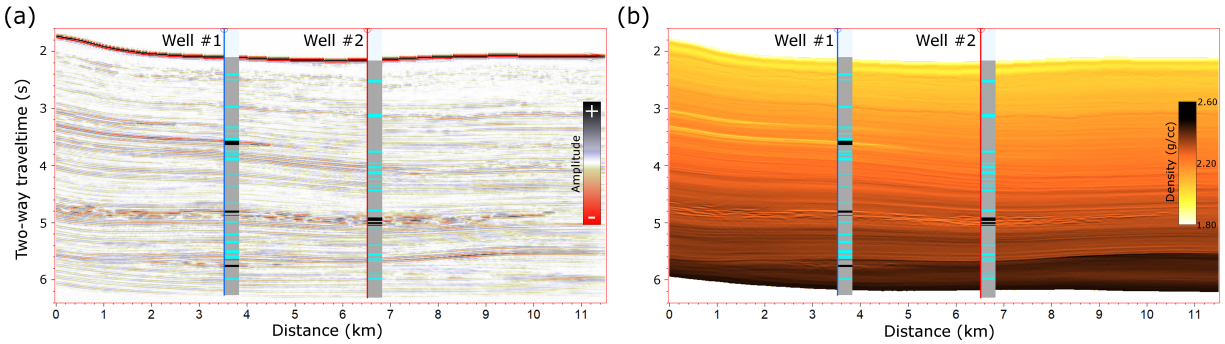

Figure 5.8: (a) Near angle stack (6-12°) generated from the model depicted in Figure 5.2(b). (b) Density attribute recovered through prestack seismic inversion. The labels for the two training wells are superimposed on both panels (gray = shale, blue = wet sand, black = hydrocarbon sand).

but realistically doing so requires a velocity model derived from the seismic data. There is generally more confidence in the TWT-depth relationship at well locations due to direct velocity measurements (i.e., sonic logs) compared to the TWT-depth relationship derived from seismic-based velocity models. As a result, I feel that the more practical approach with less uncertainty is to perform the classification in TWT and only convert the labels, or well data.

To form the training dataset, I must link the seismic attribute samples (x_i) to their corresponding labels (y_i). Linking the seismic data to well samples generally requires downscaling the well data to a similar frequency content as the seismic data (usually via a Backus average), but that is not needed in this situation. In this synthetic problem, I have the rare case where the well data sampling is much coarser than in a realistic setting, thereby inherently having lower frequency content (the depth sampling for the model is 10 m, where standard log sampling is ≈ 0.15 m). When the well data are converted to TWT, the sampling ranges from 5-12 ms (the sampling is non-uniform because of the depth-TWT conversion). The seismic inversion attributes have a default uniform sampling of 2 ms, but I determined that I could decimate the sampling to 4 ms without sacrificing any resolution. Forming the training dataset is relatively trivial here where roughly 1-3 attribute points map to one class point. This results in 1013 and 1002 labeled data points for Well #1 and #2, respectively, for a dataset of 584,644 total unlabeled data points ($< 0.2\%$ training data for either case).

5.3.6 Hyper-parameter estimation

Hyper-parameter estimation is the first stage of training machine learning algorithms, and for XGBoost, I use a standard 5-fold cross-validation (CV, Figure 2.10) on the training data with a macro-F1 scoring metric (the reason for choosing this metric is discussed below in Chapter 5.3.7). The ranges of values used for the various XGBoost hyper-parameters are provided in the following search grid:

- Learning rate: [0.10, 0.15, 0.20, 0.25, 0.30], *default = 0.30*
- Max depth: [2, 3, 4, 5, 6] *default = 6*
- Min. child weight: [1, 3, 5, 7, 9] *default = 1*
- Num. of estimators: [50, 75, 100, 125, 150] *default = 100*
- Lamda (reg): [1, 5, 10, 50, 100] *default = 1*

The ranges of these hyper-parameters are chosen to help make the XGBoost algorithm more conservative, i.e., to help prevent overfitting.

Determining hyper-parameter values for semisupervised techniques is more challenging, as discussed in Chapter 2.3. Setting the hyper-parameters for LP proves to be especially difficult for the number of nearest neighbors (p, see Eq. 2.6). The labeled data are only a small fraction of the entire dataset, and using CV on L suggests a p that is severely underestimated to adequately capture the graph structure of the data. More specifically, a p that is too small causes the Laplacian matrix (Eq. 2.7) to become so sparse that some unlabeled data *never* receive label information from their neighbors. This results in some rows in $\mathbf{F}$ corresponding to unlabeled data remaining as zero vectors (recall from Chapter 2.2.1 that the rows in $\mathbf{F}$ related to the unlabeled data are initialized to zeros). Upon convergence, each row in $\mathbf{F}$ is normalized to provide soft classifications for the unlabeled data, but this causes Python `RuntimeWarnings` if any rows are still zero vectors (i.e. divisions by zero). I am unaware of any literature at this time that provides

a systematic approach for determining p for LP, so I resort to a trial-and-error approach. In essence, I gradually increase p until I no longer receive `RuntimeWarnings`. For my problem, this is $p = 500 - 1000$, but I double it to $p = 2000$ to ensure that more than the absolute minimum information is used to construct the Laplacian matrix. This is my recommendation for setting p on other datasets: start low, increase it until `RuntimeWarnings` are no longer obtained, and then consider increasing p a bit more from this critically necessary value for added stability if the computational resources allow for it.

For setting the remaining semisupervised hyper-parameters, there is some intuition and prior work that can be leveraged. Recall that the remaining LP hyper-parameter, α, controls how much the initial label distribution is allowed to change, and setting its value can be informed by our confidence in the initial labels. Labeled data are sometimes mislabeled, and in many cases this is difficult to avoid (e.g., improper labels at the boundaries between units). Mislabeled data may be outliers, but if their classes are fixed ($\alpha = 0$), then these labeled data could propagate their false label information to nearby unlabeled data. However, if $\alpha > 0$, then this can allow a proportion of the initial labeled data, such as misclassified outliers, to take on the label information from their surroundings rather than propagate their false label information. Since I am confident in the ground truth facies model, I am comfortable setting α to a low value. A range of α values gives similar results, but I fix $\alpha = 0.10$. Self-training also poses similar challenges in determining values for its two hyper-parameters. However, Chapter 3 shows that default values for the self-training hyper-parameters work well ($S = 0.10, T = 0.50$), which continues to hold true in this study as well.

5.3.7 Evaluating performance

A benefit of this study is that I can measure the performance of the algorithms' predictions on the unlabeled data because I have a ground-truth model (Figure 5.7a). However, this requires a few steps because the machine learning predictions are in TWT, and the ground-truth model is in depth (these steps are depicted in Figure 5.9). First, I have the P-wave velocity associated with the model, so it is straightforward to convert

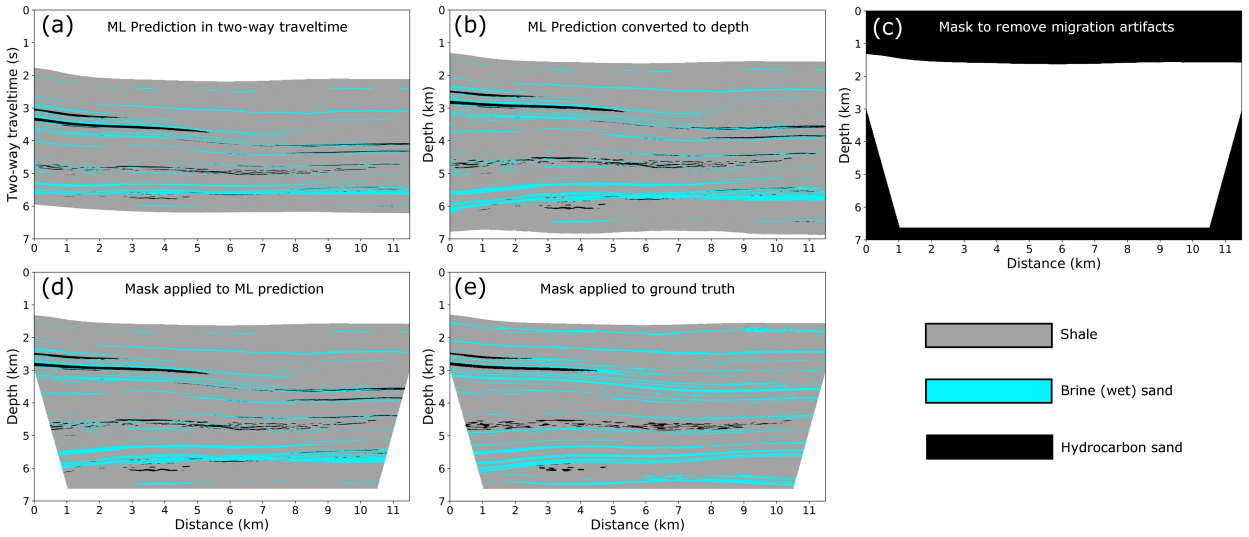

Figure 5.9: The steps needed to calculate the performance of the machine learning predictions. The (a) initial predictions in two-way traveltime are (b) converted to depth. There are migration artifacts, so a (c) mask is applied to the (d) machine learning prediction and the (e) facies model. The misclassifications computed between the prediction (d) and the model (e) are quantified by an evaluation metric.

the predictions from TWT to depth. When the predictions are converted to depth, there are some migration artifacts near the model edges (Figure 5.9b). I design a mask (Figure 5.9c) for the predictions so that these artifacts do not bias the performance evaluation (Figure 5.9d). The mask is also applied to the ground-truth model (Figure 5.9e) to facilitate the computation of evaluation metrics.

The misclassifications are commonly reported through a single value, or classification metric. Some of the most common metrics are precision, recall, and F1, which depend on elements of a confusion matrix (see Figure 5.10). Recall is a measure of how many of the known positives are predicted correctly, whereas precision measures how many of the predicted positives are actually correct. In other words, recall tries to minimize false negatives (FNs), and precision is trying to minimize false positives (FPs). For a review of classification metrics, I refer the reader to Lever et al. (2016). From a hydrocarbon exploration context, both precision and recall have practical value. One can interpret FNs as predicting shale when it is actually sand; optimizing recall will minimize the amount of reservoir volumes that are not recovered, thereby preventing the *underestimation* of reservoir volumes. FPs can be thought of as predicting sand when it is actually shale; optimizing precision minimizes the prediction of reservoirs that do not exist, thereby preventing

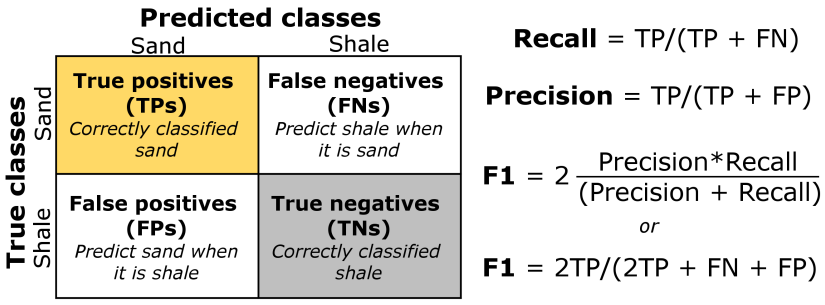

Figure 5.10: A sand-shale (two class) confusion matrix illustrating how the recall, precision, and F1 classification metrics are computed. False negatives (FNs) and false positives (FPs) are deemed misclassifications, and the type of misclassification in the denominator for a given metric specifies which type(s) of misclassification that metric is trying to minimize (i.e., FNs for recall, FPs for precision, and FNs and FPs for F1).

the *overestimation* of reservoir volumes. Precision may be more appropriate for deepwater exploration settings to reduce the cost/risk of drilling a dry hole (i.e., a FP). However, recall may be more appropriate in production settings where missed opportunities (i.e., FNs) and underestimated volumes also come with a cost. If both metrics are important, then optimizing one of these metrics individually (e.g., recall) comes at the expense of the other (precision), and vice-versa because there is this trade off between under and overestimation.

An alternative metric is the popular F1-score that takes the harmonic mean of recall and precision, which has the benefit of capturing more information at once from the confusion matrix, compared to precision or recall individually (see Figure 5.10). Given that the setup in this problem is one with limited well data, it is reasonable that this aligns with an exploration-oriented scenario where the information gained (i.e. machine learning predictions) from a limited number of wells would be used to help delineate other potential targets. Therefore, precision may be an appropriate metric here, but in the absence of an economic motive, I choose to use the balanced F1-score as the training metric (for XGBoost) and primary evaluation metric for all algorithms. That being said, I do use precision as a secondary metric which I incorporate into the discussion in Chapter 5.5.4. With regards to computing F1 (or precision) in a multi-class scenario, a separate F1 score is computed for each class, and then all the scores are combined into one value. The standard way to combine scores is to weight each F1 by the number of true instances for each class. The model (Figure 5.9e) contains

82% shale, 16% brine sand, and 2% hydrocarbon sand; so, the weighted-F1 score would give a majority of the weight to the shale, arguably the least important class. The metric I use is the macro-F1 score that gives equal weight to each of the classes in this problem (i.e., 33% for each class). The characterization of reservoir facies is undoubtedly the priority for seismic classification problems, so this metric assigns a necessary weight to the classes of interest.

5.4 Results

5.4.1 Training well #1

The first situation that I explore uses Well #1 as the labels for the training data, and the features that I use are those that come directly from the seismic inversion (i.e., AI, SI, and density). I also consider including the TWT as a feature to test if it can capture the depth dependency inherent in the base attributes. However, prior to any classification, I standardize the data using the *RobustScaler* class from `scikit-learn`. The results (with and without TWT) are shown in the first two rows of Table 5.1, and the predictions are depicted in Figure 5.11. In this instance, the self-training process appears to degrade the performance of LP. However, the performance metrics for LP appear to be higher than XGBoost, but there are significant artifacts in the predictions for all three methods, and including the TWT appears to accentuate these artifacts (see Figure 5.11).

In Chapter 5.2.2, I discuss how the inherent depth dependency of rock physics-based attributes may negatively impact classification performance because the rock properties for a given class are not unique. I suspect that the artifacts I observe in Figure 5.11 are a consequence of this. As such, I explore the effectiveness of removing the depth trends from the AI, SI, and density attributes. I demonstrate in Chapter 5.2.2 how to de-trend attributes using a 1D median filter on a single trace, but here I use a 2D median filter because the data are 2D. Figure 5.12 illustrates this process on the density attribute. The de-trending pro-

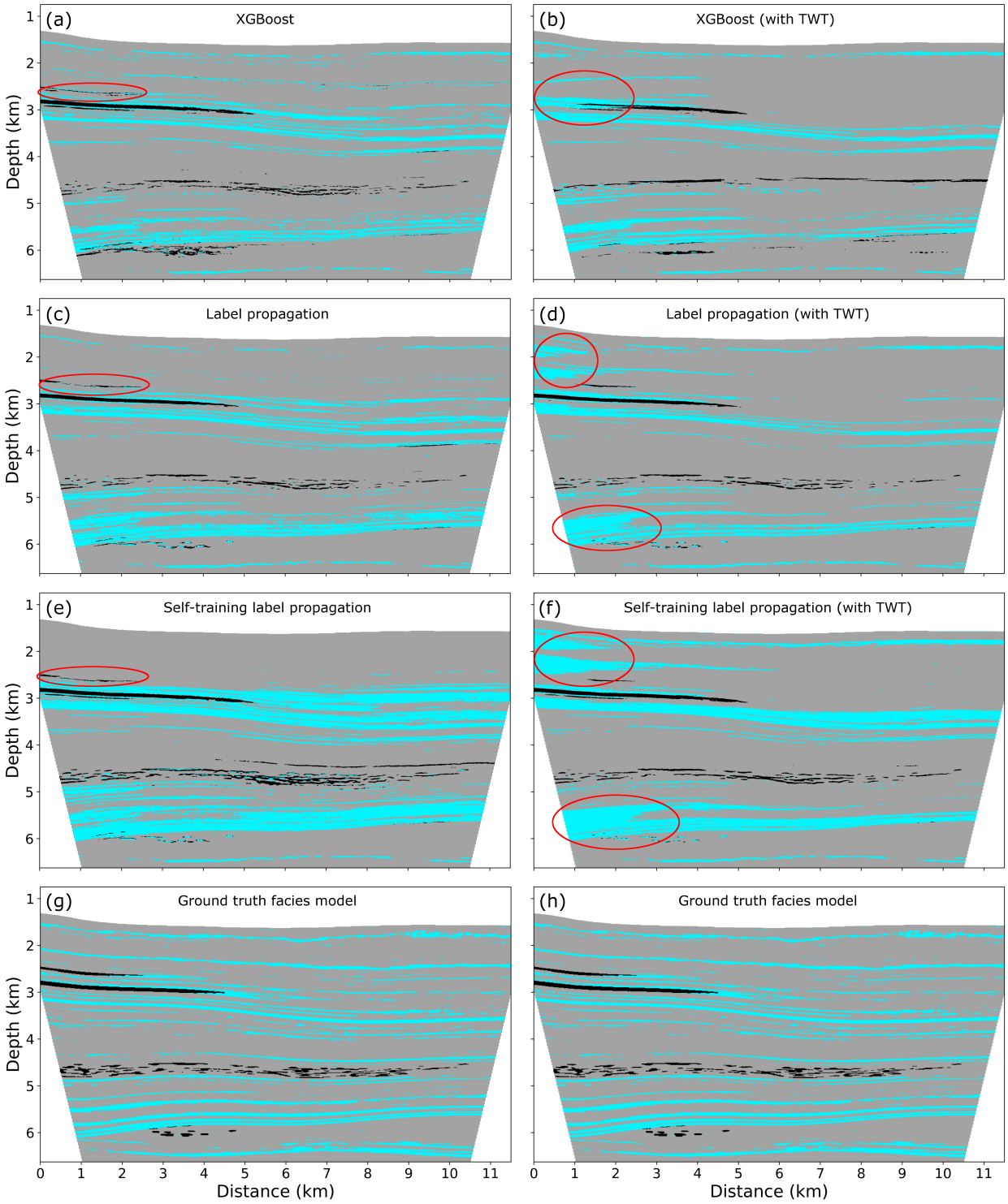

Figure 5.11: The machine learning predictions using the raw seismic attributes (AI, SI, density) as inputs and Well #1 as the training data. Panels (a, c, e) and panels (b, d, f) correspond to the results depicted in the first and second rows of Table 5.1, respectively. The red circles highlight some of the most-obvious artifacts associated with the depth dependency of the input attributes. (g, h) The ground truth model is provided at the bottom of each column for visual comparison.

Table 5.1: The unlabeled, testing data performance of each algorithm using **Well #1** as the training data. The first and last two rows correspond to the results using the raw and de-trended attributes, respectively. In the evaluation of each algorithm for a given set of attributes, six metrics are reported. While the Macro-F1 is the primary evaluation metric (in bold), I do provide other metrics for reference. In each cell, the metrics in the top row are the macro averages for precision, recall, and F1, respectively. The metrics in the bottom row are the individual per-class F1 scores (shale, wet sand, hydrocarbon sand) used to calculate the macro-F1 score. See the KEY for reference.

Input attributes	De-trending?	XGBoost	LP	Self-training LP
AI, SI, Rho	No	64.43, 60.26, **62.01** 90.59, 47.69, 47.73	69.59, 61.04, **64.54** 90.34, 49.05, 54.22	58.00, 62.78, **59.91** 86.63, 46.12, 46.97
AI, SI, Rho, TWT	No	62.15, 52.87, **56.26** 91.29, 48.21, 29.29	74.16, 61.42, **66.29** 91.54, 51.41, 55.93	67.22, 64.73, **65.68** 89.19, 51.46, 56.38
AI, SI, Rho	Yes	63.91, 63.07, **62.36** 90.45, 38.79, 57.84	72.44, 64.21, **67.28** 91.11, 43.55, 67.18	74.49, 64.19, **68.22** 90.38, 49.72, 64.55
AI, SI, Rho, TWT	Yes	67.31, 67.52, **66.67** 91.35, 49.74, 58.92	74.36, 64.99, **68.47** 91.62, 46.62, 67.16	75.20, 66.29, **69.88** 90.90, 53.39, 65.37

KEY	Macro averages for: Precision, Recall, **F1** Individual F1 for: Shale, Wet sand, HC sand

cess produces a rather drastic transformation of the attributes that can also be observed in cross-plots (i.e. plotting each attribute against each other, which produces a 3×3 grid for this example). A cross-plot of the raw attributes shows that they essentially have the appearance of measured logs and contain multi-modal distributions (Figure 5.13a). After de-trending the input attributes, one observation is that the shale class distribution is quite sharp; this is because the shale class is our background, and that is essentially what is being removed (Figure 5.13b). The hydrocarbon-bearing sand is now separated quite well from the other two classes, but it is worth noting that the degree of separability between the shale and wet sand classes is still minimal. Therefore, we can expect misclassifications to occur between the brine sand and shale classes. Another observation is that this de-trending process collapses each class to a Gaussian distribution. While this technically provides no additional benefit for LP or XGBoost because these methods do not make a Gaussian distribution assumption, it could be helpful for Bayesian-based techniques, such as a Bayes classifier. However, what potentially makes this de-trending process helpful for any method is that it improves the class separation (observed in Figure 5.13) and reduces the correlation between each attribute

by transforming the inputs.

The only potential challenge to this de-trending approach is choosing the kernel width size for the 2D median filter. However, the choice of kernel width can be quality controlled by observing the cross-plot of the training data (e.g., Figure 5.13b). If the kernel is too small, the signal from the sand classes ends up being subtracted out (this begins collapsing all classes to the same distribution). My approach is to gradually increase the kernel width until changes in the de-trended attributes are no longer observed.

The results using the de-trended input attributes (AI, SI, and density) are provided in the last two rows of Table 5.1, and Figure 5.14. These results indicate that de-trending the input attributes has a positive impact on both supervised and semisupervised methods. Given that the artifacts present in Figure 5.11 are no longer present in Figure 5.14, this suggests that the depth trends are indeed causing the artifacts in Figure 5.11. Furthermore, a comparison of the macro-F1 metrics in Table 5.1 indicates a quantitative improvement overall using de-trended rather than raw seismic attributes.

5.4.2 Training well #2

As mentioned in Chapter 5.3.1, I consider two different locations for the training well. The previous section explores using Well #1 as the training data location, and here I examine using Well #2 (see Figure 5.7a). Recall that the purpose here is to investigate the robustness of the machine learning methods to training data changes. Similar to the Well #1 scenario, I start by using the raw attribute inputs. The results are provided in the first two rows of Table 5.2, and they are relatively poor overall compared to the first two rows in Table 5.1. I show one of these results in Figure 5.15, where LP is misclassifying many areas of the section (while not shown, the other methods have identical issues). The next stage is to consider de-trending the seismic attributes. These results are provided in the last two rows of Table 5.2, and the predictions are shown in Figure 5.16. Upon comparing the results in Table 5.2, it is evident that de-trending the seismic attributes drastically improves the performance for all three algorithms.

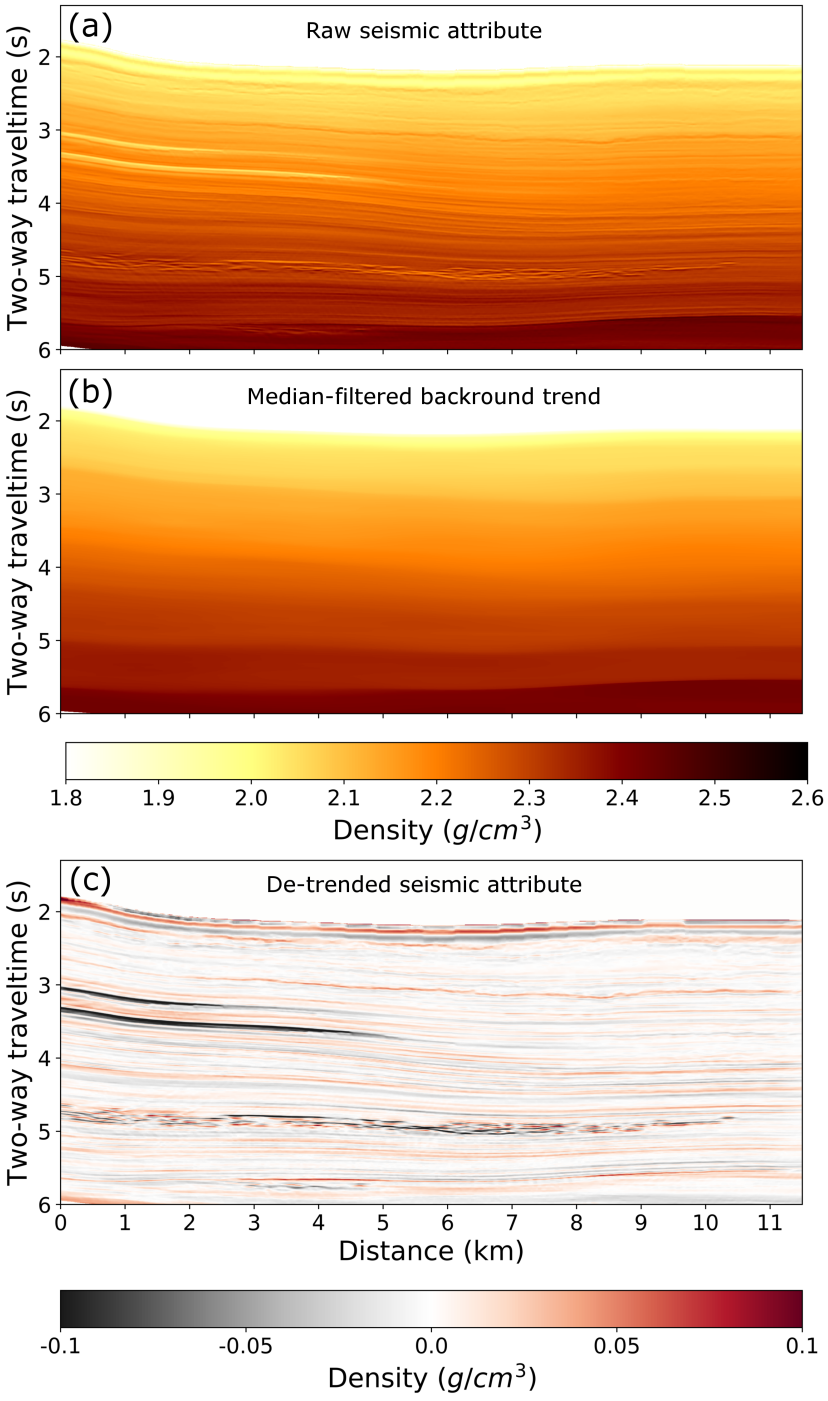

Figure 5.12: A depiction of the de-trending process for seismic attributes. A 2D median filter is applied to the (a) raw seismic attribute to extract the (b) smooth background trend. The background trend is then subtracted from the raw attribute to obtain the (c) de-trended attribute. The example shown here is for the density seismic attribute.

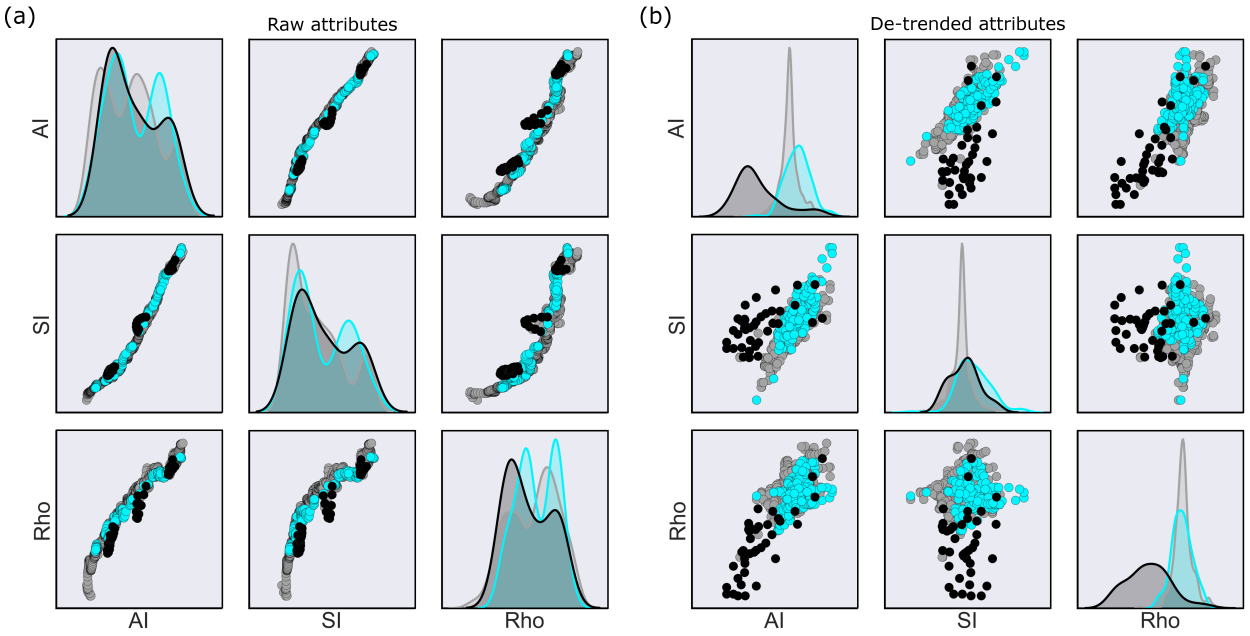

Figure 5.13: A cross-plot of the Well #1 training data using the (a) raw seismic attributes and (b) de-trended seismic attributes. The cross-plots shown have the attributes standardized in both instances. The elements on the diagonal provide the distribution of each class for each attribute. Removing the depth trend from the seismic attributes collapses each class to a Gaussian distribution.

Table 5.2: The unlabeled, testing data performance using **Well #2** as the training data. The first and last two rows correspond to the results using the raw and de-trended attributes, respectively. In the evaluation of each algorithm for a given set of attributes, six metrics are reported. While the Macro-F1 is the primary evaluation metric (in bold), I do provide other metrics for reference. In each cell, the metrics in the top row are the macro averages for precision, recall, and F1, respectively. The metrics in the bottom row are the individual per-class F1 scores (shale, wet sand, hydrocarbon sand) used to calculate the macro-F1 score. See the KEY for reference.

Input attributes	De-trending?	XGBoost	LP	Self-training LP
AI, SI, Rho	No	53.38, 52.22, **52.40** 90.02, 45.63, 21.54	53.22, 51.50, **51.71** 89.70, 46.28, 19.16	48.58, 55.87, **50.40** 85.94, 46.84, 18.41
AI, SI, Rho, TWT	No	46.18, 51.58, **47.91** 83.34, 41.79, 18.60	54.97, 53.40, **53.86** 87.08, 45.08, 29.41	48.29, 57.65, **50.92** 84.19, 46.37, 22.21
AI, SI, Rho	Yes	67.47, 56.84, **60.90** 88.46, 37.50, 56.73	68.91, 56.41, **60.90** 89.12, 40.44, 53.14	71.64, 67.27, **68.73** 88.66, 50.22, 67.31
AI, SI, Rho, TWT	Yes	50.29, 50.02, **49.42** 88.41, 42.17, 17.68	74.82, 58.10, **63.75** 90.84, 46.86, 53.57	73.00, 69.45, **70.56** 89.60, 54.55, 67.52

KEY	Macro averages for: Precision, Recall, **F1** Individual F1 for: Shale, Wet sand, HC sand

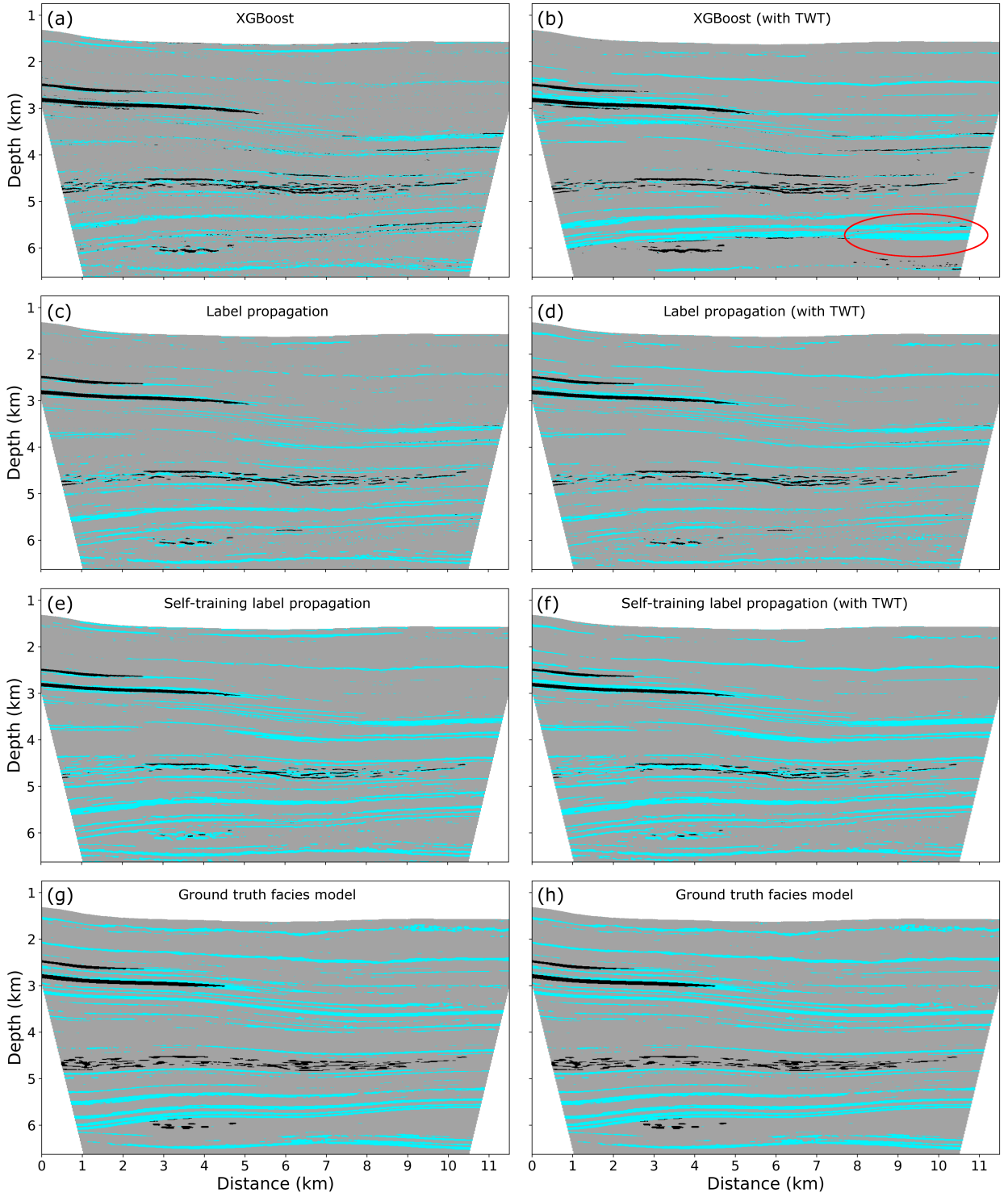

Figure 5.14: The machine learning predictions using the de-trended seismic attributes as inputs and Well #1 as the training data. Panels (a, c, e) and panels (b, d, f) correspond to the results depicted in the third and fourth rows of Table 5.1, respectively. These predictions, using de-trended inputs, indicate that many of the artifacts in Figure 5.11 are removed. However, including the TWT as input still produces an artifact for XGBoost (red circle). (g, h) The ground truth model is provided at the bottom of each column for visual comparison.

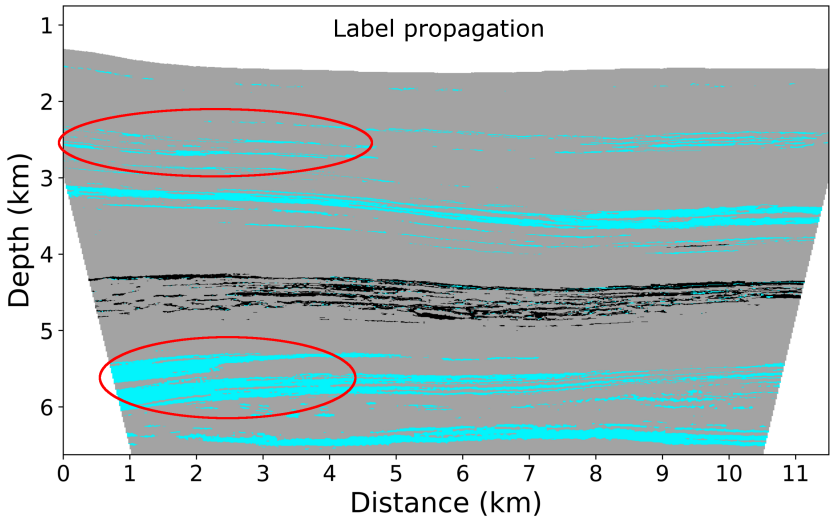

Figure 5.15: The label propagation prediction using the raw seismic attributes as inputs and Well #2 as the training data. This prediction corresponds to the LP entry in the first row of Table 5.2. The red circles highlight artifacts attributed to using the different training data, and the depth dependency of the attributes.

5.5 Discussion

Overall, these findings suggest that my coupled semisupervised method (self-training LP) can outperform a popular supervised algorithm (XGBoost) by a reasonable margin. Supervised learning methods are prone to overfitting in these minimal training data scenarios, but one potential remedy is to regularize the machine learning model to improve its generalization ability. I optimize the L2 regularization term for XGBoost, but this appears to be insufficient under these extreme circumstances. Another form of regularization is simply including more data. Including the unlabeled data into the objective function for a machine learning model (i.e., semisupervised learning) can be interpreted as a form of regularization. If the smoothness assumption of LP is satisfied (Chapter 5.5.1 below), then self-training coupled with LP can perform well. These results support the notion that semisupervised methods are better suited for problems facing scarce training data compared to supervised methods. Chapter 3 comes to the same conclusions using the same coupled self-training LP approach for well-log classification. Given that this coupled self-training LP approach is shown to be effective on two different datasets, this supports the robustness of the technique and its potential to be applied to other problems.

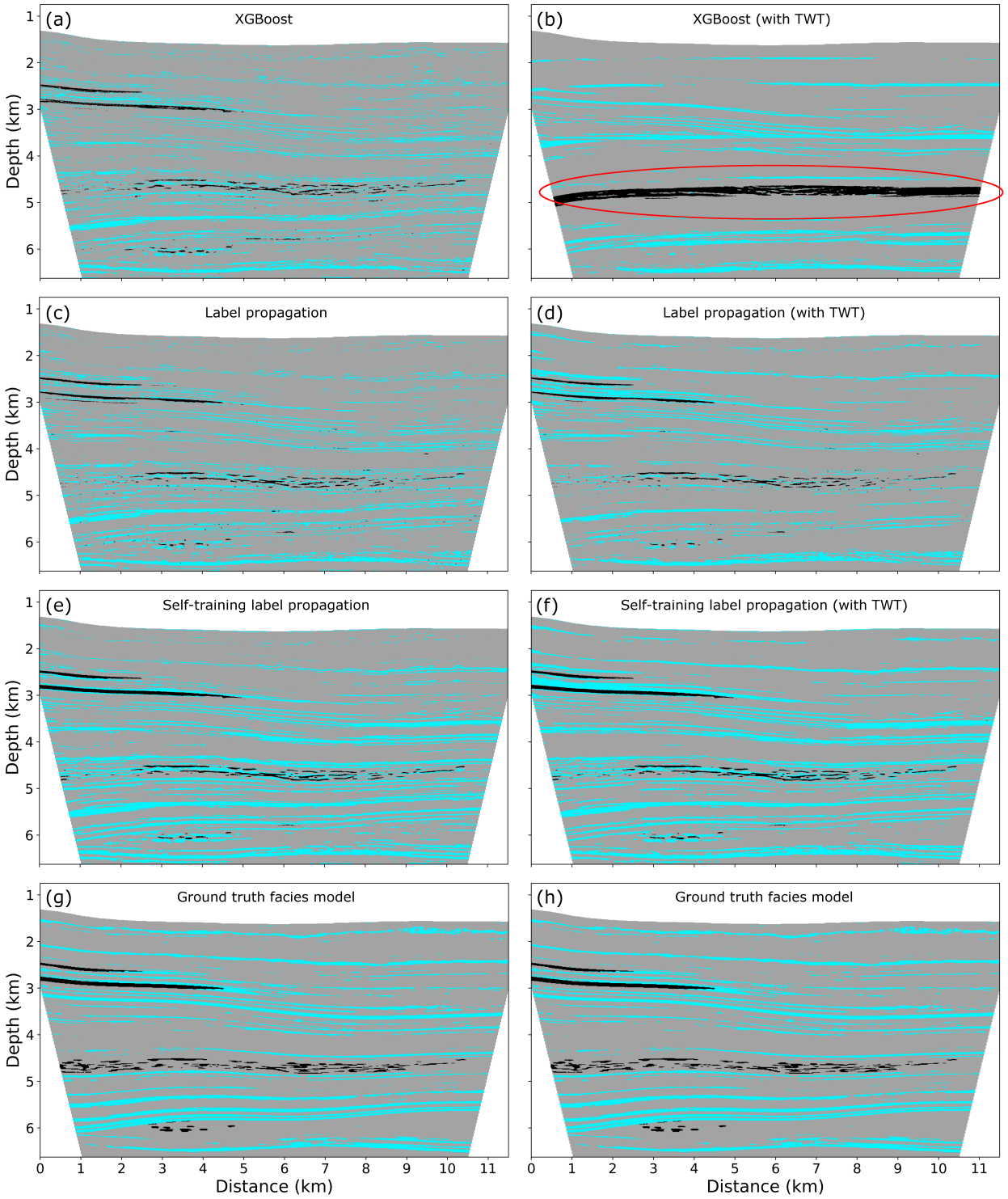

Figure 5.16: The machine learning predictions using the de-trended seismic attributes as inputs and Well #2 as the training data. Panels (a, c, e) and panels (b, d, f) correspond to the results depicted in the third and fourth rows of Table 5.2, respectively. Many of the artifacts present in Figure 5.15 are no longer present in panel (c). However, including the TWT as an input produces an artifact for XGBoost again (red circle). (g, h) The ground truth model is provided at the bottom of each column for visual comparison.

The simplicity of LP and self-training also provide a sense of transparency, which cannot be said for more complicated algorithms. Deep learning methods have many layers and hyper-parameters, making it more challenging to pinpoint what exactly is causing a problem if they are performing poorly. There are only a few sources to analyze for troubleshooting any issues with LP or self-training: the data themselves, and two hyper-parameters (for both LP and self-training). I begin my analysis below by discussing how the data themselves can negatively impact the machine learning performance and my solution for this problem.

5.5.1 Effectiveness of de-trending

The results that I present here explore many facets of a seismic petrophysical classification problem, and I begin by discussing the impact of de-trending the seismic attributes. When using the raw attributes as inputs, we see an overall poor performance for all algorithms (refer to the first two rows of Tables 5.1 and 5.2). However, upon de-trending the input attributes, we see a noticeable improvement (refer to the last two rows of Tables 5.1 and 5.2). With specific regard to the semisupervised methods, the changes in performance that we observe must relate directly to the changes in the inputs (i.e., de-trending) because the hyper-parameters for LP and self-training are kept consistent for both scenarios. I show in Figure 5.13 that de-trending the inputs is fundamentally a transformation that improves the class separation and reduces the inter-attribute correlation; it is these two phenomena that are responsible for the improvements we observe when the input attributes are de-trended.

I attribute the poor performance of LP when using the raw seismic attributes to the violation of a key assumption. LP has an inherent *smoothness* assumption, which states that nearby points in a concentrated region of the input space should share the same label; alternatively, if nearby points are separated by a low-density region, their labels need not be shared (Zhou et al., 2004; Chapelle et al., 2006). The depth trends in the raw attributes can cause points belonging to different classes to reside in the same regions of the attribute space (e.g., compaction trends causing a shallower shale and a slightly deeper sand to have the same density,

see Figure 5.1). The smoothness assumption is violated here because data points with the same properties do not always share the same label. This is an issue because it complicates the process of spreading labels from training data points to their unlabeled neighbors when the labels are conflicting in similar regions of the input space. What is happening in these conflicting label situations is the class with the most points is passing on its label to its neighbors, in some cases overwriting some of the other initial labels (this is allowed to happen 10% of the time because the hyper-parameter $\alpha = 0.10$). We can observe these phenomena in the LP prediction given in Figure 5.11(c); LP is overpredicting the presence of brine sand in areas more dominated by brine sand (e.g., 6 km depth), and the finer brine sand detail is being replaced by shale in regions dominated by shale (1-2.5 km depth). These complications with the LP result are perpetuated with self-training (see Figure 5.11e). This is also demonstrated in the well-log classification example in Chapter 3 where any issues with the underlying LP model can be perpetuated with self-training, and we observe a similar phenomenon here as well.

However, de-trending the seismic attributes helps points belonging to the same class to also reside within the same region of the input space (see Figure 5.13). This helps prevent the LP smoothness assumption from being violated, and this is noted by the improvement observed in Figure 5.14(c) compared to Figure 5.11(c). Furthermore, upon de-trending the inputs, my self-training process improves the performance of LP in every instance, which suggests that self-training can improve LP if the underlying graph is not violating the smoothness assumption. Overall, the de-trending process provides a quantitative benefit for all methods. In the Well #1 scenario, we observe up to a 10% improvement in the macro-F1 scores, with the most improvement seen with XGBoost when TWT is included (see rows 2 and 4 in Table 5.1). A similar observation is made in the Well #2 scenario, where self-training LP has improvements up to 20% in the macro-F1 scores (see Table 5.2).

Visually, the results also show that de-trending the inputs can alleviate many artifacts that occur on a broad scale. However, this process does appear to have introduced some minor artifacts in certain locations,

specifically the channel valley fills at 4.5 km depth. Notice in Figure 5.11 that these hydrocarbon-filled channel sands (denoted in black) are actually classified quite well for each method. A direct comparison of Figure 5.11 to Figure 5.14 shows that removing the background trend from the input attributes has introduced the presence of brine sands (denoted in blue) in this area at 4.5 km depth, which are misclassifications according to the ground-truth model (Figures 5.14g, 5.14h). Since these misclassifications occur for all three methods, this suggests that the de-trended attributes, not the algorithms, are the underlying issue. Not surprisingly, self-training appears to accentuate these artifacts. My technique of using a median filter to extract the background trend is a rather simplistic approach, so it is possible that a more sophisticated filter could achieve a better result here. However, despite these fine-scale artifacts, this de-trending process is still effective on a broad scale. It is worth noting that if the intent is to only classify a narrow interval, then de-trending the inputs is likely unnecessary.

5.5.2 Feature expansion

Another component of these results is that I explore feature expansion and selection. One way that I expand the input features is to simply include the TWT. The motivation behind including the TWT as a feature is to determine if it alone can capture the depth-dependency of the input attributes. The results (e.g., Figure 5.11) suggest that including the TWT cannot help the machine learning algorithms recognize the inherent depth trends of the seismic attributes. Although, upon de-trending the attributes, including the TWT always improves the performance for LP and self-training LP by 1-3% (see last two rows of Tables 5.1 and 5.2). On the contrary, including the TWT for XGBoost provides mixed results. To understand this phenomenon, it is important to know that decision tree-based methods can be conceptualized in this context as representing each class by a multi-attribute hypercube. In the true model, there are two leaf turbidites between 2 and 3 km depth towards the left of the section (see Figure 5.7a). An issue arises for XGBoost in the Well #1 case (using the raw attributes) because the training well only penetrates the deeper of these two leaf turbidites. When

including the TWT as a feature, the hypercube learned from XGBoost only thinks that the hydrocarbon sand class can occur at certain times based on the training data; this leads to a clear cutoff artifact at 2.9 km depth (see the red-circled region in Figure 5.11b). De-trending the inputs helps fix this problem for the Well #1 case, but an artifact of this nature is still observed for a brine sand at (x, z) = (10, 6) km in Figure 5.14(b). These problems are exacerbated in the Well #2 case because this well only samples the channel valley fills at 4.5 km depth (i.e. the hydrocarbon class only occurs within a narrow time window). All three methods require de-trended inputs to perform well in this case, but including the TWT limits the XGBoost classification of hydrocarbon sands to the only interval where they occur in the training data (Figure 5.16b). As such, these results suggest that one should be cautious when including TWT/depth as an attribute for decision tree methods.

Including the TWT expands the number of features by one, but I also consider additional features that can be generated from the base attributes. While these results are not provided here for brevity, they do suggest that including additional rock properties-based attributes (e.g., Lambda_Rho, Vp, Vs, Bulk modulus, and Vp/Vs) provides little to no benefit for this problem. It seems that including the TWT with the base attributes provides more benefit than including additional seismic attributes. However, the density attribute does appear to be an essential distinguishing attribute because the performance drops in most situations when the density is not included. These results perhaps demonstrate a quantitative benefit of the efforts to acquire quality far-offset seismic data to recover the density attribute from prestack seismic inversion.

5.5.3 Robustness to training data

I also consider two different locations for the training well to investigate the robustness of the machine learning methods to changes in the training data. Well #1 intersects most of the hydrocarbon-bearing units, whereas Well #2 only intersects one of the zones. A few observations can be made if we focus primarily on the de-trended results (last two rows of Tables 5.1 and 5.2). We observe a decrease in performance for both

XGBoost and LP when the training data change from Well #1 to Well #2. However, what is interesting is that the self-training LP approach achieves a comparable performance in both training data situations. These observations suggest that the self-training LP method is more robust to changes in training data.

5.5.4 Overall performance

The primary motivation for this work is to determine if the tested semisupervised techniques can achieve a better performance than supervised methods in a scarce training data scenario for seismic classification. The results using de-trended inputs (last two rows of Tables 5.1 and 5.2) offer some interesting observations on this theme. In the Well #1 case, LP outperforms XGBoost by a small margin (2-5% in macro-F1), but XGBoost and LP perform similarly in the Well #2 case if TWT is omitted. However, we see self-training LP achieving higher macro-F1 scores than XGBoost in each instance, with 3-6% improvement observed in the Well #1 case and 8% improvement (even larger if TWT is used) in the Well #2 case.

While I focus my primary analysis (above) on using the macro-F1 score as the evaluation metric on the testing data predictions, this metric lacks interpretability compared to using the individual precision or recall metrics. For instance, does a higher macro-F1 score indicate improvements in both precision and recall, or just one of those metrics? As mentioned in Chapter 5.3.7, precision is also an appropriate metric to consider for this problem, and I consider it here as a secondary metric. I also include recall for completeness (following the argument made in Chapter 5.3.7, I do still use the macro averages for these secondary metrics). It is worth noting that even though the cross-validation scoring metric used for XGBoost is macro-F1, using other scoring metrics (e.g., macro-precision) still selects XGBoost models that perform similarly to the macro-F1 chosen models. The macro precision and recall values are provided in Tables 5.1 and 5.2 (purple and red values, respectively). For brevity, I restrict the analysis here to the results of the Well #1 case using the de-trended inputs (refer to bottom two rows of Table 5.1). We generally observe that the macro-recall for XGBoost, LP, and self-training LP are relatively consistent, but self-training LP

performs 7-11% better in macro-precision (this implies that the 3-6% improvement in macro-F1 mentioned above is primarily due to significant improvements in macro-precision). These results indicate that all three algorithms have roughly the same number of false negatives in their predictions, but the semisupervised methods have much fewer false positives. The predictions in Figure 5.14 support this. The hydrocarbon class predictions for XGBoost have obvious false positives (e.g., black units predicted at 4 km depth in Figs. 5.14a and 5.14b), and this is related to XGBoost overfitting the training data. These false positives are not present in either of the LP or self-training LP predictions, which explains why these two methods have a much higher macro-precision.

The utilization of macro-averaged metrics (e.g., macro-F1 and macro-precision) for evaluating predictions leads to the observations and ultimate conclusions of this work that semisupervised methods (namely self-training LP) are outperforming XGBoost under the context of minimal training data. However, one could perhaps arrive at a different, arguably misleading, conclusion if different metrics are used, such as accuracy or weighted-F1 scores. These metrics are biased by classes with more data points, and any improvements in the least populated classes may not be adequately captured. This is critically important given that the model has class imbalance (82% shale, 16% brine sand, and 2% hydrocarbon sand). For instance, in the Well #1 case (referring to the fourth row of Table 5.1), XGBoost has a weighted-F1 = 84.16% and accuracy = 85.02%, and self-training LP has a weighted-F1 = 84.48% and accuracy = 84.64% (note: these values are not shown in Table 5.1, I am simply providing them here). The performance of both methods in this scenario using both of these metrics is nearly identical, but the associated predictions for XGBoost (Figure 5.14b) and self-training LP (Figure 5.14f) are still quite different. Here, it is clear that self-training LP recovers more accurate sand detail than XGBoost, quantified by self-training LP achieving over a 6% better hydrocarbon class F1 than XGBoost. For self-training LP, this 6% improvement in the hydrocarbon class is negligible in any weighted score because only 2% of the data are hydrocarbon sand. XGBoost performs marginally better than self-training LP using a weighted metric, such as accuracy, because XGBoost

Table 5.3: Elapsed computation time of each machine learning method associated with the de-trended attribute scenarios of Table 5.1. All computations are performed on ComputeCanada resources using a single node (2 $\times$ Intel E5-2683 v4 Broadwell @ 2.1GHz, 32 cores).

Input attributes	XGBoost	LP	Self-training LP
AI, SI, Rho	0m 56s	16m 38s	1h 24m 54s
AI, SI, Rho, TWT	1m 00s	20m 01s	2h 05m 59s

performs $< 1\%$ better on the most populous class, shale. Unfortunately, this is misleading given what we observe in the predictions themselves. If the most populous class is deemed the most important, then weighted metrics would be appropriate. However, this is not the case for this scenario because the least-populated classes have more interest. In summary, it is essential to consider which metric(s) are best suited for a given problem, otherwise, an inappropriate metric can provide misleading results, as I show here.

5.5.5 Pitfalls

While the semisupervised method, self-training LP, can outperform XGBoost from a classification standpoint, other aspects need to be considered, such as computational resources. Table 5.3 shows the computation time requirements for the de-trended attribute scenarios given in Table 5.1. XGBoost runs rather quickly, and this is because it only trains on ≈ 1000 data points. The LP method requires more computation time, which comes as no surprise because LP must learn an internal mapping on all 580k data points. If it takes on average 18 minutes for LP to run and 1 minute for XGBoost to run, then XGBoost is 18x faster here, but LP is utilizing 580x more data than XGBoost. However, what is expensive is the self-training process because each iteration of the self-training process requires me to solve for the LP spreading function, and I use five self-training iterations. For the size of this problem, the computation time requirements are still quite manageable (max. 2 hours), and these could easily be reduced if GPU nodes, or more CPU cores, are utilized (these results have been executed on one CPU node with 32 cores).

What is perhaps the most limiting factor for LP is the memory storage of the adjacency matrix. For this problem that contains 580k data points (and using 2000 nearest neighbors), LP requires ≈ 50 GB of RAM,

which is relatively easy to accommodate. However, if I do not reduce the seismic sampling from 2 to 4 ms, then the problem contains 1.16M data points, and I must double the nearest neighbors to 4000 to get comparable results; this situation requires over 200 GB of RAM. For 3D problems with several million data points, this implementation of LP in `scikit-learn` may be infeasible. The results suggest that applying self-training to LP can recover better classifications than a robust supervised approach (XGBoost), but one must consider if the accuracy improvements are worth the extra computational resources based on the size of their own problem and the available resources.

5.5.6 Recommendations

Any interested readers who seek to apply the label propagation (LP) or self-training methods to their own problem or dataset should consider a few factors such as the number of labeled data, complexity of the problem, size of the dataset, whether the machine learning problem is static or dynamic, model assumptions, and model interpretability. If a dataset has limited training data, then semisupervised methods, in general, may be the most appropriate. However, if the machine learning problem is straightforward (e.g., linearly separable classes), then semisupervised techniques may provide no improvement over supervised methods because the unlabeled data will provide no additional value (i.e., a supervised linear classifier is sufficient). However, as discussed in the previous section, the size of the dataset can be problematic for transductive methods like LP because all unlabeled data that are to be classified must be included in the generation of the adjacency matrix; this can be computationally impractical if the unlabeled data are on the order of millions, or larger. However, Liu et al. (2012) do summarize some techniques used to address the scalability issue with LP, some of which include using anchor points (Liu et al., 2010) or using a low-rank approximation of the adjacency matrix (Zhang et al., 2009). An alternative in these big data situations is to consider inductive semisupervised methods, such as semisupervised Gaussian mixture models (Chapter 2.2.3). Inductive semisupervised methods allow one to train the algorithm using a realistic subset of the unlabeled data, and

then that model can be used to predict class labels for the remaining unlabeled data not used in training.

Another essential factor is whether the machine learning problem is static or dynamic, i.e., if the dataset is fixed or if the dataset is consistently augmented with new, real-time data. In a dynamic case, LP would have to be retrained for each situation when new data need to be classified, which would be inefficient. Inductive methods would not need to be retrained in these situations where the new data have the same dimensionality as the initial training data. However, there are situations where the new data in a dynamic problem may be including additional features (e.g., an additional survey measuring a different property), and any algorithm would have to be retrained in this scenario. Nonetheless, transductive methods are best suited for static machine learning problems.

Some additional factors that are also important to consider are the machine learning model assumptions and interpretability. A benefit of the LP method that I make in this paper is that it is simple to interpret and implement, which cannot be said for more complicated semisupervised algorithms. The self-training method has similar advantages, and it has the flexibility to wrap around any algorithm that makes predictions for unlabeled data. In this paper, I exclusively apply self-training to LP, but self-training techniques can also be applied to supervised methods to improve their performance. Furthermore, how the data are distributed in feature space will be different for different datasets, so choosing a machine learning method with the appropriate model assumptions is necessary. Similar to neural networks and other non-parametric methods (e.g., SVM, decision trees), a benefit of LP is that it makes no distribution assumptions. However, as I discuss in Chapter 5.5.1, LP does have a smoothness assumption. While I am not aware of any quantitative measures to test the smoothness assumption prior to classification, a qualitative assessment of the training data can be diagnostic (e.g., Figure 5.13). In summary, I find that LP is best suited for machine learning problems with these traits: minimal training data, complex, relatively small data size, and static.

5.6 Conclusion

In this study, I propose the use of simple, straightforward semisupervised techniques for solving a synthetic seismic petrophysical classification problem with minimal training data. Supervised learning algorithms are susceptible to overfitting if the training data are minimal, and semisupervised techniques can be a potential remedy in these situations. Many semisupervised algorithms do exist, but label propagation and self-training have the advantage of having only two hyper-parameters each and they are both conceptually simple. The simplicity of label propagation and self-training allows me to easily determine any underlying causes of poor performance; the same cannot be said for more complicated techniques, such as deep learning methods. I apply these two semisupervised techniques and a supervised method, XGBoost, to a seismic petrophysical classification scenario built using the 2D SEAM model. To simulate a scarce training data scenario, I assume that there is only one well, but I vary the location of this well to determine how robust each algorithm is to training data changes. I find that the seismic attributes must have their background trends removed for any of the machine learning algorithms to perform well. Upon de-trending the inputs, label propagation and XGBoost perform similarly, but label propagation with self-training applied does outperform XGBoost. The coupled self-training label propagation approach is also shown to be quite robust to training data changes as well. These findings extend the premise to an exploration geophysics problem, namely seismic petrophysical classification, that semisupervised algorithms can provide more robust, generalized predictions than supervised methods in the presence of scarce training data.

Chapter 6

Are seismic attributes still helpful for deep learning?[1]

This short chapter is based on an SEG Expanded Abstract from 2021, which has a focus that is a minor detour from the main theme of the thesis, but additional results (that are not originally in the abstract) are included here in order for this chapter to tie into the previous work. Supervised deep learning methods have not been considered thus far to compare the semisupervised methods against because of an assumption that they would perform poorly in the minimum training data theme of this thesis. However, it is more convincing to support this notion with tangible results. As such, this chapter is a continuation of the seismic classification problem from Chapter 5, using the same dataset, but from the perspective of deep learning. The only supervised algorithm used in Chapter 5 is XGBoost, so the results presented here provide another source of comparison. A benefit of deep learning techniques is that classification can be performed directly on the seismic data. A secondary aspect of this work that I explore is determining if deep learning methods can still benefit if they are trained on seismic data *and* seismic attributes. Using the angle stacks and the seismic inversion attributes from Chapter 5, I can explore this hypothesis. I also consider using the

[1]Dunham, M.W., 2021. Are seismic attributes still helpful for deep learning?, in *SEG Technical Program Expanded Abstracts 2021*, pp. 1561-1565, Society of Exploration Geophysicists.

original seismic inversion attributes and the de-trended versions to determine the impact of de-trending. Four different training situations are explored, but the latter two situations use the same Well #1 and Well #2 training well locations in Chapter 5 (see Figure 5.7a) to facilitate comparisons to that work.

6.1 Introduction

The goal of machine learning in seismic facies classification is to map seismic data to classes of interest (e.g., seismic facies, lithofacies, or petrofacies). The two main approaches in the literature to carry out this task are unsupervised and supervised learning schemes. Unsupervised methods have included self-organizing maps (SOMs, Strecker & Uden, 2002; de Matos et al., 2007; Roden et al., 2015), generative topographic mapping (GTM, Roy et al., 2014; Qi et al., 2016), and Gaussian mixture models (GMMs, Feng et al., 2018a; Wallet & Hardisty, 2019). For supervised methods, some popular algorithms have included shallow neural networks (Saggaf et al., 2003; Ross & Cole, 2017) and support vector machines (SVM, Li & Castagna, 2004; Zhao et al., 2015). The aforementioned traditional supervised techniques are limited in their ability to train directly on the seismic data; they rely exclusively on feature engineering and feature selection as a critical part of their methodology (e.g., requiring the careful selection of seismic attributes that best relate to the classes of interest).

Deep learning techniques, however, are able to classify *and* learn the features themselves from the raw inputs (LeCun et al., 2015). In essence, deep learning techniques can be directly applied to seismic data without the need to compute attributes. Many have exploited this benefit of deep learning in recent years in the context of seismic classification. Convolutional neural networks (CNNs) treat the seismic data as an image and can extract the necessary features through filters to classify and delineate faults (Araya-Polo et al., 2017; Cunha et al., 2020), salt bodies (Waldeland et al., 2018), and seismic facies (Souza et al., 2019). Recurrent neural networks (RNNs) are another form of deep learning that are designed for sequential (time-dependent) data. While originally intended for machine translation (Graves et al., 2013; Sutskever et al.,

2014), RNNs can be extended to seismic classification by treating seismic traces as sequences (Grana et al., 2020).

It is undoubtedly an advantage over traditional techniques that deep learning methods do not require feature engineering. Recent publications show that deep learning methods trained directly on the seismic data can perform better than traditional methods that must train on seismic attributes (e.g., Souza et al., 2019). However, if the seismic attributes are also included in the training for deep learning methods, could their performance improve even more? Or does including seismic attributes provide no benefit to deep learning methods, thereby rendering seismic attributes obsolete? To the best of my knowledge, these questions have yet to be addressed in the literature. To investigate these ideas, I formulate a seismic classification problem using a subset of the Phase 1 2D SEAM model from Chapter 5. An advantage of this being a synthetic problem is that the ground truth facies model (Chapter 5.3.3) can be used to quantify the predictions of the testing data. The input data for the machine learning problem are the six angle stacks generated from the model (Chapter 5.3.4), and the prestack seismic inversion attributes. I train and evaluate the performance of an RNN when using the (1) angle stacks, (2) angle stacks + seismic inversion attributes, and (3) angle stacks + de-trended seismic inversion attributes as inputs in four different scenarios. Scenario 1 does a generic 80%-20% train-test split of all the input traces. Scenario 2 is a more realistic scenario where only ten wells (or ten traces) are used for training. The final two scenarios push RNNs to their limit by training on only a single well (one trace), which are the Well #1 and Well #2 scenarios from Chapter 5.

6.2 Data preparation

The model utilized for this study is the same subset of the Phase 1 2D SEAM model that is used in the previous chapter (see Figure 5.2). As discussed in Chapter 5.3.4, seismic data are simulated from this elastic model using Devito (Louboutin et al., 2019), the shot records are migrated using prestack time migration, and then the data are ultimately stacked into six angle stacks. Figure 6.1(a) shows one of the angle stacks.

These six angle stacks provide the base inputs for the RNN (discussed below), but one of the aspects of this work is to determine the benefit (if any) of including seismic attributes as additional inputs to the RNN. As such, the prestack seismic inversion attributes (acoustic impedance - AI, shear impedance - SI, and density - Rho) computed in Chapter 5.3.4 are also used here. The inherent depth trends of these attributes negatively impact classification performance in the previous work (Chapter 5.5.1); so for continuity, I also explore if this factor has any impact on RNN performance. Figure 6.1(b) shows the de-trended version of AI.

The seismic stacks and the inversion attributes serve as the inputs for the RNN, but I also need to establish the class labels. Recall from Chapter 5.3.3 that thresholds are applied to the shale volume (Vshale) and resistivity model properties to define a three-class model with shale, wet sand, and hydrocarbon sand classes. I use the same model in this study, however, this ground truth model is in depth and the seismic data are in two-way traveltime (TWT). So, I decide to convert the facies model to TWT and resample it so that each seismic sample from each trace has a corresponding class label (Figure 6.1c). The previous study converts the machine learning predictions from TWT to depth (see Figure 5.9), but here, I elect to leave the predictions in TWT and convert the model to TWT to measure performance.

6.3 Methodology

Traditional machine learning methods treat each data point independently, but seismic traces are inherently time-dependent due to the nature of the convolutional model ($trace = wavelet * reflectivity$). A single change in reflectivity generates a response across multiple samples in a trace due to the band-limited nature of the wavelet; so, independently mapping each seismic sample to a class may be an over-simplification. An alternative approach to this problem is to consider methods designed for sequential data, such as recurrent neural networks (RNNs). We can treat each seismic trace as a sequence, and RNNs learn by processing these sequences one sample at a time where the RNN hidden units contain information about the history of the previous samples in a sequence (LeCun et al., 2015). Theoretically, RNNs have the capability of

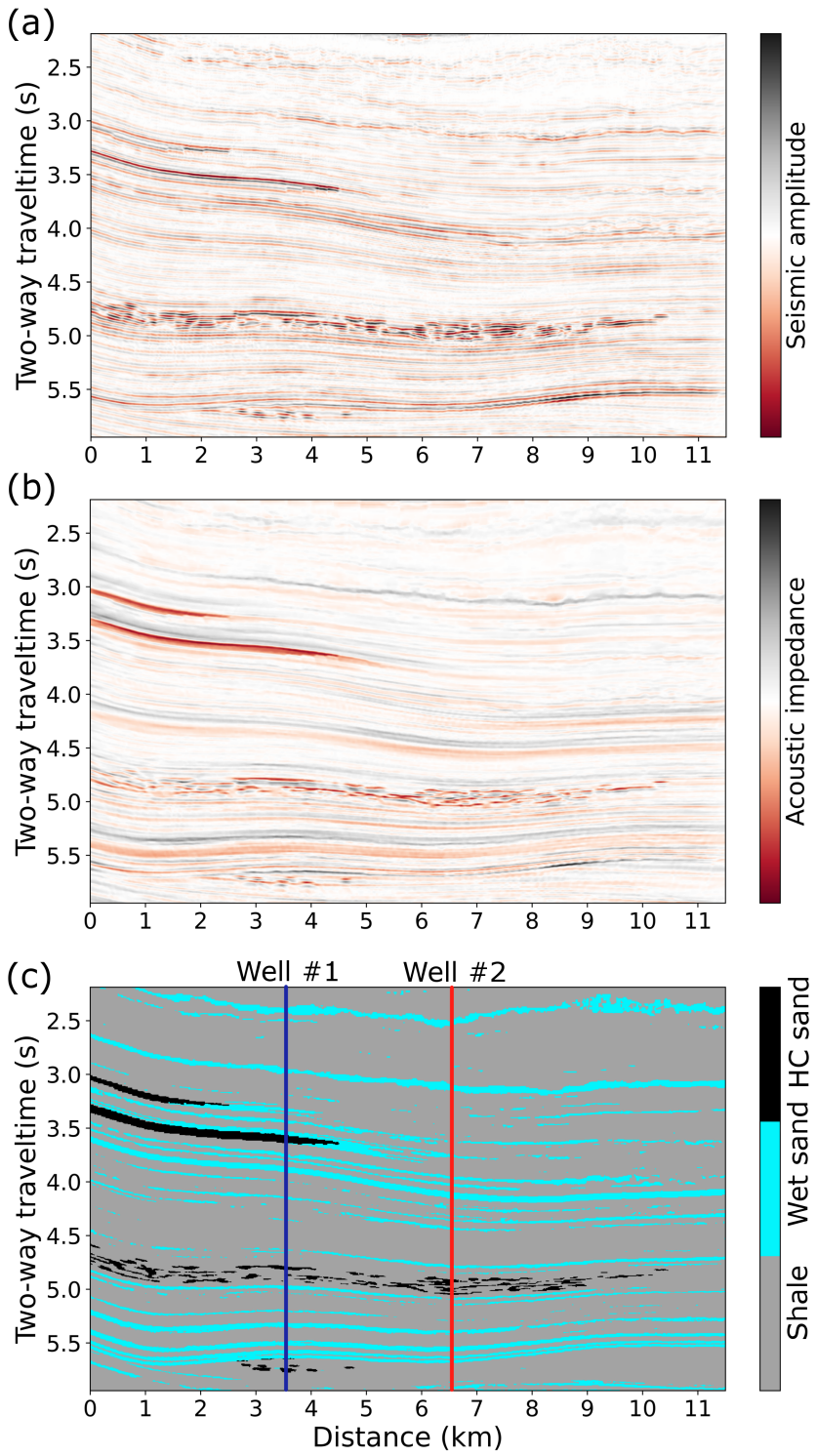

Figure 6.1: (a) Near angle stack (6-12°) generated from the model subset denoted by the red box in Figure 5.2. (b) Acoustic impedance attribute recovered through prestack seismic inversion with de-trending applied. (c) Ground truth facies model denoting three classes. The locations of Well #1 and Well #2 are indicated for reference.

processing long sequences, but they are limited to learning a history of only a few previous samples because the backpropagated gradients tend to either explode or vanish over many time steps (Pascanu et al., 2013). This limitation is overcome by improvements to the RNN unit structure, such as the long short-term memory (LSTM) unit (Hochreiter & Schmidhuber, 1997; Sak et al., 2014).

The RNN model that I use (Figure 6.2) consists of an input layer, an LSTM layer, a dropout layer, and a time-distributed layer connected to a softmax layer. The optimizer for this model is Adam (Kingma & Ba, 2014), and the loss function is sparse categorical cross-entropy. This model is inspired by the RNN model from Grana et al. (2020), which is also used for a seismic classification problem. The data for the input layer are structured as a 3D array with size = (ns, nt, nd), where ns is the number of sequences (or seismic traces), nt is the number of samples/timesteps in each sequence, and nd is the number of features (i.e., how many stacks or attributes are used). The corresponding output is structured as a 2D array of size = (ns, nt). Since each (multi-dimensional) seismic sample has a corresponding label, this is referred to as a Many-to-Many sequence problem; the purpose of the time-distributed layer is to allow the softmax operation to be applied to every sample of an input sequence (i.e., predict an output for every input sample). The distinction between this RNN model (Figure 6.2) and the one from Grana et al. (2020), is the inclusion of the dropout layer. Dropout is a form of regularization for neural networks that helps prevent overfitting by randomly removing a proportion of units (defined by the user) from a given layer at each iteration (Srivastava et al., 2014). I use default settings for many of the hyper-parameters, but two still require optimization: the number of units for the LSTM layer and the dropout value. This model is implemented using Keras with a Tensorflow backend (v. 2.3.0), and the next section summarizes the results using this model.

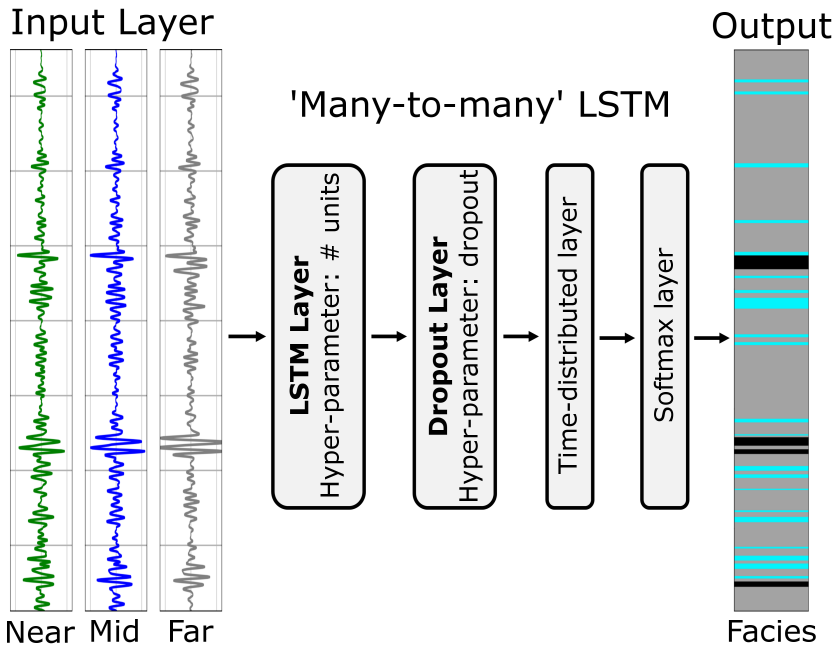

Figure 6.2: A schematic of the 'Many-to-Many' RNN model used to classify seismic data for this study. The traces shown for the input layer (6-12° Near, 18-24° Mid, and 30-36° Far angle stacks) are the output labels are extracted from the Well #1 location (see Figure 6.1c). For details regarding the RNN, refer to the text.

6.4 Results

6.4.1 Scenario 1

The first scenario that I explore is randomly splitting the total number of sequences (ns = 576) into 80% and 20% for training and testing, respectively. As mentioned previously, I consider three input situations for each scenario: (a) six angle stacks (nd = 6), (b) six angle stacks + seismic attributes (nd = 9), and (c) six angle stacks + de-trended seismic attributes (nd = 9). I tune the hyper-parameter settings using the RandomSearch class from Keras Tuner on 20% of the training data put aside for validation. For all three situations, the RNN model converges after about 50 epochs (see Scenario 1 curves in Figure 6.3a).

The training, validation, and testing scores for Scenario 1 are indicated in the top portion of Table 6.1. The accuracy metric can be misleading for imbalanced datasets, which is valid for this example (83% shale, 15% wet sand, 2% hydrocarbon sand). The macro scores provide a better representation of performance

Table 6.1: Results of the RNN performance for Scenarios 1 and 2. Each scenario has three different input data situations as stated in the text and these are denoted with (a), (b), and (c). The final column provides performance metrics of the RNN model when applied to all sequences (i.e., training and testing combined); see the key at the bottom for what each value represents. *Denotes de-trended versions of the seismic inversion attributes. **Denotes an input situation in Scenario 2 where sample weights are *not* used; the next three rows are situations where sample weights are used.

Input data		**RNN hyperparameters**	**Training accuracy**	**Validation accuracy**	**Testing accuracy**	**Global performance metrics**
Scenario 1: Train (64%), validate (16%), test (20%)						
(1a)	Angle stacks	RNN units = 128 Dropout = 0.1	95.49	94.95	95.02	90.41, 88.45, **89.41** 97.32, 86.41, 84.48
(1b)	Angle stacks, AI, SI, Rho	RNN units = 128 Dropout = 0.1	95.43	95.07	95.14	90.52, 89.10, **89.80** 97.37, 86.57, 85.45
(1c)	Angle stacks, *AI, *SI, *Rho	RNN units = 128 Dropout = 0.1	96.33	95.76	95.67	91.57, 91.08, **91.31** 97.74, 88.81, 87.38
Scenario 2: Train (1.4%), validate (0.4%), test (98.2%) – 10 training wells						
(2a)	Angle stacks**	RNN units = 64 Dropout = 0.5	84.44	86.47	84.19	65.11, 47.46, **51.38** 91.25, 26.89, 36.00
(2a)	Angle stacks	RNN units = 64 Dropout = 0.5	85.92	84.93	85.01	64.57, 72.72, **67.35** 91.24, 53.11, 57.69
(2b)	Angle stacks, AI, SI, Rho	RNN units = 80 Dropout = 0.4	87.03	87.01	86.26	65.86, 74.18, **68.50** 92.01, 57.88, 55.62
(2c)	Angle stacks, *AI, *SI, *Rho	RNN units = 80 Dropout = 0.4	90.04	90.15	88.36	73.04, 78.64, **75.39** 93.15, 64.28, 68.73

KEY	Macro averages for: Precision, Recall, **F1** Individual F1 for: Shale, Wet sand, HC sand

on imbalanced datasets (as discussed in Chapter 5.3.7). However, these metrics have been removed from the Keras (v.2) core because they can introduce complications when training is performed in batches. Consequently, this does restrict the RNN to train using accuracy as its metric (see Scenario 1 curves in Figure 6.3b), but this does not prevent me from using other metrics to evaluate the predictions. I use the trained RNN in each input situation to make predictions on all sequences (i.e., training and testing combined), and then I compute the macro precision, recall, and F1 scores, as well as the individual F1 scores for each class (last column in Table 6.1). This provides an understanding of how the algorithm performance is doing as a whole, and on a per-class basis. Despite the model being dominated by shale, the individual F1 scores for wet sand and hydrocarbon sand are still quite good in this scenario ($> 85\%$).

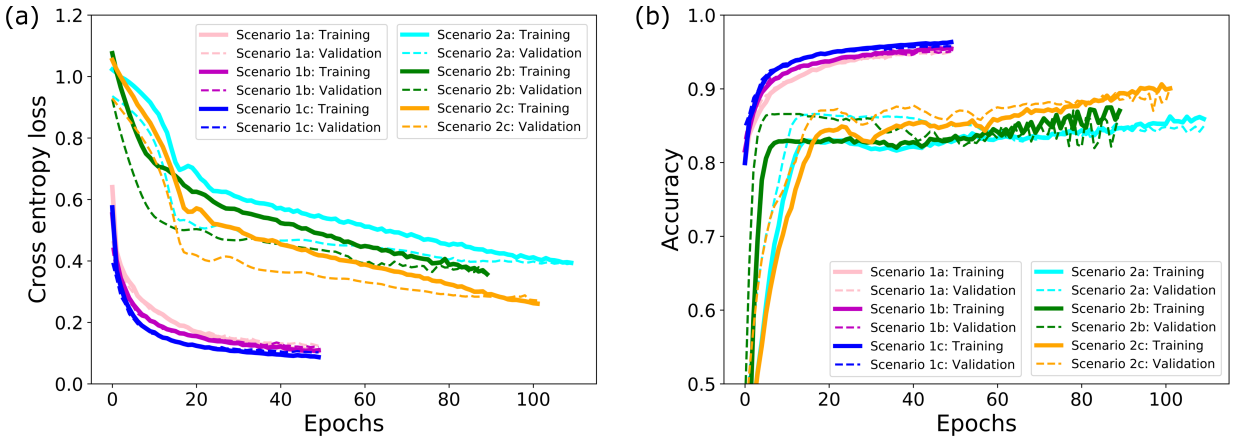

Figure 6.3: The (a) categorical cross-entropy loss and (b) accuracy curves for Scenarios 1 and 2. The performance of both of these scenarios is depicted in Table 6.1. All training curves are solid and all validation curves are dashed.

6.4.2 Scenario 2

Scenario 1 is not particularly realistic, with 80% of the model labels known a priori. Scenario 2 is more feasible where I assume that there are only ten wells (or ten sequences) available for training, and the remaining sequences are used for testing (I randomly select two of the training wells for validation and hyper-parameter tuning). The initial predictions (for all input situations) for this scenario drastically over-predict the presence of shale (i.e., there is very little sensitivity to the brine and HC sands, see the first row for Scenario 2 in Table 6.1). This phenomenon does not occur in Scenario 1, which is likely because more training data are used, and more complicated RNN models (i.e., more units) can be used without overfitting. The loss, or penalty, of misclassifying a point belonging to any particular class is the same. So what is happening here is that the model can still achieve an 83% accuracy even if it predicts everything as shale because there is no added penalty for classifying the sand classes incorrectly. To mitigate this issue in this scenario, I have to incorporate *sample weights*. These weights are inversely proportional to the number of training points per class, which give higher penalties for misclassifying the less frequent sand classes. I calibrate these weights by adjusting the values and qualitatively assessing how the predictions match the true labels for the training data. The weights that I determine for shale, wet sand, and HC sand are 0.75,

1.50, and 4.00, respectively. Using these weights significantly improves the performance of the sand classes, as seen by the second row for Scenario 2 in Table 6.1.

The sample weights are used for each of the three input scenarios, and the numerical results are provided in the bottom three rows of Table 6.1. Since far fewer data are used to train the RNN models in Scenario 2, they end up requiring more epochs to converge compared to Scenario 1 (see Scenario 2 curves in Figure 6.3a). Obvious overfitting occurs if these models run for too many epochs, so I stop training when I start to observe the training accuracy surpassing the validation accuracy.

6.4.3 Scenarios 3 and 4

The final two scenarios test the limits of the RNN model by training on only a single well (one sequence) in each case. The single wells used are Well #1 and Well #2 for Scenarios 3 and 4, respectively (see Figure 6.1c for the location of these two wells). One of the difficulties with Scenarios 3 and 4 is that they have only one sequence available for training, i.e., there are no validation sequences. This means that the RandomSearch class from Keras Tuner cannot be used to determine hyper-parameter settings because this requires more than one sequence. Another obvious consequence is that there are no validation curves, only training curves. Therefore, training the RNNs and choosing appropriate values for the hyper-parameters requires caution. The best way that I find to train the RNNs in these two instances is a trial-and-error-based approach, where I select values for hyper-parameter settings (e.g., RNN units, dropout value, epochs) and compare the single sequence predictions to the true labels in a well-view. Similar to Scenario 2, the sample weights are essential in these two scenarios (the weight values are the same as those from Scenario 2); if the sample weights are not included here, the over-prediction of shale is even more pronounced than what is seen in Scenario 2. I determine that 36 RNN-LSTM units, 0.5 for the dropout rate, and 80 epochs work well for both scenarios and all input data conditions. Table 6.2 provides the results, and while they were not particularly useful for Scenarios 3 and 4, the loss and accuracy curves are provided in Figure 6.4.

Table 6.2: Results of the RNN performance for Scenarios 3 and 4. Each scenario has three different input data situations as stated in the text and these are denoted with (a), (b), and (c). Since both of these scenarios only have one sequence available for training, there are no validation metrics. The final column provides performance metrics of the RNN model when applied to all sequences (i.e., training and testing combined); see the key at the bottom for what each value represents. *Denotes de-trended versions of the seismic inversion attributes.

Input data		**RNN hyperparameters**	**Training accuracy**	**Validation accuracy**	**Testing accuracy**	**Global performance metrics**
Scenario 3: Train (0.2%) **Well #1**, validate (N/A), test (99.8%) – 1 training well						
(3a)	Angle stacks	RNN units = 36 Dropout = 0.5	83.92	-	83.66	59.93, 55.41, **55.95** 90.81, 35.34, 41.70
(3b)	Angle stacks, AI, SI, Rho	RNN units = 36 Dropout = 0.5	82.96	-	82.75	57.52, 61.56, **55.64** 90.31, 34.19, 42.42
(3c)	Angle stacks, *AI, *SI, *Rho	RNN units = 36 Dropout = 0.5	87.54	-	84.12	62.30, 69.13, **64.20** 90.92, 46.50, 55.19
Scenario 4: Train (0.2%) **Well #2**, validate (N/A), test (99.8%) – 1 training well						
(4a)	Angle stacks	RNN units = 36 Dropout = 0.5	88.71	-	82.62	57.00, 49.96, **52.06** 90.31, 29.70, 36.16
(4b)	Angle stacks, AI, SI, Rho	RNN units = 36 Dropout = 0.5	89.35	-	82.38	56.93, 55.08, **55.07** 90.06, 33.70, 41.45
(4c)	Angle stacks, *AI, *SI, *Rho	RNN units = 36 Dropout = 0.5	88.07	-	82.79	61.62, 63.88, **62.56** 89.91, 45.24, 52.53

KEY	Macro averages for: Precision, Recall, **F1** Individual F1 for: Shale, Wet sand, HC sand

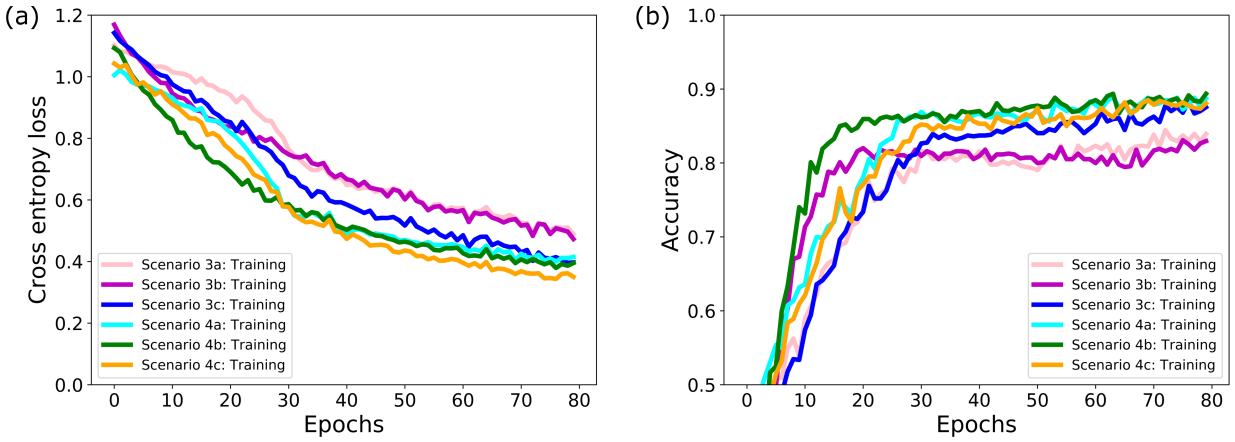

Figure 6.4: The (a) categorical cross-entropy loss and (b) accuracy curves for Scenarios 3 and 4. The performance of both of these scenarios is depicted in Table 6.2.

6.5 Discussion

6.5.1 Impact of seismic attributes on performance

The results presented here have a few noteworthy takeaways. Not surprisingly, Scenario 1 can use a more complicated model (LSTM units = 128) with less regularization (dropout = 0.1) because more training data are used. Scenarios 2-4 have much less training data, so these models require fewer LSTM units and more regularization to help prevent overfitting. In the Scenario 1 case, the performance is quite good overall without any notable differences between the predictions from each input situation (see Scenario 1 panels in Figure 6.5). Including the three seismic attributes (row 1b in Table 6.1) only gives a benefit of $< 1\%$ in macro precision, recall, and F1, and the benefit is only 1-2% in the de-trended seismic attribute case (row 1c in Table 6.1). In this circumstance, the effort required to produce the prestack seismic inversion attributes may outweigh the potential benefits.

However, different conclusions can be drawn from the more realistic training data scenarios. In Scenario 2, there is still only a minor improvement using the seismic attributes compared to just using the angle stacks (compare rows 2a and 2b in Table 6.1), but an 8% improvement in macro F1 is seen when the inputs include the de-trended attributes (row 2c in Table 6.1). The predictions in Figure 6.5 demonstrate this as well because including the de-trended seismic attributes (Figure 6.5g) has far fewer HC sand false positives compared to the other two input data scenarios. Similar observations are also seen in Scenarios 3 and 4. Including the seismic attributes continues only to provide marginal improvements in the performance, but the impact of the de-trended attributes is rather substantial. The results in Table 6.2 indicate a 9-10% improvement in macro-F1 and 10-15% performance in the individual F1 scores for the wet and HC sands when including the de-trended attributes as inputs. These improvements in the wet and HC sands are easily seen in Figure 6.6; the predictions including the de-trended seismic attributes capture far more detail than using just the angle stacks or angle stacks + seismic attributes.

These results indicate that seismic attributes can improve RNN performance when training data are limited. Overall, the seismic inversion attributes provide marginal improvements, but the de-trended versions of these attributes improve the performance quite significantly. In limited training data situations, it has also been shown that data augmentation can improve deep learning performance (Krizhevsky et al., 2012; Shi et al., 2019). Data augmentation is a trivial task for CNN-based applications (e.g., rotating, flipping, and distorting training images), but the same cannot be said for applications where the labels are derived, for instance, from well data. These results indicate that including engineered features may have a similar performance-boosting effect to data augmentation, and engineered features can be an alternative when data augmentation proves challenging.

6.5.2 Comparison of Scenarios 3 and 4 to Chapter 5 results

The purpose of training RNN models with Well #1 and Well #2 (see Figure 6.1c) is to allow a comparison of these results with those from Chapter 5. For the Well #1 case, compare the final two rows of Table 5.1 and row 3c of Table 6.2. Here, the self-training LP macro F1 score is still over 4-5% higher than the macro F1 score for the RNN model. Interestingly, the XGBoost performance is comparable to the RNN performance in this case. For the Well #2 case, compare the final two rows of Table 5.2 and row 4c of Table 6.2. The self-training LP macro F1 score is 6-8% higher than the macro F1 score for the RNN model. The difference in this case is that the RNN model performs slightly better than XGBoost. While the comparisons between Chapter 5 and this work are not necessarily apples-to-apples because the six angle stacks are used as inputs here when they are not in Chapter 5, these comparisons still provide valuable insights. The results indicate that a deep learning technique can still perform well in limited training data scenarios, but the performance is not any better or worse compared to non-deep learning algorithms, such as XGBoost. Furthermore, the semisupervised approach still achieves better performance for this seismic classification example than a supervised deep learning algorithm, such as recurrent neural networks.

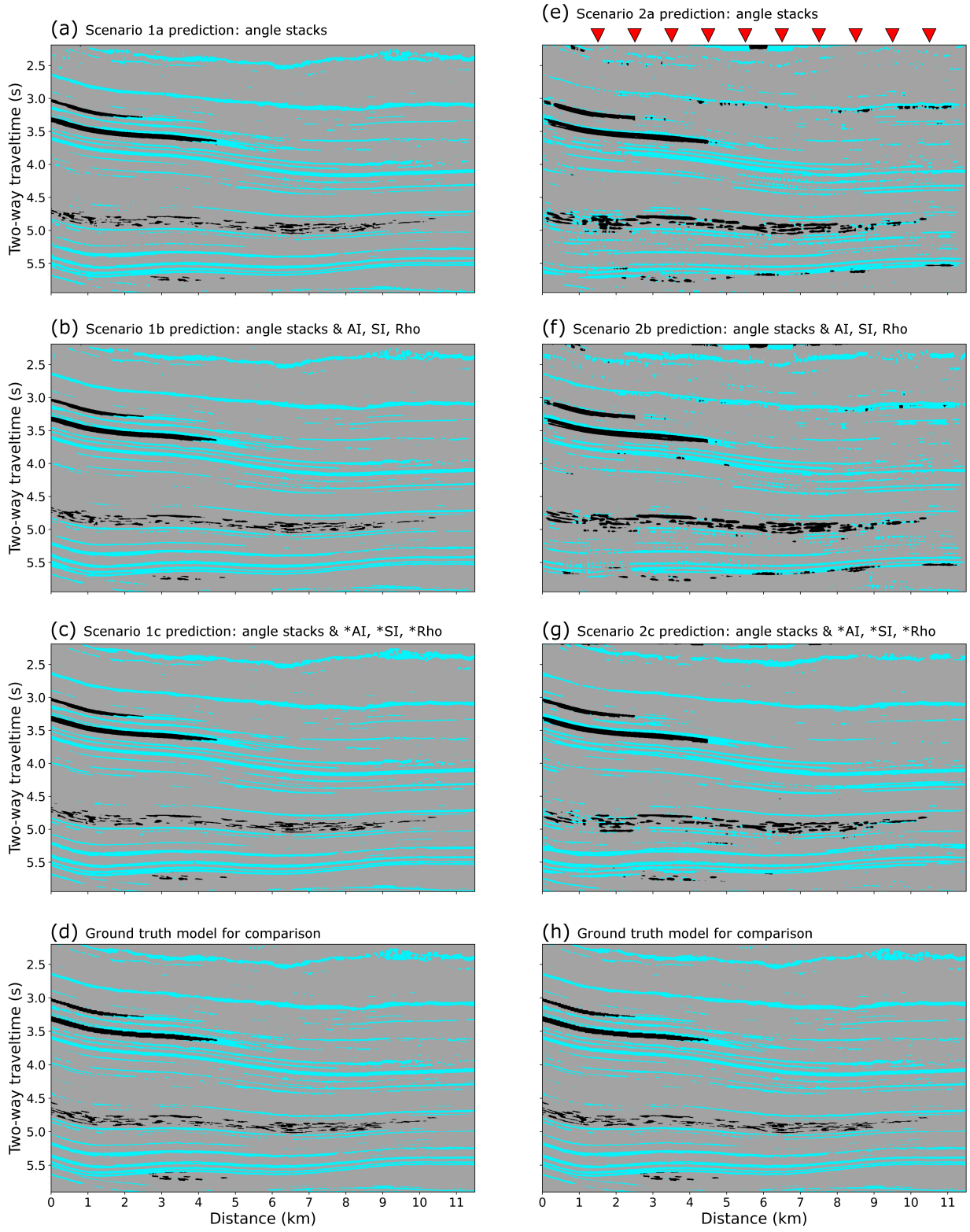

Figure 6.5: The (a, b, c) Scenario 1 and (e, f, g) Scenario 2 global predictions for each of the three input scenarios. (e) The red triangles denote the locations of the ten wells used for training. Panels (d, h) provide the ground truth model as a reference to facilitate comparisons to the predictions.

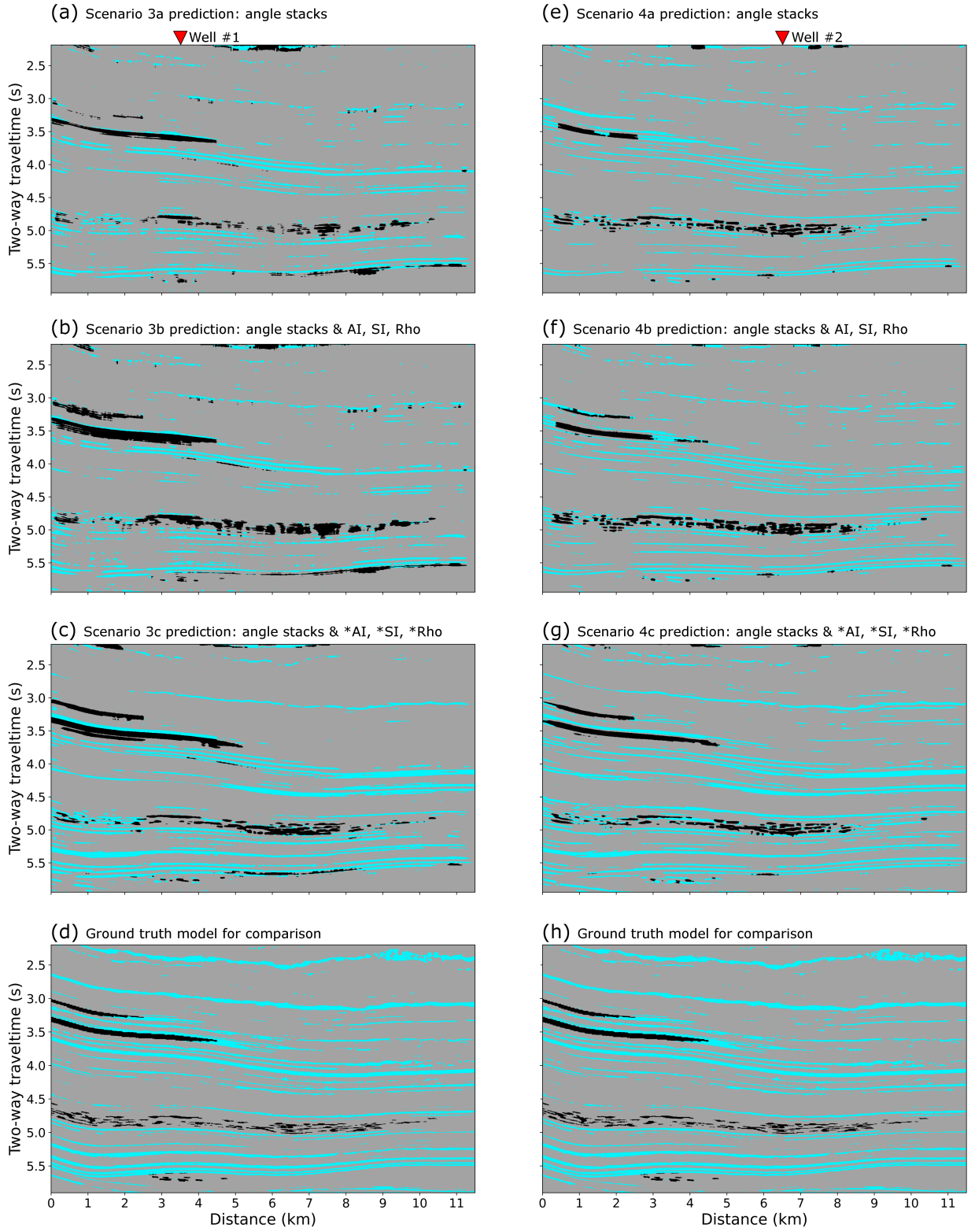

Figure 6.6: The (a, b, c) Scenario 3 and (e, f, g) Scenario 4 global predictions for each of the three input scenarios. The red triangles denote the locations of Well #1 and Well #2 used for training Scenarios 3 and 4, respectively. Panels (d, h) provide the ground truth model as a reference to facilitate comparisons to the predictions.

6.6 Conclusions

Recurrent neural networks (RNNs) have the advantage of being able to operate directly on the seismic data, but I explore if the inclusion of engineered features (i.e., seismic attributes) can improve the performance even further using a synthetic seismic classification problem. The results suggest that including seismic attributes (with background trends removed) can still improve the RNN performance, but the most significant impact is observed when the training data are limited. Many researchers have spent decades developing seismic attributes to help extract or illuminate certain features of seismic data, and perhaps there will continue to be a space for attributes in the realm of deep learning.

Chapter 7

Predictive lithology mapping using semisupervised learning: practical insights using a case study from New South Wales, Australia[1]

7.1 Introduction

The ability to produce a geologic map is an essential component of the early stages of mineral exploration. Geologic maps can inform decisions for targeting mineral deposits, such as prioritizing specific areas for more detailed mapping and sampling. Lithologic mapping generally consists of identifying outcrops, then determining the rock lithology present at these outcrop locations, and finally inferring the distribution and spatial relationships of these lithological units in regions where the bedrock is unknown or situated below cover (e.g., vegetated regions, etc.). Outcrop locations may be limited due to cost or logistical reasons,

[1]This manuscript is currently under review in the journal *Geophysics*.

so extrapolating or interpolating lithologic units to the unknown regions can be challenging. A source of information that can assist with the inference of lithologic units is remote sensing data, such as geophysics or satellite imagery. However, this manual construction of lithologic maps contains significant bias, as different interpreters can produce different maps from the same data. It is also difficult to manually reconcile the relationships between several remote sensing layers. This challenge has been addressed in recent years as geologic mapping has evolved from a purely qualitative discipline to a more hybrid approach involving quantitative, data-driven techniques such as machine learning. Machine learning-assisted lithologic mapping utilizes the same information as the traditional, manual approach, but machine learning can mitigate the source of bias discussed above and accelerate the time required to produce a product.

There are primarily two types of machine learning applied in the context of bedrock-lithology mapping. The first is unsupervised learning, which clusters the input remote sensing data without including any lithology information. One of the earliest applications involves k-means clustering of radiometric data (Pirkle et al., 1984). Other unsupervised algorithms used for geologic mapping are ISOCLUSTER (Anderson-Mayes, 2002), which is an extension of k-means, agglomerative clustering (Martelet et al., 2006), and fuzzy c-means clustering (Paasche & Eberle, 2009; Eberle & Paasche, 2012). One of the more popular algorithms is self-organizing maps (SOMs, Carneiro et al., 2012; Cracknell et al., 2014; Kuhn et al., 2019; Carter-McAuslan & Farquharson, 2021; Wu et al., 2021), and a recent study uses Gaussian mixture models (Weihermann et al., 2021). Unsupervised methods are useful approaches for extracting valuable information from remote sensing data in the absence of lithology calibration. However, when we have measurements of lithology (e.g., observations made from outcrops, grab samples, or core from drill holes), we can use this information directly to calibrate the machine learning process through a different approach.

Supervised learning methods learn a relationship between the co-located remote sensing observations and training targets (e.g., determined lithologies from outcrops) in order to make predictions for where the lithology is unknown. Some of the earlier applications utilize neural networks (Gong, 1996; Leverington,

2010), maximum likelihood classifiers (Kettles et al., 2000), minimum distance classifiers (Chen et al., 2007), and support vector machines (Waske et al., 2009; Yu et al., 2012; Cracknell & Reading, 2013, 2014). However, the types of methods that have become the de-facto standard in the last decade are ensemble-based techniques. Harris et al. (2012) and Behnia et al. (2012) combine bootstrap aggregation (bagging; Breiman, 1996) with a maximum likelihood classifier, and many researchers use random forests (Cracknell et al., 2014; Harris & Grunsky, 2015; Radford et al., 2018; Kuhn et al., 2018, 2019; Costa et al., 2019; Hood et al., 2019; Kuhn et al., 2020). The broader trend across machine learning applications as a whole is deep learning. While deep learning approaches have not yet become commonplace in bedrock-lithology mapping applications, some researchers are beginning to show that they can be effective (e.g., Wang et al., 2021).

One of the challenges to supervised learning for bedrock-lithology mapping is securing enough labeled data to train the algorithms. Other disciplines have a plethora of labeled data that can be used for machine learning problems, such as computer vision, but the same cannot be said for some geoscience-related problems. Labeled data for our applications are generally obtained by sampling the Earth in some manner, which is not trivial for cost or logistical reasons. In this context of predictive bedrock-lithology mapping, we need measurements of lithology from the ground, i.e., the bedrock. Regions with mineral exploration interest may be isolated and remote, limiting fieldwork or making fieldwork more expensive. Lithologic measurements can be determined through other means, such as geochemical analysis of grab samples or examination of core samples from drilling; however, these options are also quite costly. As a result, these problems generally contain limited amounts of labeled data. Consequently, supervised machine learning methods may be prone to a phenomenon called overfitting. This behavior occurs when a machine learning model learns from the fine detail and the noise in the training data rather than the general trends; this leads to the model performing suspiciously high on the training data, and likely performing poorly on data that the model has not seen (e.g., the testing data). Many supervised machine learning models can mitigate this overfitting phenomenon by including hyper-parameters that impose a penalty on model complexity, such as

the C hyper-parameter for support vector machines (Cortes & Vapnik, 1995). However, these measures may still be insufficient if the labeled data are severely limited.

A different approach for these limited training data situations is to use semisupervised machine learning techniques, which are a hybrid of unsupervised and supervised learning. Semisupervised algorithms train using both the labeled (e.g., co-located remote sensing and lithology data) and unlabeled (e.g., remote sensing data with unknown lithology) data, which can improve the predictions for the unlabeled data compared to supervised methods when the labeled data are scarce (Chapelle et al., 2006; Zhu & Goldberg, 2009; van Engelen & Hoos, 2020). Some geoscience-based machine learning applications have demonstrated the benefits of semisupervised learning, such as land cover classification of satellite remote sensing data (Vatsavai et al., 2005; Camps-Valls et al., 2007; Gomez-Chova et al., 2008b; Liu et al., 2013; Cui et al., 2018). Land cover and bedrock-lithology classification problems are quite similar in that they both utilize forms of remote sensing data, with the main distinction being the labels (i.e., rock types versus land cover). Given these similarities, it is natural to think that if semisupervised methods are effective for land cover classification, the same could be true for bedrock-lithology classification. However, there are no semisupervised applications to bedrock-lithology mapping in the literature to the best of my knowledge, so these ideas remain largely unexplored.

I investigate the effectiveness of semisupervised methods for bedrock-lithology mapping, specifically label propagation, using a suite of data from a region in New South Wales, Australia. Airborne radiometric and magnetic data are the input data available for this problem, and I use established feature expansion methods to generate additional inputs. A well-curated and reliable geologic map from the area provides the lithology labels for this study, allowing me to evaluate the predictions quantitatively. With these data, I subsample the geologic map in three different ways to simulate realistic training data scenarios for ML-assisted bedrock mapping. The primary motivation of this study is to determine if semisupervised methods can outperform supervised methods under typical limited training data situations. To achieve this, for each

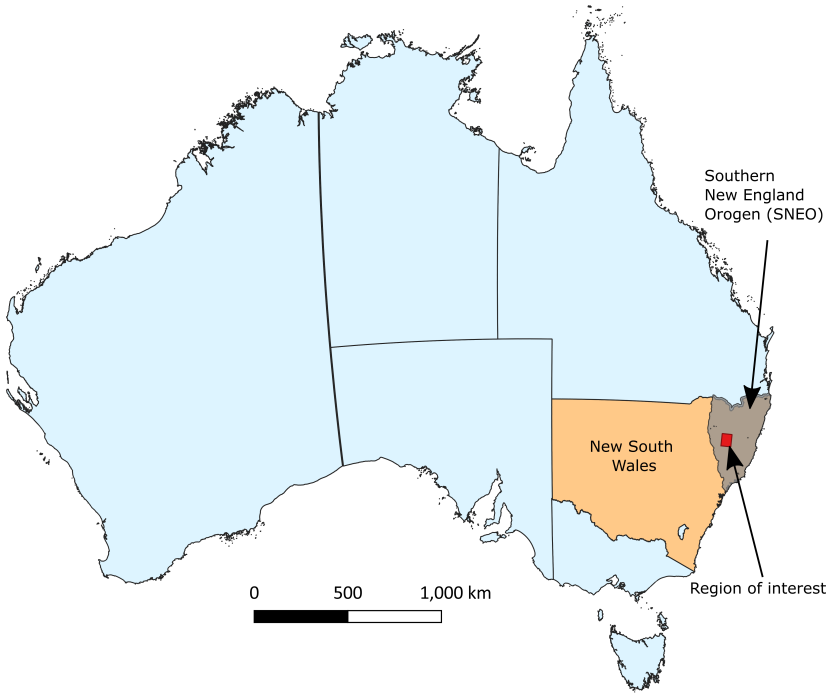

Figure 7.1: The study area (red box) relative to the continent of Australia, the state of New South Wales (NSW, orange), and the Southern New England Orogen (SNEO, gray).

of the three scenarios, I compare the label propagation (LP) performance to LightGBM and XGBoost. Furthermore, I perform unsupervised clustering of the input features using a SOM-based approach to produce a cluster map. The cluster map is used to assess whether unsupervised learning can offer additional insights when training data are still available.

7.2 Study area

My general area of focus for a bedrock-lithology classification study is in Australia due to the high-quality datasets that are publicly available. Specifically, the Geological Survey of New South Wales (GSNSW) has published a seamless GIS compilation of the best available geology data for NSW, and these data are organized into a series of layers representing each of the major lithotectonic units (Colquhoun et al., 2021). The specific region of interest that I have chosen to concentrate on is a 56×70 km area in the Southern New England Orogen (SNEO, see Figure 7.1). This area is selected because the bedrock is exposed at the surface

with little Cenozoic sedimentary cover, and there is good airborne geophysical data coverage (Chapter 7.3 below).

In this region, the geologic map contains 58 lithologic units. However, upon inspection of this map, many units share nearly identical descriptions, but are still designated as distinct units. This phenomenon is not uncommon in field mapping where different geologists in adjacent areas may describe a unit slightly differently, such that a single unit ends up being decomposed into multiple units (e.g., one individual calls it Unit A and another calls it Unit B, when both should be Unit A). I carefully examine the descriptions, spatial locations, and geologic age of all 58 lithologies and merge several units to limit redundancies and help simplify the map. The need for simplifying geologic maps for machine learning purposes is recognized in the literature (Brown et al., 2000; Porwal et al., 2003; Pereira Leite & de Souza Filho, 2009; Leverington, 2010; Yu et al., 2012; Harris et al., 2014; Kuhn et al., 2020).

The final map (Figure 7.2) has 21 lithologic units, and their associated descriptions are provided in Table 7.1. The lithologic units comprising this region of interest of the SNEO were deposited during four geologic periods. The oldest rocks in this region are the meta-sedimentary units which accumulated in an accretionary wedge of a forearc basin setting during the Late Silurian to Carboniferous. After this cycle, there was widespread felsic magmatism from the Early Permian to the Early Triassic. The Permian-aged felsic intrusives, except the Wandsworth Volcanic Group (Unit 12), are part of the Bundarra Supersuite in the west-northwest portion of the map. The Triassic-aged intrusives, except for the Fox Tor Quartz Diorite (Unit 11), are part of two different supergroups: the Moobi Supersuite to the south-southwest and the Uralla Supersuite to the east-northeast. Lastly, the youngest rocks in this region are Cenozoic-aged extrusive volcanics (basalt). For a more in-depth discussion of the geology of this region and the SNEO, I refer the interested reader to other literature (Leitch, 1974; Roberts & Engel, 1987; Ford et al., 2019; Jessop et al., 2019).

Table 7.1: Detailed descriptions of each of the geologic units present on the map in Figure 7.2 (descriptions taken from Colquhoun et al. 2021).

ID	Unit name	Description
1	Maybole Volcanics	Alkali olivine basalts with minor volcaniclastic and epiclastic units. Rare plugs, dykes and sills. Some interbedded non-volcanogenic sediments.
2	Gwydir River Monzogranite	Medium- to coarse-grained, porphyritic to equigranular biotite-hornblende monzogranite to granodiorite. Minor leucomonzogranite to leucosyenogranite.
3	Yarrowyck Granodiorite	Weakly zoned, from medium-grained, biotite-hornblende granodiorite to fine-grained, leucomonzogranite and leucosyenogranite
4	Uralla Granodiorite	Grey, coarse-grained, approximately equigranular, biotite-hornblende-(pyroxene) granodiorite to tonalite, with minor monzogranite; finer-grained, porphyritic marginal variant.
5	Terrible Vale Microgranodiorite	Texturally heterogeneous sequence with medium-grained equigranular quartz-poor granodiorite to quartz monzodiorite, with two-pyroxene dacite-tuffasite
6	Shalimar Granodiorite	Grey, fine to medium-grained, porphyritic, hornblende-biotite-quartz monzodiorite to quartz-poor granodiorite, with minor monzogranite.
7	Campbells Hill Monzogranite	Light grey, medium-grained, texturally heterogeneous, felsic biotite plagioclase-rich monzogranite to granodiorite, characterised by sparse K-feldspar megacrysts
8	Walcha Road Monzogranite	Zoned, variably porphyritic, hornblende-biotite monzogranite, and minor granodiorite.
9	Limbri Monzogranite	Medium-grained, inequigranular to equigranular, leucocratic, biotite monzogranite.
10	Moonbi Monzogranite	Weakly zoned, medium- to coarse-grained, coarsely porphyritic, hornblende-biotite monzogranite to granodiorite, minor leucomonzogranite
11	Fox Tor Quartz Diorite	Grey, medium- to coarse-grained, equigranular to slightly porphyritic, pyroxene-hornblende-biotite quartz diorite
12	Wandsworth Volcanic Group	Undifferentiated felsic volcanic rocks, minor sedimentary rocks and granite. Dominantly ignimbritic rhyolite, rhyodacite and dark crystal-lithic tuff
13	Balala Granodiorite	Texturally heterogeneous, fine- to medium-grained, equigranular, hornblende-biotite granodiorite, with minor monzogranite.
14	Rocky Glen Monzogranite	Coarse-grained biotite monzogranite; possibly minor syenogranite.
15	Namoi Tops Monzogranite	Medium-coarse grained biotite monzogranite, minor leucosyenogranite and leucomonzogranite
16	Glenclair Syenogranite	Medium-coarse-grained, porphyritic biotite-muscovite syenogranite to leucosyenogranite with lesser monzogranite-leucomonzogranite.
17	Pringles Monzogranite	Coarse-grained porphyritic to even-grained K-feldspar-rich biotite monzogranite and leucocratic syenogranite.
18	Banalasta Monzogranite	Blue-grey to buff, coarse- to very coarse-grained, strongly porphyritic biotite monzogranite.
19	Whitlow Formation	Multiply deformed, thickly bedded feldspathic- and volcanic-lithic wacke, interbedded siltstone, fine wacke and minor conglomerate. Low-grade regional metamorphism.
20	Sandon beds	Low-grade, regionally metamorphosed, multiply deformed lithic wacke, paraconglomerate, siltstone, mudstone, minor chert, spilite.
21	Sandon Association	Lithic metamorphic wacke, slate, phyllite, chert, amphibolite, metabasalt. Greywacke, sandstone, siltstone, mudstone and paraconglomerate.

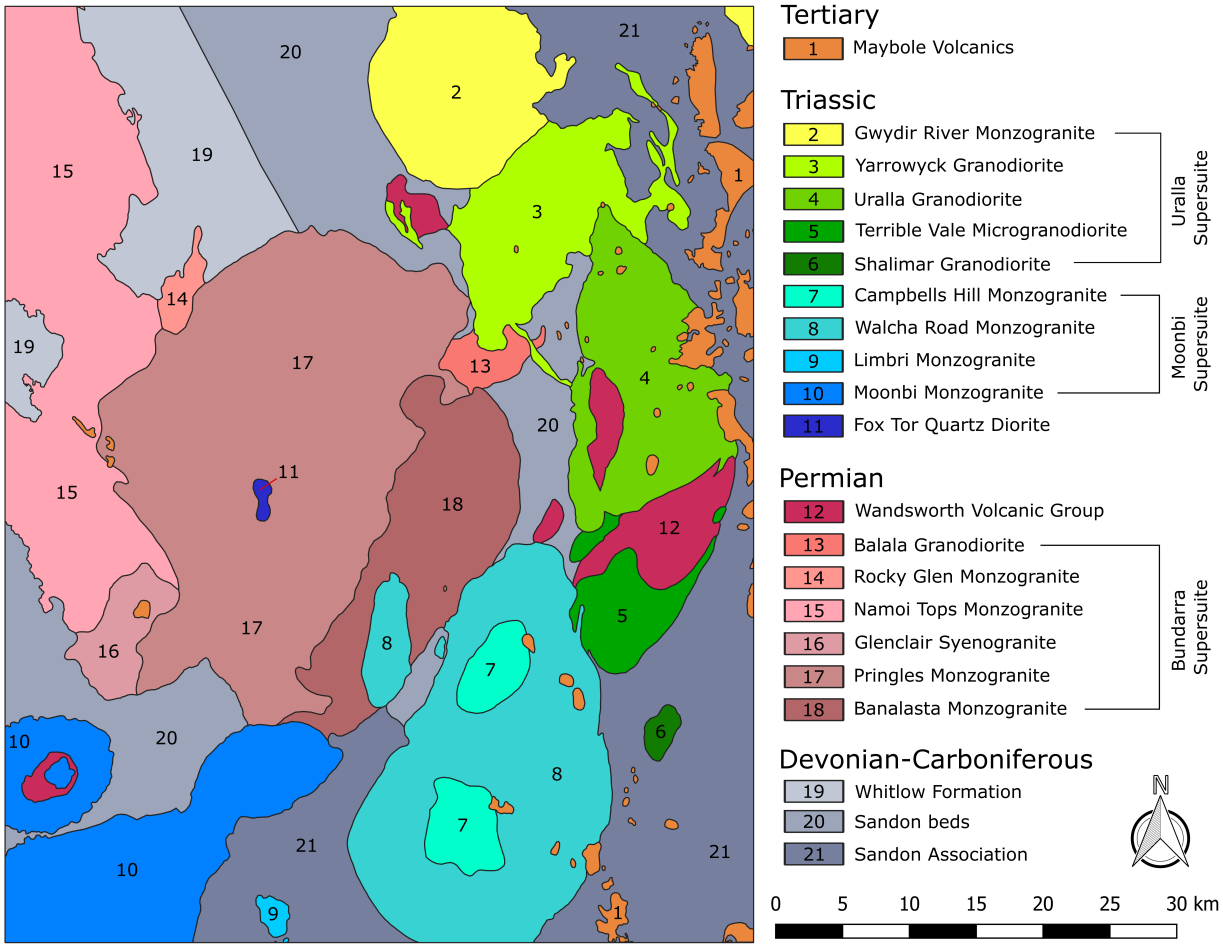

Figure 7.2: The geologic map of the 56 × 70 km region of interest in the SNEO (modified from Colquhoun et al. 2021). The associated descriptions of each unit are provided in Table 7.1.

7.3 Data

An airborne geophysical dataset containing three radiometric channels (Potassium – K %, equivalent Thorium – eTh ppm, and equivalent Uranium – eU ppm) and reduced-to-pole (RTP) magnetic data covers the 56×70 km region of interest. These data were collected in 2001 by Tesla Geophysics for the NSW Department of Mineral Resources and are made publicly available by the GSNSW on their MinView portal (https://minview.geoscience.nsw.gov.au/); this dataset is called Southern Peel. The data were collected using 250 m line spacing and are gridded to raster layers with 100 m cells using bi-cubic spline interpolation. I also considered using satellite remote sensing data (e.g., ASTER, Sentinel-2), but analysis of the normalized difference vegetation index (NDVI) indicated that a significant portion of the region is vegetated; therefore, these data are not included. As a result, there are only four input features for this machine learning problem. A limited number of features can lead to poor classification performance because there may not be enough features to discriminate the classes of interest adequately. Nonetheless, I can compute additional input features from the original inputs (i.e., feature expansion) using techniques established in geophysics and remote sensing literature.

7.3.1 Feature engineering

A common approach for expanding radiometric features is to compute ratios of the original channels, such as eTh/K, eU/K, and eU/eTh (Cracknell & Reading, 2014; Harris & Grunsky, 2015; Carter-McAuslan & Farquharson, 2021). The eU channel is quite noisy for the Southern Peel dataset, and any ratios computed from eU are also noise-prone. I elect to keep the eU channel, but do not use the eU/K or eU/eTh ratios. However, I do use eTh/K, but I take the natural logarithm of this ratio because doing so better differentiates the signals between the intrusive units. It is also common to visualize the K-eTh-eU channels as a red-green-blue (RGB) ternary image, but the only benefit that this RGB raster can provide is visualization; I cannot use this RGB image as an input feature for any of the machine learning algorithms that I consider in this

work. My way around this is to convert the RGB ternary image to a single-channel grayscale raster; this is achieved by computing a linear combination of the K-eTh-eU channels via η_KK + η_{Th}eTh + η_UeU, where η_K, η_{Th}, and η_U are the weights given to each of the radiometrics channels and they all sum to 1. Weighting each of the channels equally (0.33 for each weight) gives too much weight to the eU channel and introduces noise into the grayscale image. I experiment with a few different weight settings, and find (η_K, η_{Th}, η_U) = (0.6, 0.3, 0.1) gives good results (see Feature 12 in Table 7.2, and Figure 7.3e).

There are several techniques for potential field methods to expand the number of features such as spatial derivatives, upward continuation, residuals, and more. I compute three additional features from the RTP magnetics: upward continuation, a residual, and a pseudogravity residual. Upward continuation simulates the RTP signal at a higher elevation (akin to a low-pass filter), and I use a height of 500 m. I compute a residual feature by subtracting an upward continuation response from the original RTP signal; this removes longer wavelength features (akin to a high-pass filter), and I use a height of 1000 m. The other feature that I compute is pseudogravity (Baranov, 1957; Blakely, 1995), a signal that approximates the gravity data response based on magnetics data. To better accentuate the signals in the pseudogravity, I compute its residual after subtracting an upward continued field with a height of 100 m.

A common magnetics feature in the mineral exploration industry is the first vertical derivative (1VD), which is helpful for mapping structure, complexity, and texture. However, recent publications (Kuhn et al., 2019, 2020) show that the 1VD may not be diagnostic of lithology for pixel-wise machine learning applications. These authors conclude that magnetic lineaments highlighted in the 1VD are likely at a smaller scale than the lithological domains. Recognizing this, I posit that features that can quantify the behavior of the 1VD in a local neighborhood of a given pixel may be more useful for this application. Rather than include the RTP 1VD directly, I elect to quantify its textural information using gray-level co-occurrence matrix (GLCM) features (Haralick et al., 1973). This process first requires converting the input data to gray levels (i.e., quantized to a set of integers). Computing GLCM textures is a sliding window approach where

a GLCM is computed for a given window (e.g., 9×9), and texture attributes are calculated from the GLCM and assigned to the cell in the center of the window (see Hall-Beyer, 2017a, for details). The GLCM features that I choose to compute from the RTP 1VD are the energy, difference entropy, and inverse difference moment (a.k.a. homogeneity) with a window size of 7×7. The size of the window impacts how crisp (small) or blurry (large) the resulting texture attributes become, and I find 7×7 windows to be a good balance for this problem.

The use of GLCM texture attributes applied to satellite imagery is well-documented. For example, Hall-Beyer (2017b) computes GLCM textures from Landsat data for a land-cover classification problem, and Radford et al. (2018) use GLCM textures from radar imagery to serve as inputs to a lithology classification problem. Conversely, computing texture attributes from geophysical data is less established. Yu et al. (2012) compute neighborhood statistics (mean and variance) from apparent susceptibility data. Another example uses a GLCM attribute (entropy) computed from RTP magnetics data in a workflow involving other filters to locate prospective gold deposits (Holden et al., 2008). Therefore, the potential of using GLCM texture attributes computed from geophysical data is not fully realized in the context of machine learning problems, and I explore their efficacy in this study.

7.3.2 Bedrock lithology classification

The radiometric, magnetic, and magnetic texture features form the inputs for my machine learning problem (12 total, see Table 7.2). A subset of these input features is provided in Figure 7.3. Each of these features is projected to the Geocentric Datum of Australia (GDA) Map Grid of Australia (MGA) zone 56 (EPSG = 28356) with coincident 100-m cells. Thereby, each input feature has 560 x 700 cells, which amounts to 392,000 total cells. The associated targets for this ML problem are the categorical data represented as the lithologies from Figure 7.2. Geologic maps are commonly digitally represented as a series of polygons for each lithology in a vector data format such as a shapefile (.shp). To use these data for this problem, they

are converted to a raster using a process called rasterization, and the resulting raster has 100-m cells aligned with the input features. As a result, each pixel can be thought of as a 12-dimensional vector containing a value from each of the input features, with its associated target being an integer representing one of the 21 possible lithologies. With these data, I conduct a series of machine learning experiments to reflect various input conditions representative of realistic bedrock mapping problems. In each experiment, I assume that the a priori knowledge of lithology is limited to a discrete number of locations, and these locations vary with each experiment.

The first scenario (Scenario A) reflects an early exploration stage where a field mapping campaign is conducted along transects. Here, field geologists walk along a particular bearing and record the observed lithology along the traverse. For this situation, there are four transects: two are oriented north-south, and two are oriented east-west. However, these observations are not made continuously along each transect because obstructions (brush, dense vegetation, roads, etc.) can hinder observations from being made. I capture this phenomenon by randomly decimating lithology observations along these four transects, which produces a scenario with 1258 total training data (see Figure 7.4a). It is noteworthy that five of the smaller lithologic units on the map (Classes 6, 9, 11, 13, 14) are not sampled in these training data.

The second scenario (Scenario B) also reflects an early exploration stage, but in this instance, reconnaissance field mapping is performed at 70 outcrop locations distributed across the map. At each of these outcrop locations, the field geologist indicates that they are confident that the mapped lithology remains consistent within 250-m of the field station. I represent this with a circular polygon (with a radius of 250-m) around each station, which results in 1371 training data upon rasterizing the polygons (see Figure 7.4b). Note that each lithology contains at least one station, and each station is on average 6 km from another.

The third situation (Scenario C) represents a mature exploration stage that utilizes more extensive geological information. Here, I treat the labeled data as representing lithologies determined from grab samples or the first intercepted lithology from diamond drill hole (DDH) samples. These sampling locations are spa-

Table 7.2: The 12 features used as inputs for the bedrock-lithology machine learning problem. Those indicated with an * are the original four inputs, and the rest are determined via feature expansion methods. The features indicated with an ** are shown in Figure 7.3. The Number column is used as a short-hand to refer to particular features in Table 7.4. The Abbreviation column denotes the names that are used to refer to certain features in subsequent plots.

Number	Feature description	Abbreviation
1*	Magnetics reduction to pole	**Magnetics RTP
2	Magnetics RTP upward continuation with 500 m	Magnetics RTP upcont
3	Magnetics RTP pseudogravity residual with 100 m	**Magnetics RTP pseudogravity
4	Magnetics RTP residual with 1000 m	Magnetics RTP residual
5	Difference entropy texture from magnetics RTP 1st vertical derivative	Texture difference entropy
6	Energy texture from magnetics RTP 1st vertical derivative	Texture energy
7	Homogeneity texture from magnetics RTP 1st vertical derivative	**Texture homogeneity
8*	Radiometrics potassium channel (in %)	Radiometrics (K)
9*	Radiometrics thorium channel (in ppm)	Radiometrics (Th)
10*	Radiometrics uranium channel (in ppm)	Radiometrics (U)
11	Radiometrics ratio log(Th/K)	**Radiometrics ratio
12	Radiometrics RGB ternary image converted to single-channel grayscale	**Radiometrics composite

tially distributed across the map, and there are 1250 points in total (see Figure 7.4c). The main distinction between this situation and the second scenario is that the training data in the second scenario are situated in isolated clusters, whereas here, those training data are more evenly dispersed across the map.

These subsampled lithology pixels and their co-located feature vectors form the labeled data for machine learning, and the remaining feature vectors are the unlabeled data for which I predict the lithology labels. The goal of each of these scenarios is to produce geologic maps from limited direct observations and to compare the performance of supervised to semisupervised algorithms. A benefit of this study is that I have a high-quality geologic map against which I can evaluate the machine learning predictions. While this is a useful means for evaluating the machine learning performance, such a polished geologic map may not exist in a realistic exploration scenario. In such instances, it would still be beneficial to have something to use for qualitative comparisons. This introduces the fourth situation (Scenario D) where I take advantage of unsupervised learning, which clusters the input data into its natural groups independently of any lithological constraints. I qualitatively compare cluster results to the predictions from supervised and semisupervised learning to determine if incorporating the limited training data helps the machine learning recognize detail that the clustering cannot.

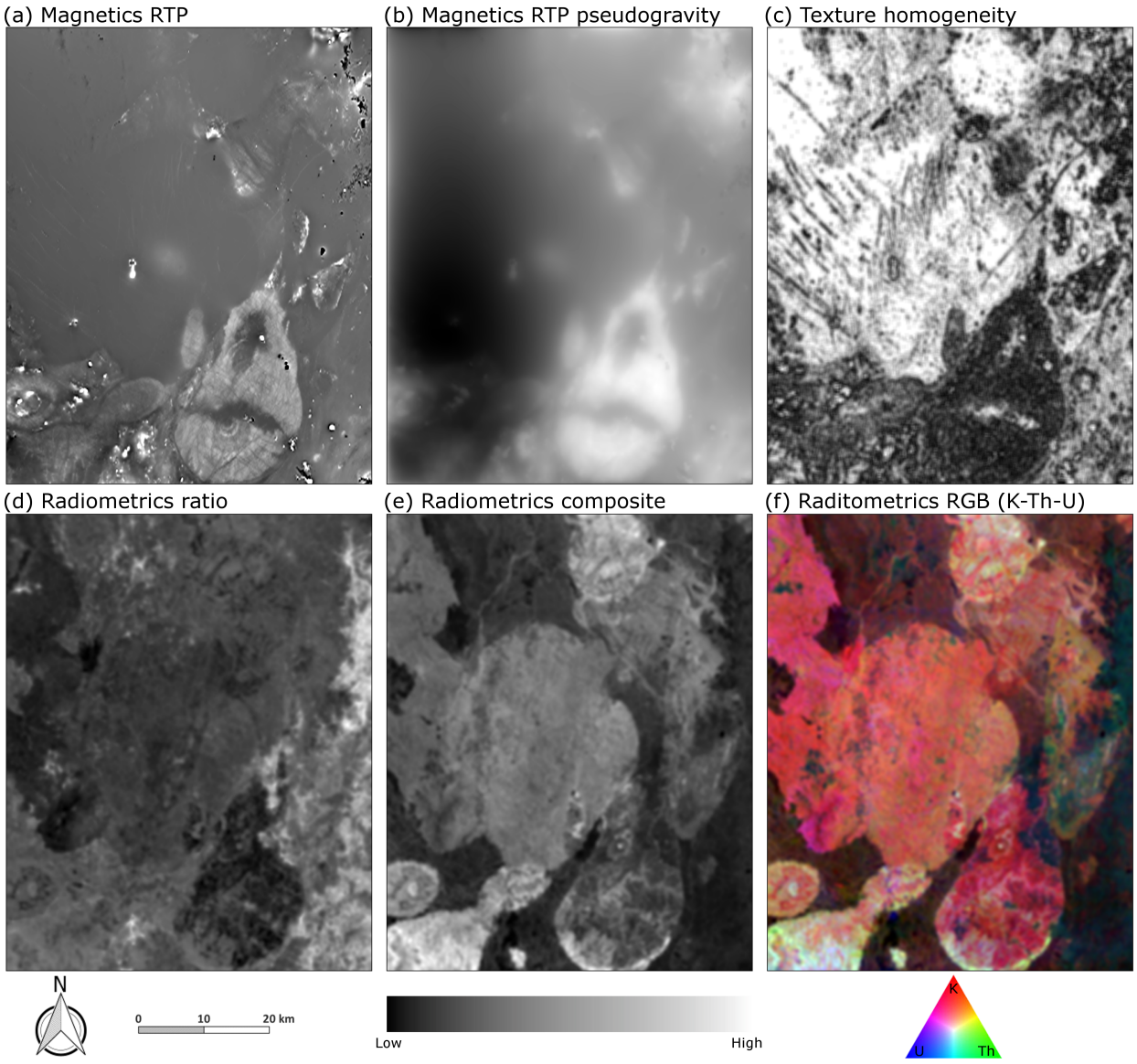

Figure 7.3: A subset of the input features with 100-m cell sizes. Panels a-e share the same relative grayscale colorbar. Panel (f) is an RGB composite of the three radiometric channels with K = red, Th = green, and U = blue (see bottom right).

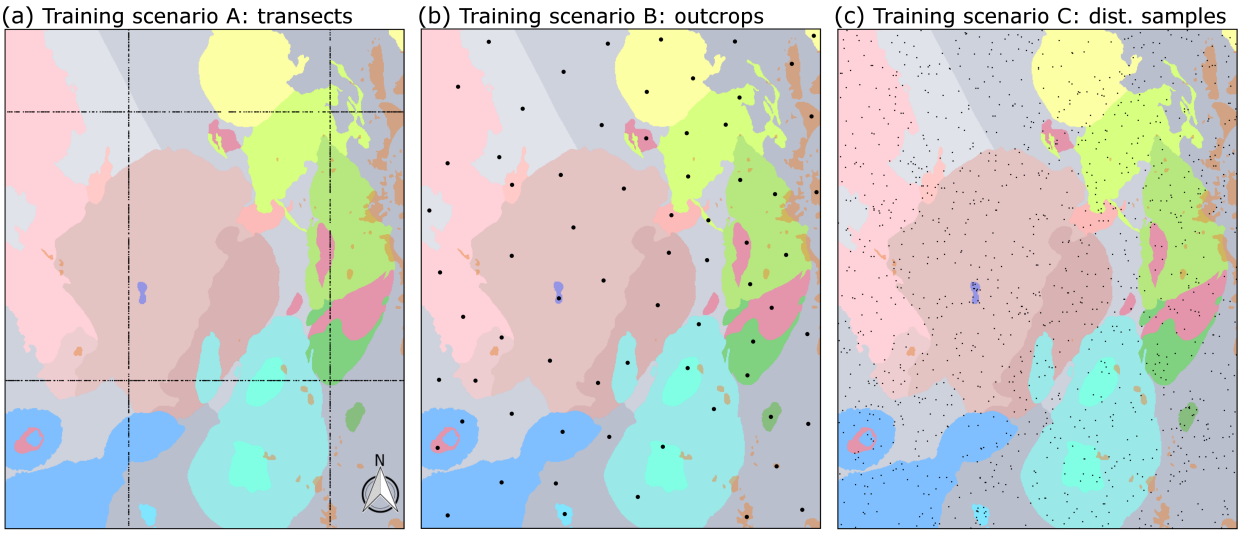

Figure 7.4: Three experiments are conducted where I assume that we know the lithology at a limited number of locations (i.e., the training data), and these locations vary with each scenario. The three training scenarios represent lithologic sampling from (a) transects, (b) outcrop polygons, and (c) distributed grab samples. The black dots indicate where the lithology is sampled from the geologic map. The remaining points are assumed to be unlabeled data for each machine learning experiment and the goal in each scenario is to make predictions for these data. The lithologic units have 50% transparency to make the training points stand out.

7.4 Methodology

7.4.1 Machine learning techniques

Performing the machine learning analysis for the four scenarios described above requires three different kinds of machine learning techniques: unsupervised, semisupervised, and supervised. I utilize the label propagation method with a nearest-neighbors kernel for semisupervised learning (Chapter 2.2.1, Eq. 2.6) and the XGBoost and LightGBM methods (Chapter 2.1.3) for supervised learning. One of the simplest methods that is easy to implement for unsupervised learning is k-means clustering (MacQueen, 1967). However, one of the disadvantages of k-means is that the clusters have no topological properties, i.e., there is no information on how one cluster relates to another. Clusters in this context will represent proxies for lithologic units, so having knowledge of which clusters are closely related to each other (or not) could aid in their interpretation.

I employ self-organizing maps (SOMs), a topologically constrained unsupervised technique that is well-

established (Kohonen, 1982, 1998, 2013). The SOM is represented by a series of nodes that are typically organized onto a 2D mesh. Each node also has a weight vector with the same dimensionality as the input data. Training the SOM nodes is an iterative, two-step process. The process begins by choosing a random input data vector x_i, and then determining the SOM node that is closest; this node, m_c, is called the best matching unit (BMU). The weight vector of this BMU is then moved in the direction toward x_i by the distance $[x_i - m_c]$ scaled by a learning rate, α; this is the first step. The second step consists of identifying the nodes on the 2D mesh that lie within the neighborhood of the BMU, as defined by $h_{c,j}$, and then updating the weights of these nodes (m_j) so they also move in the direction towards x_i. The following formula encapsulates these steps for learning the weights for the SOM nodes,

$$m_j = m_j + \alpha(t)h_{c,j}(t)[x_i - m_j], \quad \forall j \in neighborhood \ \ of \ \ m_c \tag{7.1}$$

where t is an integer representing iterations. Once this process is repeated for each data point x_i, this marks one iteration. The learning rate α and neighborhood function $h_{c,j}$ decrease and shrink, respectively, as iterations progress, which eventually leads the algorithm to a state of convergence. After the SOM is trained, each x_i can be assigned to the SOM node with the most similar weight vector. Therefore, each SOM node can be considered a cluster of input data, where neighboring SOM nodes in the 2D mesh also represent similar clusters.

SOM meshes typically contain 100s or 1000s of nodes based on the established heuristic of $5\sqrt{N}$, where N is the total number of data (Vesanto & Alhoniemi, 2000), but in reality, we expect far fewer clusters if they represent proxies for lithologic units. As such, it is common to perform a secondary clustering on the weights to merge SOM nodes and produce an interpretable number of clusters to analyze. Some techniques that have been used for the secondary clustering of SOM nodes are k-means (Carneiro et al., 2012; Carter-McAuslan & Farquharson, 2021) and agglomerative clustering (AC, Cracknell et al., 2015), where the Davies-Bouldin

Index (DBI, Davies & Bouldin, 1979) is used to select an appropriate number of clusters. The DBI metric measures a ratio of intra-cluster distances to inter-cluster distances, therefore, lower scores result in a better clustering. I mimic the approach of Cracknell et al. (2015), where AC is used in conjunction with SOMs. To select the number of clusters for AC, I use the DBI metric and the silhouette coefficient (Rousseeuw, 1987). SuSi is the Python package (Riese et al., 2020) used for the SOM implementation, and I use a mesh size of 50×50 nodes (this roughly follows the heuristic based on the number of data in this problem) with the remaining parameters set to their defaults. Lastly, I use the *AgglomerativeClustering* class in `scikit-learn` as the implementation for AC.

7.4.2 Training, testing, and evaluation

The supervised and semisupervised methods both have hyper-parameters that must be tuned during training. For the supervised methods, I use 5-fold cross-validation (Figure 2.10) on the training data (Figure 7.4), and the models with the highest cross-validation score are used to make predictions for the entire map. Hyper-parameter tuning for semisupervised methods is more challenging, but there are three possible strategies. The first is simply a trial-and-error, heuristic-based approach to trying different hyper-parameter settings and seeing which work best. The second strategy is to use the standard cross-validation approach designed for supervised methods (i.e., using the labeled data only). For semisupervised methods, the validation fold in cross-validation can be treated as unlabeled data to include during training, and the goal is to still predict labels for the validation fold and evaluate the performance. The problem with this strategy is that some semisupervised hyper-parameters relate to the density of the data as a whole, and only using the labeled data would drastically underestimate the value for these parameters (e.g., the nearest-neighbors hyper-parameter for LP). However, we can mitigate this issue by augmenting the validation fold with all (or a fraction) of the unlabeled data from the problem (Figure 2.11). When predictions are made on this augmented unlabeled data set, only those corresponding to the validation fold can actually be evaluated, but

including the unlabeled data can help inform the learning for semisupervised algorithms. That being said, the increased computational cost of including the unlabeled data in cross-validation could be substantial. Luckily, label propagation only has two hyper-parameters that must be set, and I use the heuristic-based strategy from Chapter 5.3.6 for setting the number of nearest neighbors to 100. That leaves α as the only remaining hyper-parameter, and I use the modified cross-validation approach (Figure 2.11) with five folds to determine α.

Cross-validation and evaluating the predictions on the testing data requires classification metrics to quantify performance (Lever et al., 2016). The classes in this problem are imbalanced, so accuracy and weighted metrics will give more weight to the classes with more points. Even if specific lithologic units have small extents, this does not necessarily mean these units are any less valuable; the smaller, isolated lithologies could indeed be targets for mineral deposits. I choose to weigh each class equally using macro-based metrics rather than give larger units more weight and smaller units less weight. For the cross-validation scores, I use macro-F1, and for evaluating the testing data predictions (the entire map), I use macro precision, recall, and F1.

In addition to computing classification metrics for the map prediction, we can also measure the uncertainty in these predictions. Each of the supervised and semisupervised algorithms considered can produce what are called *class membership probabilities* (CMPs). The CMPs from any given algorithm are represented as a $u \times K$ matrix, where u is the number of (unlabeled) data and K is the number of classes encountered during training. The final classification for any given data point is the class with the highest CMP, but this CMP could be the maximum by only a narrow margin (e.g., other classes could contain near-equal probabilities). In this situation, there would be low confidence (high uncertainty) in the prediction result, but in the case of the highest CMP having a large margin, this may be a point with higher confidence. However, all this information is lost if only the predictions are analyzed, highlighting the importance of assessing the CMPs. It is possible to analyze the CMPs for each class (i.e., produce a raster for each class),

Table 7.3: A summary of the numerical results for Scenarios A, B, and C. The first data column provides the cross-validation scores during training. The values provided are the means and standard deviations (in *italics*) computed from five cross-validation scores. The second column provides the testing (map) performance. A precision, recall, and F1 testing score is computed for each class (21 scores total), and this column provides the means and standard deviations (in *italics*) computed from the 21 individual class scores. However, the macro testing metrics for Scenario A are only computed using 16 of the 21 individual class scores (see Chapter 7.5.1). The values in **bold** indicate the algorithm that performs best for that metric. *Only ten features are used as inputs to prevent overfitting artifacts. **Hyper-parameter tuning using semisupervised cross-validation is inconclusive to determine α, so a fixed value of 0.5 is used.

Machine learning method	**Training (cross-validation) scores**	**Testing data (map) performance**		
	Macro F1	**Macro precision**	**Macro recall**	**Macro F1**
Scenario A: Transects				
XGBoost*	61.95 ± *8.90*	53.03 ± *18.91*	53.55 ± *13.28*	50.85 ± *13.02*
LightGBM*	66.59 ± *7.68*	54.68 ± *19.43*	53.78 ± *13.36*	51.95 ± *13.82*
LP**	91.58 ± *3.32*	**60.89** ± *19.89*	**60.84** ± *16.25*	**58.01** ± *16.07*
Scenario B: Outcrop polygons				
XGBoost	84.87 ± *8.25*	49.01 ± *20.93*	52.68 ± *16.50*	48.60 ± *18.25*
LightGBM	89.42 ± *7.19*	53.71 ± *22.89*	55.29 ± *18.77*	52.19 ± *20.75*
LP**	99.81 ± *0.16*	**59.63** ± *24.23*	**64.58** ± *14.95*	**58.28** ± *18.63*
Scenario C: Distributed samples				
XGBoost	61.95 ± *13.81*	70.77 ± *15.44*	69.52 ± *17.59*	69.53 ± *16.07*
LightGBM	63.07 ± *15.23*	**74.20** ± *13.87*	70.68 ± *18.01*	**71.83** ± *15.52*
LP	69.56 ± *4.55*	73.32 ± *17.27*	**72.19** ± *16.69*	71.71 ± *16.15*

but capturing this information into a single number for each point may be more efficient. One method for quantifying the prediction uncertainty is the normalized information entropy (Kuhn et al., 2018, 2019, 2020) given by,

$$H_i = -\frac{1}{K}\sum_{k=1}^{K} p_{i,k} \log p_{i,k} \tag{7.2}$$

where $p_{i,k}$ is the CMP of class k for the data point i. The value of H is at its maximum when the CMPs are equally probable across all classes, and it is at its minimum when a probability of 1.0 belongs to one of the classes. In the following sections, I use the classification metrics and associated lithology predictions in conjunction with entropy to assess the results of Scenarios A, B, and C.

7.5 Results

A summary of the numerical results for Scenarios A, B, and C is provided in Table 7.3. In this table, I provide the cross-validation performance on the training data and the testing (map) performance for each algorithm. There are no metrics for Scenario D in Table 7.3, given that Scenario D is an unsupervised analysis reflecting a situation with no a priori labeled data. In the subsections below, I provide a brief description of the results for each Scenario.

7.5.1 Scenario A: Transects

The first situation that I explore is Scenario A, where the training data represent sampling taken along four different transects (Figure 7.4a). Using all 12 features as inputs to train both supervised methods produces erroneous results. The predictions for LightGBM (see red polygons, Figure 7.5a) show recognizable artifacts in the north-western portion of the map (while not shown, XGBoost has the same artifacts). These artifacts have a long-wavelength behavior to them, and upon inspection of the feature importances for LightGBM and XGBoost (Figure 7.6, top row), it is clear that these methods are overfitting the smoothest input features (magnetics RTP upcont and pseudogravity). To fix these artifacts, I must remove both of these input features. This makes the feature importances on the remaining ten inputs for LightGBM and XGBoost more evenly distributed (Figure 7.6, bottom row), and the subsequent predictions no longer contain the long-wavelength artifacts (predictions and entropy for LightGBM are shown in Figures 7.7a and 7.7b, respectively). The cross-validation and testing data performances for LightGBM and XGBoost, using the ten input features, are provided in Table 7.3. A unique aspect of this scenario is that the training data do not sample every class on the basemap, as the transects do not intersect five of the units: 6, 9, 11, 13, and 14 (see Figures 7.2 and 7.4a). As a result, these five units on the map get classified as one of the other 16 classes encountered during training, which is easily seen by looking at the rows of the confusion matrix corresponding to these five classes (see Figure 7.8a). This translates to the classification metrics being 0% for these five units on the

testing (map) data. Rather than let these zeros bias the macro metric calculations, the testing data metrics in Table 7.3 only come from the 16 classes that can be predicted by the algorithms.

Unlike the two supervised methods, label propagation can train using all 12 input features without any of the long-wavelength artifacts noted above (see Figures 7.7c and 7.7d). However, the complication with this scenario is that the semisupervised cross-validation scores to determine α are not trustworthy. This is ultimately a consequence of the spatial distribution of the training data. Points from the same class that are in close proximity to each other will likely have highly correlated input features. Given that the training data for this scenario are taken along transects, we can expect neighboring training samples of the same class to be highly correlated. The nature of label propagation is to spread its labels to the unlabeled data, and in this instance, validation folds are strongly correlated to the training folds (Figure 2.11b). The consequence here is that regardless of which α is used, the cross-validation scores are unrealistically high, which leads me not to trust the hyper-parameter selection. Rather than use the α with the highest cross-validation score, which I suspect is not trustworthy, I instead fix it to a conservative value of 0.5. I provide the cross-validation score associated with $\alpha = 0.5$ in Table 7.3, and notice the disparity between the cross-validation performance for the supervised methods versus LP. Conventional knowledge would attribute an inflated training performance to overfitting, but that is not the case here; the phenomenon observed is attributed to highly correlated training data. Nonetheless, this LP model performs well on the testing data as indicated by the predictions (Figure 7.7c) and confusion matrix (Figure 7.8b).

7.5.2 Scenario B: Outcrop polygons

The next scenario that I simulate is the situation where the labeled data are obtained from 70 outcrop stations, which are represented as circular polygons with a radius of 250-m (Figure 7.4b). Unlike Scenario A, the supervised methods in this situation can train with all 12 features as inputs without generating prominent artifacts. The feature importance graph suggests that LightGBM is overfitting the pseudogravity feature

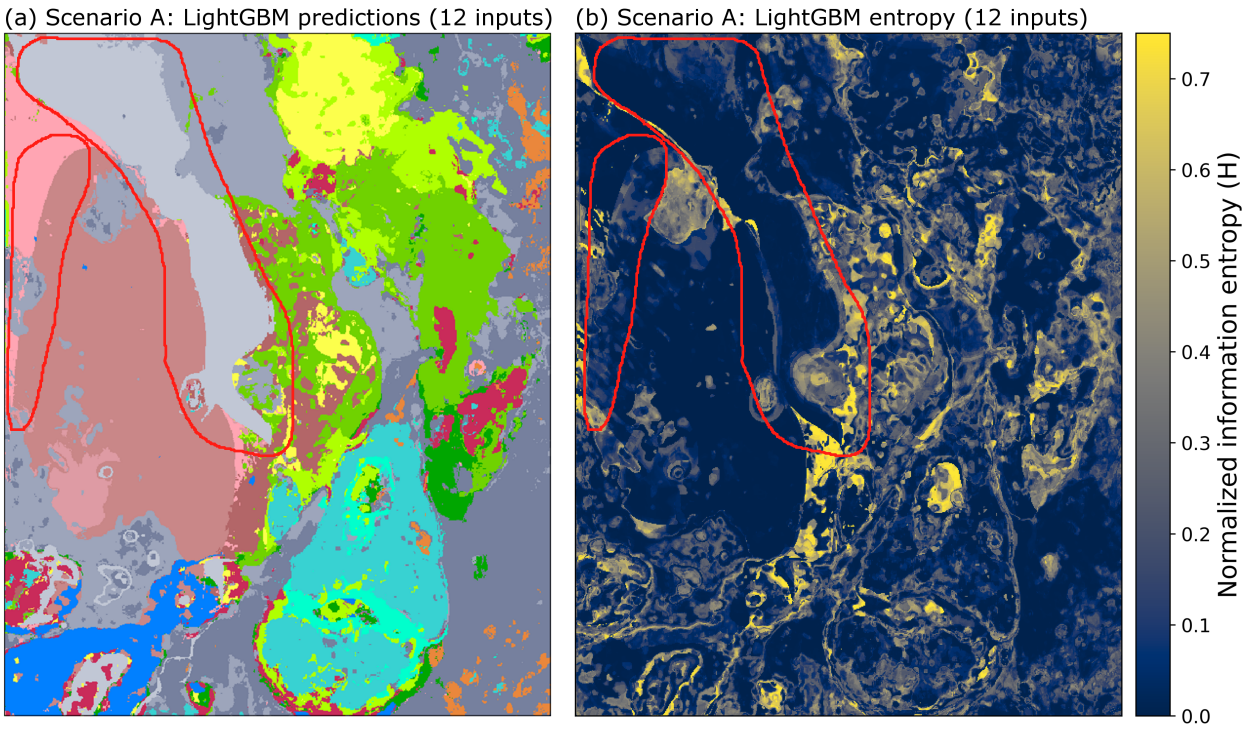

Figure 7.5: The (a) predictions and (b) entropy for LightGBM in Scenario A using all 12 input features. There are noticeable long-wavelength artifacts in the western-portion of the map (denoted by red polygons), indicating that the LightGBM model is overfitting the training data.

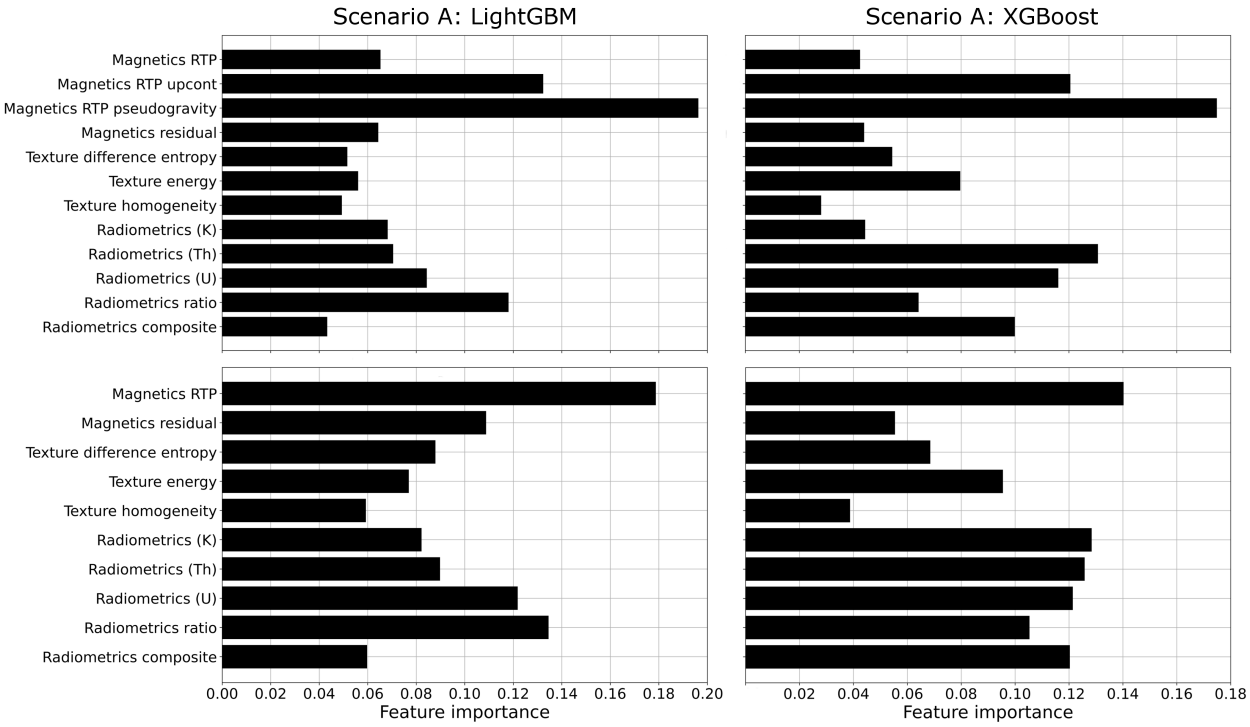

Figure 7.6: The feature importances for LightGBM and XGBoost in Scenario A using (top row) 12 inputs and (bottom row) 10 inputs. Figure 7.5 shows LightGBM overfitting the smoother input features, Magnetics RTP pseudogravity and upcont, so these two features are removed to mitigate these artifacts. The feature importances for LightGBM and XGBoost are much more balanced with these two features removed (bottom row).

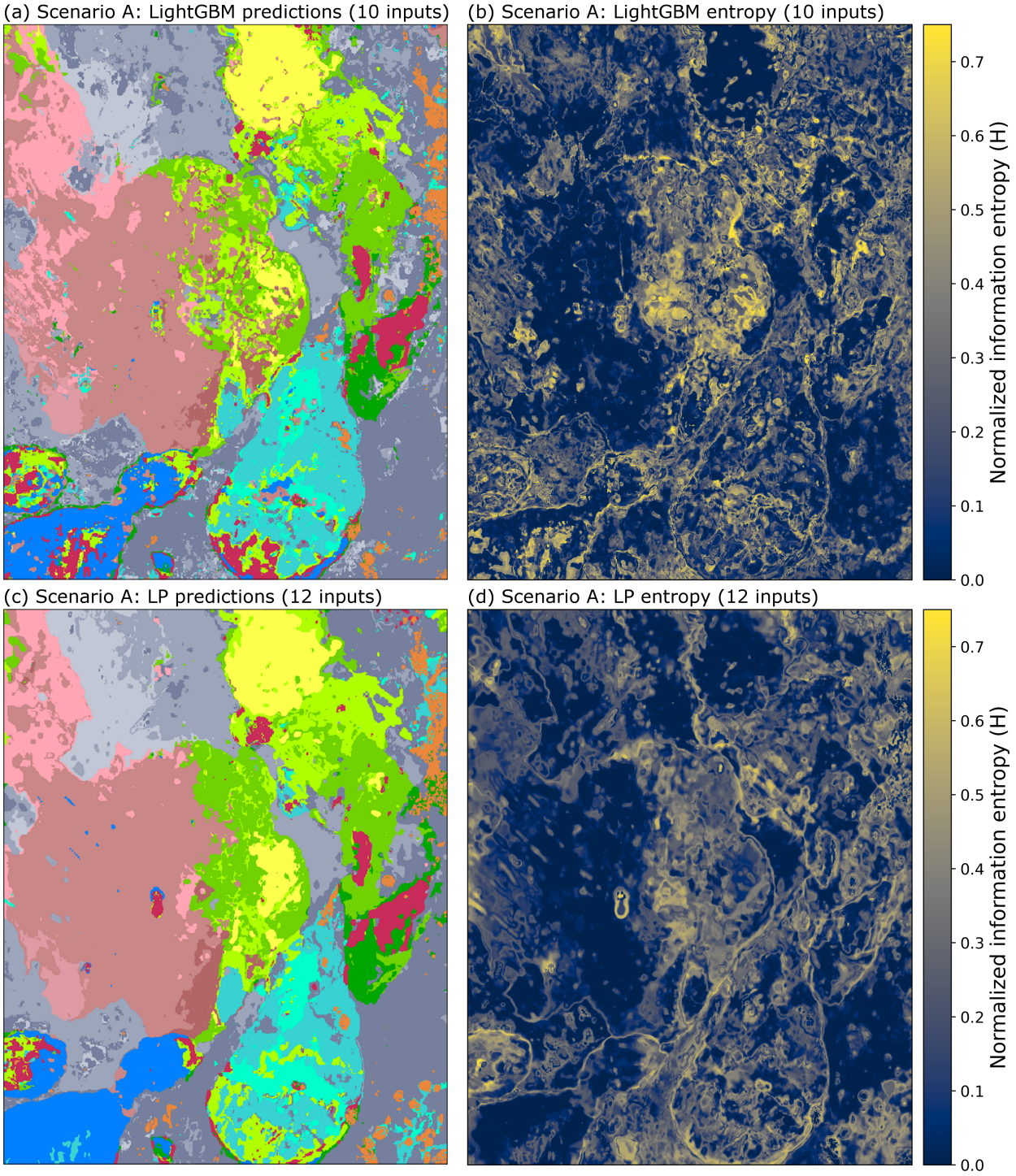

Figure 7.7: The Scenario A predictions and entropy for LightGBM (a, b) and LP (c, d). The predictions and entropy shown for LightGBM are for when 10 input features are used to prevent overfitting artifacts (compare to Figure 7.5). LP is able to utilize all 12 input features without producing the artifacts seen by LightGBM. The LP predictions are generally less noisy and slightly more accurate than the LightGBM predictions. Refer to Figure 7.2 for the lithologic unit key, Figure 7.4(a) for the Scenario A training data locations, and Table 7.3 for the numerical results.

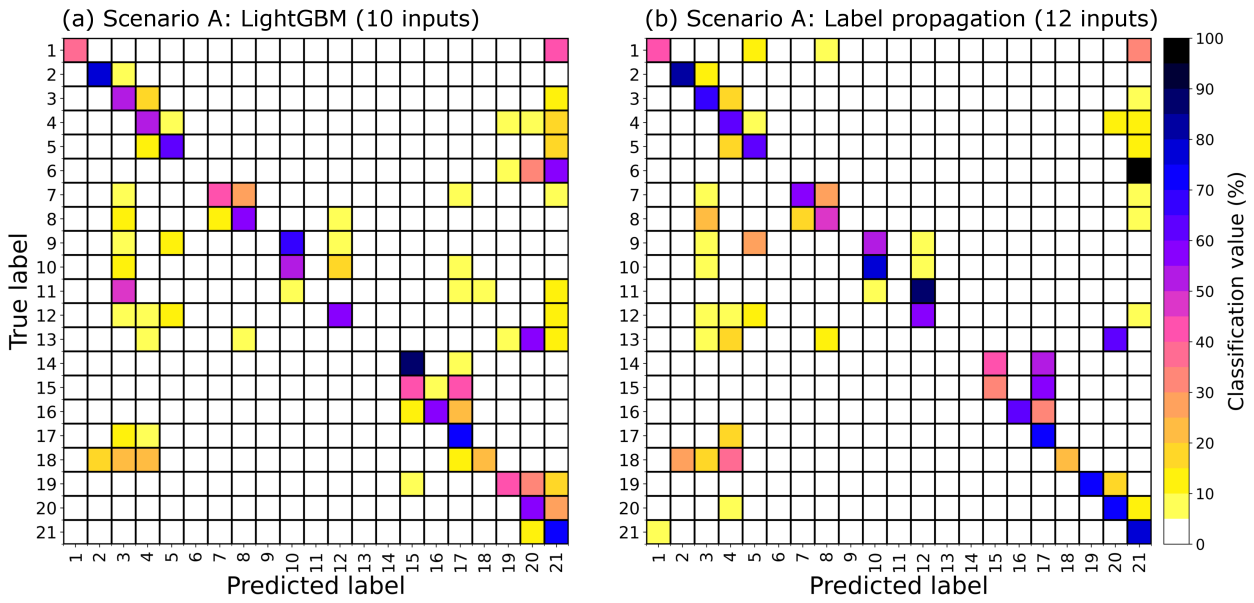

Figure 7.8: The Scenario A confusion matrices for (a) LightGBM and (b) LP. These confusion matrices are computed from the testing data predictions (Figures 7.7a and 7.7c) and the known geologic map (Figure 7.2). The training data only sample 16/21 classes, so five of the classes cannot be predicted for (6, 9, 11, 13, and 14).

(Figure 7.9a), but the predictions and entropy (Figures 7.10a and 7.10b) are generally devoid of the long-wavelength artifacts observed in Figure 7.5.

Much like Scenario A, the training data from Scenario B are still strongly correlated. The semisupervised cross-validation for α is once again indeterminate as all the cross-validation scores are nearly 100%. While the correlation among labels appears to impact LP the most, the supervised methods also seem to be affected with cross-validation scores near 90% (see Table 7.3). Out of caution, I once again fix α to a value of 0.5. The resulting model still performs well on the testing data (Figure 7.10c), and the confusion matrices in Figure 7.11 show that the LP model performs better than LightGBM on most classes.

7.5.3 Scenario C: Distributed samples

For the third situation that I explore, the labeled data are assumed to represent lithologies determined from 1250 grab samples spatially distributed across the map (Figure 7.4c). When the training data are spatially distributed in this manner, this reduces the correlation among training points for a given class and better

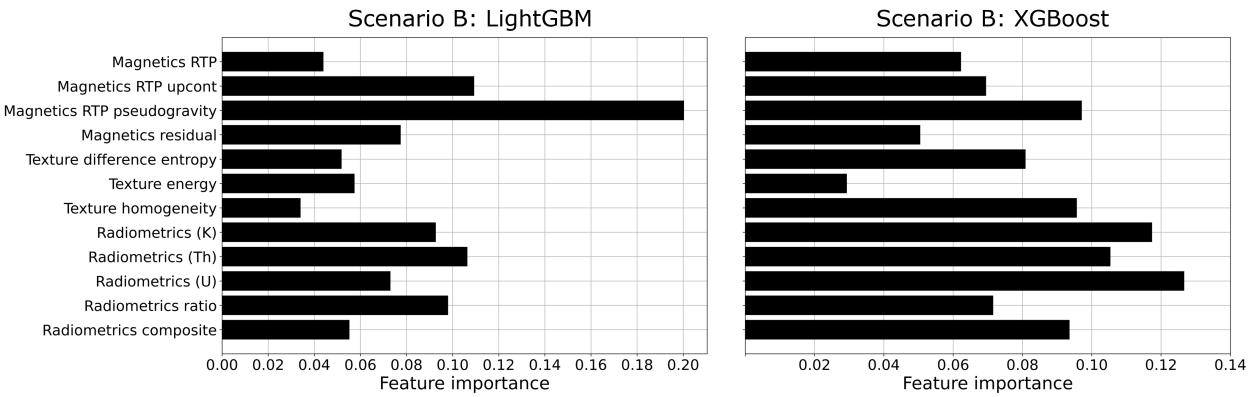

Figure 7.9: The feature importances for LightGBM (left) and XGBoost (right) in Scenario B. LightGBM has a significant weight placed on the magnetics RTP pseudogravity, but the associated predictions (Figure 7.10a) do not contain the same overfitting artifacts encountered in Scenario A. The feature importances for XGBoost are relatively balanced.

captures their distribution. As a result, Scenario C produces the best performance on the map compared to Scenarios A and B, and the cross-validation scores more closely reflect the testing performance for each algorithm (see Table 7.3). The supervised methods continue to have imbalanced feature importances (Figure 7.12), but this seems to have little impact on the predictions as they appear relatively clean (Figure 7.13a). The semisupervised cross-validation for α is finally insightful in this scenario, as the CV scores have reliable values (e.g., not 99%), and there is a clear maximum mean CV score associated with a corresponding low CV standard deviation (see Figure 7.14). The cross-validation suggests an $\alpha = 0.75$, which performs well on the testing data. Interestingly, the testing data performance, predictions, and associated confusion matrices for LP and LightGBM are comparable in this Scenario (see Figures 7.13 and 7.15).

7.5.4 Scenario D: Unsupervised clustering

The final scenario aims to define the natural groups, or clusters, in the data that are independent of lithological constraints from training data. Figure 7.16 shows some key components of the SOM-AC workflow. The histogram plot (Figure 7.16a) is a convenient way to visualize how many points are assigned to each SOM node. There are many points assigned to boundary nodes, which is a well-known consequence of 2D sheet-like meshes that can lead to SOM nodes not accurately capturing the topology of the data space

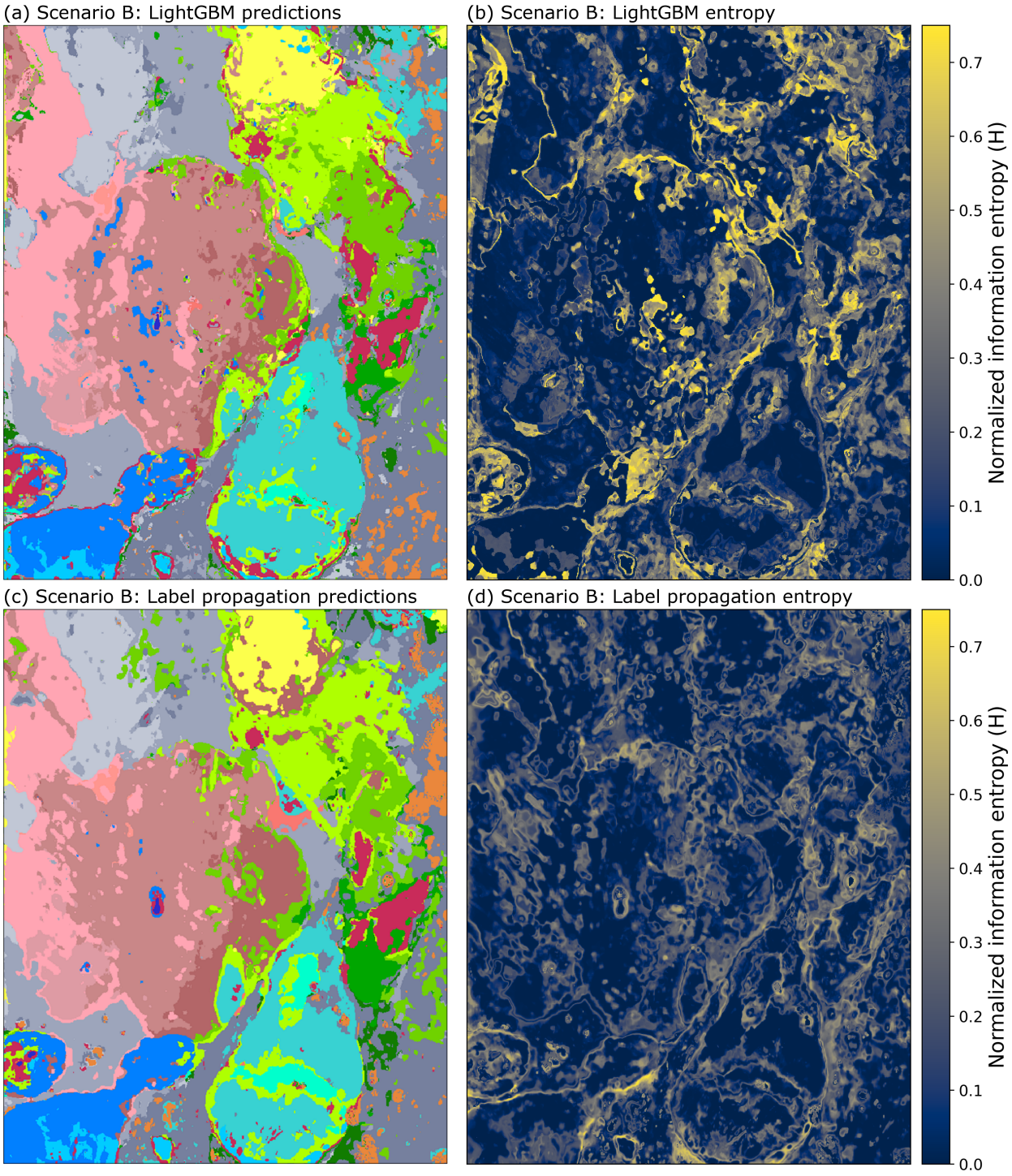

Figure 7.10: The Scenario B predictions and entropy for LightGBM (a, b) and LP (c, d). The LP predictions are generally less noisy and slightly more accurate than the LightGBM predictions. For both methods, the entropy is effective at indicating misclassified areas and unit boundaries. Refer to Figure 7.2 for the lithologic unit key, Figure 7.4(b) for the Scenario B training data locations, and Table 7.3 for the numerical results.

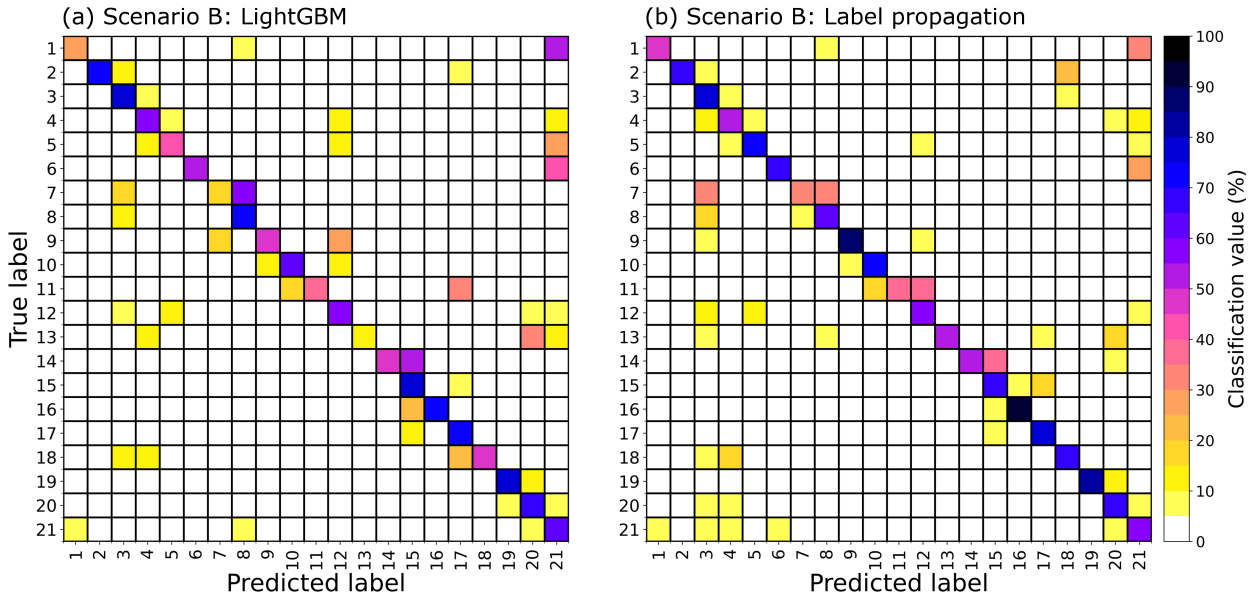

Figure 7.11: The Scenario B confusion matrices for (a) LightGBM and (b) LP. These confusion matrices are computed from the testing data predictions (Figures 7.10a and 7.10c) and the known geologic map (Figure 7.2). The LP confusion matrix shows higher values on the diagonal than the LightGBM confusion matrix for most units, which explains the higher metrics for LP in Table 7.3.

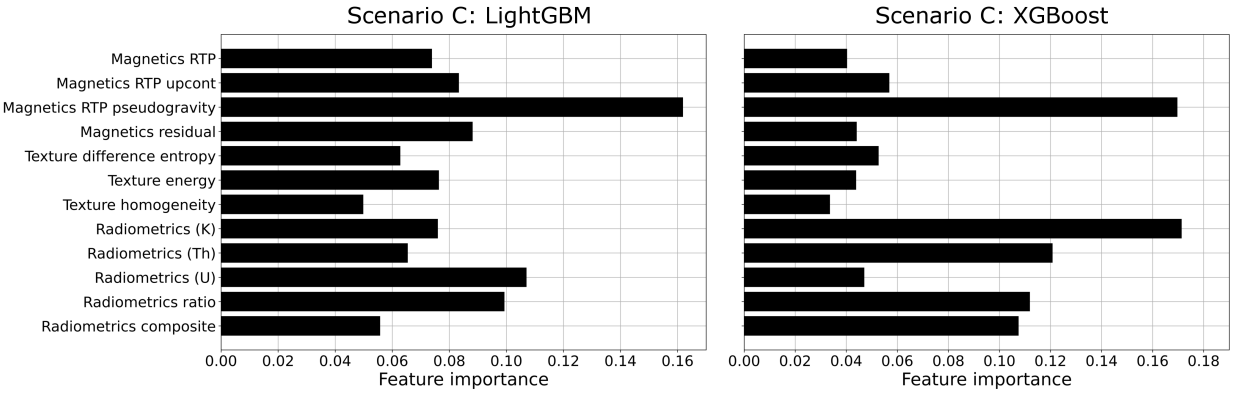

Figure 7.12: The feature importances for LightGBM (left) and XGBoost (right) in Scenario C. Both supervised algorithms are sensitive to particular input features, but this does not appear to cause any artifacts in the predictions.

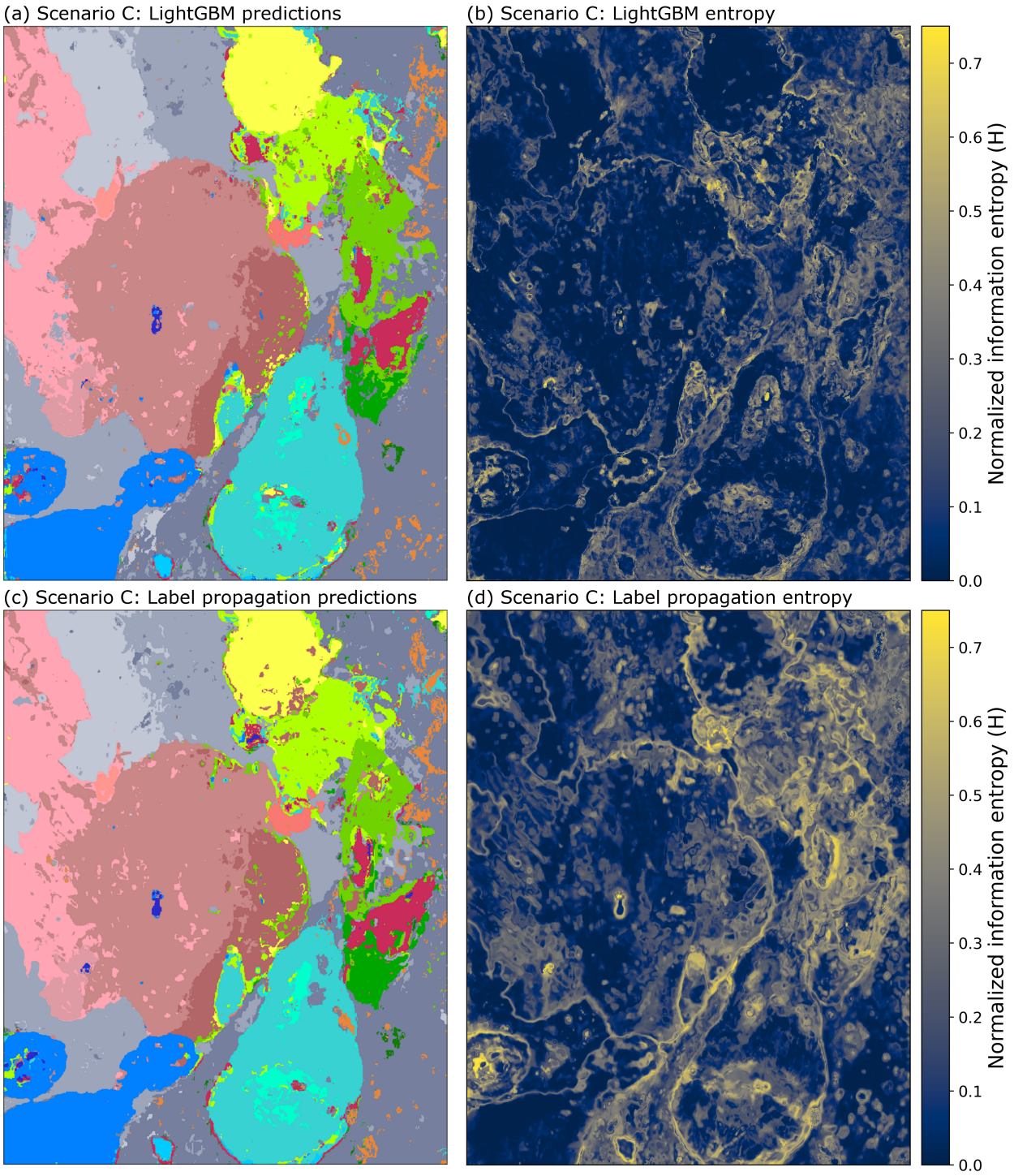

Figure 7.13: The Scenario C predictions and entropy for LightGBM (a, b) and LP (c, d). The results from this scenario produce the best predictions compared to those from Scenarios A and B. The testing data performance is also comparable for LightGBM and LP. Despite the predictions being rather clean, the entropies suggest significant uncertainty in most of the Uralla Supersuite and the north-western portion of Unit 10. Refer to Figure 7.2 for the lithologic unit key, Figure 7.4(c) for the Scenario C training data locations, and Table 7.3 for the numerical results.

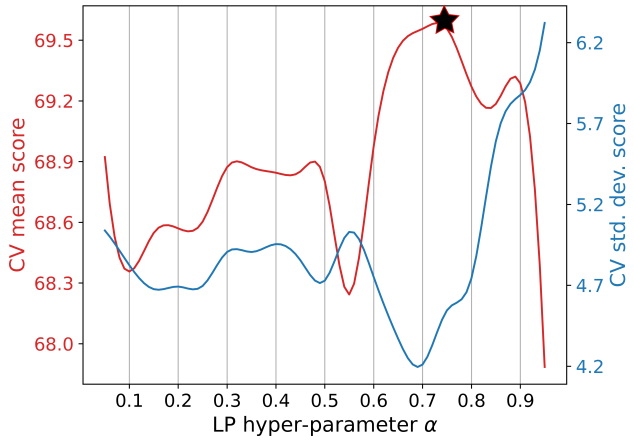

Figure 7.14: The semisupervised cross-validation (Figure 2.11b) mean and standard deviation scores to determine α for label propagation in Scenario C. The chosen $\alpha = 0.75$ has the highest mean score, and also has a relatively low standard deviation score.

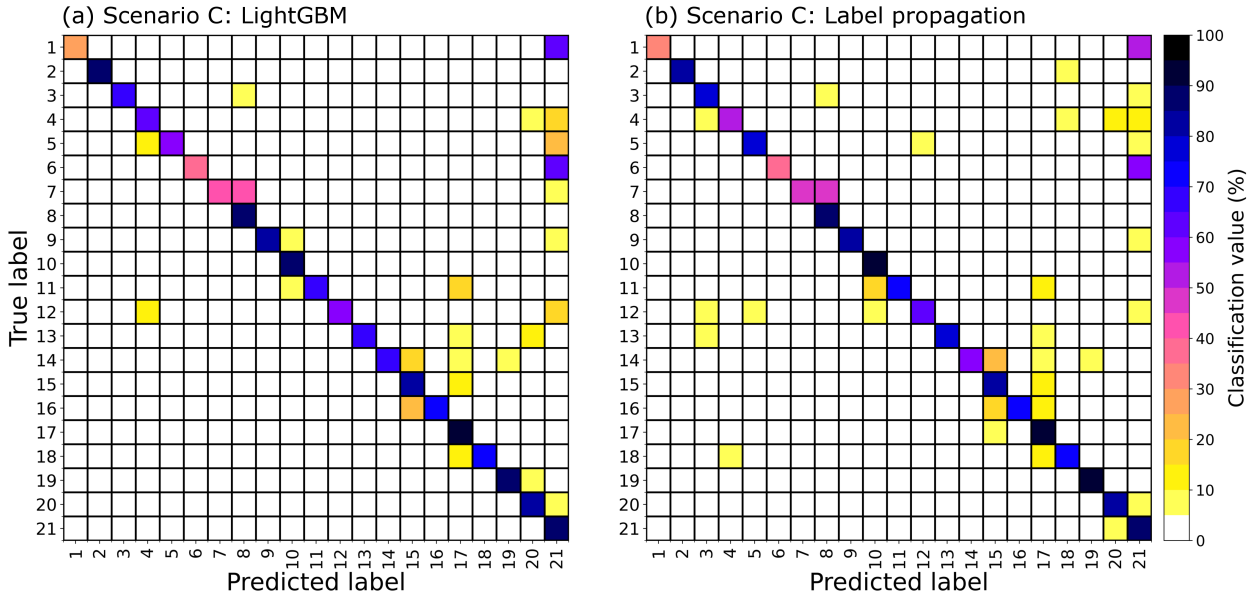

Figure 7.15: The Scenario C confusion matrices for (a) LightGBM and (b) LP. These confusion matrices are computed from the testing data predictions (Figures 7.13a and 7.13c) and the known geologic map (Figure 7.2). The confusion matrices for both algorithms are comparable, which aligns with the similar performance of both algorithms indicated in Table 7.3.

(Mount & Weaver, 2011; Carter-McAuslan & Farquharson, 2021). A solution is to use a toroidal-based mesh for the SOM nodes, but the SuSi Python package currently does not have this functionality. Figure 7.16(b) is the U-matrix, which shows how far the SOM weight vectors are to their adjacent neighbors in Euclidean distance. Regions of larger distances in the U-matrix could signify heterogeneous lithologic units or perhaps boundaries between units in this context.

The next two panels in Figure 7.16 are related to the secondary clustering of the SOM nodes using AC. Figure 7.16(c) shows two clustering metrics, DBI and the silhouette score, for a range of cluster values from 5-20. The optimal cluster setting minimizes the DBI and maximizes the silhouette score, which suggests that 9 clusters are optimal. The appearance of these 9 secondary clusters projected onto the 2D SOM node map is given in Figure 7.16(d). The U-matrix is also superimposed onto Figure 7.16(d) with 50% transparency to help indicate which clusters contain the large U-matrix distances (e.g., Clusters 3, 7, and 9). I still analyze a few other cluster settings, such as 11 and 13, and find that they have a closer resemblance to the geologic map than 9 clusters. However, I am only able to make this determination because I have prior knowledge of the true map; without this knowledge, I would be drawn to the 9 clusters option. In Figure 7.17(a), I show the cluster map using 9 clusters, but I also compare this result to using 13 clusters in Figure 7.17(b).

7.6 Discussion

7.6.1 Analysis of Scenarios A, B, and C

Even though each of the experiments has roughly the same amount of training data (1258, 1371, and 1250 for Scenarios A, B, and C, respectively), the ultimate predictions for the entire map look different for each scenario. Scenario A does not sample five of the smaller igneous intrusive units (6, 9, 11, 13, and 14), which end up classified as one of the 16 classes that are present in the training data. For both supervised and semisupervised algorithms, Units 9, 11, and 14 are classified as other intrusive units (refer to Figures 7.2,

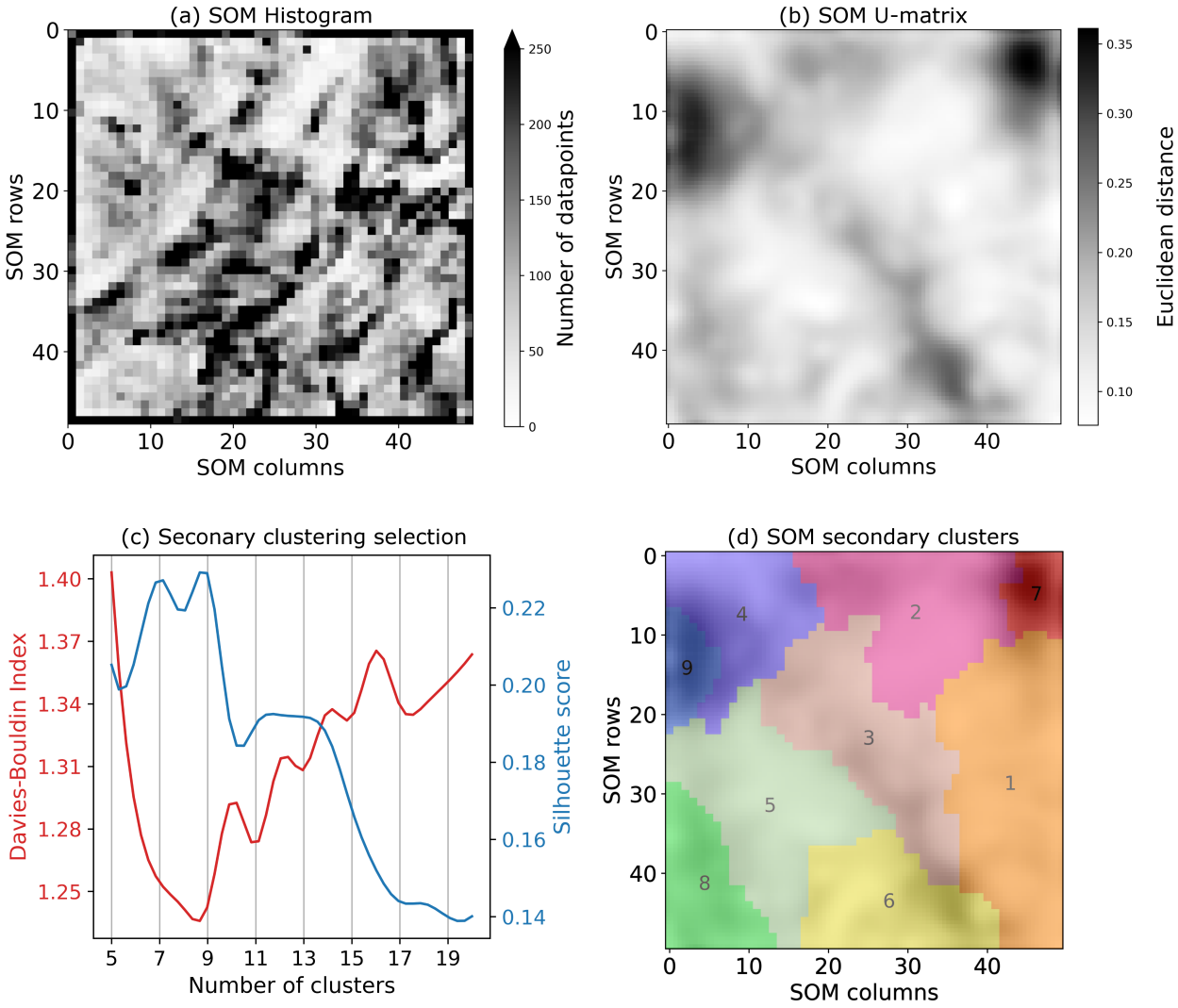

Figure 7.16: The key components of the SOM-AC workflow for the unsupervised clustering performed in Scenario D. The (a) histogram and (b) U-matrix are standard outputs from a trained SOM model (see text for details). A secondary clustering of the SOM using AC (c) suggests that 9 clusters are optimal based on the DBI and silhouette score metrics. Panel (d) shows the distribution of these 9 clusters on the 2D SOM node map with the U-matrix superimposed at 50% transparency.

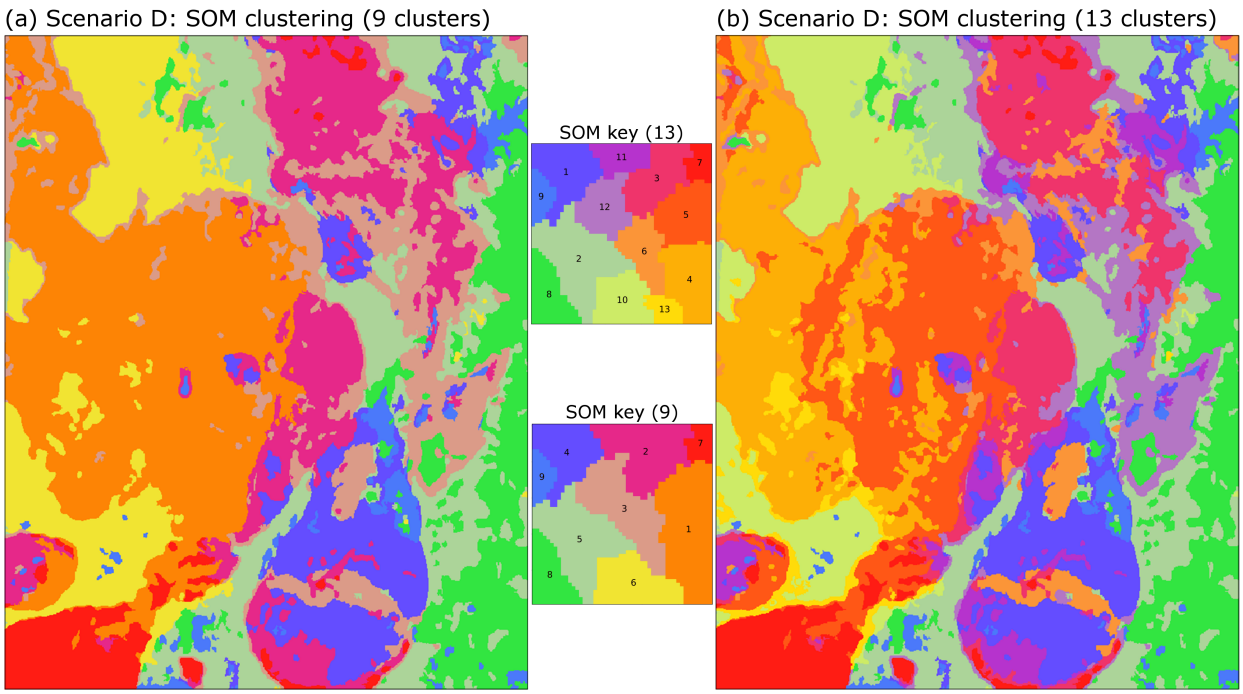

Figure 7.17: The Scenario D SOM-AC cluster maps using (a) 9 clusters and (b) 13 clusters. The locations of the clusters in the 2D SOM node space (see the keys) are consistent between 9 and 13 clusters, with certain clusters from Panel (a) being decomposed into two clusters for Panel (b). These results also bear a strong resemblance to the geologic map in Figure 7.2.

7.7, and 7.8). However, Units 6 and 13 are assigned to background Units 21 and 20, respectively, and the entropies for LightGBM and LP also indicate that they are rather confident in these classifications (Figure 7.7). This suggests that the signals for Units 6 and 13 are likely too subtle with respect to the background units to be confidently classified as one of the intrusive units (this is also supported by the clustering in Figure 7.17). One of the most-obvious misclassifications for all algorithms in Scenario A is Units 17 and 18 of the Bundarra Supersuite (center of the map) being classified as units of the Uralla Supersuite (Figures 7.7 and 7.8). There are, however, noticeable differences overall between the predictions of LightGBM and LP. LightGBM overpredicts the presence of Unit 12 (red unit) in the Moonbi Supersuite, whereas LP does not contain these misclassifications. Generally, the LP predictions are less noisy than those from LightGBM, which is undoubtedly the case for the background units (Figure 7.7). These observations show why LP outperforms the supervised methods on the testing data by $6-7\%$ for all macro metrics (Table 7.3). Similar

conclusions can be drawn regarding Scenario B. There are regions where all algorithms struggle, such as the north-eastern area of the map and the north-western portion of Unit 10, but LP performs much better on Units 5, 13, and 18 (see Figures 7.10 and 7.11). This experiment shows LP performing $6-9\%$ better than LightGBM for all macro metrics (Table 7.3). Scenario C is where we begin to see some different phenomena occur with regards to performance. All the algorithms have their best testing (map) performance in this experiment, and depending on the algorithm and metric, some improvements are on the order of $10-20\%$ relative to Scenarios A and B (see Table 7.3). The predictions are relatively noise-free (Figure 7.13), and the confusion matrices show high values situated along the diagonal (Figure 7.15). Despite the training data still being quite limited, LP only marginally outperforms XGBoost and has comparable performance to LightGBM.

The distinctions in map performance between Scenarios A and B and Scenario C are a result of how the training data are distributed spatially. In Scenarios A and B, the training samples are incredibly localized, which creates situations where the ML predictions can be interpreted as having to perform *extrapolation* spatially (see Figure 7.4). Despite the highly correlated training data creating challenges for semisupervised cross-validation, the LP method performs better than the supervised methods under these extrapolation-style scenarios. On the contrary, the training samples for Scenario C are well distributed, which creates a situation where the ML predictions can be interpreted as performing *interpolation* spatially (see Figure 7.4c). The testing (map) performance is the best for all algorithms in this scenario, likely because the well-distributed training data better capture the distribution of each lithologic unit making the learning of decision boundaries much easier. This significantly improves the performance of the SL methods to the point where SSL appears not to provide any added value; it is sufficient to use SL methods. Cracknell & Reading (2014) explore at length the impact of training data spatial distribution; they arrive at the same conclusion that as the training data become more dispersed, the performance on the testing data improves. This has practical implications because if a company plans on using ML to produce lithological maps in greenfield areas, they can use

this prior knowledge to design field campaigns with better spatial sampling to ensure a better ML product downstream. However, it is not always possible to secure spatially distributed training samples due to cost or logistical factors; in this circumstance, these findings suggest that semisupervised methods are preferred. As a whole, these results can help guide users toward whether SL or SSL is more appropriate based on the training data situation in their own problems.

Even though I show that semisupervised algorithms can perform better than supervised algorithms under more challenging circumstances, the predictions for both types of methods are still not perfect. This could be attributed to several factors, such as the spatial distribution of the training samples discussed above, but another aspect to be mindful of is that the training samples are themselves typically an approximation made by a geoscientist. These factors, among others, can undoubtedly introduce errors, noise, and uncertainties into the machine learning predictions. These uncertainties in the predictions can be expressed as a significant spread in the CMPs (i.e., the algorithm does not have confidence in the class assignment for a given data point), which I quantify in this study using information entropy.

Generally, high entropy values correlate well with misclassifications, unit boundaries, or heterogeneous areas within units. The misclassification of Units 17 and 18 of the Bundarra Supersuite as being units of the Uralla Supersuite from Scenario A is a perfect example (center of the map, Figure 7.7). Even though the predictions for all algorithms are incorrect in this area, the entropy values are significantly elevated here, suggesting that these predictions are untrustworthy (Figures 7.7b and 7.7d). The entropies can also sometimes indicate if a boundary between units is abrupt or transitional. Unit boundaries with abrupt transitions exhibit a spatially isolated spike in entropy, such as the boundaries between the Bundarra Supersuite and Moonbi Supersuite units with the background meta-sediment units (see Figures 7.10b, 7.10d, 7.13b, 7.13d). However, a transitional unit boundary exhibits elevated values of entropy distributed over a larger spatial region. Some examples that we observe are between the meta-sediment Units 19 and 20 in the northern portion of the map, and between the shared boundary of intrusive Unit 16 and Units 15 and 17 in the southwest

(see Figures 7.10b and 7.10d). We can also gain insights regarding the heterogeneous nature of units from the entropy information. Homogeneous units that are correctly classified tend to have low entropy values (e.g., most of the Bundarra Supersuite in Figures 7.10 and 7.13). By contrast, heterogeneous units tend to have higher entropy values, and they can still be correctly classified. Some examples include most of the Uralla Supersuite in the east, and the north-western portion of Unit 10 (refer to Figures 7.13b and 7.13d).

I am able here to distinguish high entropy values associated with misclassifications and heterogeneity because I have the basemap to evaluate and compare against. In reality, this basemap would not be available because that is what I am trying to produce; so, the misclassifications and heterogeneous areas may be indistinguishable. Nonetheless, regardless of their underlying source, a high entropy prediction represents regions with uncertainty that need refinement. From a practical point of view, these regions could indicate focus areas for subsequent field campaigns (i.e., obtaining more training data). For instance, gathering additional training data could enhance the resolution of uncertain (smooth) unit boundaries, or reveal the presence of additional units in an area assumed to be one homogeneous unit. In the context of this study, it seems that additional training data may help reduce the uncertainty in the eastern portion of the map.

7.6.2 Analysis of Scenario D

My purpose for performing unsupervised learning on this dataset is to see how well the clustering can recover map patterns independent of any label information and to assess the value of applying unsupervised learning when training data are still available. The SOM-AC results in Figure 7.17 appear to strongly resemble the geologic map in Figure 7.2. When comparing the clustering results to the geologic map, we generally observe three phenomena: (1) some clusters correlate well with a single unit on the map, (2) some clusters represent more than one geologic unit, and (3) single map units are represented by multiple clusters. Recall that the resulting clusters have a topology associated with them, in that neighboring clusters on the SOM grid will be more similar than those that are more distant. The three clusters in the bottom-left portion

of the SOM grid (e.g., clusters 5, 6, and 8 in Figure 7.17a) correlate reasonably well with the background meta-sedimentary units on the map (Units 19, 20, and 21); given that these three clusters are in the same region of the SOM grid indicates that they are similar as well. This helps explain why the SL and SSL performance on these three background units is generally quite good.

Given that the optimum number of clusters is 9 and the total number of units on the map is 21, certain clusters must represent more than one geologic unit, and there are several instances of this. One obvious example is one cluster representing Units 14, 15, 16, and 17 (or just Units 14, 15, and 16 in the Figure 7.17b case). If all of these units can be captured by a single cluster, then perhaps the training data associated with these units could be merged. If each of these units is indeed all part of the same cluster, then this would help explain why there is some confusion/misclassification in the predictions for these four units in Scenarios A, B, and C. Despite this observation, the SL and SSL algorithms are still capable of reasonably distinguishing each of these units from each other (see Figure 7.10). Another example is one cluster representing Units 2 and 18 on the map. Much like before, this may explain some of the misclassifications observed (e.g., parts of Unit 2 being misclassified as Unit 18 in Figure 7.10c), but the SL and SSL algorithms can still adequately distinguish between these two units. One more example that stands out for multiple reasons is Cluster 6 (Figure 7.17b), which encapsulates Unit 7 and portions of Unit 3. The classification results in Figure 7.10 show halos of Unit 3 predicted around the two locations of Unit 7; this may be happening because both units share the same cluster, and misclassifications are likely. Interestingly, the southern portion of Unit 7 on the map does not relate well with how the cluster is defined in this area (i.e., the cluster has a much more pronounced east-west extent). Upon inspection of the input data, this cluster may be responding to signals that do not necessarily relate to lithologic changes. The RTP magnetics data in Figure 7.3(a) show two demagnetization anomalies within Unit 8, which could represent alteration, and Cluster 6 directly captures these two anomalies. The northern demagnetization anomaly roughly coincides with the northern portion of Unit 7, but the southern anomaly does not align with the southern portion of Unit 7 (compare Figures 7.2

Table 7.4: Results showing the impact of feature expansion on Scenario C. The purpose is to quantify the impact that the texture features have on the testing data performance compared to if the magnetics RTP 1VD is used. Including the RTP 1VD provides no benefit, or worsens performance, and using the texture attributes only provides a minor benefit. The last column has the same values from Table 7.3. *Refer to Table 7.2 for the features representing the integers 1-12.

Machine learning method	**Input features (4 tot): *1, 8-10**	**Input features (9 tot): *1-4, 8-12**	**Input features (10 tot): *1-4, 8-12, RTP 1VD**	**Input features (12 tot): *1-12**
XGBoost	0.4915	0.6800	0.6789	0.6953
LightGBM	0.5165	0.7100	0.7103	0.7183
LP	0.5156	0.7080	0.6931	0.7171

and 7.17). This potential alteration signal also manifests itself in the SL and SSL predictions as well and is something to be mindful of (see Figures 7.7 and 7.10).

The third situation that occurs is instances where one unit on the map is characterized by more than one cluster. One example is the north-western portion of Unit 10, which comprises roughly three clusters (see Figures 7.2 and 7.17). The more obvious examples are Units 3 and 4 being represented by several clusters. What is interesting in each of these circumstances is that they are all coincident with regions that contain high entropy in Scenarios A, B, and C.

For this dataset, using SL and SSL with the limited training data, I am able to recover more lithologic detail than the cluster results, which may just be a consequence of the number of clusters being fewer than the number of classes in the training data. As mentioned above, there are multiple instances where the SL and SSL approaches can distinguish several lithologic units from each other that are only described by a single cluster. However, this does not mean that the clustering results are without value. The clusters clearly indicate heterogeneous areas that correlate with high-entropy regions in the SL and SSL predictions. In other words, the clustering results can help calibrate expectations on how good (or poor) the subsequent SL and SSL results might be, given that certain behaviors in the clustering still manifest themselves in the SL and SSL results.

7.6.3 Feature expansion and texture attributes

A secondary objective of this study is to evaluate the impact of the GLCM texture attributes on this problem. The feature importance measurements for Scenarios A, B, and C (Figures 7.6, 7.9, and 7.12, respectively) indicate that there is a noticeable contribution from the three texture features, but their importance is relatively less overall compared to the other nine features. However, to truly quantify the impact of texture features on the problem, I investigate how they affect the testing (map) performance, but I do so in the greater context of feature expansion. To demonstrate this, I perform additional simulations with all three algorithms using different combinations of input features in Scenario C. I start by only using the original four features, which are the magnetics RTP and the three radiometric channels (K, Th, U). Second, I include the expanded features for magnetics and radiometrics, but I exclude the texture features, giving nine inputs. In the next situation, I use the same nine previous features, but I include the magnetics RTP 1VD for the first time. Then for the final situation, I use all 12 inputs (i.e., including the texture attributes, but not the RTP 1VD) as I do for the main part of this study. These four situations allow me to isolate the performance changes attributed to certain features or groups of features.

The results from each of these four situations are provided in Table 7.4, and there are a few notable observations. The performance boost is most significant when going from the initial four input features (Column 1) to including the magnetic and radiometric expanded features (Column 2). Not surprisingly, including the RTP 1VD provides no benefit or worsens performance. These results confirm the conclusions from recent publications (Kuhn et al., 2019, 2020) that the 1VD may not be diagnostic of lithologic changes. What is particularly interesting, however, is that including the texture attributes only provides a minor benefit (compare Columns 2 and 4). While including the texture attributes is better than using the RTP 1VD, these results show that this is only marginally the case (compare Columns 3 and 4). So the question remains, are the texture attributes computed from the RTP 1VD impactful enough to justify their inclusion? Performance boosts of $< 1\%$ would suggest that the answer to this question is perhaps no. However, the fact that

the texture attributes are able to make an improvement (when the RTP 1VD could not) at least shows that the texture attributes can extract some meaningful information out of the RTP 1VD. I see this as a move in the right direction of finding new input features that could be useful for these types of problems. Perhaps it is worth exploring other types of local neighborhood, moving window-type features such as spatial autocorrelation (Anselin, 1995) or Tamura's texture features (Tamura et al., 1978), which is a subject of future work.

7.7 Conclusion

This paper presents a comprehensive study involving unsupervised, supervised, and semisupervised machine learning techniques for bedrock-lithology classification using a publicly available dataset from New South Wales, Australia. Using the geophysical data and geologic map provided for the study area, I compare the performance of supervised and semisupervised algorithms in three different scenarios where the known lithology data are varied in each situation. I also consider a fourth scenario where I perform an unsupervised data analysis to cluster the data. The value of clustering is assessed by comparing the recovered clusters to the known geologic map and to the previous supervised and semisupervised results. Using an existing reliable geologic map allows me to evaluate the results, which can help calibrate my understanding of which algorithms are preferable under different circumstances.

The outcomes of this work have several practical implications. When the training data are well distributed spatially, the performance improves and is comparable for both supervised and semisupervised algorithms. However, if the training data are spatially sparse, or localized, then semisupervised methods perform better. These valuable findings can help guide users toward supervised or semisupervised algorithms based on the training data situation in their own problems. The information entropy values that can be calculated from supervised or semisupervised class membership probabilities can also be used to guide decision-making. High entropy values indicate regions where there are uncertainties in the lithology predic-

tions that need refinement, and these regions could suggest focus areas for subsequent field campaigns. For this dataset, the supervised and semisupervised predictions can accurately distinguish between more lithologic units than the cluster results, but the clusters still provide valuable insights. In this case, the cluster map indicated heterogeneous regions that directly correlated with high entropy areas in the supervised and semisupervised predictions. So regardless of training data availability, I recommend performing an unsupervised analysis of data because it may reveal lithologies not previously seen in the training data, or it may help explain why misclassifications occur in supervised or semisupervised predictions.

Chapter 8

Conclusion

8.1 Overall discussion and recommendations

At the beginning of this thesis, there is a quote by R. Buckminster Fuller that reads "...doing more with less". This quote has several philosophical interpretations related to overconsumption and efficiency, but it also summarizes the theme of this thesis quite succinctly. The premise of semisupervised learning is to improve the classification of datasets with limited training data, i.e., doing more with less. Many geoscience applications are inherently challenged with limited data, three of which are considered in this thesis: well-log classification, seismic classification, and bedrock-lithology mapping. The primary research question that I seek to address with this work is if semisupervised algorithms can perform better than supervised methods in the context of these three examples. If semisupervised methods can *consistently* perform better across different datasets, this makes a stronger case for the robustness of the SSL algorithms in question and their broader applicability.

The results show that semisupervised methods can outperform state-of-the-art supervised learning approaches, such as XGBoost and LightGBM, when applied to various geophysical problems. In the well-log example from Chapter 3, LP does not outperform XGBoost, but the coupled self-training LP technique

surpasses the performance of XGBoost with visibly less chaotic predictions shown in the well views. The ssGMM technique in Chapter 4 also achieves a better performance and visibly less noisy predictions than XGBoost. A similar phenomenon is observed in the seismic classification example from Chapters 5 and 6. The results from Chapter 5 indicate that LP outperforms XGBoost and RNNs (Chapter 6), but only marginally so. Compared to LP, the coupled self-training LP method outperforms XGBoost and RNNs by a larger margin, and is also shown to be more robust to changes in training data. For the bedrock-lithology example in Chapter 7, the relative performance of LP and the supervised methods (XGBoost and LightGBM) depends on the spatial distribution of the training data. If the training data are spatially localized, then LP surpasses the performance of both SL methods.

In summary, the semisupervised methods are generally able to outperform the supervised techniques, but this is not *always* the case. When model assumptions are violated, such as when the seismic attributes are not de-trended in Chapter 5, LP performs just as poorly as XGBoost, and self-training further degrades the performance of LP. Before obtaining the results in Chapter 7, I was operating under the assumption that if a problem had limited training data, then supervised methods would be prone to overfitting, and semisupervised methods would produce better-generalized predictions. Much of the results in this thesis support this premise, but the findings from Chapter 7 suggest a noteworthy exception. Scenario C (Chapter 7.5.3) shows that even though the training data are limited, the performance between LP and the SL methods is still comparable. This phenomenon is attributed to the spatially dispersed sampling of the training data, which allows the distribution of each class to be better captured, facilitating the determination of decision boundaries. As stated in Chapter 7.5.3, this essentially turns the machine learning predictions into more of an interpolation-like problem because the training data are well constrained. In this situation, including the unlabeled data during training (i.e., SSL) appears to provide no additional benefit.

The exception above suggests that the criteria for determining whether SL or SSL methods would be preferred for a problem need to be revised (see Figure 8.1). If the training data are numerous, then SL

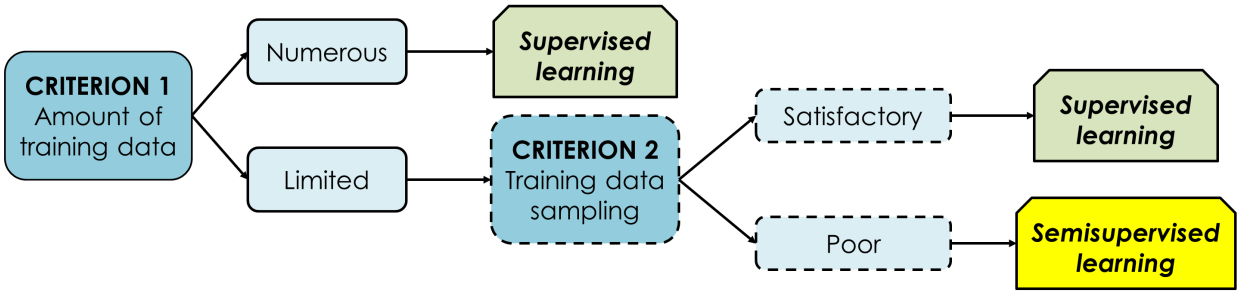

Figure 8.1: A high-level flowchart showing the criteria for determining whether supervised or semisupervised learning is preferred.

is undoubtedly favored. At the onset of this thesis, I assumed that SSL methods would be favorable if the training data were scarce. However, it seems that the training data sampling/distribution also has an influence. If the training data are sampled well, despite being few in number, supervised learning may still be appropriate. The results presented in this thesis indicate that SSL is preferred when the training data sampling is poor (e.g., training data coming from a single well, or spatially sparse measurements). If SSL algorithms are determined to be most suitable for a problem, the next question becomes which algorithm(s) to consider. Different semisupervised methods have different model assumptions (as discussed in Chapter 2.2), so the data need to be assessed to determine which assumptions are (in-)valid. For instance, many of the lithologic units in Chapter 7 are not Gaussian distributed, which violates assumptions for ssGMM, so this method is not considered for that problem. However, there may be systematic approaches to determine which SSL algorithms are best suited, which is a subject of future work (see below).

A secondary focus of this thesis is hyper-parameter estimation for SSL methods. As stated in Chapters 1.2 and 2.3, there is a lack of consensus in the literature on preferred approaches for determining hyper-parameters for SSL algorithms. Throughout the body chapters of this thesis, there are additional treatments for determining the best strategies for SSL hyper-parameter estimation, and I summarize them here. Chapter 3 shows that standard k-fold cross-validation can select optimal hyper-parameters (σ, α) for LP, but not always. However, fixed settings for self-training seem to work well. In Chapter 4, the standard k-fold cross-validation approach makes poor selections for ssGMM. I explore using repeated k-fold cross-validation

and a new selection strategy that also takes the standard deviation scores into account. The best ssGMM hyper-parameter selections are achieved using the new selection strategy, but the selections are still relatively unstable, likely because only the training data are used. Chapter 5 uses heuristic-based approaches for estimating the hyper-parameters for LP and default settings for self-training, and both are effective. Chapter 7 introduces the new semisupervised k-fold cross-validation technique, which is used to estimate α for LP (the number of nearest neighbors, p, is set using the heuristic from Chapter 5). This approach is inconclusive for two scenarios because the labeled data are highly correlated, but the semisupervised k-fold cross-validation works well for estimating α in Scenario C (Figure 7.14).

Several different approaches are explored for SSL hyper-parameter estimation, but a few techniques appear to be preferred. Chapters 3 and 4 show that hyper-parameter estimation using (supervised) k-fold cross-validation (Figure 2.10) can be unstable, and should perhaps be avoided if possible. The semisupervised k-fold cross-validation (Figure 2.11) technique looks promising, but it requires more testing to determine its applicability. The difficulty with this approach is the significant computational demand it requires; augmenting the validation fold with all the unlabeled data can drastically increase the computation time and memory requirements. However, this is not necessarily an issue for problems with fewer data. The well-log dataset used in Chapters 3 and 4 is relatively small, and while this is speculation, I expect that the semisupervised k-fold cross-validation approach would improve the stability of the SSL hyper-parameter estimation in those chapters. Another technique that works surprisingly well is the trial-and-error, heuristic-based approach. The default parameters for self-training work well in Chapters 3 and 5, and the heuristic scheme to determine p for LP is effective in Chapters 5 and 7. Given these observations, my recommendation is to use heuristics where possible, and then use semisupervised k-fold cross-validation to estimate any remaining hyper-parameters. Semisupervised k-fold cross-validation is expensive, so estimating some hyper-parameters with heuristics can actually reduce the overall computational burden. These recommendations may be most appropriate for the three SSL methods considered in this thesis, but they may still apply

to other SSL methods.

In many geoscience disciplines, semisupervised methods are rarely (or never) considered, as evidenced by the lack of published examples in the literature. The outcomes of this work help fill this gap, but they also raise the awareness of SSL methods. It is established that SSL is not appropriate for every problem, but this thesis shows that SSL can be an effective tool in certain situations. The far-reaching impact of this work is for other researchers to consider the applicability of SSL to their own problems, as well as have an understanding of how to estimate the hyper-parameters for SSL techniques.

8.2 Future work

This thesis explores applying SSL techniques to geoscience domains that have traditionally never considered SSL as an option. However, there are several directions to improve this work as well as challenges that remain to be reconciled. Some of the methods and concepts in this thesis are relatively new and continue to require additional testing. One example is the semisupervised k-fold cross-validation technique; this concept needs to be applied to more datasets to determine if it can improve the stability of hyper-parameter estimation for SSL methods compared to the standard (supervised) k-fold cross-validation. An ideal place to start would be with smaller problems, such as the well-log examples of Chapters 3 and 4, and using this new hyper-parameter estimation strategy on LP and ssGMM. Another method that needs further testing is self-training. Chapters 3 and 5 show that self-training is effective when applied to LP, but the impact that self-training has on LP performance in Chapter 7 is $\pm 1\%$ (these results are not shown). More work is needed to understand why self-training is not working for Chapter 7, but works well for the previous examples.

There are also several improvements needed for the ssGMM method, both critical and auxiliary. The input data (de-trended seismic attributes) in Chapter 5 are Gaussian distributed (see Figure 5.13b), so this would make ssGMM an ideal candidate to apply to this problem. However, when I try using ssGMM, I encounter several instabilities that I have been unable to resolve. Computer science is not my domain of

expertise, so perhaps collaborations with someone with this expertise could assist in resolving these instabilities. Once this issue is resolved, there are a few components to consider adding to ssGMM to improve its functionality. The code is currently limited to representing each class by a single multivariate Gaussian distribution, but there is the potential to extend the method to handle bimodal or multimodal distributions (i.e., more than one Gaussian distribution representing each class). Another feature that could improve ssGMM is incorporating transition probabilities from hidden Markov models. Several researchers (Lindberg & Grana, 2015; Feng et al., 2018a,b) apply this concept to unsupervised Gaussian mixture models to constrain the machine learning output by preventing geologically implausible predictions (e.g., brine-saturated sand predicted stratigraphically above gas-saturated sand). Given that these ideas have been applied to unsupervised GMMs, it is plausible to think they could also be extended to ssGMM.

Another concept that could build on this work is *meta-learning*, which refers to when a machine learning algorithm learns from the output of other machine learning algorithms (Feurer et al., 2015). This idea could be helpful to learn information from the data to indicate which SSL method(s) would be most appropriate. Recall that different SSL methods have different assumptions (e.g., manifold, smoothness, and cluster assumptions) about how the data are distributed. Knowing which assumptions are (in-)valid *before* choosing an SSL algorithm and carrying out a project could save time and resources. Li et al. (2019b) propose a meta-learning scheme that attempts to accomplish this task. First, they use a few different clustering algorithms that are sensitive to different unlabeled data distributions (e.g., k-means for recognizing the cluster assumption, spectral clustering for recognizing the manifold assumption, etc.). Next, they compute a series of metrics for each clustering algorithm. Then, the clustering algorithm that recovers the best metric scores can indicate which assumption is best realized, and by extension, which SSL algorithm is most suitable. This meta-learning approach proposed by Li et al. (2019b) could help determine which SSL methods to use instead of blindly testing a collection of algorithms.

Bibliography

Al-Bulushi, N., King, P. R., Blunt, M. J., & Kraaijveld, M., 2009. Development of artificial neural network models for predicting water saturation and fluid distribution, *Journal of Petroleum Science and Engineering*, **68**(4), 197–208.

Al-Bulushi, N. I., King, P. R., Blunt, M. J., & Kraaijveld, M., 2012. Artificial neural networks workflow and its application in the petroleum industry, *Neural Computing and Applications*, **21**(3), 409–421.

Aleardi, M. & Ciabarri, F., 2017. Application of different classification methods for litho-fluid facies prediction: a case study from the offshore Nile Delta, *Journal of Geophysics and Engineering*, **14**(5), 1087–1102.

Alfarraj, M. & AlRegib, G., 2019. Semisupervised sequence modeling for elastic impedance inversion, *Interpretation*, **7**(3), SE237–SE249.

Alpaydin, E., 2010. *Introduction to machine learning*, MIT Press, 2nd edn.

Anderson-Mayes, A.-M., 2002. Strategies to improve information extraction from multivariate geophysical data suites, *Exploration Geophysics*, **33**(2), 57–64.

Anselin, L., 1995. Local indicators of spatial association - LISA, *Geographical Analysis*, **27**(2), 93–115.

Araya-Polo, M., Dahlke, T., Frogner, C., Zhang, C., Poggio, T., & Hohl, D., 2017. Automated fault detection without seismic processing, *The Leading Edge*, **36**(3), 208–214.

Asghar, S., Choi, J., Yoon, D., & Byun, J., 2020. Spatial pseudo-labeling for semi-supervised facies classification, *Journal of Petroleum Science and Engineering*, **195**(1), 107834.

Aster, R. C., Borchers, B., & Thurber, C. H., 2005. *Parameter estimation and inverse problems*, Elsevier Academic Press.

Avseth, P., Mukerji, T., & Mavko, G., 2005. *Quantitative seismic interpretation: Applying rock physics tools to reduce interpretation risk*, Cambridge University Press.

Bagheri, M. & Riahi, M. A., 2015. Seismic facies analysis from well logs based on supervised classification scheme with different machine learning techniques, *Arabian Journal of Geosciences*, **8**(9), 7153–7167.

Baldwin, J. L., Bateman, R. M., & Wheatley, C. L., 1990. Application of a neural network to the problem of mineral identification from well-logs, *The Log Analyst*, **31**(5), 279–293.

Baranov, V., 1957. A new method for interpretation of aeromagnetic maps: Pseudo-gravimetric anomalies, *Geophysics*, **22**(2), 359–382.

Behnia, P., Harris, J. R., Rainbird, R. H., Williamson, M. C., & Sheshpari, M., 2012. Remote predictive mapping of bedrock geology using image classification of Landsat and SPOT data, western Minto Inlier, Victoria Island, Northwest Territories, Canada, *International Journal of Remote Sensing*, **33**(21), 6876–6903.

Benaouda, D., Wadge, G., Whitmarsh, R. B., Rothwell, R. G., & MacLeod, C., 1999. Inferring the lithology of borehole rocks by applying neural network classifiers to downhole logs: an example from the Ocean Drilling Program, *Geophysical Journal International*, **136**(2), 477–491.

Berge, T. B., Aminzadeh, F., de Groot, P., & Oldenziel, T., 2002. Seismic inversion successfully predicts reservoir, porosity, and gas content in Ibhubesi Field, Orange Basin, South Africa, *The Leading Edge*, **21**(4), 338–348.

Bestagini, P., Lipari, V., & Tubaro, S., 2017. A machine learning approach to facies classification using well logs, in *SEG Technical Program Expanded Abstracts 2017*, pp. 2137–2142, Society of Exploration Geophysicists.

Bishop, C. M., 2006. *Pattern recognition and machine learning*, Information Science and Statistics, Springer-Verlag.

Bishop, C. M., Svensén, M., & Williams, C. K. I., 1998. The generative topographic mapping, *Neural Computation*, **10**(1), 215–234.

Blakely, R. J., 1995. *Potential theory in gravity and magnetic applications*, Cambridge University Press.

Breiman, L., 1996. Bagging predictors, *Machine Learning*, **26**(2), 123–140.

Breiman, L., 2001. Random forests, *Machine Learning*, **45**(1), 5–32.

Brown, W. M., Gedeon, T. D., Groves, D. I., & Barnes, R. G., 2000. Artificial neural networks: A new method for mineral prospectivity mapping, *Australian Journal of Earth Sciences*, **47**(4), 757–770.

Camps-Valls, G., Marsheva, T. V. B., & Zhou, D., 2007. Semi-supervised graph-based hyperspectral image classification, *IEEE Transactions on Geoscience and Remote Sensing*, **45**(10), 3044–3054.

Carneiro, C. d. C., Fraser, S. J., Crósta, A. P., Silva, A. M., & Barros, C. E. d. M., 2012. Semiautomated geologic mapping using self-organizing maps and airborne geophysics in the Brazilian Amazon, *Geophysics*, **77**(4), K17–K24.

Carter-McAuslan, A. & Farquharson, C., 2021. Predictive geologic mapping from geophysical data using self-organizing maps: A case study from Baie Verte, Newfoundland, Canada, *Geophysics*, **86**(4), B249–B264.

Chapelle, O., Schölkopf, B., & Zien, A., 2006. *Semi-supervised learning*, MIT Press, Cambridge, MA, 1st edn.

Chen, C., Zhang, Q., Ma, Q., & Yu, B., 2019. LightGBM-PPI: Predicting protein-protein interactions through LightGBM with multi-information fusion, *Chemometrics and Intelligent Laboratory Systems*, **191**, 54–64.

Chen, T. & Guestrin, C., 2016. XGBoost: A scalable tree boosting system, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, ACM, San Francisco, California, USA.

Chen, X., Warner, T. A., & Campagna, D. J., 2007. Integrating visible, near-infrared and short-wave infrared hyperspectral and multispectral thermal imagery for geological mapping at Cuprite, Nevada, *Remote Sensing of Environment*, **110**(3), 344–356.

Choi, J., Kim, B., Kim, S., & Byun, J., 2017. Probabilistic facies analysis using 3D crossplot of stochastic forward-modeling results, in *SEG Technical Program Expanded Abstracts 2017*, pp. 3077–3081, Society of Exploration Geophysicists.

Coléou, T., Poupon, M., & Azbel, K., 2003. Unsupervised seismic facies classification: A review and comparison of techniques and implementation, *The Leading Edge*, **22**(10), 942–953.

Colquhoun, G. P., Hughes, K. S., Deyssing, L., Ballard, J. C., Folkes, C. B., Phillips, G., Troedson, A. L., & Fitzherbert, J. A., 2021. New South Wales Seamless Geology dataset, version 2.1 [Digital Dataset], Geological Survey of New South Wales, Department of Regional NSW, Maitland.

Cortes, C. & Vapnik, V., 1995. Support-vector networks, *Machine Learning*, **20**(3), 273–297.

Costa, I. S. L., Tavares, F. M., & de Oliveira, J. K. M., 2019. Predictive lithological mapping through

machine learning methods: a case study in the Cinzento Lineament, Carajás Province, Brazil, *Journal of the Geological Survey of Brazil*, **2**(1), 26–36.

Cracknell, M. J. & Reading, A. M., 2013. The upside of uncertainty: Identification of lithology contact zones from airborne geophysics and satellite data using random forests and support vector machines, *Geophysics*, **78**(3), WB113–WB126.

Cracknell, M. J. & Reading, A. M., 2014. Geological mapping using remote sensing data: A comparison of five machine learning algorithms, their response to variations in the spatial distribution of training data and the use of explicit spatial information, *Computers and Geosciences*, **63**, 22–33.

Cracknell, M. J., Reading, A. M., & McNeill, A. W., 2014. Mapping geology and volcanic-hosted massive sulfide alteration in the Hellyer–Mt Charter region, Tasmania, using random forests and self-organising maps, *Australian Journal of Earth Sciences*, **61**(2), 287–304.

Cracknell, M. J., Reading, A. M., & de Caritat, P., 2015. Multiple influences on regolith characteristics from continental-scale geophysical and mineralogical remote sensing data using Self-Organizing Maps, *Remote Sensing of Environment*, **165**, 86–99.

Cui, B., Xie, X., Hao, S., Cui, J., & Lu, Y., 2018. Semi-supervised classification of hyperspectral images based on extended label propagation and rolling guidance filtering, *Remote Sensing*, **10**(515), 1–18.

Cunha, A., Pochet, A., Lopes, H., & Gattass, M., 2020. Seismic fault detection in real data using transfer learning from a convolutional neural network pre-trained with synthetic seismic data, *Computers and Geosciences*, **135**, 1–9.

Das, V., Pollack, A., Wollner, U., & Mukerji, T., 2019. Convolutional neural network for seismic impedance inversion, *Geophysics*, **84**(6), R869–R880.

Davies, D. L. & Bouldin, D. W., 1979. A cluster separation measure, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-1**(2), 224–227.

de Matos, M. C., Osorio, P. L., & Johann, P. R., 2007. Unsupervised seismic facies analysis using wavelet transform and self-organizing maps, *Geophysics*, **72**(1), 9–21.

Dempster, A. P., Laird, N. M., & Rubin, D. B., 1977. Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society*, **39**(1), 1–38.

Dietterich, T. G., 2000. Ensemble methods in machine learning, in *Multiple Classifier Systems*, vol. 1857 of **Lecture Notes in Computer Science**, pp. 1–15, Springer Berlin Heidelberg.

Dubois, M. K., Byrnes, A. P., Bohling, G. C., & Doveton, J. H., 2006. Multiscale geologic and petrophysical modeling of the giant Hugoton Gas Field (Permian), Kansas and Oklahoma, U.S.A., in *Giant hydrocarbon reservoirs of the world: From rocks to reservoir characterization and modeling*, pp. 307–353, eds Harris, P. M. & Weber, L. J., AAPG Memoir 88/SEPM Special Publication.

Dubois, M. K., Bohling, G. C., & Chakrabarti, S., 2007. Comparison of four approaches to a rock facies classification problem, *Computers and Geosciences*, **33**(5), 599–617.

Duda, R. O., Hart, P. E., & Stork, D. G., 2001. *Pattern Classification*, John Wiley and Sons Inc., 2nd edn.

Dunham, M. W., 2021. Are seismic attributes still helpful for deep learning?, in *SEG Technical Program Expanded Abstracts 2021*, pp. 1561–1565, Society of Exploration Geophysicists.

Dunham, M. W., Malcolm, A., & Welford, J. K., 2020a. Improved well-log classification using semisupervised label propagation and self-training, with comparisons to popular supervised algorithms, *Geophysics*, **85**(1), O1–O15.

Dunham, M. W., Malcolm, A., & Welford, J. K., 2020b. Improved well log classification using semisu-

pervised Gaussian mixture models and a new hyper-parameter selection strategy, *Computers and Geosciences*, **140**, 1–12.

Dunham, M. W., Malcolm, A., & Welford, J. K., 2021. A seismic petrophysical classification study of the 2-D SEAM model using semisupervised techniques and detrended attributes, *Geophysical Journal International*, **227**(2), 1123–1142.

Eberle, D. G. & Paasche, H., 2012. Integrated data analysis for mineral exploration: A case study of clustering satellite imagery, airborne gamma-ray, and regional geochemical data suites, *Geophysics*, **77**(4), B167–B176.

Fatti, J. L., Smith, G. C., Vail, P. J., Strauss, P. J., & Levitt, P. R., 1994. Detection of gas in sandstone reservoirs using AVO analysis: A 3-D seismic case history using the Geostack technique, *Geophysics*, **59**(9), 1362–1376.

Fawcett, T., 2006. An introduction to ROC analysis, *Pattern Recognition Letters*, **27**(8), 861–874.

Fehler, M. & Keliher, P. J., 2011. *SEAM Phase 1: Challenges of subsalt imaging in Tertiary basins, with emphasis on deepwater Gulf of Mexico*, Society of Exploration Geophysicists.

Feng, R., Luthi, S. M., Gisolf, D., & Angerer, E., 2018a. Reservoir lithology classification based on seismic inversion results by Hidden Markov Models: Applying prior geological information, *Marine and Petroleum Geology*, **93**, 218–229.

Feng, R., Luthi, S. M., Gisolf, D., & Angerer, E., 2018b. Reservoir lithology determination by hidden Markov random fields based on a Gaussian mixture model, *IEEE Transactions on Geoscience and Remote Sensing*, **56**(11), 6663–6673.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F., 2015. Efficient and

robust automated machine learning, in *Advances in Neural Information Processing Systems*, vol. 28, pp. 1–9, Curran Associates, Inc.

Ford, A., Peters, K. J., Partington, G. A., Blevin, P. L., Downes, P. M., Fitzherbert, J. A., & Greenfield, J. E., 2019. Translating expressions of intrusion-related mineral systems into mappable spatial proxies for mineral potential mapping: Case studies from the Southern New England Orogen, Australia, *Ore Geology Reviews*, **111**, 102943.

Freund, Y. & Schapire, R. E., 1996. Experiments with a new boosting algorithm, in *Machine Learning: Proceedings of the 13th International Conference*, pp. 148–156, Morgan Kauffman, San Francisco, CA.

Friedman, J. H., 2001. Greedy function approximation: A gradient boosting machine, *The Annals of Statistics*, **29**(5), 1189–1232.

Gallop, J., 2006. Facies probability from mixture distributions with non-stationary impedance errors, in *SEG Technical Program Expanded Abstracts 2006*, pp. 1801–1805, Society of Exploration Geophysicists.

Gieseke, F., Airola, A., Pahikkala, T., & Kramer, O., 2014. Fast and simple gradient-based optimization for semi-supervised support vector machines, *Neurocomputing*, **123**, 23–32.

Gomez-Chova, L., Bruzzone, L., Camps-Valls, G., & Calpe-Maravilla, J., 2008a. Semi-supervised remote sensing image classification based on clustering and the mean map kernel, in *IEEE International Geoscience and Remote Sensing Symposium*, pp. IV391–IV394, IEEE.

Gomez-Chova, L., Camps-Valls, G., Munoz-Mari, J., & Calpe, J., 2008b. Semisupervised image classification with Laplacian support vector machines, *IEEE Geoscience and Remote Sensing Letters*, **5**(3), 336–340.

Gong, P., 1996. Integrated analysis of spatial data from multile sources: using evidential reasoning and

artificial neural network techniques for geological mapping, *Photogrammetric Engineering and Remote Sensing*, **62**(5), 513–523.

Goodfellow, I., Bengio, Y., & Courville, A., 2016. *Deep learning*, vol. 1, MIT Press.

Görnitz, N., Lima, L. A., Varella, L. E., Müller, K.-R., & Nakajima, S., 2018. Transductive regression for data with latent dependence structure, *IEEE Transactions on Neural Networks and Learning Systems*, **29**(7), 2743–2756.

Grana, D., Lang, X., & Wu, W., 2017. Statistical facies classification from multiple seismic attributes: comparison between Bayesian classification and Expectation-Maximization method and application in petrophysical inversion, *Geophysical Prospecting*, **65**(2), 544–562.

Grana, D., Azevedo, L., & Liu, M., 2020. A comparison of deep machine learning and Monte Carlo methods for facies classification from seismic data, *Geophysics*, **85**(4), WA41–WA52.

Graves, A., Mohamed, A., & Hinton, G., 2013. Speech recognition with deep recurrent neural networks, in *International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, IEEE, Vancouver, BC, Canada.

Gupta, V. K. & Ramani, N., 1980. Some aspects of regional-residual separation of gravity anomalies in a Precambrian terrain, *Geophysics*, **45**(9), 1412–1426.

Hall, B., 2016. Facies classification using machine learning, *The Leading Edge*, **35**(10), 906–909.

Hall, M. & Hall, B., 2017. Distributed collaborative prediction: Results of the machine learning contest, *The Leading Edge*, **36**(3), 267–269.

Hall-Beyer, M., 2017a. GLCM Texture: A Tutorial v. 3.0 March 2017.

Hall-Beyer, M., 2017b. Practical guidelines for choosing GLCM textures to use in landscape classification tasks over a range of moderate spatial scales, *International Journal of Remote Sensing*, **38**(5), 1312–1338.

Hand, D. J. & Till, R. J., 2001. A simple generalisation of the area under the ROC curve for multiple class classification problems, *Machine Learning*, **45**(2), 171–186.

Haralick, R. M., Shanmugam, K., & Dinstein, I., 1973. Textural features for image classification, *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-3**(6), 610–621.

Hardisty, R. & Wallet, B. C., 2017. Unsupervised seismic facies from mixture models to highlight channel features, in *SEG Technical Program Expanded Abstracts 2017*, pp. 2289–2293, Society of Exploration Geophysicists.

Harris, J. R. & Grunsky, E. C., 2015. Predictive lithological mapping of Canada's North using Random Forest classification applied to geophysical and geochemical data, *Computers and Geosciences*, **80**, 9–25.

Harris, J. R., Grunsky, E. C., He, J., Gorodetzky, D., & Brown, N., 2012. A robust, cross-validation classification method (RCM) for improved mapping accuracy and confidence metrics, *Canadian Journal of Remote Sensing*, **38**(1), 69–90.

Harris, J. R., He, J. X., Rainbird, R., & Behnia, P., 2014. A comparison of different remotely sensed data for classifying bedrock types in Canada's Arctic: Application of the robust classification method and random forests, *Geoscience Canada*, **41**(4), 557–584.

Hastie, T., Tibshirani, R., & Friedman, J., 2009. *The elements of statistical learning*, Springer Series in Statistics, Springer-Verlag, 2nd edn.

Hochreiter, S. & Schmidhuber, J., 1997. Long short-term memory, *Neural Computation*, **9**(8), 1735–1780.

Holden, E.-J., Dentith, M., & Kovesi, P., 2008. Towards the automated analysis of regional aeromagnetic data to identify regions prospective for gold deposits, *Computers and Geosciences*, **34**(11), 1505–1513.

Hood, S. B., Cracknell, M. J., Gazley, M. F., & Reading, A. M., 2019. Improved supervised classification of bedrock in areas of transported overburden: Applying domain expertise at Kerkasha, Eritrea, *Applied Computing and Geosciences*, **3-4**, 100001.

Huang, L., Dong, X., & Clee, T. E., 2017. A scalable deep learning platform for identifying geologic features from seismic attributes, *The Leading Edge*, **36**(3), 249–256.

Jessop, K., Daczko, N. R., & Piazolo, S., 2019. Tectonic cycles of the New England Orogen, Eastern Australia: A review, *Australian Journal of Earth Sciences*, **66**(4), 459–496.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y., 2017. LightGBM: A highly efficient gradient boosting decision tree, in *Advances in Neural Information Processing Systems*, vol. 30, pp. 3149–3157, Curran Associates, Inc.

Kettles, I. M., Rencz, A. N., & Bauke, S. D., 2000. Integrating Landsat, geologic, and airborne gamma ray data as an aid to surficial geology mapping and mineral exploration in the Manitouwadge area, Ontario, *Photogrammetric Engineering and Remote Sensing*, **66**(4), 437–445.

Keynejad, S., Sbar, M. L., & Johnson, R. A., 2019. Assessment of machine-learning techniques in predicting lithofluid facies logs in hydrocarbon wells, *Interpretation*, **7**(3), SF1–SF13.

Kim, Y., Hardisty, R., Torres, E., & Marfurt, K. J., 2018. Seismic-facies classification using random forest algorithm, in *SEG Technical Program Expanded Abstracts 2018*, pp. 2137–2142, Society of Exploration Geophysicists.

Kingma, D. P. & Ba, J., 2014. Adam: A method for stochastic optimization, *arXiv E-prints*, pp. 1–15.

Kireeva, N., Baskin, I. I., Gaspar, H. A., Horvath, D., Marcou, G., & Varnek, A., 2012. Generative topographic mapping (GTM): Universal tool for data visualization, structure-activity modeling and dataset comparison, *Molecular Informatics*, **31**(3), 301–312.

Kohonen, T., 1982. Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, **43**(1), 59–69.

Kohonen, T., 1998. The self-organizing map, *Neurocomputing*, **21**(1), 1–6.

Kohonen, T., 2013. Essentials of the self-organizing map, *Neural Networks*, **37**(1), 52–65.

Krizhevsky, A., Sutskever, I., & Hinton, G. E., 2012. ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems 25*, pp. 1097–1105, eds Pereira, F., Burges, C. J. C., Bottou, L., & Weinberger, K. Q., Curran Associates, Inc.

Krstajic, D., Buturovic, L. J., Leahy, D. E., & Thomas, S., 2014. Cross-validation pitfalls when selecting and assessing regression and classification models, *Journal of Cheminformatics*, **6**(10), 1–15.

Kuhn, S., Cracknell, M. J., & Reading, A. M., 2018. Lithologic mapping using Random Forests applied to geophysical and remote-sensing data: A demonstration study from the Eastern Goldfields of Australia, *Geophysics*, **83**(4), B183–B193.

Kuhn, S., Cracknell, M. J., & Reading, A. M., 2019. Lithological mapping in the Central African Copper Belt using Random Forests and clustering: Strategies for optimised results, *Ore Geology Reviews*, **112**, 103015.

Kuhn, S., Cracknell, M. J., Reading, A. M., & Sykora, S., 2020. Identification of intrusive lithologies in volcanic terrains in British Columbia by machine learning using random forests: The value of using a soft classifier, *Geophysics*, **85**(6), B249–B258.

Kumar, T., Seelam, N. K., & Rao, G. S., 2022. Lithology prediction from well log data using machine learning techniques: A case study from Talcher coalfield, Eastern India, *Journal of Applied Geophysics*, **199**, 104605.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P., 1998. Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, **86**(11), 2278–2324.

LeCun, Y., Bengio, Y., & Hinton, G., 2015. Deep learning, *Nature*, **521**, 436–444.

Lee, S., Choi, J., Yoon, D., & Byun, J., 2018. Automatic labeling strategy in semi-supervised seismic facies classification by integrating well logs and seismic data, in *SEG Technical Program Expanded Abstracts 2018*, pp. 2166–2170, Society of Exploration Geophysicists.

Leitch, E. C., 1974. The geological development of the southern part of the New England Fold Belt, *Journal of the Geological Society of Australia*, **21**(2), 133–156.

Lever, J., Krzywinski, M., & Altman, N., 2016. Point of significance: classification evaluation, *Nature Methods*, **13**(8), 603–604.

Leverington, D. W., 2010. Discrimination of sedimentary lithologies using Hyperion and Landsat Thematic Mapper data: a case study at Melville Island Canadian high arctic, *International Journal of Remote Sensing*, **31**(1), 233–260.

Li, G., Qiao, Y., Zheng, Y., Li, Y., & Wu, W., 2019a. Semi-supervised learning based on generative adversarial network and its applied to lithology recognition, *IEEE Access*, **7**, 67428–67437.

Li, J. & Castagna, J., 2004. Support vector machine (SVM) pattern recognition to AVO classification, *Geophysical Research Letters*, **31**(2), 1–4.

Li, Y. & Anderson-Sprecher, R., 2006. Facies identification from well logs: A comparison of discriminant analysis and naïve Bayes classifier, *Journal of Petroleum Science and Engineering*, **53**, 149–157.

Li, Y.-F., Wang, H., Wei, T., & Tu, W.-W., 2019b. Towards automated semi-supervised learning, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4237–4244.

Lima, L. A., Görnitz, N., Varella, L. E., Vellasco, M., Müller, K.-R., & Nakajima, S., 2017. Porosity estimation by semi-supervised learning with sparsely available labeled samples, *Computers and Geosciences*, **106**, 33–48.

Lindberg, D. V. & Grana, D., 2015. Petro-elastic log-facies classification using the Expectation-Maximization algorithm and Hidden Markov Models, *Mathematical Geosciences*, **47**, 719–752.

Liu, M., Jervis, M., Li, W., & Nivlet, P., 2020. Seismic facies classification using supervised convolutional neural networks and semisupervised generative adversarial networks, *Geophysics*, **85**(4), O47–O58.

Liu, W., He, J., & Chang, S.-F., 2010. Large graph construction for scalable semi-supervised learning, in *27th International Conference on Machine Learning*, pp. 679–686, Haifa, Israel.

Liu, W., Wang, J., & Chang, S.-F., 2012. Robust and scalable graph-based semisupervised learning, *Proceedings of the IEEE*, **100**(9), 2624 – 2638.

Liu, Y., Zhang, B., Wang, L.-m., & Wang, N., 2013. A self-trained semisupervised SVM approach to the remote sensing land cover classification, *Computers and Geosciences*, **59**(1), 98–107.

Liu, Z., Cao, J., Lu, Y., Chen, S., & Liu, J., 2019. A seismic facies classification method based on the convolutional neural network and the probabilistic framework for seismic attributes and spatial classification, *Interpretation*, **7**(3), SE225–SE236.

Livieris, I. E., Kanavos, A., Tampakas, V., & Pintelas, P., 2018. An ensemble SSL algorithm for efficient chest X-ray image classification, *Journal of Imaging*, **4**(7), 1–17.

Louboutin, M., Lange, M., Luporini, F., Kukreja, N., Witte, P. A., Herrmann, F. J., Velesko, P., & Gorman,

G. J., 2019. Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration, *Geoscientific Model Development*, **12**(3), 1165–1187.

MacQueen, J. B., 1967. Some methods for classification and analysis of multivariate observations, in *Proceedings of the Fifth Symposium on Math, Statistics, and Probability*, pp. 281–297, University of California Press, Berkeley, CA.

Maiti, S., Tiwari, R. K., & Kümpel, H.-J., 2007. Neural network modelling and classification of lithofacies using well log data: A case study from KTB borehole site, *Geophysical Journal International*, **169**(2), 733–746.

Malvić, T., Velić, J., Horváth, J., & Cvetković, M., 2010. Neural networks in petroleum geology as interpretation tools, *Central European Geology*, **53**(1), 97–115.

Martelet, G., Truffert, C., Tourlière, B., Ledru, P., & Perrin, J., 2006. Classifying airborne radiometry data with agglomerative hierarchical clustering: A tool for geological mapping in context of rainforest (French Guiana), *International Journal of Applied Earth Observation and Geoinformation*, **8**(3), 208–223.

McCormack, M. D., 1991. Neural computing in geophysics, *The Leading Edge*, **10**(1), 11–15.

Meng, Z., Merkurjev, E., Koniges, A., & Bertozzi, A. L., 2017. Hyperspectral image classification using graph clustering methods, *Image Processing On Line*, **7**, 218–245.

Mount, N. J. & Weaver, D., 2011. Self-organizing maps and boundary effects: quantifying the benefits of torus wrapping for mapping SOM trajectories, *Pattern Analysis and Applications*, **14**, 139–148.

Mukerji, T., Jørstad, A., Avseth, P., Mavko, G., & Granli, J. R., 2001. Mapping lithofacies and pore-fluid probabilities in a North Sea reservoir: Seismic inversions and statistical rock physics, *Geophysics*, **66**(4), 988–1001.

Navidi, W., 2010. *Statistics for engineers and scientists*, McGraw-Hill, 3rd edn.

Ng, A., Jordan, M., & Weiss, Y., 2002. On spectral clustering: analysis and an algorithm, in *Advances in Neural Information Processing Systems*, vol. 14, pp. 1–8, MIT Press.

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T., 2000. Text classification from labeled and unlabeled documents using EM, *Machine Learning*, **39**(2), 103–134.

Paasche, H. & Eberle, D. G., 2009. Rapid integration of large airborne geophysical data suites using a fuzzy partitioning cluster algorithm: a tool for geological mapping and mineral exploration targeting, *Exploration Geophysics*, **40**(3), 277–287.

Paisley, J. W., 2017. ColumbiaX Machine Learning Lecture 13: Boosting, in *CSMM.102x Machine Learning*.

Pascanu, R., Mikolov, T., & Bengio, Y., 2013. On the difficulty of training recurrent neural networks, in *30th International Conference on Machine Learning*, vol. 28 of **Proceedings of Machine Learning Research**, pp. 1310–1318, PMLR, Atlanta, Georgia, USA.

Pereira Leite, E. & de Souza Filho, C. R., 2009. Artificial neural networks applied to mineral potential mapping for copper-gold mineralizations in the Carajás Mineral Province, Brazil, *Geophysical Prospecting*, **57**(6), 1049–1065.

Pirkle, F. L., Howell, J. A., Wecksung, G. W., Duran, B. S., & Stablein, N. K., 1984. An example of cluster analysis applied to a large geologic data set: Aerial radiometric data from Copper Mountain, Wyoming, *Mathematical Geology*, **16**(5), 479–498.

Porwal, A., Carranza, E. J. M., & Hale, M., 2003. Artificial neural networks for mineral-potential mapping: A case study from Aravalli Province, Western India, *Natural Resources Research*, **12**(3), 155–171.

Qi, J., Lin, T., Zhao, T., Li, F., & Marfurt, K., 2016. Semisupervised multiattribute seismic facies analysis, *Interpretation*, **4**(1), SB91–SB106.

Radford, D. D. G., Cracknell, M. J., Roach, M. J., & Cumming, G. V., 2018. Geological mapping in Western Tasmania using radar and random forests, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, **11**(9), 3075–3087.

Raeesi, M., Moradzadeh, A., Ardejani, F. D., & Rahimi, M., 2012. Classification and identification of hydrocarbon reservoir lithofacies and their heterogeneity using seismic attributes, logs data and artificial neural networks, *Journal of Petroleum Science and Engineering*, **82-83**(1), 151–165.

Riese, F. M., Keller, S., & Hinz, S., 2020. Supervised and semi-supervised self-organizing maps for regression and classification focusing on hyperspectral data, *Remote Sensing*, **12**(7), 1–23.

Roberts, J. & Engel, B. A., 1987. Depositional and tectonic history of the southern New England Orogen, *Australian Journal of Earth Sciences*, **34**(1), 1–20.

Roden, R., Smith, T., & Sacrey, D., 2015. Geologic pattern recognition from seismic attributes: Principal component analysis and self-organizing maps, *Interpretation*, **3**(4), SAE59–SAE83.

Rogers, S. J., Fang, J. H., Karr, C. L., & Stanley, D. A., 1992. Determination of lithology from well logs using a neural network, *AAPG Bulletin*, **76**(5), 731–739.

Rosenberg, C., Hebert, M., & Schneiderman, H., 2005. Semi-supervised self-training of object detection models, in *2005 Seventh IEEE Workshops on Applications of Computer Vision*, vol. 1, pp. 29–36, IEEE.

Ross, C. P. & Cole, D. M., 2017. A comparison of popular neural network facies-classification schemes, *The Leading Edge*, **36**(4), 340–349.

Rousseeuw, P. J., 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, *Journal of Computational and Applied Mathematics*, **20**, 53–65.

Roy, A., Dowdell, B. L., & Marfurt, K. J., 2013. Characterizing a Mississippian tripolitic chert reservoir

using 3D unsupervised and supervised multiattribute seismic facies analysis: An example from Osage County, Oklahoma, *Interpretation*, **1**(2), SB109–SB124.

Roy, A., Romero-Peláez, A. S., Kwiatkowski, T. J., & Marfurt, K. J., 2014. Generative topographic mapping for seismic facies estimation of a carbonate wash, Veracruz Basin, southern Mexico, *Interpretation*, **2**(1), SA31–SA47.

Saggaf, M. M. & Nebrija, E. L., 2000. Estimation of lithologies and depositional facies from wire-line logs, *AAPG Bulletin*, **84**(10), 1633–1646.

Saggaf, M. M., Toksöz, M. N., & Marhoon, M. I., 2003. Seismic facies classification and identification by competitive neural networks, *Geophysics*, **68**(6), 1984–1999.

Sak, H., Senior, A., & Beaufays, F., 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling, in *Interspeech 2014*, pp. 338–342, International Speech Communication Association, Singapore.

Shi, Y., Wu, X., & Fomel, S., 2019. SaltSeg: Automatic 3D salt segmentation using a deep convolutional neural network, *Interpretation*, **7**(3), SE113–SE122.

Sigdel, M., Dinç, I., Dinç, S., Sigdel, M. S., Pusey, M. L., & Aygün, R. S., 2014. Evaluation of semi-supervised learning for classification of protein crystallization imagery, in *IEEE Southeastcon 2014*, pp. 1–6, Lexington, KY, USA.

Smith, G. C. & Gidlow, P. M., 1987. Weighted stacking for rock property estimation and detection of gas, *Geophysical Prospecting*, **35**(9), 993–1014.

Souza, J. F. L., Santos, M. D., Magalhães, R. M., Neto, E. M., Oliveira, G. P., & Roque, W. L., 2019. Automatic classification of hydrocarbon “leads” in seismic images through artificial and convolutional neural networks, *Computers and Geosciences*, **132**, 23–32.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research*, **15**(1), 1929–1958.

Strecker, U. & Uden, R., 2002. Data mining of 3D poststack seismic attribute volumes using Kohonen self-organizing maps, *The Leading Edge*, **21**(10), 1032–1037.

Sun, X., Liu, M., & Sima, Z., 2020. A novel cryptocurrency price trend forecasting model based on LightGBM, *Finance Research Letters*, **32**, 101084.

Sutskever, I., Vinyals, O., & Le, Q. V., 2014. Sequence to sequence learning with neural networks, in *Advances in Neural Information Processing Systems 27*, pp. 3104–3112.

Tamayo, D., Silburt, A., Valencia, D., Menou, K., Ali-Dib, M., Petrovich, C., Huang, C. X., Rein, H., van Laerhoven, C., Paradise, A., Obertas, A., & Murray, N., 2016. A machine learns to predict the stability of tightly packed planetary systems, *The Astrophysical Journal Letters*, **832**(2), 1–5.

Tamura, H., Mori, S., & Yamawaki, T., 1978. Textural features corresponding to visual perception, *IEEE Transactions on Systems, Man, and Cybernetics*, **8**(6), 460–473.

Taner, M. T. & Sheriff, R. E., 1977. *Seismic stratigraphy - applications to hydrocarbon exporation*, chap. Application of amplitude, frequency, and other attributes to stratigraphic and hydrocarbon determination, pp. 301–328, American Association of Petroleum Geologists Memoir 26.

Taner, M. T., Koehler, F., & Sheriff, R. E., 1979. Complex seismic trace analysis, *Geophysics*, **44**(6), 1041–1063.

Theodoridis, S., 2015. *Machine learning: a Bayesian and optimization perspective*, Academic Press Inc.

Torlay, L., Perrone-Bertolotti, M., Thomas, E., & Baciu, M., 2017. Machine learning - XGBoost analysis of language networks to classify patients with epilepsy, *Brain Informatics*, **4**(3), 159–169.

Tuia, D. & Camps-Valls, G., 2009. Semisupervised remote sensing image classification with cluster kernels, *IEEE Geoscience and Remote Sensing Letters*, **6**(2), 224–228.

van Engelen, J. E. & Hoos, H. H., 2020. A survey on semi-supervised learning, *Machine Learning*, **109**, 373–440.

Vatsavai, R. R., Shekhar, S., & Burk, T. E., 2005. A semi-supervised learning method for remote sensing data mining, in *17th IEEE International Conference on Tools with Artificial Intelligence*, pp. 1–5, IEEE.

Vesanto, J. & Alhoniemi, E., 2000. Clustering of the self-organizing map, *IEEE Transactions on Neural Networks*, **11**(3), 586–600.

Vocaturo, E., Perna, D., & Zumpano, E., 2019. Machine learning techniques for automated melanoma detection, in *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 2310–2317.

von Luxburg, U., 2007. A tutorial on spectral clustering, *Statistics and Computing*, **17**, 395–416.

Waldeland, A. U., Jensen, A. C., Gelius, L.-J., & Solberg, A. H. S., 2018. Convolutional neural networks for automated seismic interpretation, *The Leading Edge*, **37**(7), 529–537.

Wallet, B. C. & Hardisty, R., 2019. Unsupervised seismic facies using Gaussian mixture models, *Interpretation*, **7**(3), SE93–SE111.

Wallet, B. C., de Matos, M. C., Kwiatkowski, J. T., & Suarez, Y., 2009. Latent space modeling of seismic data: an overview, *The Leading Edge*, **28**(12), 1454–1459.

Wang, D., Zhang, Y., & Zhao, Y., 2017. LightGBM: An effective miRNA classification method in breast cancer patients, in *Proceedings of the 2017 International Conference on Computational Biology and Bioinformatics*, ICCBB 2017, pp. 7–11, Association for Computing Machinery, New York, NY, USA.

Wang, G., Carr, T. R., Ju, Y., & Li, C., 2013. Identifying organic-rich Marcellus Shale lithofacies by support vector machine classifier in the Appalachian Basin, *Computers and Geosciences*, **64**, 52–60.

Wang, M. & Wu, Q., 2017. Research of advanced GTM and is application to gas-oil reservoir identifcation, *International Journal of Pattern Recognition and Artificial Intelligence*, **31**(5), 1–18.

Wang, Z., Zuo, R., & Liu, H., 2021. Lithological mapping based on fully convolutional network and multi-source geological data, *Remote Sensing*, **13**(23), 4860.

Waske, B., Benediktsson, J. A., Árnason, K., & Sveinsson, J. R., 2009. Mapping of hyperspectral AVIRIS data using machine-learning algorithms, *Canadian Journal of Remote Sensing*, **35**(1), S106–S116.

Weihermann, J. D., Ferreira, M. P., de Castro, L. G., Ferreira, F. J. F., & Silva, A. M., 2021. Retrieving geological units with unsupervised clustering of gamma-ray spectrometry data, *Journal of Applied Geophysics*, **184**, 104225.

West, B. P., May, S. R., Eastwood, J. E., & Rossen, C., 2002. Interactive seismic facies classification using textural attributes and neural networks, *The Leading Edge*, **21**(10), 1042–1049.

Weston, J., Leslie, C., Ie, E., Zhou, D., Elisseeff, A., & Noble, W. S., 2005. Semi-supervised protein classification using cluster kernels, *Bioinformatics*, **21**(15), 3241–3247.

Wu, G., Chen, G., Cheng, Q., Zhang, Z., & Yang, J., 2021. Unsupervised machine learning for lithological mapping using geochemical data in covered areas of Jining, China, *Natural Resources Research*, **30**(2), 1053–1068.

Xing, X., Yu, Y., Jiang, H., & Du, S., 2013. A multi-manifold semi-supervised Gaussian mixture model for pattern classification, *Pattern Recognition Letters*, **34**(16), 2118–2125.

Xiong, W., Ji, X., Ma, Y., Wang, Y., AlBinHassan, N. M., Ali, M. N., & Luo, Y., 2018. Seismic fault detection with convolutional neural network, *Geophysics*, **83**(5), 97–103.

Yan, H., Zhou, J., & Pang, C. K., 2017. Gaussian mixture model using semisupervised learning for probabilistic fault diagnosis under new data categories, *IEEE Transactions on Instrumentation and Measurement*, **66**(4), 723–733.

Yarowsky, D., 1995. Unsupervised word sense disambiguation rivaling supervised methods, in *33rd Annual Meeting of the Association for Computational Linguistics*, pp. 189–196.

Yu, L., Porwal, A., Holden, E.-J., & Dentith, M. C., 2012. Towards automatic lithological classification from remote sensing data using support vector machines, *Computers and Geosciences*, **45**, 229–239.

Zhang, D., Qian, L., Mao, B., Huang, C., Huang, B., & Si, Y., 2018a. A data-driven design for fault detection of wind turbines using random forests and XGboost, *IEEE Access*, **6**, 21020–21031.

Zhang, G., Wang, Z., & Chen, Y., 2018b. Deep learning for seismic lithology prediction, *Geophysical Journal International*, **215**(2), 1368–1387.

Zhang, K., Kwok, J. T., & Parvin, B., 2009. Prototype vector machine for large scale semi-supervised learning, in *26th International Conference on Machine Learning*, pp. 1233–1240, Montreal, Canada.

Zhang, X. & Lee, W., 2006. Hyperparameter learning for graph based semi-supervised learning algorithms, in *Advances in Neural Information Processing Systems*, vol. 19, pp. 1–8.

Zhao, T., 2018. Seismic facies classification using different deep convolutional neural networks, in *SEG Technical Program Expanded Abstracts 2018*, pp. 2046–2050, Society of Exploration Geophysicists.

Zhao, T., Jayaram, V., Roy, A., & Marfurt, K. J., 2015. A comparison of classification techniques for seismic facies recognition, *Interpretation*, **3**(4), SAE29–SAE58.

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., & Schölkopf, B., 2004. Learning with local and global consistency, in *Advances in Neural Information Processing Systems*, vol. 16, pp. 321–328, MIT Press.

Zhu, X. & Goldberg, A. B., 2009. *Introduction to semi-supervised learning*, Synthesis lectures on artificial intelligence and machine learning, Morgan & Claypool.