Rock Facies Characterization Using Machine Learning Algorithms

by

Zhili Wei

A dissertation submitted to the Department of Earth and Atmospheric Sciences,
College of Natural Sciences and Mathematics
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Geophysics

Chair of Committee: Hua-wei Zhou

Committee Member: Yingcai Zheng

Committee Member: Jonny Wu

Committee Member: August Lau

University of Houston

December 2019

ACKNOWLEDGMENTS

My special thanks go to my advisor, Dr. Huawei Zhou, for his guidance and support throughout this endeavor, and for his willingness to share the wisdom of life with me. My sincerest appreciation goes to Dr. Yingcai Zheng, Dr. Jonny Wu, and Dr. August Lau for their numerous invaluable contributions to this dissertation. I would also like to thank Dr. Hao Hu for all his help and encouragement when frustrations or setbacks occurred. I also want to thank all the people in our group for their technical efforts and suggestions. Last but certainly not least, my deepest gratitude goes to my parents for their constant encouragement and total support.

ABSTRACT

In the upstream field of exploration and production of hydrocarbons, the characterization of rock facies is critical for estimating rock physical properties, such as porosity and permeability, and for reservoir detection and simulation. The precise identification of rock properties is closely related to the net pay thickness determination of reservoirs, and is thus a definitive factor in the drilling decision-making process.

In this dissertation, I applied five different machine learning algorithms to characterize rock facies with various techniques and strategies using a field dataset. The input dataset is acquired from the Panoma gas field in southwest Kansas. It contains five wireline logs and two geological indicators with a corresponding rock facies type at a sampling interval of half a foot. The total dataset has nine different rock facies. Besides exact facies there are also adjacent facies (facies with similar features that can be considered as the same one) used in classification tasks. The machine learning algorithms I used are Support Vector Machine (SVM), k-Nearest Neighbors (KNN), Decision Trees, Artificial Neural Network (ANN), and Convolutional Neural Network (CNN). Under each methodology, I also tested the method with various techniques including feature engineering, regularization, and padding strategy, in order to improve the final results. I created two benchmark datasets for training and predicting, respectively, and all the algorithms are trained and predicted on the same dataset. The results were evaluated by the F-1 score, where F-1 is a metric used to quantify the average prediction accuracy.

The results show that, for classifying exact rock facies, the Decision Trees had the best performance with a 0.66 F-1 score. For classifying adjacent facies, KNN with feature engineering technique achieved the highest F-1 score, 0.94. For each specific facies, facies #9 (Phylloidalgal bafflestone) has the highest averaged F-1 score. Facies #5 (mudstone-limestone) has the lowest averaged F-1 score which indicates it is the hardest facies to classify. My results indicate that machine learning algorithms have great application potential in automatic rock facies characterization with high accuracy and efficiency. It could significantly enhance the rock physical property estimation process and meanwhile reduce manual effort.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction of Dataset and Data Statistic, Preprocessing and Preparing

1.1 Introduction

Facies, as a geologic term, is defined as a way to distinguish bodies of rock into mappable units in terms of physical properties, composition, formation, or various other attributes (Moore and Liou 1979). There are many types of facies; in this dissertation, facies represent the lithofacies. Facies classification is an essential element of geologic investigations and involves using recorded attributes and measurements to assign a class or type to rock samples. Defining a rock facies precisely can contribute to a better understanding of the depositional environment penetrated by a wellbore. Although core samples of reservoir rocks are the ideal source of lithofacies information, cores are not commonly accessible due to high cost and limited availability compared to the number of wells in the field. Therefore, developing an efficient and accurate method for classifying lithofacies in wells without cores is valuable. It is a common way to extrapolate rock lithofacies from the cored wells to the uncored wells by a comparison of wire-line loggings measurements of petrophysical properties. However, since the number of wells could be numerous, manually classifying lithofacies for each single well could be time-consuming and even unfeasible. While this classification process can be accelerated by applying cutoff rules to wire-line log curves by facies and cross-plots of log curves (see Figure 1.1), it continues to be inefficient.

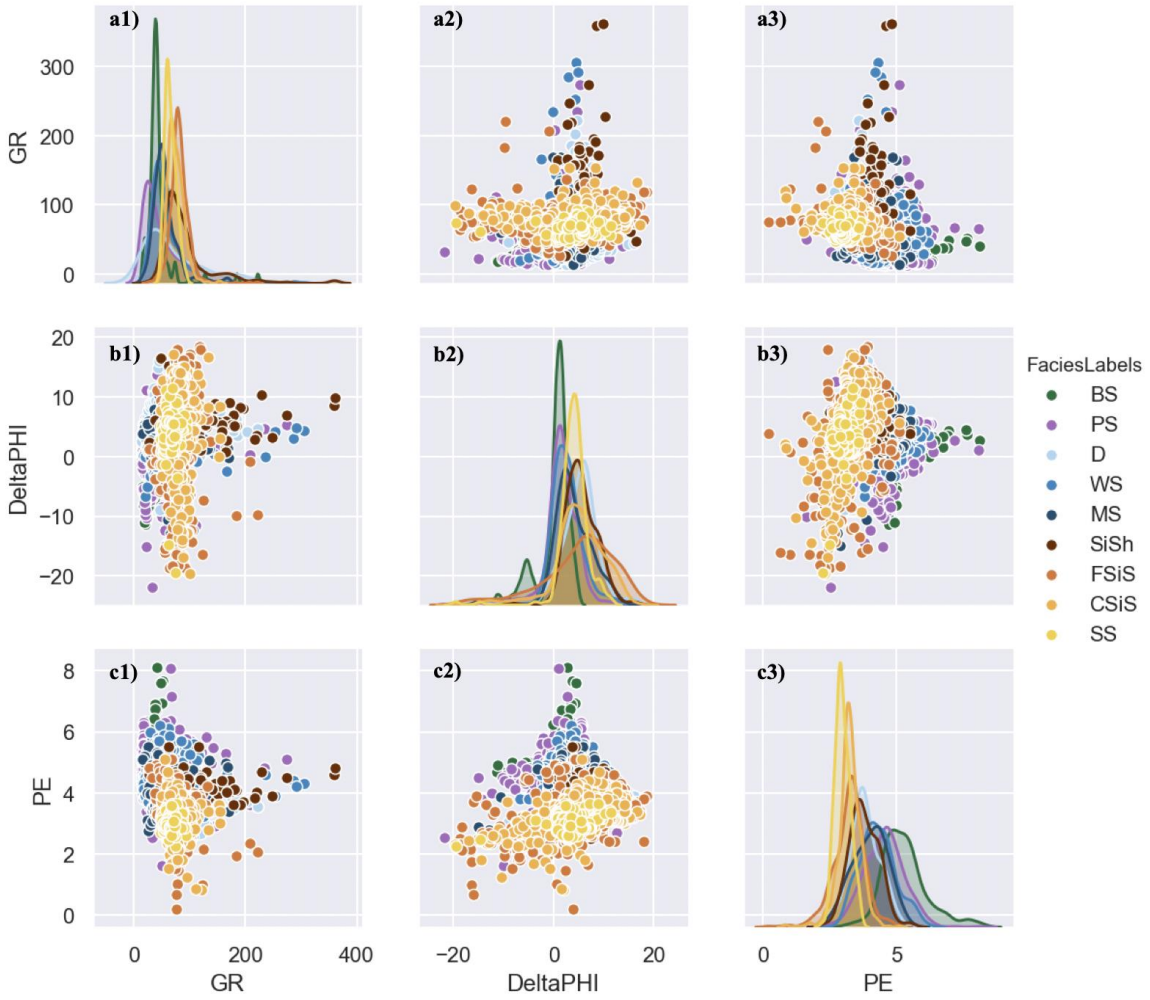

Figure 1. 1 Cross-plots of three wireline logs (gamma ray (GR), neutron-density porosity difference (DELTAPHI) and photoelectric effect (PE)) of well "SHRIMPLIN". Each color represents a specific facies class, and information of all lithofacies is shown in Table 1. The diagonal panels are the facies distribution over one specific wireline log. Facies groups usually overlap with each other and form crowded clouds with fuzzy boundaries, which make it hard to interpret. Facies labels represent specific facies that can be found details in table 1.1.

Accurate classification of lithofacies and their appropriate representation in a 3D cellular geologic model is critical to understanding the field because permeability and fluid saturations for a given porosity and height above free water vary considerably among lithofacies (Dubois et al., 2006). Because the availability of core is limited compared to the number of wells in the field, a method for estimating lithofacies in wells without cores is necessary. In this case, core

lithofacies are extrapolated from the cored wells (training wells) to the uncored wells through comparison of physical rock properties measured by wire-line logs.

The conventional method of manually assigning lithofacies in half-foot (0.15 m) segments on a well-by-well basis is impractical in the Panoma modeling project due to the immense volume being considered (hundreds of wells sampling a 250-foot (75 m) interval, Figure 1.2). Applying cutoff rules to wire-line log curves by facies and cross plots of log curve data can augment the process, but it is a very tedious and time-consuming process.

At the early stage of facies classification study, much work (Wolf and Pelissier-Combescure 1982; Gill et al. 1993; Kapur et al. 1998) has focused on using multivariate statistical approaches to solve the problem of facies classification along with wire-line log measurements. In the 1980s and early 1990s, some approaches utilized the idea of grouping and clustering of similar features for facies classification. For example, Delfiner et al. (1987) and Busch et al. (1987) applied a discriminant function analysis to estimate facies. Gill et al. (1993) used multivariate clustering and correlation of zones between wells to determine facies.

Meanwhile, many machine learning (ML) algorithms, such as neural networks, were developed and used in the computer vision industry in applications such as facial recognition and object detection (Lawrence et al. 1997). Rumelhart et al. (1986) proposed the backpropagation algorithm for the multilayer perceptron based on the forward propagation of neural networks. The backpropagation process continually adjusts the weights and thresholds between neurons until output errors are reduced to within an allowable range, or until a predetermined number of training iterations is reached.

Owing to the fast development of both computational hardware performance and software algorithms, ML has become a productive trend for the oil and gas industry. Due to the capability

of ML solving the problem of nonlinear classification, considerable research efforts have been devoted to applying ML to rock facies classification. Baldwin et al. (1990) applied a neural network to identify minerals from well logs. Rogers et al. (1992) and Kapur et al. (1998) used a neural network to predict facies from the core and log data. Cuddy (1997) did a comprehensive review of the application of the mathematics of fuzzy logic to petrophysics, while Dubois et al. (2007) compared four ML approaches on rock facies classification.

In 2016, the Society of Exploration Geophysics (SEG) hosted an ML contest to characterize rock facies and published a field dataset (Hall 2016; Hall and Hall 2017). In this dataset, each sample contains seven features and a known facies type. This makes it a verifiable dataset for evaluating the performance of ML classifications. The contest received about 300 entries from individuals or groups all over the world in four months. Those talented researchers have submitted many different machine learning classifiers to the contest and improved the results a lot compared to the initial submission from Hall (2016). The winner of the contest, LA Team, achieved a median accuracy of 0.6388. Their solution, along with all the submissions, is available online at https://github.com/seg/2016-ml-contest.

In this dissertation, I will first split the SEG dataset into training and testing datasets, and then test five ML algorithms on them. Although the SEG contest results can be a good external reference, since I split the data differently than the SEG contest, it may not be appropriate to compare my results with the SEG contest results directly. ML algorithms I used include the Support Vector Machine, k-Nearest Neighbors, Decision Trees, Artificial Neural Network, and Convolutional Neural Network. I tested the performance of each algorithm of classifying rock facies and applied different techniques and strategies to enhance the performance. Finally, I will compare and analyze the results for future work. Some of this work was published in the

journal Pure and Applied Geophysics (Wei et al. 2019). That work is shown in Chapter 7 of this dissertation.

1.2 Dataset Overview

In this section, I will review our dataset in three major ways: research area geological setting, rock facies classes, and rock facies petrophysical features.

1.2.1 Research Area Geological Setting

In this section, I first briefly overview the geological setting of the test site. Figure 1.2 shows a map of the Hugoton and Panoma fields, the largest region of gas production in North America, located in southwest Kansas (Dubois et al. 2007). Modeling the gas reservoir in the Panoma Field is part of a long-term project of the Kansas Geological Survey and an industry consortium, studying the Hugoton and Panoma fields in southwest Kansas and northwest Oklahoma. Having produced 963 billion m^3 (34 tcf) gas from 412,000 wells, the fields constitute the largest gas-producing area in North America. Gas from the Panoma is produced from the upper seven marine (M)-continental sedimentary cycles of the Permian, Wolfcampian, Council Grove Group, and classifying these rocks is the subject of this research.

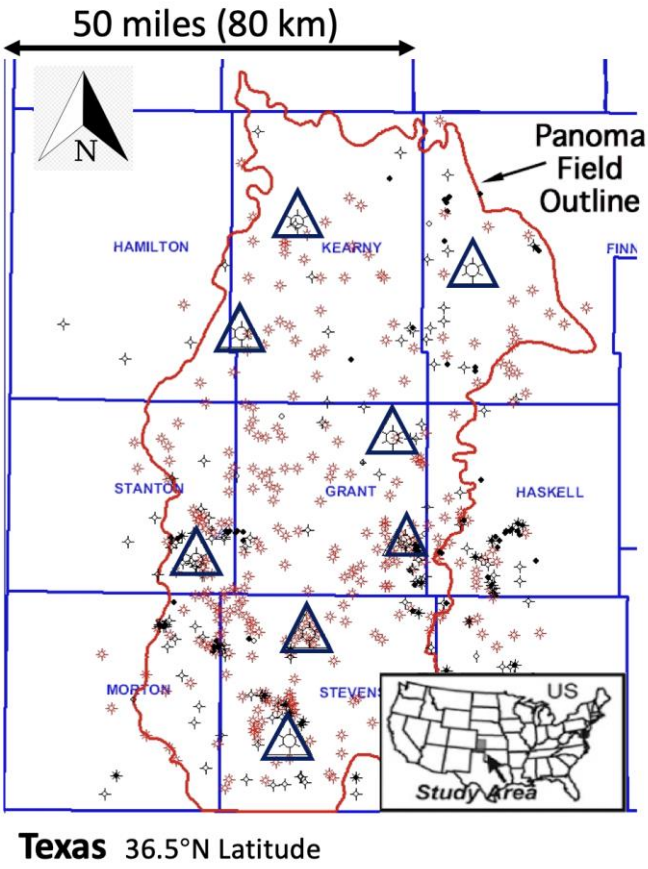

Figure 1. 2 Study area map. Locations of eight wells in our dataset are marked by large triangles and 515 wells without lithofacies information are marked as red or black stars (Dubois et al. 2004).

1.2.2 Rock Facies Classes

The dataset acquired from Hugoton and Panoma fields consists of five wireline log measurements, two indicator variables, and a facies label sampling at every half-foot (0.15 m). In this dataset, sedimentary rocks are subdivided into nine rock facies classes shown in Table 1.1.

Table 1. 1 Nine classes of lithofacies and corresponding abbreviated labels.

Facies label	1	2	3	4	5
Abbreviated Label	SS	CSiS	FSiS	SiSh	MS
Facies	Nonmarine sandstone	Nonmarine coarse siltstone	Nonmarine fine siltstone	Marine siltstone and shale	Mudstone (limestone)

Facies label	6	7	8	9
Abbreviated Label	WS	D	PS	BS
Facies	Wackestone (limestone)	Dolomite	Packstone-grainstone (limestone)	Phylloidalgal bafflestone (limestone)

Sedimentary rocks in this study are subdivided into nine rock facies (classes), and the term facies as used throughout this paper means lithofacies (lithology). Facies were distinguished based on rock type (siliciclastic or carbonate) and texture (Folk (1954), grain size for siliciclastics, and Dunham (1962), classification for carbonates), and they represent a continuum that can be lumped or split in many ways. We settled on nine classes (facies) by balancing three objectives: (1) maximum number of facies distinguishable by petrophysical wire-line log curves; (2) minimum number of facies needed to precisely represent the physical variability of the reservoir; and (3) class distinction by core petrophysical properties (porosity-permeability relationships and capillary pressure). Three facies are of continental origin, sandstone (SS), coarse siltstone (CSiS) and shaley fine siltstone (FSiS), and six are of M origin, M siltstone (SiSh), carbonate mudstone (MS), wackestone (WS), fine-crystalline dolomite (D), packstone (PS), and grainstone (BS). Facies codes generally, but not perfectly, reflect the position in the continuum of facies and increasing permeability for a given porosity for M facies codes 4–9. In the non-marine (NM) realm, CSiS generally has higher permeability than FSiS

for a given porosity. Adjacent facies codes are generally close neighbors in terms of petrophysical properties.

Because facies boundaries are continuous rather than discrete, measured properties for different classes overlap in the feature space. As shown in Table 1.2, I created a list of adjacent facies because the facies codes do not entirely reflect a continuum. Each facies class has adjacent facies which are considered to be the same. I use this standard to evaluate the adjacent facies prediction results.

Table 1. 2 Facies and adjacent facies labels. These facies are not discrete and gradually blend into one another. Some have neighboring facies that are rather close. The table lists the facies, their abbreviated labels, and their approximate neighbors.

Facies	1	2	3	4	5	6	7	8	9
Abbreviated Label	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS
Adjacent Facies	2	1,3	2	5	4,6	5,7	6,8	6,7,9	7,8

1.2.3 Rock Facies Petrophysical Features

Digital measurements were recorded at half-foot intervals from devices of a variety of physical properties. As shown in Figure 1.3, we utilize five feature elements derived from wire-line log measurements, including natural Gamma Ray (GR), neutron and density porosity average (PHI), neutron porosity and density porosity difference (DeltaPHI), photoelectric effect (PE) and apparent true resistivity log base (ILD_log10).

Gamma Ray (GR) measures γ-ray emissions from radioactive formations. Different formations will have different γ-ray signatures. Gamma Ray Logs are often used for correlation, shale content evaluation, and mineral analysis. Resistivity (ILD_log10) measures

the ability of the subsurface materials to resist or inhibit electrical conduction. Photoelectric effect (PE) measures the emission of electrons from formations when shooting with incident photons. Average neutron-density porosity (PHIND) measures a formation's porosity by analyzing neutron energy losses in porous formations. Neutron energy loss will occur mostly in the part of the formation that has the highest hydrogen concentration. Neutron-density porosity difference (DeltaPHI) provides porosity difference in formation based on neutron logs.

I mentioned in section 1.2.1 that my study area is a gas field. Therefore, the logs are susceptible to fluid effects. In particular, the neutron, density, delta-phi, and resistivity logs values will be affected. To allow my study to be relevant for quick analysis of large well log datasets, here I make no attempt to filter the fluid-related effects. A similar approach was taken by the SEG machine learning contestants (Hall 2016; Wei et al. 2019).

Two crucial additional feature elements derived from geologic data incorporate geologic knowledge in the variable mix. A nonmarine-marine (NM–M) indicator variable is assigned to each sample interval, which indicates fundamentally different depositional environments for sample sedimentary rocks. It is a binary value with 1 and 2 and based on human interpretation. Relative position (RELPos) is the position of a sample concerning the base of its respective formation segment. These two essential geologic constraining variables provide new attributes from depositional environments and vertical stacking patterns aspects and both play important roles in facies classification.

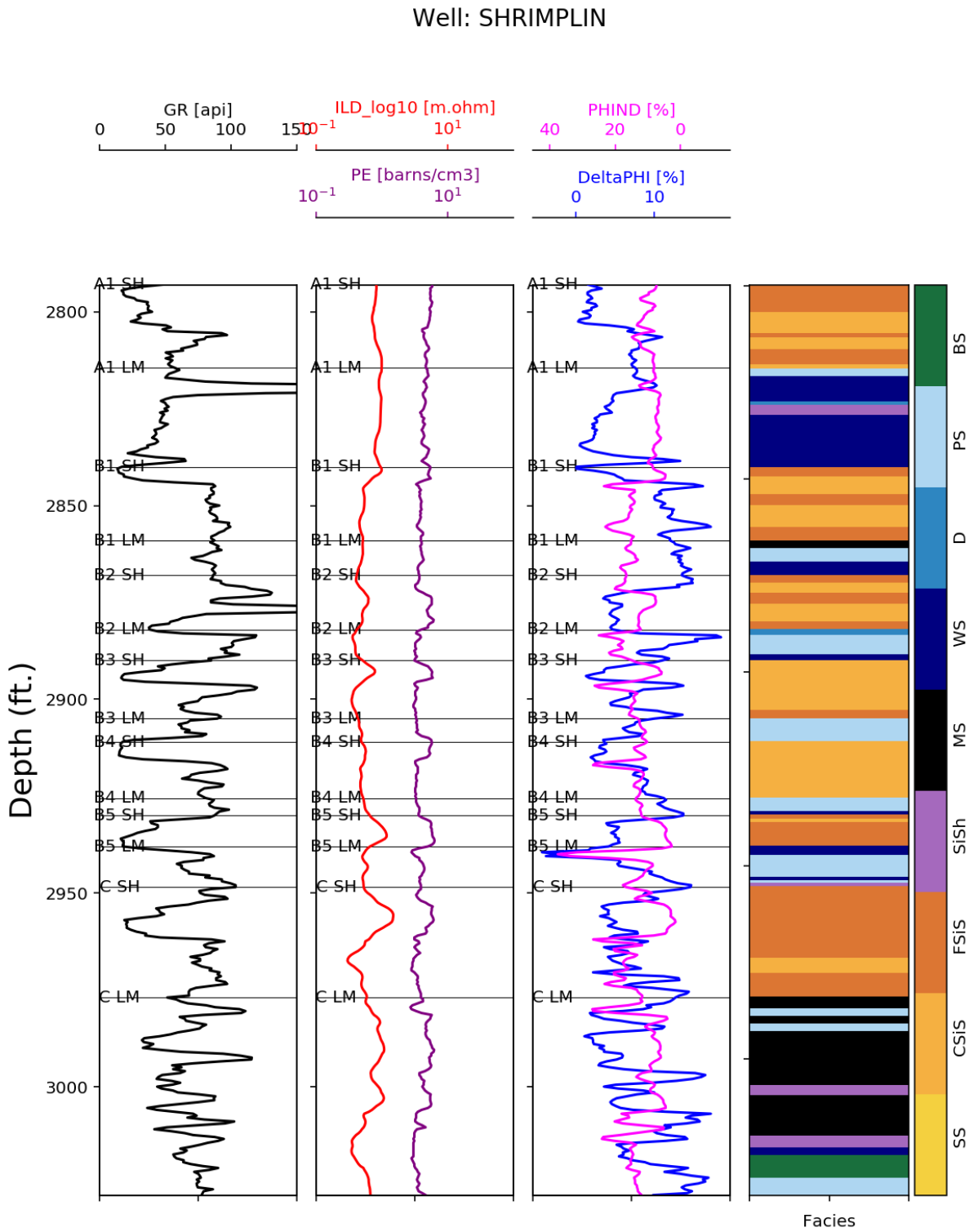

Figure 1. 3 Wire-line log curves plot for "SHRIMPLIN" well along with facies classes sampled at half-feet (0.15m). Five wire-line log curves are gamma ray (GR), resistivity (ILD_log10), photoelectric effect (PE), neutron-density porosity difference (DELTAPHI), and average neutron-density porosity (PHIND). The color indicates the assigned facies, which are explained in Table 1.1. Formation or member level tops are also marked in the figure.

The dataset contains eight wells with a total of 4149 examples. Dataset statistics are listed in Table 1.3.

Table 1. 3 Dataset statistics of the mean and standard deviation (Std dev.) values of seven features: gamma ray (GR), resistivity (ILD_log10), photoelectric effect (PE), neutron-density porosity difference (DELTAPHI), average neutron-density porosity (PHIND), nonmarine-marine (NM–M) and relative position (RELPos) for each rock class.

Facies label	Count	GR		ILD_log10		DeltaPHI	
		Mean	Std dev.	Mean	Std dev.	Mean	Std dev.
1	268	64.475	9.346	0.381	0.216	3.538	3.066
2	940	74.100	14.178	0.563	0.138	6.046	5.170
3	780	79.880	17.090	0.546	0.132	5.152	7.695
4	271	87.584	44.828	0.768	0.150	6.500	3.291
5	296	60.409	34.417	0.838	0.201	4.739	4.400
6	582	54.367	30.651	0.880	0.213	3.293	3.257
7	141	57.595	43.150	0.475	0.209	6.171	4.494
8	686	44.768	31.636	0.777	0.286	2.152	3.825
9	185	43.682	28.714	0.602	0.322	1.025	3.966
All	4149	64.934	30.303	0.660	0.253	4.402	5.275

Facies label	PHIND		PE		NM-M		RELPos	
	Mean	Std dev.	Mean	Std dev.	Mean	Std dev.	Mean	Std dev.
1	14.798	3.898	2.913	0.279	1	0	0.466	0.225
2	14.694	6.615	3.215	0.431	1..006	0.080	0.481	0.260
3	19.857	8.781	3.132	0.621	1.021	0.142	0.567	0.327
4	11.620	3.401	3.802	0.543	1.982	0.135	0.406	0.207
5	9.398	4.736	3.995	0.627	1.963	0.189	0.587	0.288
6	7.722	3.139	4.234	0.679	1.995	0.072	0.502	0.270
7	14.024	5.758	3.671	0.595	1.993	0.084	0.602	0.274
8	9.816	4.610	4.519	0.746	1.983	0.131	0.582	0.316
9	12.803	4.309	5.296	0.839	2.0	0.0	0.464	0.232
All	13.201	7.133	3.725	0.896	1.518	0.499	0.522	0.287

1.3 Data Preparation for Machine Learning Algorithms

1.3.1 Stratified Split

The goal of stratified sampling is that by splitting one data set, each split is similar concerning something to avoid bias (Leuenberger and Kanevski 2015). In a classification problem, it is often chosen to ensure that the train and test sets have approximately the same distribution of samples of each class as the complete set. As a result, stratified sampling is almost the same as random sampling if the dataset has a large amount of each class. However, if one class is not sufficient in the data set, like the dataset I used in this dissertation, then stratified sampling will yield a similar class distribution in the train and test sets to avoid systematic bias.

In this dissertation, as shown in figure 1.4 and 1.5, I split the dataset into a testing dataset with 412 samples and a training dataset with 3737 sample. The ratio for training vs. testing data is about 90/10. This ratio is relatively high compared to a typical training-testing data split. However, it is reasonable to do this way since the dataset I use is relatively small, I tried to let the machine learning algorithms capture the dataset patterns as much as possible, which means I need a training dataset as big as it could be. Meanwhile, I also need a testing dataset to validate the trained algorithms. I have to set the ratio to be 90/10 as a trade-off.

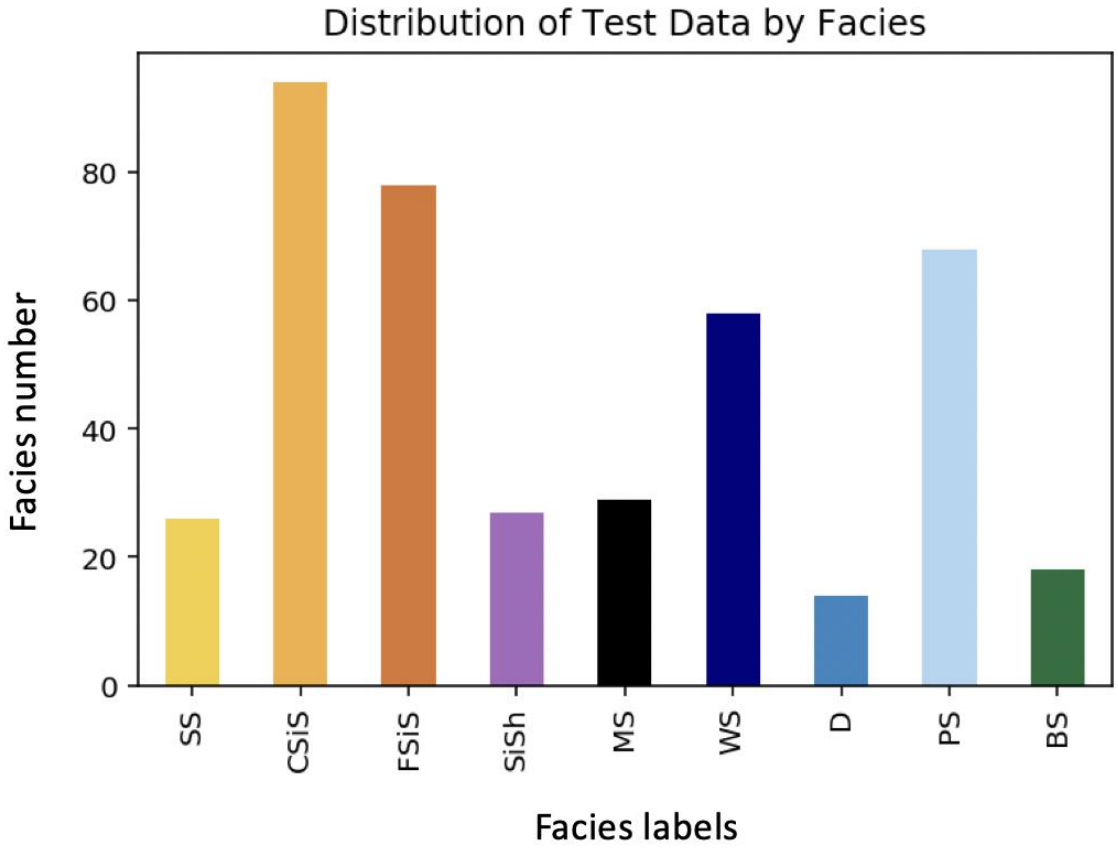

Figure 1. 4 Testing dataset facies distribution. Facies labels represent specific facies which are explained in table 1.1.

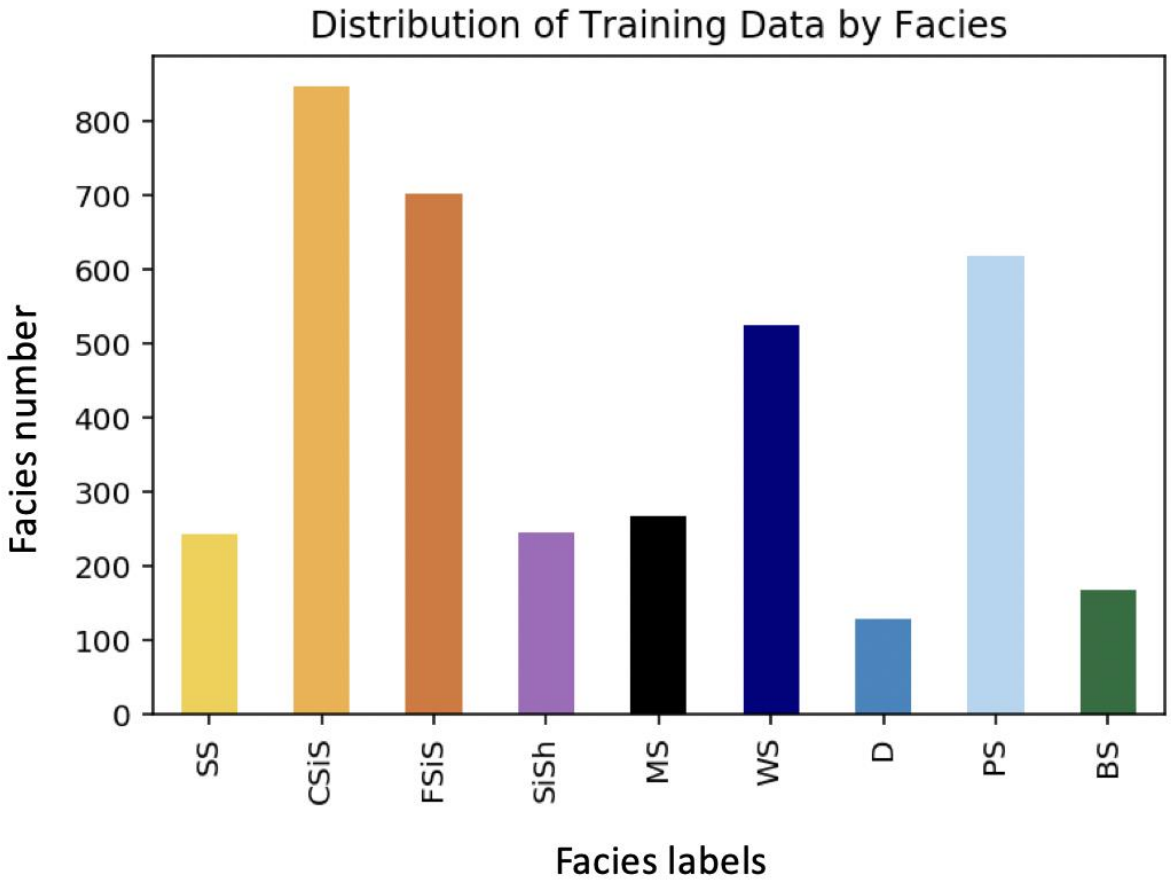

Figure 1. 5 Training dataset facies distribution. Facies Labels represent specific facies, which are explained in table 1.1.

1.3.2 Data Cleaning

Data cleaning is a processing step that tends to remove 'NAN' data values (Fayyad 1996). "NAN" means "Not a number", it is a special case in floating-point representations of a real number as well as in floating-point operations. Even though some machine learning algorithms can deal with "NAN" data, applying data cleaning can potentially minimize the negative effect caused by contaminated "NAN" data. I examined our whole dataset and find that some PE values have "NAN" data. To maintain the consistency of our dataset, I replace the "NAN" values with the PE mean value.

1.3.3 Feature Scaling

The purpose of feature scaling is to normalize the variables or features of data so they can fall into the same range, like from -1 to 1 (Youn and Jeong 2009). It is also known as data normalization in data processing and is generally performed during the data preprocessing step. In most cases, the range of values of raw data varies widely, which will cause objective functions may not work properly without normalization in machine learning algorithms. For example, many machine learning classifiers use Euclidean distance between two points to represent distance. If one of the features has a dominant range of values, the calculated distance will be governed by this particular feature, which might dysfunction the classifier algorithm. Another reason for using feature scaling is that gradient descent converges much faster with feature scaling than without it.

2. Introduction of Machine Learning

Machine learning is a new and very big topic. In this dissertation, I focus on applying the classification machine learning methods to do rock facies classification tasks. Fernandez-Delgado et al. (2014) and his colleagues did an experiment using 179 classifiers on 121 different datasets and trying to analyze the performance of each classifier. Their published paper is titled “Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?”. The conclusion is simple and I agree with it: there is no best classifier, only the most suitable one.

Then, before I set out to introduce the five classification algorithms I used in this dissertation, I will first introduce the three most popular sets of machine learning classification: supervised versus unsupervised learning, online versus batch learning, instance-based versus model-based learning.

2.1 Types of Machine Learning Systems

There are many criteria to classify different types of machine learning systems into broad categories, such as whether or not they are trained with given labels (supervised, unsupervised, semi-supervised, and reinforcement learning); whether or not they can learn incrementally on the data streaming (online versus batch learning); whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model (instance-based versus model-based learning). Among all these different types of machine learning systems, I will briefly introduce supervised, unsupervised, semi-

supervised, and reinforcement Learning based on the fact that the problem I deal with in this dissertation is a supervised learning problem.

2.1.1 Supervised Learning

In supervised learning, the training data you put into the algorithm includes desired answers, called labels. A typical supervised learning task is classification. Like in my dissertation, the dataset is labeled a facies type at a half foot interval, I have to feed the seven features into the algorithms and output a facies class.

Another typical application is to predict a target numeric value, such as the price of a house, given a set of features (location, year of built, floor space, etc.) called predictors. This sort of task is called regression.

Here are some of the most important and popular supervised learning algorithms: k-Nearest Neighbors, Logistic Regression, Linear Regression, Support Vector Machines, Decision Trees, Random Forests, Artificial Neural Networks (Ye et al. 2009; Schwenker and Trentin 2014; Al-Jamimi et al. 2018; Abdughani et al. 2019; Mao et al. 2019).

2.1.2 Unsupervised Learning

In unsupervised learning, the training data is unlabeled (Langkvist et al. 2014; Hubner et al. 2018; Miller et al. 2018). The system tries to learn without prior information. Here are some of the most important unsupervised learning algorithms: 1) Clustering: k-Means, Hierarchical Cluster Analysis (HCA), Expectation Maximization; 2) Visualization and dimensionality

reduction: Principal Component Analysis (PCA), Locally-Linear Embedding (LLE), t-distributed Stochastic Neighbor Embedding (t-SNE)

2.1.3 Semi-supervised Learning

Some algorithms can deal with partially labeled training data. This is called semi-supervised learning (Rout et al. 2017; Liu et al. 2018; Kim et al. 2019).

2.1.4 Reinforcement Learning

Reinforcement learning is mainly composed of Agent, Environment, State, Action, and Reward. After the agent performs an action, the environment will transit to a new state, and a reward signal (positive or negative reward) will be given for the new state environment. Subsequently, the agent performs new actions according to certain strategies according to the rewards of the new state and environmental feedback. The above process is a way for the agent and the environment to interact through states, actions, and rewards. Through reinforcement learning, the agent can know what state it is and what action it should take to maximize its reward. (Jahoda et al. 2001; Cohen 2008; Yau et al. 2012; Al-Rawi et al. 2015; Chen et al. 2015; Zhang et al. 2018; Jaafra et al. 2019; Mammeri 2019; Mason and Grijalva 2019).

2.2 Machine Learning in Geophysics

In 2006, Hinton and Salakhutdinov (2006) published a paper showing how to train a deep neural network capable of recognizing handwritten digits with state-of-the-art precision (>98%). Since then, machine learning has fast-forwarded and gradually conquered the industry.

Nowadays, machine learning is evolving rapidly and has provided numerous successful applications in computer vision, speech processing, and artificial intelligence problems. In particular, machine learning provides great potentials in imaging inverse problems such as de-noising (Burger et al. 2012; Xie et al. 2012), rock facies recognition (Wei et al.), super-resolution (Dong et al. 2016), compressed sensing (Sun et al. 2016), and X-ray computed tomography (Wang 2016; Würfl et al. 2016). For these reasons, there is great promise in using this approach to approximate complex and highly nonlinear functions.

More specified in the geophysics field: in both exploration and earth geophysics, they have large amounts of exploration or earthquake data, and lots of problems remain unsolved are nonlinear problems (Mora 1989; Zhou 2006). And this makes a perfect situation for applying machine learning algorithms. For example, machine learning in geophysics is suitable for problems for which existing solutions require a lot of limitations or assumptions that are hard to satisfy in real life, like seismic processing and horizon picking (Jia and Ma 2017; Reynen and Audet 2017; Chaki et al. 2018; Li et al. 2018; Malfante et al. 2018; Wrona et al. 2018). Machine learning can also solve complex problems for which there is no satisfactory solution at all using a traditional approach, like the nonuniqueness caused by the ill-posed matrix in seismic inversion problems (Teng 2015; Zeng et al. 2018; Giannakis et al. 2019; Kovachki and Stuart 2019; Ray and Myer 2019). Machine learning can help geophysicists getting insights about complex problems and large amounts of data.

I have briefly introduced machine learning so far, and I can't test all machine learning classification algorithms. The goal of my dissertation is first trying to find out whether machine learning is feasible to classify rock facies or not. And it is the reason I chose to use the most popular and representative classification algorithms which have different principles. I also try

to improve our algorithms by incorporating multiple techniques, which means the algorithms should be easy to implement. Based on these criteria, finally I choose Support Vector Machine (SVM), k-Nearest Neighbors (KNN), Decision Trees, Artificial Neural Network (ANN) and Convolutional Neural Network (CNN) in my dissertation. I will perform analysis using the five selected ML algorithms in the subsequent chapters 3 to 7.

2.3 Common Parts of Machine Learning Algorithms

Most ML algorithms contain three common parts: How to measure the error (objective or loss function)? How to reduce the error (optimize function) and how to evaluate the result?

2.3.1 How to Measure The Error?

The equations to formulate or represent the machine learning problems are also called the objective function, and some people prefer to use “loss function”.

2.3.2 How to Reduce The Error?

Assume I already have an objective function, and then the next step should naturally be “how to reduce the error”. So the process to optimize the objective function is called optimization. There are also many optimization methods like gradient descent.

2.3.3 How to Evaluate The Result?

With each final result, I always need to find a way to evaluate the results. A much better way to assess the performance of a classifier is using the confusion matrix. The general idea is to count the number of times instances of class A are classified as class B, and F-1 score, as defined in equation 2-1, is often used to evaluate the classification algorithm results.

I followed the ML industry standard (Baluja et al. 2000) of measuring the test results by using the F-1 score as follows:

$$\frac{1}{F1} = \frac{1}{2}\left(\frac{1}{Precision} + \frac{1}{Recall}\right), \qquad (2-1)$$

where *Precision* and *Recall* are expressed in Equations (2-2) and (2-3):

$$Precision = \frac{True\ positive}{True\ positive + False\ positive}, \qquad (2-2)$$

$$Recall = \frac{True\ positive}{True\ positive + False\ negative}, \qquad (2-3)$$

where *true positive* is the number of facies correctly predicted, *false positive* is the number of facies which are not x-class but are predicted as x-class, and *false negative* is the number of facies which are x-class but are predicted as not x-class. These terms are illustrated in figure 2.1.

I use the following two criteria to judge the success of facies prediction:

(1) Average F-1 score of all facies;

(2) Average F-1 score of all adjacent facies.

		Predicted/Classified	
		Negative	*Positive*
Actual	*Negative*	True negative	False positive
	Positive	False negative	True positive

Figure 2. 1 Illustration of precision and recall.

Now I have introduced the measure of success factor F-1 score and in the next step, I will apply it to evaluate our classification results on the field dataset.

3. Support Vector Machine

3.1 Introduction

A Support Vector Machine (SVM) is a compelling and versatile machine learning algorithm, capable of performing linear or nonlinear classification, regression, and even outlier detection (Nishitsuji and Exley 2019). It is one of the most popular algorithms in machine learning. SVMs are particularly well suited for the classification of complex but small or medium-sized datasets.

In the machine learning category, SVM are supervised learning algorithms used for classification and regression analysis (Suykens and Vandewalle 1999). For example, given a set of training examples having two categories, apples and oranges, and an SVM training algorithm builds a model that assigns new samples to apples or oranges, making it a non-probabilistic binary linear. After the training process, an SVM model is a representation of mapping the samples of different categories into space as points that are divided by a clear margin that is as wide as possible. New samples are then mapped into that same space and predicted to belong to a category based on which side of the margin they fall into.

In addition to performing linear classification, SVM can also efficiently perform a non-linear classification using the kernel trick, which is implicitly mapping their inputs into high-dimensional feature spaces and reduce computation.

Vladimir N. Vapnik and Alexey Ya invented the original SVM algorithm. Chervonenkis in 1963. In 1992, Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik proposed a way to create nonlinear classifiers by applying the kernel function to maximum-margin

hyperplanes (Boser et al. 1992). Corinna Cortes and Vapnik introduced the current standard incarnation (soft margin) in 1993 and published in 1995 (Cortes and Vapnik 1995).

3.1.1 Theory

Let me use a simple example to illustrate the way SVM works. In a two-dimensional space, I can use a linear function to separate the sample points. If a linear function can separate the sample points, the data is said to be linearly separable; otherwise it is called linear indivisible. This linear function is a point in a dimension space, a line in a two-dimensional space, and a plane in a three-dimensional space. They are all called "hyperplanes".

As shown in the figure below, in a two-dimensional space, a line separates the two types of samples, the red and yellows circles, which is the hyperplane:

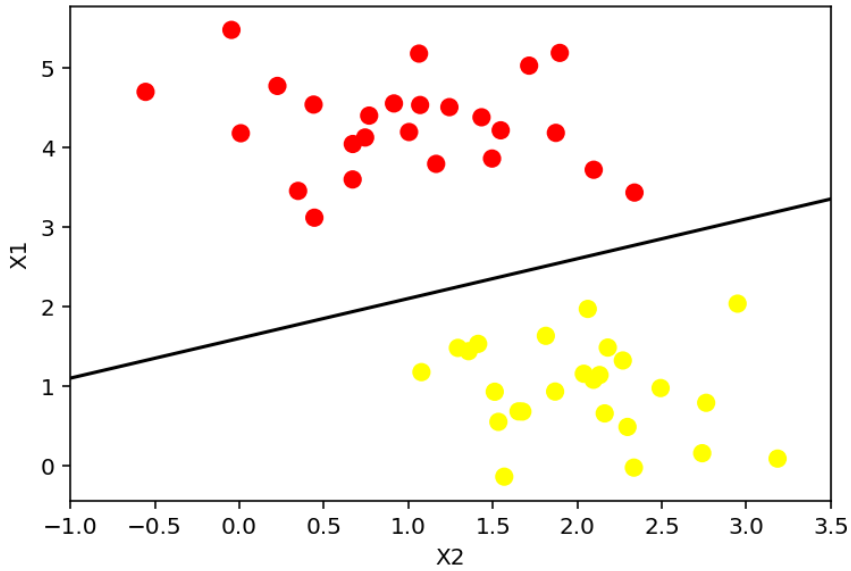

Figure 3. 1 An illustration of a hyperplane separates two types of samples. Yellow and red circles represent different types of samples.

Mathematically I can express the line by

$$w_1 x_1 + w_2 x_2 + b = 0, \qquad (3-1)$$

let f(x) equal to

$$f(x) = w_1 x_1 + w_2 x_2 + b, \qquad (3-2)$$

when a point coordinates make f(x) > 0, which means the point is above the line, it belongs to the circle. When it makes f(x) < 0, then it belongs to cross. Think about this situation: the point falls exactly on the line. Looking at the picture below, if you move the line slightly and it is another hyperplane. They both can separate the two types of points. The hyperplane is not unique. But I want to find out the hyperplane which can distinguish the two types of points that are the most open and the clearest, and use such a hyperplane to classify the points of the unknown category, which is the most reliable.

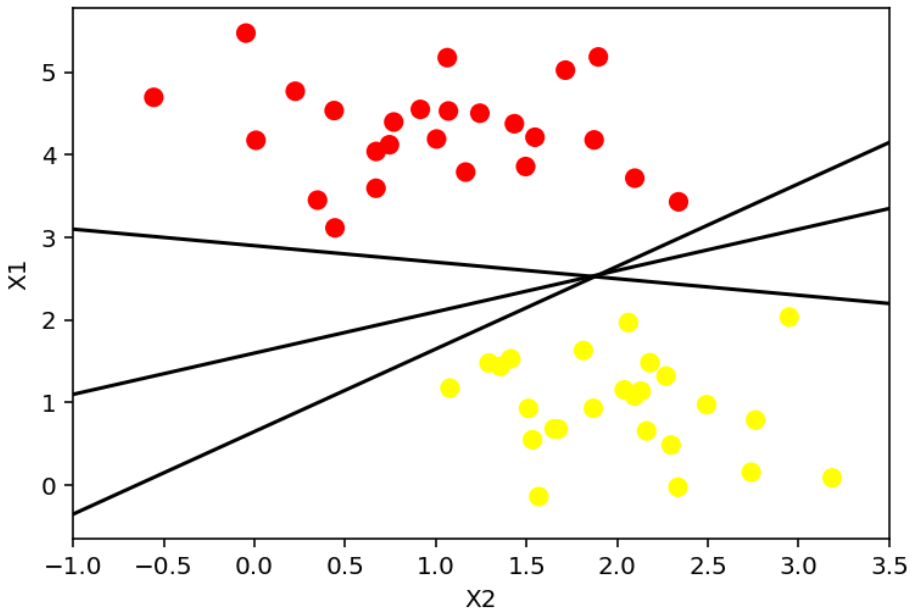

Figure 3. 2 An illustration of multiple hyperplanes that can separate two types of samples. Yellow and red circles represent different types of samples.

Earlier I talked about the mathematical expression of the hyperplane in two-dimensional space:

$$f(x) = w_1x_1 + w_2x_2 + b, \tag{3-3}$$

and in 3D space it is

$$f(x) = w_1x_1 + w_2x_2 + w_3x_3 + b. \tag{3-4}$$

In any n-dimensional space, the universal classification function will be

$$f(x) = w_1x_1 + \cdots + w_nx_n + b = (w_1, \ldots, w_n)\begin{pmatrix} x_1 \\ \cdots \\ x_n \end{pmatrix} + b = W^TX + b. \tag{3-5}$$

So, no matter how many dimensions it is, I can represent them all in vector form

$$f(x) = W^TX + b. \tag{3-6}$$

In order to quantify the expression, I use the letter y to represent the category to which a point belongs, y is called the result label, then use y=1 to represent the red dots, and y= -1 to represent the yellow dots, that is, the point below the line, the value of y is -1, the point above the line, the value of y is 1. It yields the following equation

$$y = \begin{cases} 1, f(x) \geq 0 \\ -1, f(x) < 0 \end{cases}. \tag{3-7}$$

3.1.2 Geometric Spacing

The geometric spacing means the distance from the point to the line. In fact, the geometric spacing, is equal to the function interval divided by a paradigm (Mudur and Koparkar 1984).

$$interval = \frac{\hat{r}}{\|W^T\|}, \tag{3-8}$$

$$\|W^T\| = \sqrt[p]{w_1^p + w_2^p + \cdots + w_n^p}, \tag{3-9}$$

where $\|W^T\|$ is called the paradigm of $W^T = (w_1, w_2, \ldots, w_n)$. The geometric spacing is better than the function interval. Imagine that I scale the values of the coefficients w and b in f(x) proportionally, the line remains the same, but the function interval changes and the scale is enlarged or reduced. When scaling w and b, the distance from the point to the line is still constant.

Besides, the geometric spacing of all samples to a hyperplane is the smallest of the geometric distances from all points to the hyperplane.

I now know that the geometric spacing is used to evaluate the superiority of the hyperplane—the higher the geometric spacing, the more precise the hyperplane will separate the sample. So, the hyperplane I want to select is a plane whose geometric spacing to a set of sample points is the largest – it is called the maximum interval classifier.

3.1.3 Maximum Interval Classifier

Below are three steps how SVM looks for hyperplanes.

1: Use geometric spacing to measure the distance from the point to the hyperplane;

2: In a set of samples, the distance from the point closest to the hyperplane is taken as the distance from the set of samples to the hyperplane;

3: The hyperplane that is being searched for can maximize the distance between the set of samples and it, that is, the best way to separate the samples.

Converting the above three into a mathematical language,

$$① \; r = \frac{\hat{r}}{\|W^T\|} = \frac{|f(x)|}{\|W^T\|} = \frac{yf(x)}{\|W^T\|}, \qquad (3-10)$$

$$② \; r = min r_i (i = 1,2, \dots, n), which\ is\ y_i(W^T x_i + b) \geq \hat{r}, \qquad (3-11)$$

$$③ \; \max r = \max \frac{\hat{r}}{\|W^T\|}. \qquad (3-12)$$

In this way, the problem is transformed into a mathematical problem of maximization. To facilitate the derivation of price and optimization, I can make the function interval |f(x)|=yf(x)=1, which is the fixed function interval, value is 1.

$$min \frac{1}{2} \|W^T\|^2,$$

$$s.t.\ y_i(W^T x_i + b) \geq 1, i = 1,2, \dots, n. \qquad (3-13)$$

3.1.4 Lagrangian Multiplier Method of Inequality

In mathematical optimization, Lagrange multipliers is the method used to find the local maxima and minima of a function subject to equality constraints. First, I construct the Lagrangian function from equation(x) and (y) and yield

$$F(w, b, a) = \frac{1}{2} \|W^T\|^2 - \sum_{i=1}^{n} a_i * [y_i(W^T x_i + b) - 1]. \qquad (3-14)$$

Make $C(w) = \max_{a_i > 0} F(w, b, a)$, $C(w)$ means I can maximize the F function through adjusting a. When the constraint is not satisfied, which means $y_i(W^T x_i + b) < 1$; if a is infinite, then $C(w)$ is infinite; When the constraint is satisfied, the maximum of F is $C(w) = \frac{1}{2} \|W^T\|^2$.

So the original proposition transformed to

$$\min_{w,b} C(w) = \min_{w,b} \max_{a_i > 0} F(w, b, a) = p^*. \qquad (3-15)$$

3.1.5 Lagrange Dual Problem

In the above, I have transformed the constraint problem of inequality into a p* problem.

$$\min_{w,b} \max_{a_i>0} F(w,b,a) = p^*. \qquad (3-16)$$

But it is still a difficult problem to solve because I have to solve the max problem of inequality constraints first, and then find the minimum value on w. It will be much easier if I change the order, first solve the minimum value about w, and then solve the inequality constraint problem about a. Which is,

$$\max_{a_i>0} \min_{w,b} F(w,b,a) = d^*. \qquad (3-17)$$

This d* is the dual form of p*, which is the dual form of the original problem, but unfortunately, the duals are not necessarily equal because,

$$d^* \leq p^*. \qquad (3-18)$$

This inequality has a name called Weak Duality; the smallest of the maximum is greater than or equal to the largest of the minimum. In convex optimization theory, there is a Slater theorem. When this theorem is satisfied, the dual gap will disappear, namely

$$d^* = p^*. \qquad (3-19)$$

This is called strong duality. Fortunately, equation 3-17 satisfies the Slater theorem here. What is the Slater theorem? If the convex optimization problem has a strictly feasible solution, that is, there is x, which is satisfied.

$$f_i(x) < 0, i = 1,2,\dots,m\,,$$

$$Ax = b. \qquad (3-20)$$

Then the optimization problem has strong duality (the inequality constraint in the original question is $f(x) \leq 0$).

Now that Slater's theorem is satisfied, I have p*=d*, but there is still a problem. Since both p* and d* may have more than one solution, I have to guarantee that the solution of p* = d* is the optimal solution. Luckily, when the Slater condition is satisfied and F (w, b, a) is derivative for both w and b, the KKT condition is satisfied. Then I can use SMO (Sequential Minimal Optimization) algorithm to solve equation x and get the value of w and b. The principle of KKT condition and SMO is beyond our scope, so I put them in Appendix A for reference.

3.2 Results of Facies Classification

In this section, I will show the confusion matrix of facies prediction results, heat map, and adjacent facies prediction results.

A confusion matrix is a matrix representation of prediction results on a classification problem. The number of correct and incorrect predictions are listed in the matrix with count values and broken down by each class. A confusion matrix is an important tool for the performance evaluation of classification models. From the confusion matrix, various evaluation indicators such as true positive rate, false positive rate, true negative rate, false negative rate, accuracy rate, accuracy rate and F index can be calculated. In particular, the confusion matrix distinguishes between false positives and false negatives, which can be used to estimate the expected loss caused by misclassification of the classification model.

In machine learning, a heatmap is a graphical representation of the confusion matrix to visually present the summation of prediction results. Heatmaps are used in various kinds of analytics. In this dissertation, I use heatmap as a visualization technique to compare the results of predicted facies and true facies. As shown in figure 3.3, if the numbers along diagonal are large, it means the method I use has done an excellent job in classification. If there is non-zero value off the diagonal, then it means some facies are misclassified.

Table 3. 1 Confusion matrix of SVM facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	21	5								26
CSiS	14	57	22	1						94
FSiS	6	24	45	1		1		1		78
SiSh		1		18		7			1	27
MS	1	1		6	1	14	2	4		29
WS	1		1	8	1	32	4	8	3	58
D				2	1	1	8	2		14
PS		1	1	5		16	6	25	14	68
BS				3				4	11	18
Precision	0.91	0.68	0.68	0.51	0.03	0.38	0.50	0.47	0.60	0.58
Recall	0.53	0.81	0.60	0.59	0.13	0.60	0.12	0.55	0.60	0.61
F1	0.67	0.74	0.64	0.55	0.05	0.47	0.19	0.51	0.60	0.59

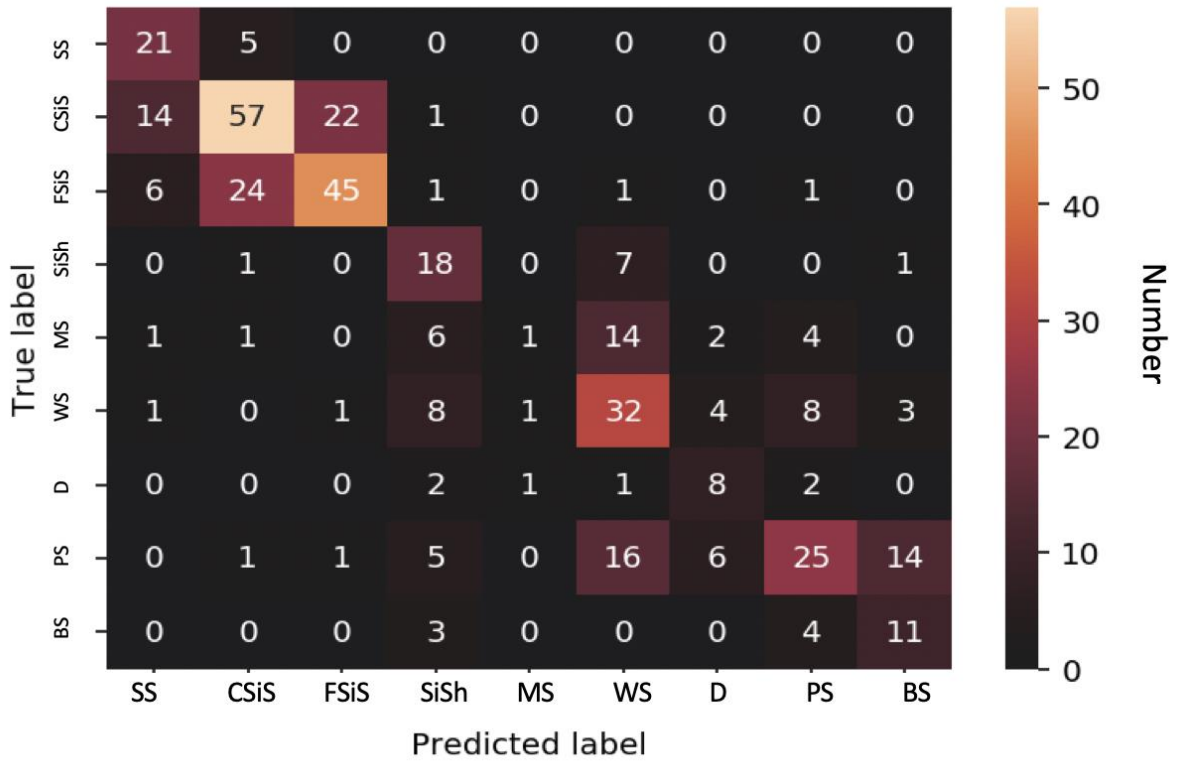

Figure 3. 3 Heat map of SVM facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

Table 3. 2 Confusion matrix of SVM adjacent facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	26									26
CSiS		93		1						94
FSiS	7		68			1	1	1		78
SiSh		1		17		6	2		1	27
MS	1	1			21		2	4		29
WS	1		1	9		44			3	58
D				2	1		11			14
PS		1		3	4			60		68
BS				3					15	18
Precision	0.74	0.97	0.99	0.49	0.81	0.86	0.69	0.92	0.79	0.87
Recall	1.00	0.99	0.87	0.63	0.72	0.76	0.79	0.88	0.83	0.86
F1	0.85	0.98	0.93	0.55	0.76	0.81	0.73	0.90	0.81	0.86

From table 3.1 we can see that the F-1 scores of exact facies of the SVM algorithm are 0.59. The MS facies (mudstone-limestone) has the lowest F-1 score indicates that it is the most difficult facies to classify by the SVM algorithm. Considering the fuzzy nature of the facies and inherent error in the data, I consider being close (within one facies) is nearly as good as being precise. I listed the adjacent facies classification results in table 3.2 and the F-1 score of SVM is 0.86. I will continue to show the results of k-Nearest Neighbors and compare the results of the two algorithms in the next chapter.

4. K-Nearest Neighbors

In pattern recognition, the k-Nearest Neighbors algorithm (KNN) is a non-parametric method used for classification and regression (Keller et al. 1985). KNN is a type of instance-based learning, where it doesn't need a training process. The new sample distance function is only approximated locally, and all computation is deferred until classification. The KNN algorithm is among the most straight forward of all machine learning algorithms. In both cases, the first step is to normalize the input data, and then calculate the distance of all examples the input in the feature space, then choose the k nearest neighbors from all distances. The output will be a class or a single value depends on whether KNN is used for classification or regression.

Specifically, in KNN classification, the output is a class that is classified by a majority group class among its k nearest neighbors (k is a positive integer). In KNN regression, the output is the property value that is the average of the values of its k nearest neighbors.

Both for classification and regression, a useful technique can be used to assign a weight to the contributions of the neighbors, so that the closer neighbors contribute more to the average than the more distant ones. For example, a standard weighting scheme consists in giving each neighbor a weight of $1/d$, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for KNN classification) or the object property value (for KNN regression) is known. This set of objects can be treated as the training set for the algorithm, though no specific training step is required.

4.1 Introduction

The K-nearest neighbor algorithm is a basic classification and regression method. In the classification problem, the KNN algorithm assumes that the instance category of a given training set has been determined. For the new instance, the KNN algorithm classifies the new instance by majority voting, etc. according to the category of its k nearest neighbor training set instances to make predictions. The three essential elements of the KNN algorithm are: the choice of k value (i.e., how many training instances points to take as a neighbor in the new instance), the distance measurement method (Euclidean distance, Manhattan distance, etc.) and the classification decision rules (standard methods). It is the category in which the number of occurrences of the category in the k neighbor training instances is the largest as the input new instance.

4.1.1 Algorithm Implementation

The input are the training data set $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, where $x_i \in \chi \subseteq R^n$ is the feature vector $y_i \in \gamma \subseteq \{c_1, c_2, c_3, \dots, c_k\}$ of the training instance, which is the category of the training instance. And the output will be x category of the new input instance y. The algorithm can be summarized in the following steps:

(1) According to the given distance metric, find the x nearest k points in the training set T, and the neighborhood covering the k points is recorded as $N_k(x)$.

(2)$N_k(x)$ according to the classification decision rule (If the majority vote, x the category to which it belongs y:

$$\mathrm{y} = \arg\max_{c_j} \sum_{x_i \in N_k(x)} I(y_i = c_j), i = 1,2,3,\dots,N; j = 1,2,3,\dots,k\,, \qquad (4-1)$$

where I is the indicator function, only $y_i = c_j$ I have a value of 1, and 0 otherwise.

4.1.2 Distance Measurement

The more commonly used distance metric is the Euclidean distance, which can be defined using other more general L_p distances or Minkowski distances. Wherein space is provided χ for the number n is a real vector space R^n,

$$x_i,\ x_j\ \in \chi, x_i = \left(x_i^{(1)}, x_i^{(2)}, x_i^{(3)}, \dots, x_i^{(n)}\right)^T, x_j = \left(x_j^{(1)}, x_j^{(2)}, x_j^{(3)}, \dots, x_j^{(n)}\right)^T. \qquad (4-2)$$

Distance L_p of (x_i, x_j) can be defined in several ways. For example, And Minkowski distance is

$$L_p(x_i, x_j) = \left(\sum_{l=1}^{n} \left|x_i^{(l)} - x_j^{(l)}\right|^p\right)^{1/p}. \qquad (4-3)$$

Euclidean distance is

$$L_p(x_i, x_j) = \left(\sum_{l=1}^{n} \left|x_i^{(l)} - x_j^{(l)}\right|^2\right)^{1/2}. \qquad (4-4)$$

Manhattan distance is

$$L_p(x_i, x_j) = \sum_{l=1}^{n} \left|x_i^{(l)} - x_j^{(l)}\right|. \qquad (4-5)$$

4.1.3 K Value Selection

Generally, a smaller k value is selected first, and then cross-validation is performed to select an optimal k value. When the k value is small, the overall model becomes complicated, and it is sensitive to the training instance points of the neighbors and prone to over-fitting. When the value of k is significant, the model tends to be simple. At this time, the farther training instance points will also play a predictive role, and it is easy to appear underfitting.

4.1.4 Classification Decision Rule

Majority voting rules are often used. For a given instance $x \in \chi$, the K nearest neighbor points constituting a set of training examples $N_k(x)$, when the cover $N_k(x)$ class region is c_j, the misclassification rate is

$$\frac{1}{k}\sum_{x_i \in N_k(X)} i(y_i \neq c_j) = 1 - \frac{1}{k}\sum_{x_i \in N_k(X)} i(y_i = c_j). \qquad (4-6)$$

To minimize misclassification rate the $\sum_{x_i \in N_k(X)} i(y_i = c_j)$ should reach the maximum, thus the majority of Voting rules meet equivalence, and empirical risk is minimized.

4.2 Facies Featuring Engineering

In classification problems having relatively high dimensional feature space without a priori statistical knowledge of objects being classified, the KNN approach provides simplicity and ease of use. Because many elements of the feature vectors have wide-ranging values that

are not normally distributed, determining similarities by using simple distance metrics between raw or normalized elements is not appropriate.

In the real world, it would be unusual to have neutron-density cross-plot porosity (i.e. PHIND) without the corresponding raw input curves, namely bulk density and neutron porosity, as I have in this contest dataset. So as part of the feature engineering process, I back-calculate estimates of those raw curves from the provided DeltaPHI and PHIND curves. One issue with this approach is that cross-plot porosity differs between vendors, tool strings, and software packages, and it is not known exactly how the PHIND in this dataset was computed. So I make the assumption here that PHIND ≈ sum of squares porosity, which is usually an adequate approximation of neutron-density crossplot porosity. That equation looks like this:

$$PHIND \approx \sqrt{\frac{NPHI^2 + DPHI^2}{2}}, \qquad (4-7)$$

and it is assumed here that DeltaPHI is

$$DeltaPHI = NPHI - DPHI. \qquad (4-8)$$

The functions below use the relationships from the above equations to estimate NPHI and DPHI (and consequently RHOB). Once I have RHOB, I can use it combined with PE to estimate apparent grain density (RHOMAA) and apparent photoelectric capture cross-section (UMAA), which are useful in lithology estimations from well logs.

$$NPHI = DPHI + DeltaPHI, \qquad (4-9)$$

$$DPHI = (\sqrt{4 * PHIND^2 - DeltaPHI^2} - DeltaPHI)/2, \qquad (4-10)$$

$$RHOB = 2.71 - \left(\frac{\mathrm{DP}HI}{100}\right) * 1.71, \qquad (4-11)$$

$$RHOMAA = (RHOB - \frac{PHIND}{100})/(1 - \frac{PHIND}{100}). \quad (4-12)$$

Table 4. 1 Features used in KNN classification.

Feature name	Symbol	Feature name	Symbol
Gamma ray	GR	Neutron porosity	NPHI
Resistivity	ILD_log10	Density porosity	DPHI
Photoelectric effect	PE	density	RHOB
Neutron-density porosity difference	DELTAPHI	Apparent grain density	RHOMAA
Average neutron-density porosity	PHIND	Apparent photoelectric capture cross-section	UMAA

4.3 Results of Facies Classification Without Feature Engineering

In this section, I will show the confusion matrix of facies prediction results, heat map, and adjacent facies prediction results of KNN without feature engineering.

Table 4. 2 Confusion matrix of KNN (without feature engineering) facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	11	15								26
CSiS	4	70	20							94
FSiS	2	20	54	1						78
SiSh			1	18		4		4		27
MS	1	1		3	9	10	1	4		29
WS		1	1	5	6	36	1	7	1	58
D				1		1	8	4		14
PS		1		2	3	10		52		68
BS							2	3	13	18
Precision	0.61	0.65	0.71	0.60	0.50	0.59	0.67	0.69	0.93	0.66
Recall	0.42	0.74	0.69	0.67	0.31	0.62	0.57	0.76	0.72	0.66
F1	0.50	0.69	0.70	0.63	0.38	0.61	0.62	0.73	0.81	0.65

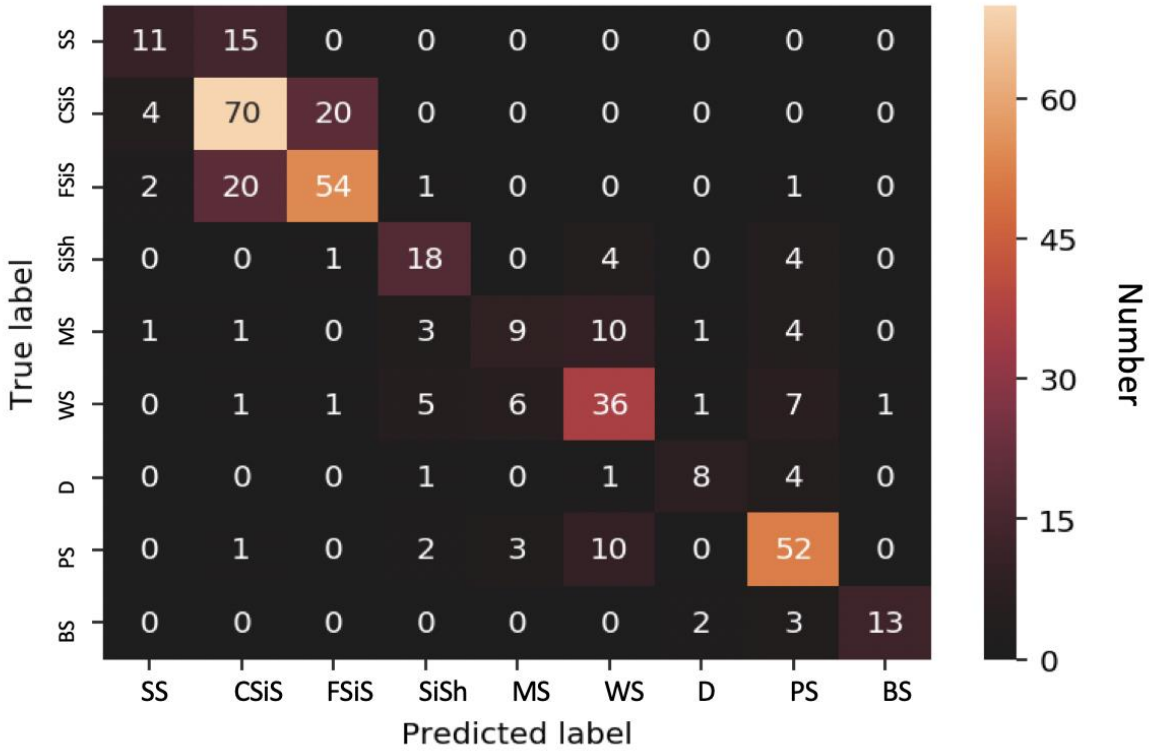

Figure 4. 1 Heat map of blind test results of KNN (without feature engineering). Facies labels represent specific facies which are explained in table 1.1.

Table 4. 3 Confusion matrix of KNN (without feature engineering) adjacent facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	26									26
CSiS		94								94
FSiS	2		74	1				1		78
SiSh			1	18		4		4		27
MS	1	1			22		1	4		29
WS		1	1	5		50			1	58
D				1			13			14
PS		1		2	3			62		68
BS									18	18
Precision	0.90	0.97	0.97	0.67	0.88	0.93	0.93	0.87	0.95	0.92
Recall	1.00	1.00	0.95	0.67	0.76	0.86	0.93	0.91	1.00	0.92
F1	0.95	0.98	0.96	0.67	0.81	0.89	0.93	0.89	0.97	0.92

4.4 Results of Facies Classification With Feature Engineering

In this section, I will show the confusion matrix of facies prediction results, heat map, and adjacent facies prediction results of KNN with feature engineering.

Table 4. 4 Confusion matrix of KNN (feature engineering) facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	5	21								26
CSiS		79	14	1						94
FSiS		15	60	1		1		1		78
SiSh		1		22		2		2		27
MS		2		3	18	3	1	2		29
WS		1	1	3	4	40	2	6	1	58
D							14			14
PS			1	2	3	11	2	48	1	68
BS							1		17	18
Precision	0.61	0.65	0.71	0.60	0.50	0.59	0.67	0.69	0.93	0.66
Recall	0.42	0.74	0.69	0.67	0.31	0.62	0.57	0.76	0.72	0.66
F1	0.50	0.69	0.70	0.63	0.38	0.61	0.62	0.73	0.81	0.66

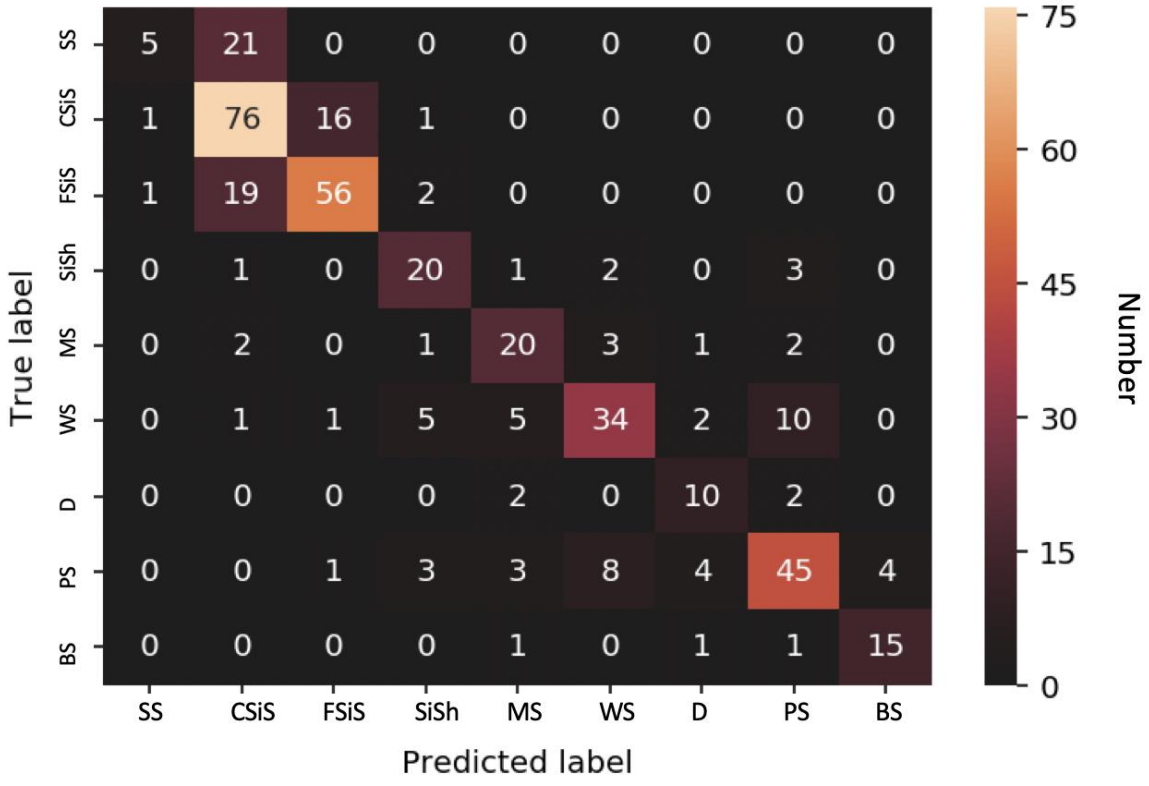

Figure 4. 2 Heat map of blind test results of KNN (feature engineering). Facies labels represent specific facies which are explained in table 1.1.

Table 4. 5 Confusion matrix of KNN (feature engineering) adjacent facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	26									26
CSiS		93		1						94
FSiS			75	1		1		1		78
SiSh		1		22		2		2		27
MS		2			24		1	2		29
WS		1	1	3		52			1	58
D							14			14
PS			1	2	3			62		68
BS									18	18
Precision	1.00	0.96	0.97	0.76	0.89	0.95	0.93	0.93	0.95	0.94
Recall	1.00	0.99	0.96	0.81	0.83	0.90	1.00	0.91	1.00	0.94
F1	1.00	0.97	0.97	0.79	0.86	0.92	0.97	0.92	0.97	0.94

In this chapter, I applied the k-Nearest Neighbors algorithm in our classification task. Then I utilize the feature engineering techniques to improve our results. As shown in table 4.2 and 4.3, before applying feature engineering, the F-1 scores of exact facies and adjacent facies are 0.65 and 0.92, respectively; after feature engineering applied, as shown in table 4.4 and 4.5, the F-1 scores of exact facies and adjacent facies are improved to 0.66 and 0.94, respectively. The results show that our feature engineering techniques work: generating more features is a way to add dimensions to our data space, which could create better performance. However, the improvement is not significant, the reason partial could be that the additional features are all derived from original features, in other words, the additional features are linearly or non-linearly related to the original data feature, the data space doesn't add in more freedom. Finally, compared to the SVM algorithm, the KNN methods generally have higher results. In the next chapter, I will introduce the decision tree algorithm.

5. Decision Trees

Like SVMs, Decision Trees are also versatile machine learning algorithms that can perform both classification and regression tasks, and even multi-output tasks (Safavian and Landgrebe 1991). They are potent algorithms, capable of fitting complex datasets. In this chapter, I will start by discussing how to train, visualize, and make predictions with Decision Trees. Then I will introduce the CART training algorithm used in this dissertation, and I will discuss how to regularize trees, some limitations of Decision Trees, and then show the classification results of decision trees.

5.1 The Basic Idea of The Decision Trees

A decision tree is a basic classification and regression method. It can be regarded as a set of **if-then** rules. It can also be considered as a conditional probability distribution defined in feature space and class space.

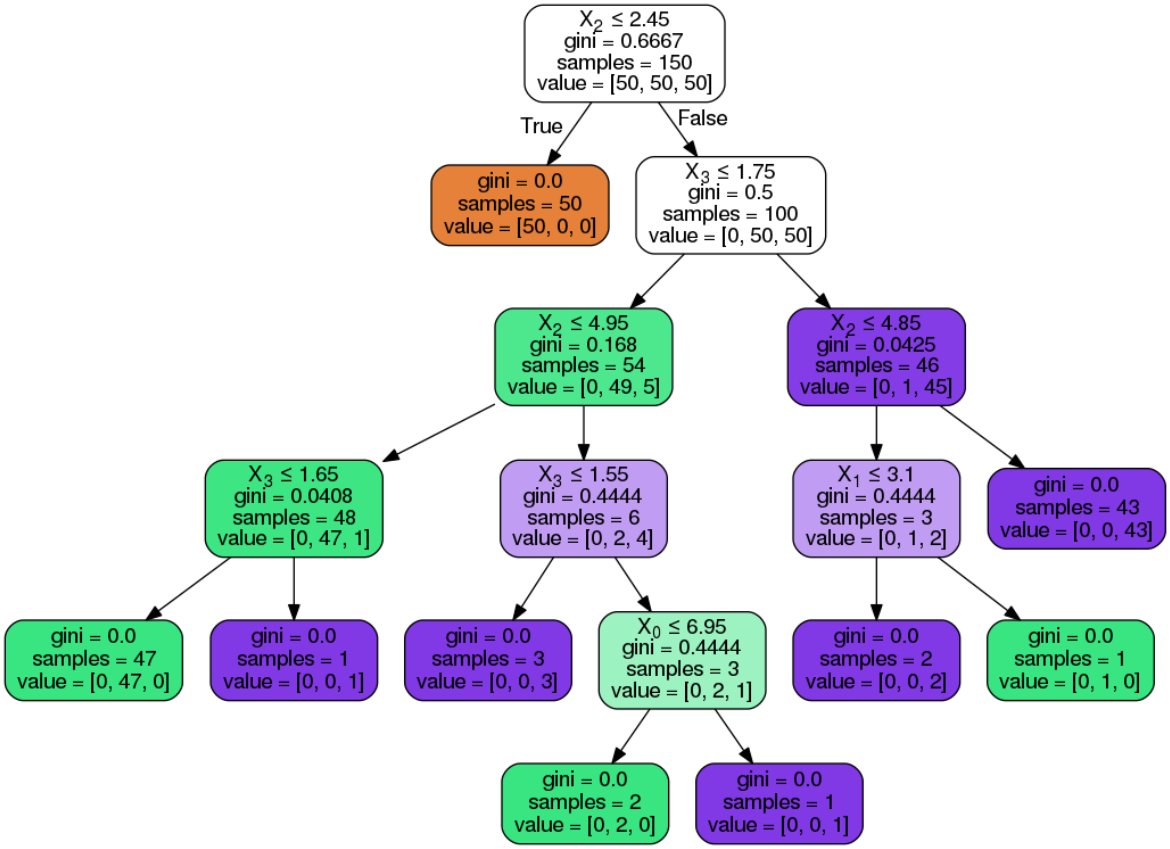

Figure 5. 1 An example of Decision Trees. X with subscripts are input sample features. Gini is a value which can quantify the performance of the Decision Tress splitting.

The process of converting a decision tree to an if-then rule is as follows: 1) Construct a rule from the root node of the decision tree to each path of the leaf node; 2) The characteristics of the internal nodes of the path correspond to the conditions of the rule; 3) The conclusion of the class corresponding to the leaf node.

The path of the decision tree has an important property: mutually exclusive and complete; that is, each sample is covered by only one path. The decision tree learning algorithm is mainly composed of three parts: feature selection, decision tree generation, pruning of decision trees. Below, the theoretical introduction of these three aspects.

If I want to separate a group of people, should I first separate them by age, by gender, or by weight? And what criteria should I use to determine the classification ability of a feature? I need to introduce a concept here: information gain.

5.1.1 Entropy

Let's first introduce the concept: entropy. In information theory and probability theory, entropy is used to represent the measure of "random variable uncertainty" (Shannon 1997). Let X be a finite state discrete random variable whose probability distribution is

$$P(X = x_i) = p_i\,, i = 1,2, \dots, n, \qquad (5-1)$$

then the entropy of the random variable *X* is defined as

$$H(X) = -\sum_{i=1}^{n} p_i \log\,(p_i). \qquad (5-2)$$

The greater the entropy, the greater the uncertainty of the random variable. When the random variable has only 0 and 1 values when the hypothesis is. $P(X = 1) = p$

Thus,

$$H(X) = -p\log(p) - (1-p)\log(1-p)\,, \qquad (5-3)$$

then

$$\frac{\partial H(X)}{\partial p} = -\log\left(\frac{p}{1-p}\right). \qquad (5-4)$$

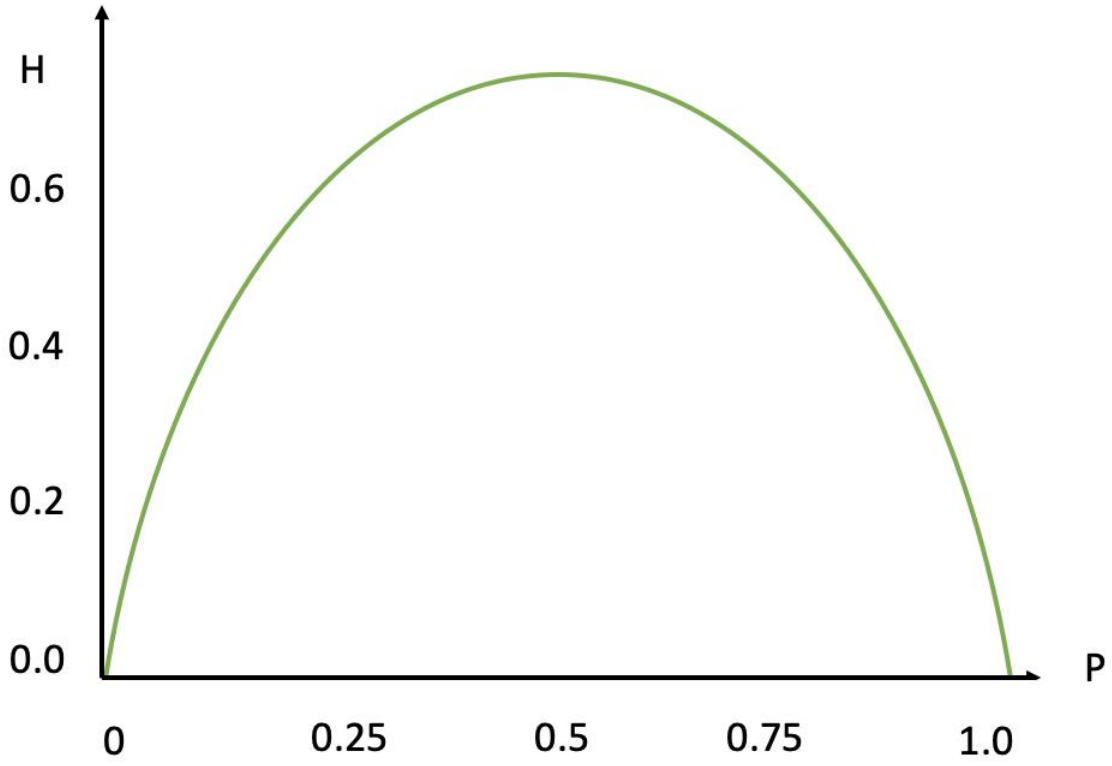

Figure 5. 2 Relationship between probability P and entropy H.

It can be seen from figure 5.2 that when p=0.5, the entropy takes the largest value, and the random variable has the largest uncertainty.

5.1.2 Conditional Entropy

Random variables X under the given conditions, the random variable Y conditional entropy $H(Y|X)$ is defined as:

$$H(Y|X) = \sum_{i=1}^{n} p_i \, H(Y|X = \mathrm{x}_i). \qquad (5-5)$$

Among them, $p_i = P(X = x_i)$,

$$\sum_{n=1}^{l} a_i = 1. \qquad (5-6)$$

5.1.3 Information Gain

Gain information is represented by that the information of the characteristic X-Y is such that the class information uncertainty reduced extent. The information gain of feature A on training data set D is $g(D, A)$ defined as the difference between the empirical entropy of set D $H(D)$ and the empirical conditional entropy $H(D|A)$ of D under the given condition A, i.e.

$$g(D, A) = H(D) - H(D|A). \qquad (5-7)$$

In general, the difference between entropy $H(Y)$ and conditional entropy $H(Y|X)$ is called mutual information.

The method of feature selection according to the information gain criterion is: for the training data set D, the information gain of each feature is calculated and compared with their size, thereby selecting the feature with the largest information gain.

Suppose the training data set is D, the sample size is |D|, and there are K categories C_k, $|C_k|$ is the sample number in category C_k,

Feature A has n different values $a_1, a_2, \ldots a_n$. Dataset D may be divided into n subsets $D_1, D_2, \ldots, D_n$ according to the values of feature A. $|D_i|$ is the sample number in subset D_i. The set of category C_k in subset D_i is noted as D_{ik} and $|D_{ik}|$ is the sample number of D_{ik}.

The algorithm for information gain is as follows: Input: training data set D and feature A; Output: A feature information gain of the training data set D, (D, A).

(1) Calculate the empirical entropy data set D. $H(D)$

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log \frac{|C_k|}{|D|}. \quad (5-8)$$

(2) Wherein A computing experience conditional entropy of the data set D. $H(D|A)$

$$H(D|A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{D_i} \log \frac{|D_{ik}|}{D_i}. \quad (5-9)$$

(3) Calculate information gain

$$g(D, A) = H(D) - H(D|A). \quad (5-10)$$

5.1.4 Information Gain Ratio

With the information gain as the feature selection criterion, there is a problem that the feature with a larger value is selected. This problem can be corrected by using the information gain ratio.

The information gain ratio of feature A to training data set D is defined as the ratio of its information gain to the entropy of training set D with respect to the value of feature A,

$$gR(D|A) = \frac{g(D, A)}{H_A(D)}, \quad (5-11)$$

among them,

$$H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log \frac{|D_i|}{|D|}, \qquad (5-12)$$

5.2 Decision Trees Generation Algorithms

There are many variants of the decision tree generation algorithm. Here are several classic implementation algorithms: the ID3 algorithm, the C4.5 algorithm, and the CART algorithm. The main difference between these algorithms is that the selection criteria for feature selection on the classification nodes are different. Let's take a closer look at the specific implementation of the algorithm.

5.2.1 ID3 Algorithm

The core of the ID3 algorithm is to apply information gain criteria to feature selection at each node of the decision tree. The specific approach is: Starting from the root node, calculating the information gain of all possible features for the node, selecting the feature with the largest information gain as the feature of the node, and constructing the child node by different values of the feature;

Recursively calling the above methods on the child nodes to build a decision tree; Until the information gain of all features is small or no features are available.

5.2.2 C4.5 Algorithm

The difference between the C4.5 algorithm and the ID3 algorithm is that it uses the information gain ratio for feature selection in the process of producing decision trees.

5.2.3 CART Algorithm

The classification and regression tree (CART), like the C4.5 algorithm, evolved from the ID3 algorithm. The CART hypothesis decision tree is a binary tree that divides the feature space into finite elements by recursively dividing each feature and determines the predicted probability distribution of these elements.

In the CART algorithm, for the regression tree, the square error minimization criterion is adopted; for the classification tree, the Gini index minimization criterion is adopted.

5.2.4 Square Error Minimization

Suppose you have divided the input space into M units R_1 , R_2 , … , R_m and every unit Rm have a fixed output value C_m so that the regression tree can be expressed as

$$f(x) = \sum_{m=1}^{M} c_m \, I(x \in R_m). \qquad (5-13)$$

When the partition of the input space is determined, the squared error can be used $\sum_{xi \in R_m}(y_i - f(x_i))^2$ to represent the prediction error of the regression tree for the training data.

5.3 Gini Index

In the classification problem, if there are K categories and pk the probability that the sample points belong to the first category k, then the Gini index of the probability distribution is defined as

$$Gini(p) = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2 . \qquad (5-14)$$

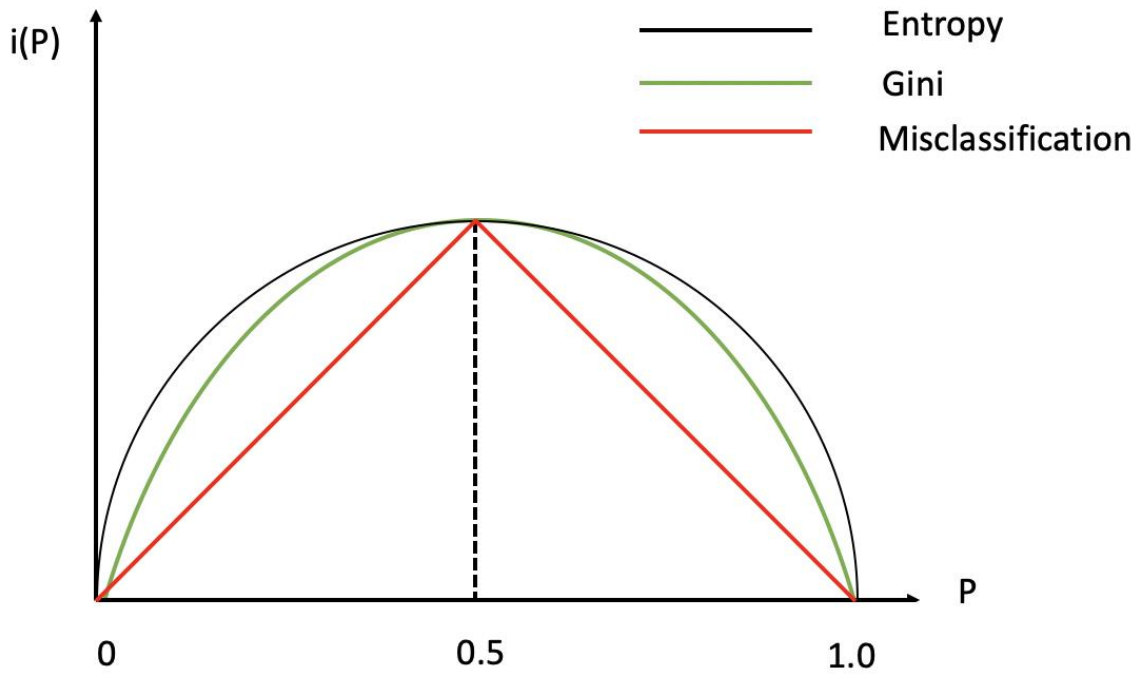

Figure 5. 3 Entropy and Gini distribution over the possibility. P is the possibility, i(P) is the corresponding out of a given P either for entropy or Gini.

5.4 Pruning of Decision Trees

If a complete decision tree is established for the training set, the model will be too much for the training data, and most of the noise is fitted, that is, the phenomenon of over-fitting occurs. To avoid this problem, there are two solutions: 1) When the number of entropy reductions is less than a certain threshold, the creation of the branch is stopped. This is a greedy algorithm. 2) Create a complete decision tree first, and then try to eliminate the extra nodes, that is, using the method of branching.

There is a potential problem with Method 1: It is possible that the creation of a branch will not cause a large drop in entropy, but subsequent sub-branches may cause a large decrease in entropy. Therefore, I prefer to use the method of pruning.

The pruning of the decision tree is achieved by minimizing the loss function of the decision tree as a whole. Based on improving the information gain, by imposing a penalty on the complexity T of the model, the definition of the loss function is obtained:

$$C_\alpha(T) = \sum_{t=1}^{|T|} N_t \, H_t(T) + \alpha|T|, \qquad (5-15)$$

α reflects the trade-offs between model training set fit and model complexity. The process of pruning is to select the model with the smallest loss function when α is fixed. The specific algorithm is as follows:

1. Calculate the empirical entropy of each node;

2. Recursively retracting from the leaf node of the tree. If all the leaf nodes of a parent node are merged, so that the loss function can be reduced, pruning is performed, and the parent node is changed into a new leaf node;

3. Return 2 until you cannot continue the merge.

5.5 Results of Facies Classification

In this section, I will show the confusion matrix of facies prediction results, heat map, and adjacent facies prediction results of decision tress algorithm.

Table 5. 1 Confusion matrix of decision trees facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	17	8	1							26
CSiS	4	71	18		1					94
FSiS	2	22	49		3			2		78
SiSh		1		18	3			5		27
MS		1	1	3	11	8		5		29
WS		1	1	4	9	35		8		58
D					1		8	5		14
PS			1	2	2	13	3	46	1	68
BS							2	1	15	18
Precision	0.74	0.68	0.69	0.67	0.37	0.62	0.62	0.64	0.94	0.66
Recall	0.65	0.76	0.63	0.67	0.38	0.60	0.57	0.68	0.83	0.66
F1	0.69	0.72	0.66	0.67	0.37	0.61	0.59	0.66	0.88	0.66

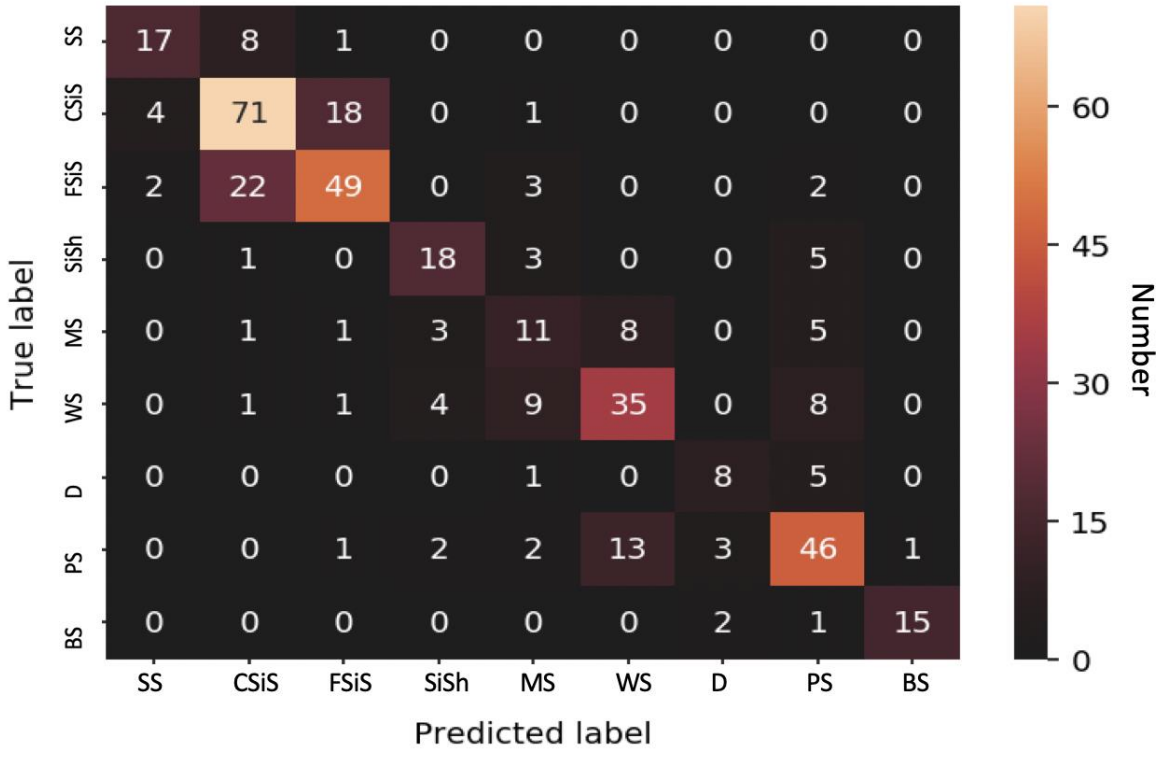

Figure 5. 4 Heat map of blind test results of decision trees. Facies labels represent specific facies which are explained in table 1.1.

Table 5. 2 Confusion matrix of decision trees adjacent facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	25		1							26
CSiS		93			1					94
FSiS	2		71		3			2		78
SiSh		1		21				5		27
MS		1	1		22			5		29
WS		1	1	4		52				58
D					1		13			14
PS			1	2	2			63		68
BS									18	18
Precision	0.93	0.97	0.95	0.78	0.76	1.00	1.00	0.84	1.00	0.92
Recall	0.96	0.99	0.91	0.78	0.76	0.90	0.93	0.93	1.00	0.92
F1	0.94	0.98	0.93	0.78	0.76	0.95	0.96	0.88	1.00	0.92

As shown in table 5.1 and 5.2, the F-1 scores of exact facies and adjacent facies of the decision tree algorithm are 0.66 and 0.92, respectively. The results indicate that in our classification task decision tree has almost the same performance as KNN has. I have published the decision tree model to Github (https://github.com/weixiongdi/PyFacies) for reference. In the next chapter, I will introduce the artificial neural network algorithm.

6. Artificial Neural Networks

In this chapter, I will introduce artificial neural networks, starting with a quick tour of the evolvement history of ANN architectures. Then I will implement ANN with Keras to tackle our facies classification problem. I will also implement some techniques like regularization and drop out which can improve the neural network performance and analyze the results at the end of this chapter.

6.1 Introduction: From Biological to Artificial Neurons

ANNs are one of the most popular machine learning algorithms today. They are versatile, powerful, and scalable (Hornik et al. 1989). With the development of ANN algorithms, nowadays ANN has many different branches which can tackle large and highly complex machine learning tasks, such as images classification (e.g., Faces Recognition), powering speech recognition services (e.g., Amazon's Echo), recommending the best videos to watch to hundreds of millions of users every day (e.g., Netflix), or learning to beat the world champion at the game of Go (DeepMind’s AlphaGo).

ANNs are powerful machine-learning algorithms (LeCun et al. 2015; Goodfellow et al. 2016) that provide numerous successful applications in computer vision, speech processing, and artificial intelligence problems. In particular, ANNs provide great potentials in imaging inverse problems such as de-noising (Burger et al. 2012; Xie et al. 2012), rock facies recognition (Wei et al.), super-resolution (Dong et al. 2016), compressed sensing (Sun et al. 2016), and X-ray computed tomography (Wang 2016; Würfl et al. 2016). For these reasons,

there is great promise in using this approach to approximate complex functions that are highly nonlinear.

ANNs were first introduced back in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts. In their landmark paper, "A Logical Calculus of Ideas Immanent in Nervous Activity," McCulloch and Pitts proposed a simplified computational model combine with biological neurons in animal brains to perform complex computations using propositional logic. They invented the first artificial neural network architecture and since then many other architectures have been proposed and developed.

Scientists had a widespread belief that humans would soon be conversing with truly intelligent machines due to the early success of ANNs until the 1960s. However, after several tests, people realized that this promise would go unfulfilled (at least for quite a while), then ANNs fell out of scientists' favor and funding quickly flew out. This situation didn't change until the early 1980s, when new network architectures were invented, and better training techniques were developed, which led to a revival of interest in ANNs. But by the 1990s, most researchers favored powerful alternative machine learning techniques such as Support Vector Machines (see Chapter 3), as they can offer better results and stronger theoretical foundations. Finally, in the 21st century we are now witnessing yet another wave of interest in ANNs and this wave is getting stronger and stronger.

Thanks to Moore' s Law and the gaming industry, which has produced powerful GPU cards, computing power has increased tremendously since the 1990s. Now it is possible to train large neural networks in a reasonable amount of time. And also thanks to the relatively small tweaks and improvement of training algorithms, the algorithm results have a huge enhancement compared to original algorithms used in the 1990s.

Of course, there are still some theoretical limitations of ANNs that need scientists to devote effort to, however in practice, some limitations have turned out to be benign. For example, theoretically, ANN training algorithms were likely to get stuck in local optima, but it turns out to be rather rare in practice.

6.1.1 Perceptron

What is a neural network? Before I jump into introducing modern neuron networks, I will first describe a type of artificial neuron called a perceptron, then explain the more commonly used artificial neuron model – sigmoid neurons. Sigmoid neurons were evolved from perceptrons, so it is worth taking the time first to understand perceptrons in order to understand why sigmoid neurons are defined the way they are. Perceptrons were first proposed in the 1950s and 1960s by the scientist Frank Rosenblatt, following the earlier work by Warren McCulloch and Walter Pitts (Mcculloch and Pitts 1990). A perceptron takes several binary inputs, X_1, X_2 …, and produces a single binary output:

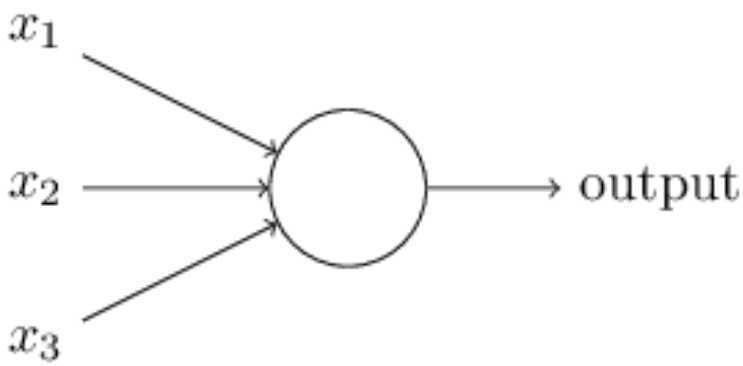

Figure 6. 1 Illustration of the perceptron.

The single binary output, 0 or 1, is determined by whether the weighted sum $\sum_j w_j x_j$ is less than or greater than some threshold value. A threshold is a real number, which is a parameter of the neuron. To express it in algebraic terms,

$$output = \begin{cases} 0 \; if \; \sum_j w_j x_j \leq threshold \\ 1 \; if \; \sum_j w_j x_j > threshold \end{cases}. \qquad (6-1)$$

The first change is to write $\sum_j w_j x_j$as a dot product, $\mathrm{W.X} = \sum_j w_j x_j$, where W and X are vectors of the weights and inputs, respectively. Then define the perceptron's bias, b= -threshold , and move the threshold to the other side of the inequality and to replace it by bias. Equation (6-1) can be rewritten as

$$output = \begin{cases} 0 \; if \; w.x + b \; \leq 0 \\ 1 \; if \; w.x + b \; > 0 \end{cases}. \qquad (6-2)$$

Suppose I have a network of perceptrons that I'd like to use to learn to solve some problems. To see how learning might work, suppose I make a small change in some weight (or bias) in the network. What I would like is a small change in weight can only cause a small corresponding change in the output from the network, and this property will make learning possible. Figure 6.2 shows the learning process schematically.

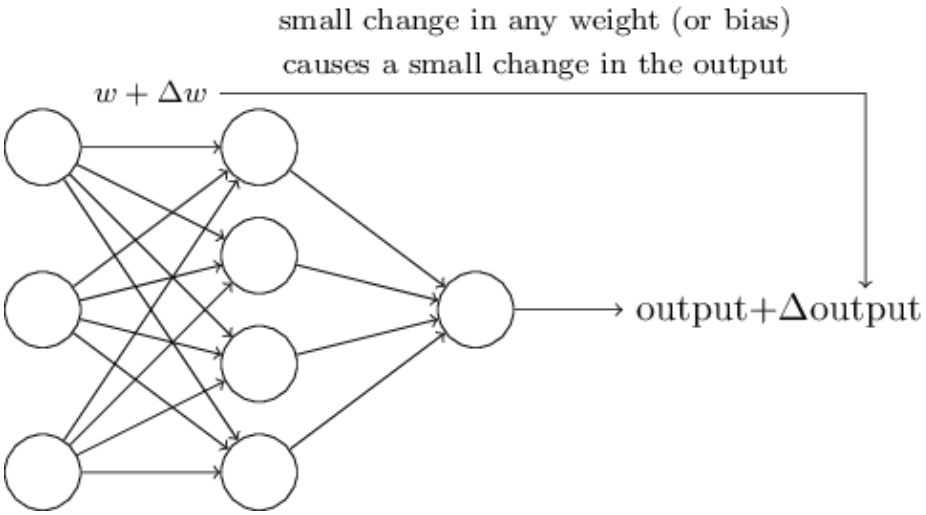

Figure 6. 2 Small change in weights (w) can cause a small change in neural network output.

If we take a close look at equation 6-2, we will find that small change in weights or bias of any perceptrons in the network can sometimes cause the output change drastically, say from 0 to 1 completely. Luckily, scientists have found a solution to overcome this problem by introducing a sigmoid neuron. Sigmoid neurons are a new type of neurons that are similar to perceptrons, but they are modified to make sure that small changes in the weights and bias cause only a small change in the output. This small change is the crucial fact that will allow a network of sigmoid neurons to learn.

Just like a perceptron, the sigmoid neuron has weights for each input, W_1, W_2,…, and overall bias, b. But the output is not 0 or 1. Instead, it's $\sigma(W.X + b)$, where σ is called the sigmoid function, and is defined by

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \qquad (6-3)$$

To put it all more explicitly, the output of a sigmoid neuron with inputs X_1, X_2 … weights W_1, W_2, and bias b is

$$\frac{1}{1+\exp\left(-\sum_j w_j x_j - b\right)}. \qquad (6-4)$$

The smoothness of σ means that small changes Δw_j in the weights and Δb in the bias will produce a small change $\Delta output$ in the output from the neuron. In fact, calculus tells us that $\Delta output$ is well approximated by

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j}\Delta w_j + \frac{\partial output}{\partial b}\Delta b\,. \qquad (6-5)$$

The goal is to find an algorithm that can find a group of weights and biases so that the output from the network approximates y(x) for all training inputs x. I can define a cost function as follows to quantify how well the outputs approximate the labels,

$$C(w,b) = \frac{1}{2n}\sum_x \|y(x) - a\|^2\,. \qquad (6-6)$$

So our training algorithm will aim to minimize the cost C(w,b) as a function of the weights and biases. In other words, I want to find a set of weights and biases which make the cost as small as possible. I will do that using an algorithm known as gradient descent.

6.1.2 ANN Architecture

I will focus on the neural network as a machine learning algorithm in this dissertation. The theoretical justification for this algorithm is based on the Universal Approximation Theorem (Hornik et al. 1989) where, loosely speaking, any function can be approximated by a neural network.

Generally, ANNs are composed of input, hidden, and output layers. As depicted in Figure 6.3, each hidden layer has multiple neural with connections to previous and following layers neural. Each connection has a weighted parameter. The input data will be transformed or

adjusted to implement on the input layer, which is followed by a varying number of hidden layers, and the result will be computed at the output layer. The training process is a way to interactively update the weighted parameters to minimize the error between predicted and input ones.

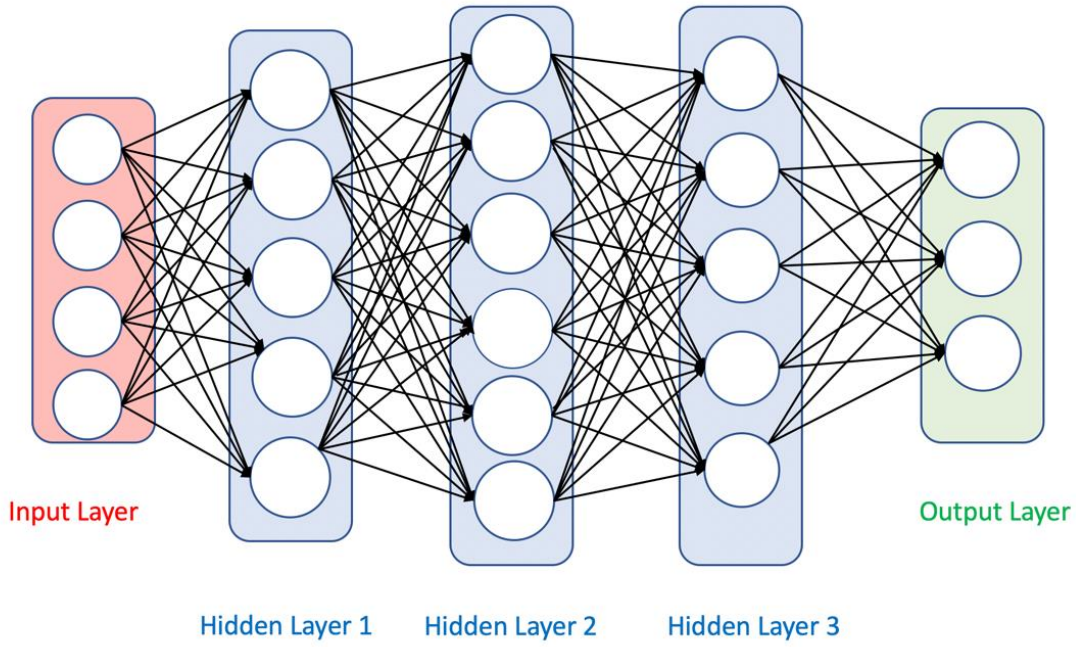

Figure 6. 3: Topology of a neural network.

6.1.3 Gradient Descent

In the section 6.1.1, I defined a cost function and the goal of training a neural network is to find a set of weights and biases which minimize the cost function C(w,b). To simplify the equation let's suppose I am trying to minimize some function, C(v). This could be any function of many variables, $v=v_1,v_2\ldots$ Let's start with function C with only two variables, which I will call v_1 and v_2.

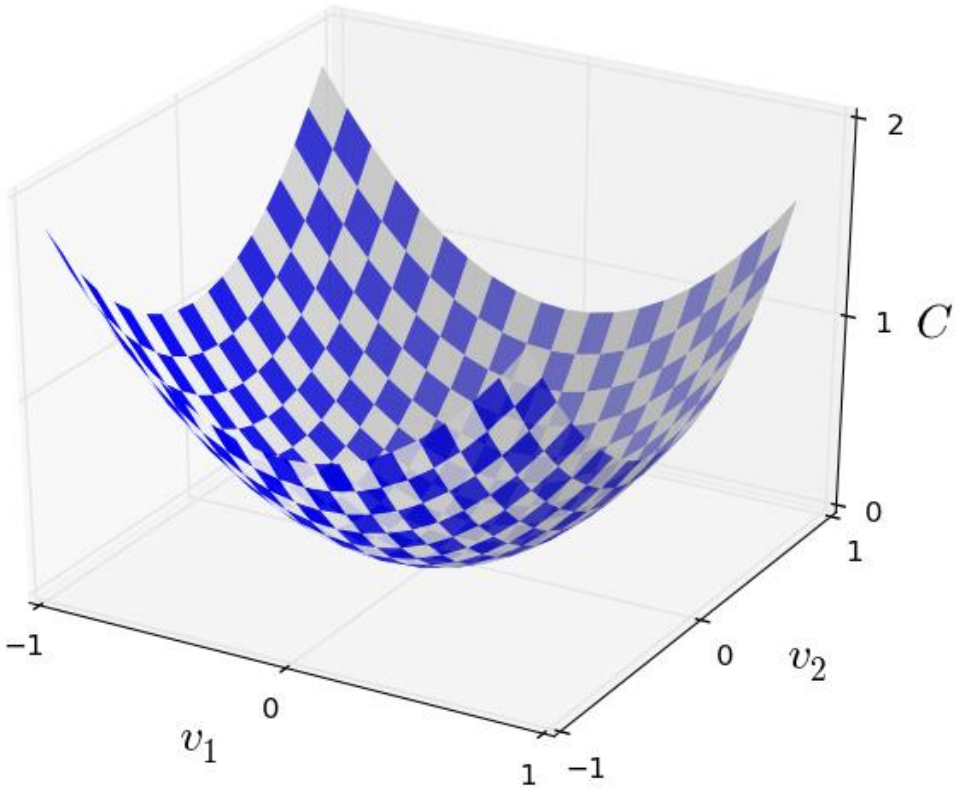

Figure 6. 4: An example of gradient descent. C is cost function, v_1 and v_2 are two variables of cost function C.

The analytical solution would be to use calculus to find the minimum of the cost function. It might work when C is a function of just one or a few variables. However, in neural networks, I will often want far more variables - the biggest neural networks have cost functions that depend on millions or even billions of weights and biases in an extremely complicated way. Using calculus to minimize that would be a nightmare and just wouldn't work!

Let me use an analogy to address the question more precisely. Let's think about what happens when I move the ball a small amount Δv_1 in the v_1 direction, and a small amount Δv_2 in the v_2 direction. Calculus tells us that C changes as follows:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2. \qquad (6-7)$$

I am going to find a way of choosing $\Delta v_1$and Δv_2 so as to make ΔC negative. First let me define Δv to be the vector of changes in V, $\Delta v = (\Delta v_1, \Delta v_2)^T$, where T is again the transpose operation. I will also define the gradient of C to be the vector of partial derivatives, $(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2})^T$. I denote the gradient vector by ΔC, i.e.:

$$\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T, \qquad (6-8)$$

with these definitions, the expression 6-8 for ΔC can be rewritten as

$$\Delta C \approx \nabla C . \Delta v. \qquad (6-9)$$

This equation helps explain why ΔC is called the gradient vector: ΔC relates changes in v to changes in C, just as I would expect something called a gradient to do. But what's exciting about the equation is that it lets us see how to choose Δv to make ΔC negative. In particular, suppose I choose

$$\Delta v = -\eta \nabla C, \qquad (6-10)$$

where η is a small, positive parameter (known as the learning rate). Then Equation 6-9 tells us that$\Delta C \approx -\eta \nabla C * \nabla C = -\eta \|\nabla C\|^2$ Because$\|\nabla C\|^2 \geq 0$, this guarantees that$\Delta C \leq 0$, i.e., C will always decrease, never increase

$$v' = v - \eta \nabla C. \qquad (6-11)$$

I've explained gradient descent when C is a function of just two variables. But everything works just as same as when C is a function of many more variables.

So in the training process of a neural network, I will use gradient descent to find the weights w_k and biases b_l, which minimize the cost in Equation 6-5. Let's restate the gradient descent update rule by replacing the variables v_j with the weights and biases. In other words, our original position has components w_k and b_l, and the gradient vector ∇C has corresponding components $\frac{\partial C}{\partial w_k}$ and $\frac{\partial C}{\partial b_l}$. Writing out the gradient descent update rule in terms of components, I have

$$w_k' = w_k - \eta \frac{\partial C}{\partial w_k}, \tag{6-12}$$

$$b_l' = b_l - \eta \frac{\partial C}{\partial b_l}. \tag{6-13}$$

By repeatedly applying this update rule, I can iteratively update the w_k and b_l and finally find a minimum of the cost function.

6.1.4 Backpropagation Algorithm

Formally, in all neural network methods, backpropagation is used to propagate the misfit error of loss function back through the whole network to update the neural network parameter $\Theta_{ij}^{(l)}$. This process is called the "training process" for the neural network, and the data used is called training data. After implementing the training process, I use the updated $\Theta_{ij}^{(l)}$ to predict the results of a new test dataset. The $\Theta_{ij}^{(l)}$ is often updated by gradient descent method which is expressed as

$$\Theta_{ij}^{(l)}new = \Theta_{ij}^{(l)} - \alpha * \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta), \tag{6-14}$$

where $\Theta_{ij}^{(l)}$ is the neural parameter of i_{th} sample, j_{th} neural of l_{th} layer; α is learning rate which controls the speed of gradient descent, and $J(\Theta)$ is the loss function. The subscript "new" represents the neural parameter after updating. First, I randomly initialize all $\Theta_{ij}^{(l)}$ then update them iteratively through backpropagation.

I have a total of *m* samples in the dataset and L layers of neural networks. During the training process, the input *i-th* sample data is $x^{(i)}$ and the labeled output is $y^{(i)}$. For each sample, I first forward propagate each feature data through the whole neural network and then start from the last layer, the L_{th} layer, to calculate the error $\delta_i^{(L)}$,

$$\delta_i^{(L)} = a^{(L)} - y^{(i)}, \qquad (6-15)$$

where $a^{(L)}$ is the output of the L_{th} layer activation function; $y^{(i)}$ is the i_{th} sample labeled output and $\delta_i^{(L)}$ is the error between the output of the L_{th} layer activation function and the labeled output. Starting from the last layer, I backpropagate the error to layers L-1, L-2 and so on. First, I define an intermediate variable $\Delta_{ij}^{(l)}$

$$\Delta_{ij}^{(l)}new = \Delta_{ij}^{(l)} + a_j^{(l)}\delta_i^{(l+1)}, \qquad (6-16)$$

where $\Delta_{ij}^{(l)}$ is the intermediate variable related to the i_{th} sample, j_{th} neural of the l_{th} layer and $a_j^{(l)}$ is the activation function output of the j_{th} neural in the l_{th} layer. All $\Delta_{ij}^{(l)}$ initially, are set to zero, $\Delta_{ij}^{(l)}new$ means the updated $\Delta_{ij}^{(l)}$. Using the updated $\Delta_{ij}^{(l)}$, I can calculate the $D_{ij}^{(l)}$ in Equation (6-17),

$$D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)}new + \lambda\Theta_{ij}^{(l)} \qquad if\ j \neq 0,$$

$$D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)}new \qquad if\ j = 0. \qquad (6-17)$$

Where λ is the regularization parameter, and m is the total number of data samples. I can prove (Werbos 1994) that $D_{ij}^{(l)}$ is the partial derivative result of the loss function $J(\Theta)$

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}. \qquad (6-18)$$

With $D_{ij}^{(l)}$, I can rewrite Equation (6-14) as Equation (6-19) and implement gradient descent methods to update parameters $\Theta_{ij}^{(l)}$,

$$\Theta_{ij}^{(l)}new = \Theta_{ij}^{(l)}old - \alpha * D_{ij}^{(l)}. \qquad (6-19)$$

6.2 Results: Facies Classification Results with Original ANN

I built a simple neural network from scratch and incorporated it with gradient descent and backpropagation and apply it to our dataset. The results are shown below.

Table 6. 1 Confusion matrix of a normal neural network facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	1	25								26
CSiS		67	26	1						94
FSiS		24	52	1		1				78
SiSh			1	19		6		1		27
MS		2		4	1	14	1	7		29
WS			2	6	1	31	1	17		58
D				4		1	6	3		14
PS		1		3		20	1	41	2	68
BS				2		1		8	7	18
Precision	1.00	0.56	0.64	0.47	0.50	0.42	0.67	0.53	0.78	0.58
Recall	0.04	0.71	0.64	0.70	0.03	0.53	0.43	0.60	0.39	0.55
F1	0.07	0.63	0.65	0.57	0.06	0.47	0.52	0.57	0.52	0.56

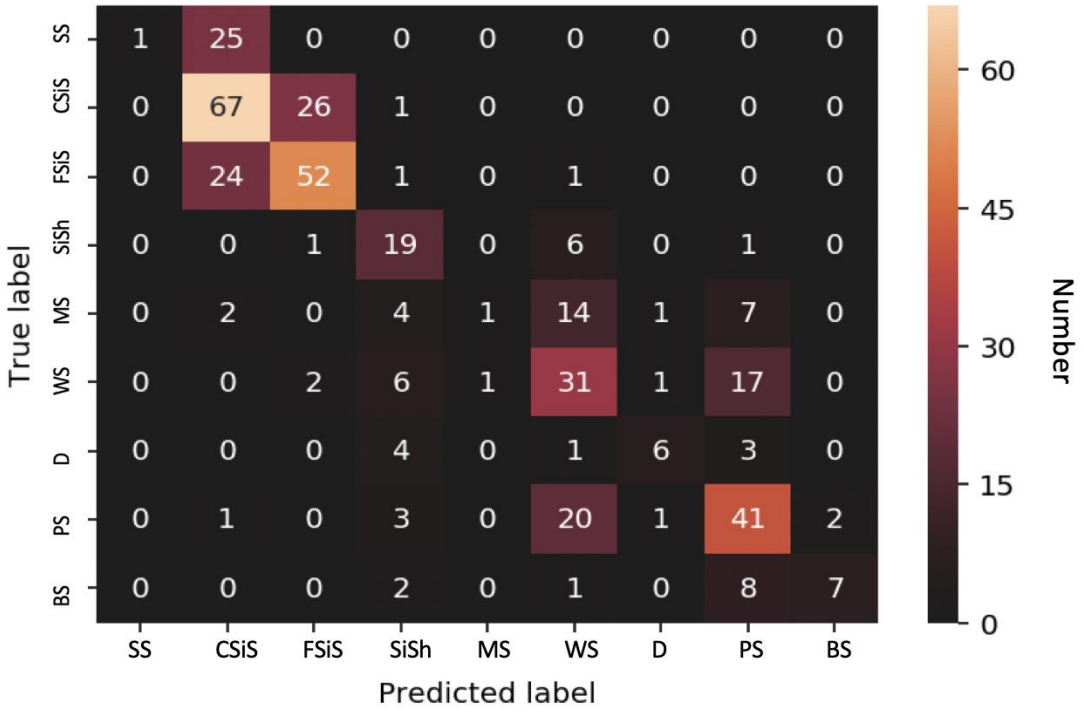

Figure 6. 5 Heat map of blind test results of a normal neural network. Facies labels represent specific facies which are explained in table 1.1.

Table 6. 2 Confusion matrix of a normal neural network adjacent facies prediction results. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	26									26
CSiS		93		1						94
FSiS			76	1		1				78
SiSh			1	19		6		1		27
MS		2			19		1	7		29
WS			2	6		50				58
D				4			10			14
PS			1	3				64		68
BS				2		1			15	18
Precision	1.00	0.97	0.96	0.53	1.00	0.86	0.91	0.89	1.00	0.91
Recall	1.00	0.99	0.97	0.70	0.66	0.86	0.71	0.94	0.83	0.90
F1	1.00	0.98	0.97	0.60	0.79	0.86	0.80	0.91	0.91	0.90

As shown in table 6.1 and 6.2, the F-1 scores of exact facies and adjacent facies of the ANN algorithm are 0.56 and 0.90, respectively. The normal neural network I used didn’t show a satisfactory performance considering it is one of the most famous and also powerful machine learning algorithms. In the next section, I will try to enhance the ANN learning performance and then improve the final classification results.

6.3 Improving the Way Neural Networks Learn

6.3.1 Regularization

The previous results show that my normal neural network model didn’t do a satisfactory job in facies classification. Neural network models often contain a large number of free parameters, even if such a model agrees well with the available data, the predicting results may still be bad. This phenomenon is called “overfitting”. It is because there's enough freedom in

the model that it can fit almost any data set of the given size, without capturing any genuine insights into the underlying phenomenon. I want our model has the ability to make predictions in situations it hasn't been exposed to before.

Fortunately, techniques known as regularization can reduce overfitting, even when I have a fixed network and fixed training data. In this section, I will describe one of the most commonly used regularization techniques, L2 regularization. The application of L2 regularization is to add an extra term, called the regularization term to the cost function. The new cot function will be like

$$\mathrm{C} = \frac{1}{2n}\sum_{x}\|y - a^{L}\|^{2} + \frac{\lambda}{2n}\sum_{w} w^{2}\,. \qquad (6-20)$$

In both cases, I can write the regularized cost function as

$$\mathrm{C} = C_{0} + \frac{\lambda}{2n}\sum_{w} w^{2}\,. \qquad (6-21)$$

Where C_0 is the original, unregularized cost function. Intuitively, from equation 6-21, we can tell that in order to minimize the cost function, the non-negative regularization term has to be small, which means the network prefers to learn small weights. In other words, regularization is a process try to find the balance between achieving small weights and minimizing the original cost function. The value of λ decides the relative importance of the two elements of the compromise: when λ is small I prefer to minimize the original cost function, but when λ is large I prefer small weights.

How do small weights affect neural networks? The smallness of the weights means that the behavior of the network won't change too much if I change a few random inputs. That makes it difficult for a regularized network to learn the effects of local noise in the data. It's an

empirical fact that regularized neural networks usually generalize better than unregularized networks. Indeed, researchers continue to write papers where they try different approaches to regularization, compare them to see which works better, and attempt to understand why different approaches work better or worse. I don't have an entirely satisfactory systematic understanding of what's going on, merely incomplete heuristics and rules of thumb.

6.3.2 Results: Facies Classification Results with L2 Regularization

Below are the results of our original neural network with L2 regularization.

Table 6. 3 Confusion matrix of ANN with regularization facies prediction results after L2 regularization. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	15	11								26
CSiS	10	62	21	1						94
FSiS	1	26	49	1		1				78
SiSh			1	18		7		1		27
MS		1	1	5	1	14	1	6		29
WS			2	5	1	34	2	14		58
D				3		2	6	3		14
PS			1	3		15	2	45	2	68
BS				2		1		7	8	18
Precision	0.58	0.62	0.65	0.47	0.50	0.46	0.55	0.59	0.80	0.58
Recall	0.58	0.66	0.63	0.67	0.03	0.59	0.43	0.66	0.44	0.58
F1	0.58	0.64	0.64	0.55	0.06	0.52	0.48	0.62	0.57	0.58

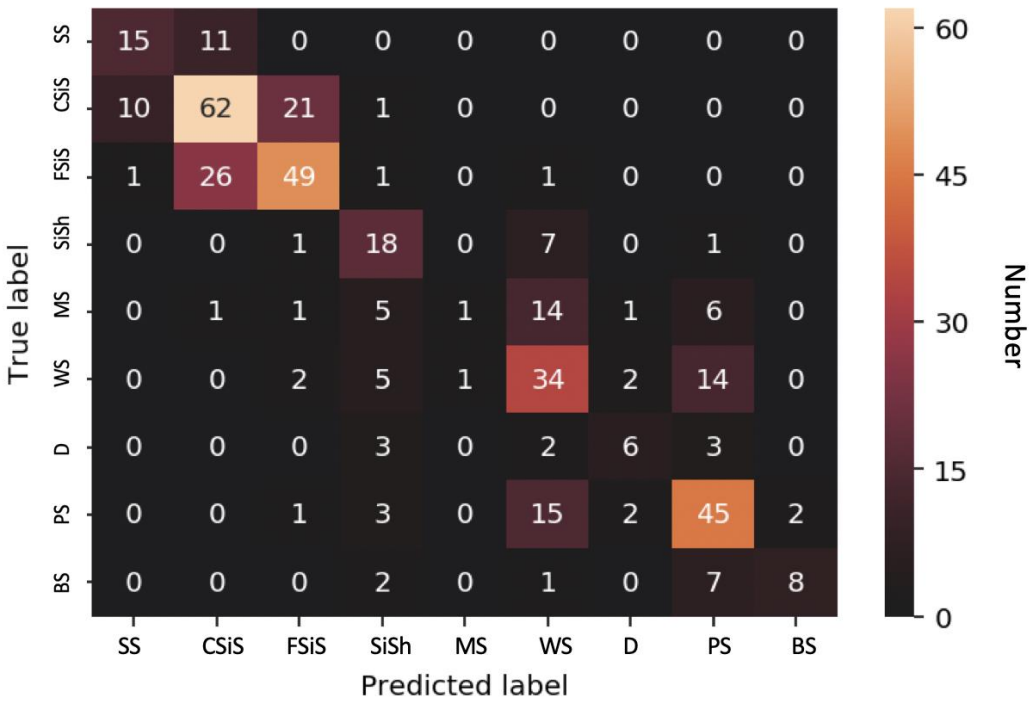

Figure 6. 6 Heat map of blind test results of ANN with regularization. Facies labels represent specific facies which are explained in table 1.1.

Table 6. 4 Confusion matrix of ANN with regularization adjacent facies prediction results after L2 regularization. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	26									26
CSiS		93		1						94
FSiS	1		75	1		1				78
SiSh			1	18		7		1		27
MS		1	1		20		1	6		29
WS			2	5		51				58
D				3			11			14
PS			1	3				64		68
BS				2		1			15	18
Precision	0.96	0.99	0.94	0.55	1.00	0.85	0.92	0.90	1.00	0.91
Recall	1.00	0.99	0.96	0.67	0.69	0.88	0.79	0.94	0.83	0.91
F1	0.98	0.99	0.95	0.60	0.82	0.86	0.85	0.92	0.91	0.91

6.3.3 Drop Out

In this section I will use another technique called “drop out” to improve the neural network results. Dropout is another technique for regularization. Dropout doesn't modify the cost function, instead, it modifies the network itself. Let me describe the basic mechanics of how dropout works.

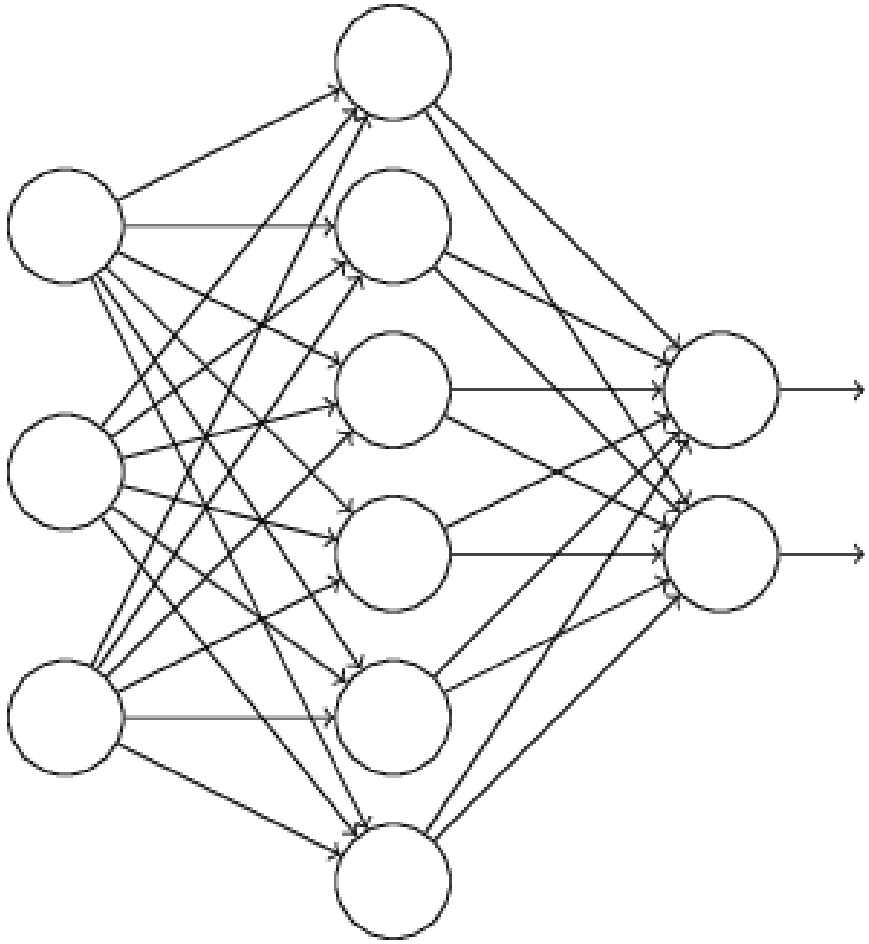

Figure 6. 7 A normal ANN architecture.

Particularly, the training process of neural networks is forward-propagating input x through the network, and then backpropagating to determine the contribution to the gradient. With dropout, I start by randomly inactivating a small portion of hidden neurons in the network while leaving the input and output neurons untouched. After doing this, I will sill have weights being updated through the training process, the only difference is that the weights are updated at different times.

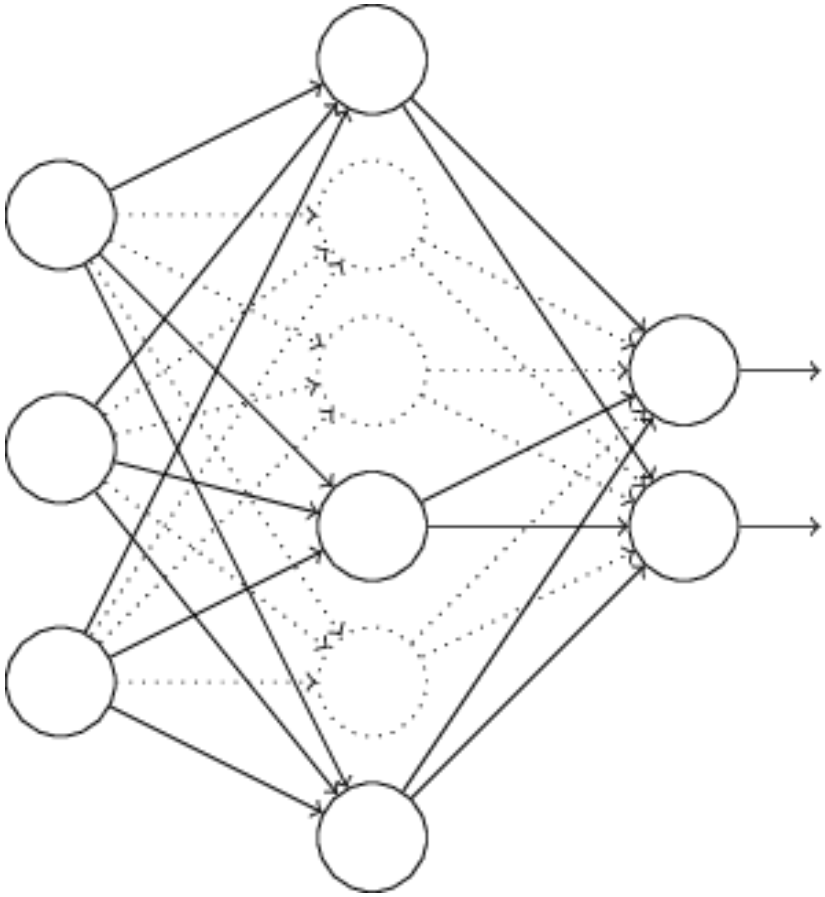

Figure 6. 8 An example of drop-out architecture.

By repeating this process over and over, our network will learn a set of weights and biases. Of course, those weights and biases will have been learned under conditions in which half the hidden neurons were dropped out.

This dropout procedure may seem strange, but how does it help with regularization? What does this have to do with dropout? Heuristically, when I drop out different sets of neurons, it's rather like I am training different neural networks. And so the dropout procedure is like averaging the effects of a very large number of different networks. The different networks will overfit in different ways, and so, hopefully, the net effect of dropout will be to reduce overfitting.

A related heuristic explanation for dropout is given in one of the earliest papers to use the technique in “ImageNet Classification with Deep Convolutional Neural Network”, by Krizhevsky et al. (2017): "This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons."

Dropout has a very successful history in improving the performance of neural networks. The original paper “Improving neural networks by preventing co-adaption of feature detectors”, by Hinton et al. (2012), introducing the technique applied it to MNIST digit classification. The paper noted that the best result anyone had achieved up to that point using such an architecture was 98.4 percent classification accuracy on the test set. They improved that to 98.7 percent accuracy using a combination of dropout and a modified form of L2 regularization. Dropout has yielded impressive results in tasking including image and speech recognition, and natural language processing. It has been especially useful in training large, deep networks, where the problem of overfitting is often acute.

6.3.4 Results: Facies Classification Results with Drop Out

Below are the results of our original neural network with regularization and dropout techniques.

Table 6. 5 Confusion matrix of facies prediction results with drop out and regularization applied. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	15	11								26
CSiS	10	58	25	1						94
FSiS	1	25	50	1		1				78
SiSh			1	19		6		1		27
MS		2		5	1	13	1	7		29
WS			2	6		33	2	15		58
D				1		2	8	3		14
PS			1	3		14	2	45	3	68
BS				1		2		6	9	18
Precision	0.58	0.60	0.63	0.51	1.00	0.46	0.62	0.58	0.75	0.61
Recall	0.58	0.62	0.64	0.70	0.03	0.57	0.57	0.66	0.50	0.58
F1	0.58	0.61	0.64	0.59	0.07	0.51	0.59	0.62	0.60	0.59

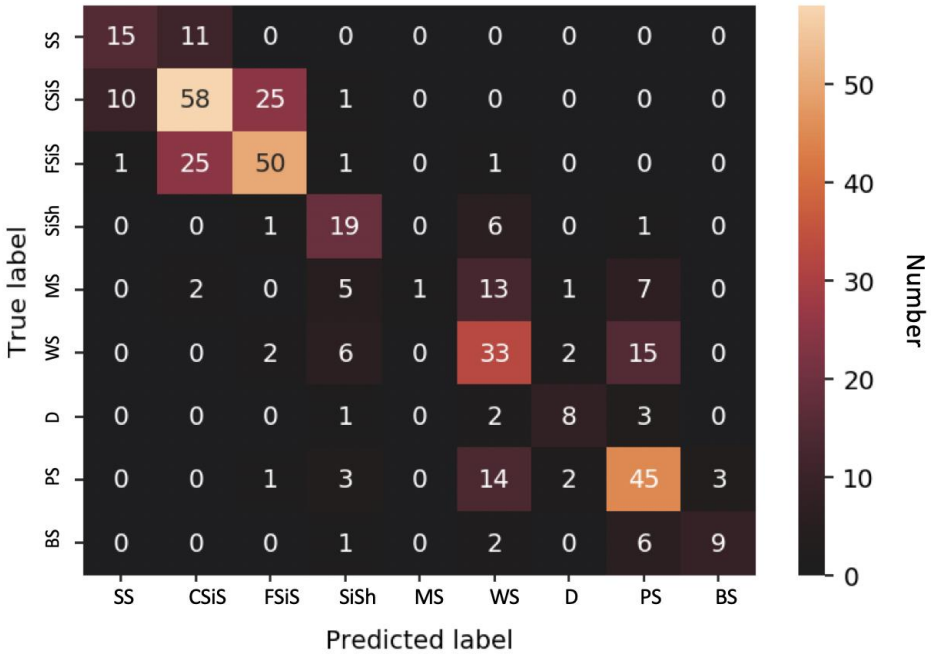

Figure 6. 9 Heat map of blind test results of ANN with drop out and regularization. Facies labels represent specific facies which are explained in table 1.1.

Table 6. 6 Confusion matrix of adjacent facies prediction results drop out and regularization applied. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	26									26
CSiS		93		1						94
FSiS	1		75	1		1				78
SiSh			1	19		6		1		27
MS		2			19		1	7		29
WS			2	6		50				58
D				1			13			14
PS			1	3				64		68
BS				1		2			15	18
Precision	0.96	0.98	0.95	0.59	1.00	0.85	0.93	0.89	1.00	0.91
Recall	1.00	0.99	0.96	0.70	0.66	0.86	0.93	0.94	0.83	0.91
F1	0.98	0.98	0.96	0.64	0.79	0.85	0.93	0.91	0.91	0.91

After I applied regularization and dropout techniques, our neural network generally performed better in terms of F-1 score. As shown in table 6.3, 6.4, 6.5 and 6.6, the exact facies F-1 score increased from 0.56 to 0.58, the 0.59; and adjacent facies F-1 score increased from 0.90 to 0.91, then remains 0.91 when dropout method applied. Compared to the previous several methods, the ANN didn't perform very well. In the next chapter, I will explore more and enhance the classification results by using a convolutional neural network.

7. Convolutional Neural Networks

In this section, I will briefly introduce the CNN algorithm of classifying rock facies and our proposed data padding strategies. The work showed in this chapter is published on journal Pure and Applied Geophysics with the article title “Characterizing Rock Facies Using Machine Learning Algorithm Based on a Convolutional Neural Network and Data Padding Strategy”.

7.1 Convolutional Layers and Pooling Layers

Generally, convolutional layers are the multiplication results of the input matrix and filters. Filters, or kernels, work as feature identifiers to classify different features of input data, which are usually images. As being seen in Figure 7.1, the filters scan through the input data matrix and generate convolutional layers. For each partial space it scans, the filter does a matrix element-wise multiplication with the partial data space and output a scalar. If the output scalar is significant, it means the filter and the partially scanned area of the image have the same features. The filters can detect lower-order features like edges, corners, or curves. When increasing the number of filters, the filters can detect more complicated features. The last convolutional layer output is the processed input of the previous convolutional layers. In the first step, I initialize the parameters of all filters, then update each filter parameter through the training process by back-propagating the errors through the whole CNN system to update the values of filter parameters.

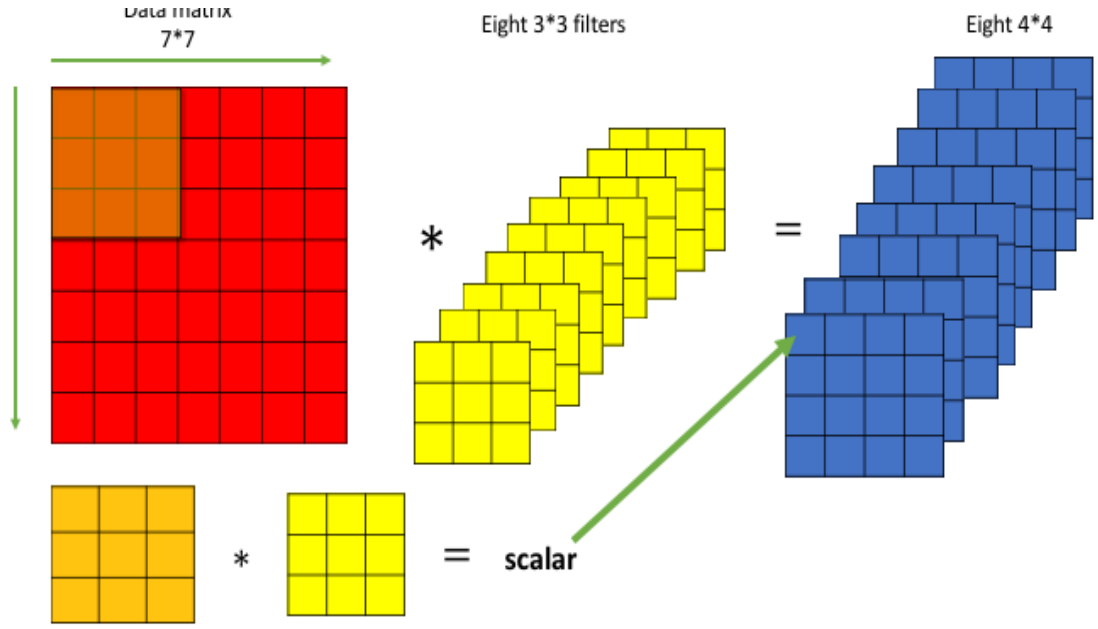

Figure 7. 1 Convolutional layers. For each filter (yellow box), it scans through the input data space (red boxes) from left to right and then top to bottom; each time it performs a matrix element-wise multiplication. The scalar output is an element in a convolutional layer. When the scanning process is done, the output forms a convolutional layer (blue box). Each filter will generate one convolutional layer. The program users normally decide the number of filters.

Pooling layers are used to reduce the spatial dimensions of the input data (Scherer et al. 2010). For example, as shown in Figure 7.2, a 2x2 maximum pooling layer filter scans through the input data, and for each 2x2 area, it outputs the largest of the four numbers. The pooling layer effectively combines several values into a single one, hence decreasing the chance of overfitting because some values in the data will be lost in the pooling phase.

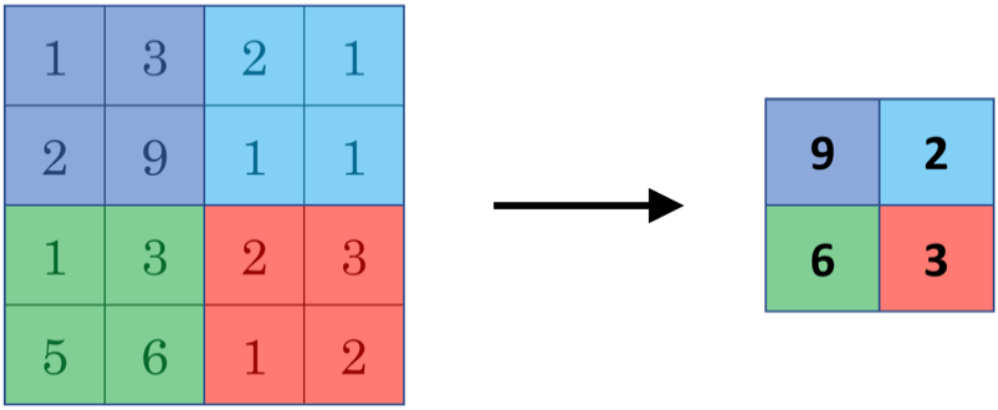

Figure 7. 2 Maximum pooling layers. The box in the left represents the convolutional layer, which is the input for the maximum pooling method. The left box is separated into four sections by four different colors, then the maximum pooling method selects the biggest number of each division and yields a maximum pooling layer.

7.2 Data Padding

In the ML field, data augmentation is often used to generate additional training data sets in order to aggregate training set size when training data is insufficient (Mikołajczyk and Grochowski 2018). For example, for an image of a cat, I can flip the image from left to right to create a new image of a cat. It is still the same cat but in a different position. The flipped cat image could be considered as a new image. As a result, the dataset is expanded. For 1-D data, data padding is a data augmentation technique by unit permutation. In our case, I use seven different features to predict rock facies. For 1-D CNN, the input should be a 1x7 matrix for one sample.

I propose a new method which is called “data padding” to maximize the ability of the CNN algorithm. I expand our dataset from 1-D to 2-D to be implemented by the CNN. Our “data padding” strategy is inspired by both industry data augmentation methods and the CNN algorithm itself. The CNN can recognize the features or the connections of an image, so I can

expand our input example to generate more features and help improve the ability of the network to recognize the right facies correctly.

During the implementation process, I test three types of data padding strategies: equal padding and shift padding. For one input example, I have seven features that form a 1x7 matrix (in Figure 7.3). For equal padding, I repeat the 1-D matrix six times and create a 7x7 matrix (see Figure 7.3a). For shift padding, I shuffled each row by one space each time (see Figure 7.3b). The number in each cell from 1 to 7 demonstrates seven data features.

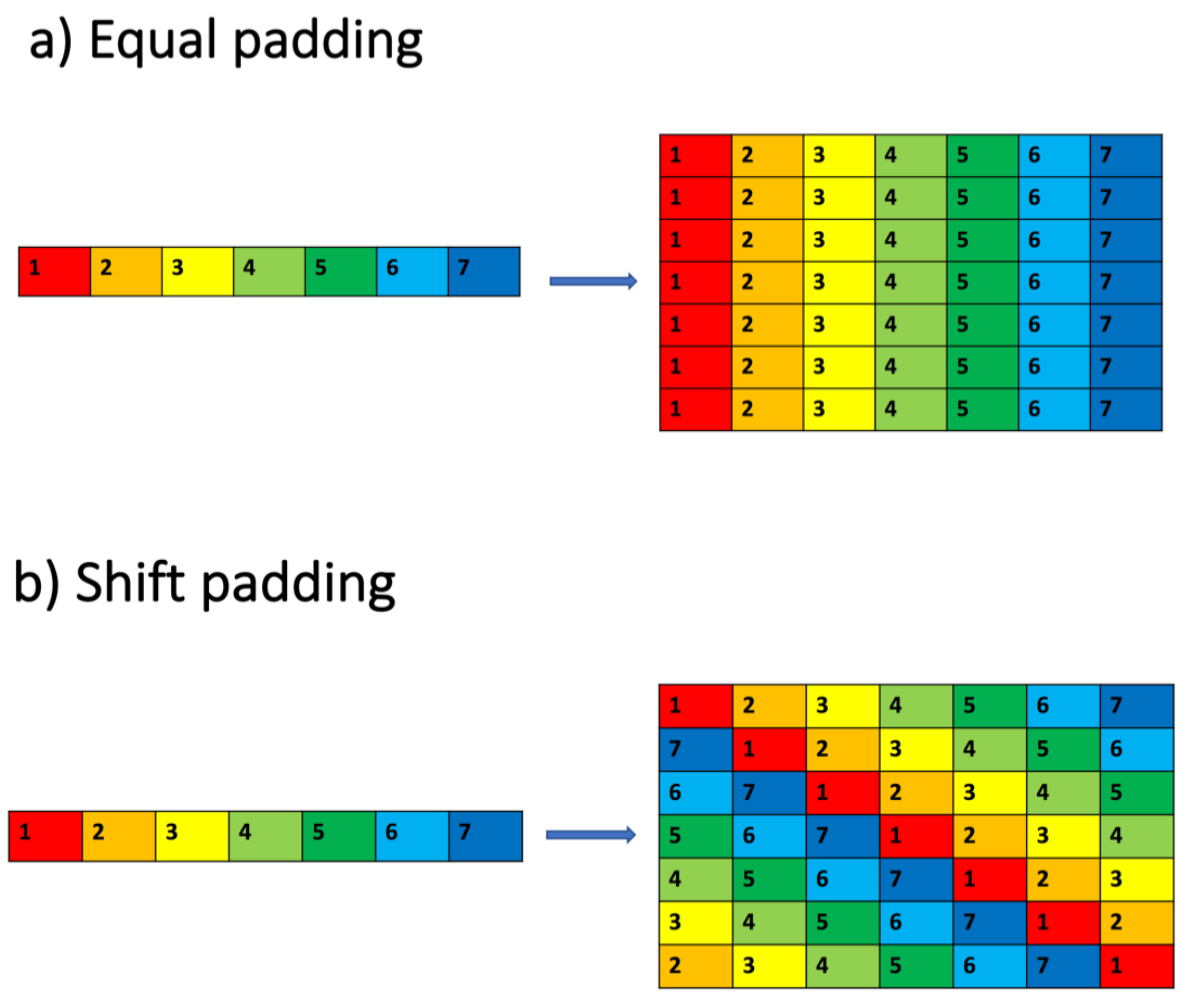

Figure 7. 3 Three padding strategies: a) equal padding, b) shift padding. The colors and corresponding numbers in each box represent seven features of the input sample, which are five wireline logs and two geological indicators.

7.3 Results: Facies Classification Results with Padding Strategies

7.3.1 Shift Padding

In this section, I will show the confusion matrix of facies prediction results, heat map, and adjacent facies prediction results of CNN with shift padding strategy.

Table 7. 1 Confusion matrix of facies prediction results with CNN shift padding. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	16	10								26
CSiS	6	68	20							94
FSiS	1	23	50	1	1	1		1		78
SiSh			1	20		5		1		27
MS	1	1			4	18	2	3		29
WS	1		1	3	3	43		7		58
D						1	9	4		14
PS		1	1	2	3	18	1	40	2	68
BS							2	1	15	18
Precision	0.64	0.66	0.68	0.77	0.36	0.50	0.64	0.70	0.88	0.64
Recall	0.62	0.72	0.64	0.74	0.14	0.74	0.64	0.59	0.83	0.64
F1	0.63	0.69	0.66	0.75	0.20	0.60	0.64	0.64	0.86	0.64

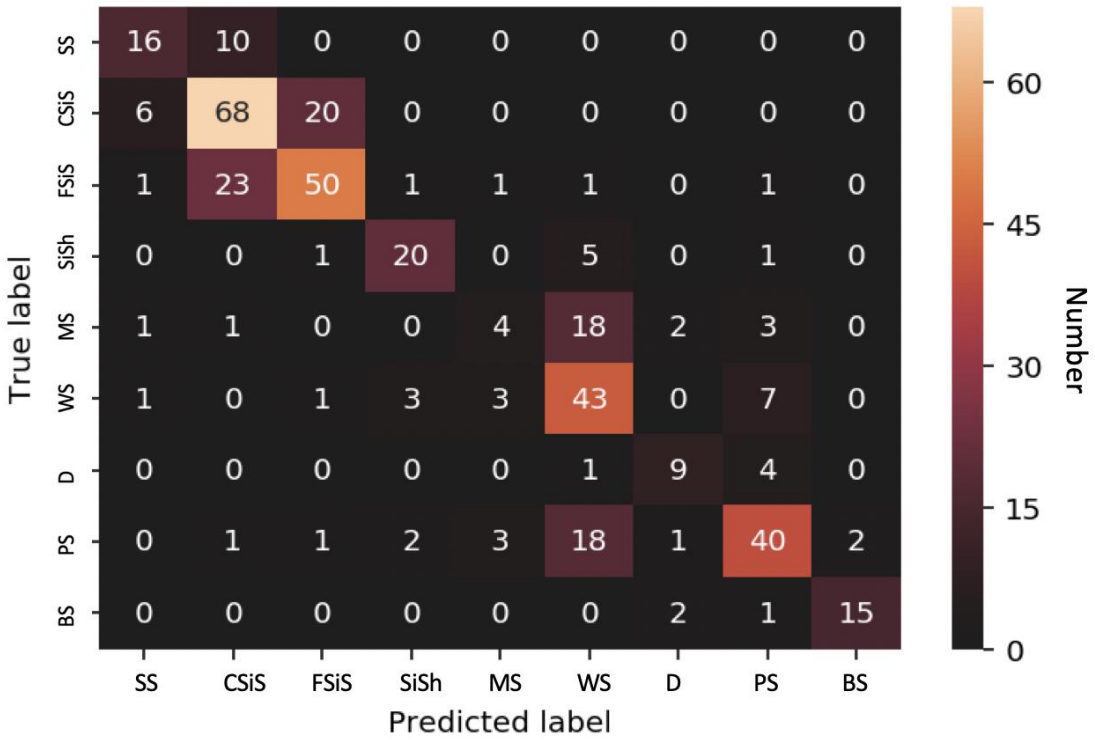

Figure 7. 3 Heat map of blind test results of CNN shift padding. Facies labels represent specific facies which are explained in table 1.1.

Table 7. 2 Confusion matrix of adjacent facies prediction results with CNN shift padding. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	26									26
CSiS		94								94
FSiS	1		73	1	1	1		1		78
SiSh			1	20		5		1		27
MS	1	1			22		2	3		29
WS	1		1	3		53				58
D							14			14
PS		1	1	2	3			61		68
BS									18	18
Precision	0.90	0.98	0.96	0.77	0.85	0.90	0.88	0.92	1.00	0.92
Recall	1.00	1.00	0.94	0.74	0.76	0.91	1.00	0.90	1.00	0.92
F1	0.95	0.99	0.95	0.75	0.80	0.91	0.93	0.91	1.00	0.92

7.3.2 Equal Padding

In this section, I will show the confusion matrix of facies prediction results, heat map, and adjacent facies prediction results of CNN with equal padding strategy.

Table 7. 3 Confusion matrix of facies prediction results with CNN equal padding. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	16	10								26
CSiS	5	68	21							94
FSiS	1	25	48	1	2			1		78
SiSh			1	21		1		4		27
MS		1	1	1	10	9	1	6		29
WS	1	1		6	11	27		12		58
D					3		8	3		14
PS			1	1	3	10		48	5	68
BS								1	17	18
Precision	0.70	0.65	0.67	0.70	0.34	0.57	0.89	0.64	0.77	0.64
Recall	0.62	0.72	0.62	0.78	0.34	0.47	0.57	0.71	0.94	0.64
F1	0.65	0.68	0.64	0.74	0.34	0.51	0.70	0.67	0.85	0.64

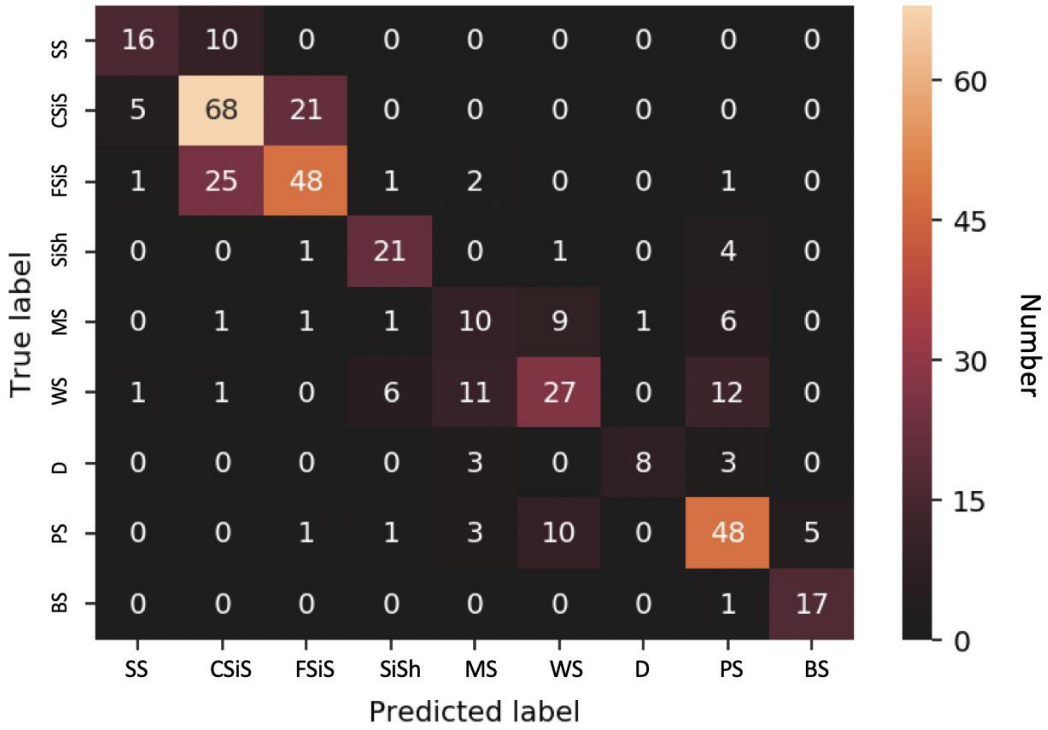

Figure 7. 4 Heat map of blind test results of CNN equal padding. Facies labels represent specific facies which are explained in table 1.1.

Table 7. 4 Confusion matrix of adjacent facies prediction results with CNN equal padding. Facies labels represent specific facies which are explained in table 1.1.

	Predicted facies									
True Facies	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	26									26
CSiS		94								94
FSiS	1		73	1	2			1		78
SiSh			1	21		1		4		27
MS		1	1		20		1	6		29
WS	1	1		6		50				58
D					3		11			14
PS			1	1	3			63		68
BS									18	18
Precision	0.93	0.98	0.96	0.72	0.71	0.98	0.92	0.85	1.00	0.91
Recall	1.00	1.00	0.94	0.78	0.69	0.86	0.79	0.93	1.00	0.91
F1	0.96	0.99	0.95	0.75	0.70	0.92	0.85	0.89	1.00	0.91

In this chapter, as shown in table 6.1, 6.2, 6.3 and 6.4, I successfully pushed up the F-1 score to 0.64 by using CNN incorporated with padding strategy. By padding our 1-D data into 2-D “image like” data, I have fully exploited the potential of CNN and achieved a better result compared to ANN. In the next chapter, I will make conclusions about our total results and make an analysis of the results thoroughly.

8. Discussion and Conclusions

In this dissertation, I have investigated the ability of facies classification of five machine learning algorithms. I have applied Support Vector Machine, k-Nearest Neighbors with and without feature engineering, Decision Trees, an Artificial Neural Network with the normal setting, with regularization and with regularization plus dropout, and Convolutional Neural Network. Under each algorithm category, I applied different techniques to increase the classification F-1 score. To better visualize the final results, I have complied with all the results of exact and adjacent facies classification results of all algorithms in Table 8.1 and 8.2, and I will analyze the results thoroughly in this section. Finally, I will discuss the future application of our methods.

Table 8. 1 Exact facies classification results of all algorithms. Facies labels represent specific facies which are explained in table 1.1.

Facies	SVM	KNN		DT	ANN			CNN		
		No FE	FE		Normal	Regularize	Dropout	Padding shift	Padding equal	Average F-1 score
SS	0.67	0.50	0.50	0.69	0.07	0.58	0.58	0.63	0.65	0.54
CSiS	0.74	0.69	0.69	0.72	0.63	0.64	0.61	0.69	0.68	0.68
FSiS	0.64	0.70	0.70	0.66	0.65	0.64	0.64	0.66	0.64	0.66
SiSh	0.55	0.63	0.63	0.67	0.57	0.55	0.59	0.75	0.74	0.63
MS	0.00	0.38	0.38	0.37	0.06	0.06	0.07	0.20	0.34	0.21
WS	0.47	0.61	0.61	0.61	0.47	0.52	0.51	0.60	0.51	0.55
D	0.19	0.62	0.62	0.59	0.52	0.48	0.59	0.64	0.70	0.55
PS	0.51	0.73	0.73	0.66	0.57	0.62	0.62	0.64	0.67	0.64
BS	0.60	0.81	0.81	0.88	0.52	0.57	0.60	0.86	0.85	0.72
Overall	0.59	0.65	0.66	0.66	0.56	0.58	0.59	0.64	0.64	
SVM= support vector machine; KNN= k-nearest neighbors; DT= decision trees; ANN=artificial neural network; CNN= convolutional neural network; FE= feature engineering										

Table 8. 2 Adjacent facies classification results of all algorithms. Facies labels represent specific facies which are explained in table 1.1.

Facies	SVM	KNN		DT	ANN			CNN		
		No FE	FE		Normal	Regularize	Dropout	Padding shift	Padding equal	Average F-1 score
SS	0.85	0.95	1.00	0.94	1.00	0.98	0.98	0.95	0.96	0.96
CSiS	0.98	0.98	0.97	0.98	0.98	0.99	0.98	0.99	0.99	0.98
FSiS	0.93	0.96	0.97	0.93	0.97	0.95	0.96	0.95	0.95	0.95
SiSh	0.55	0.67	0.79	0.78	0.60	0.60	0.64	0.75	0.75	0.68
MS	0.76	0.81	0.86	0.76	0.79	0.82	0.79	0.80	0.70	0.79
WS	0.81	0.89	0.92	0.95	0.86	0.86	0.85	0.91	0.92	0.89
D	0.73	0.93	0.97	0.96	0.80	0.85	0.93	0.93	0.85	0.88
PS	0.90	0.89	0.92	0.88	0.91	0.92	0.91	0.91	0.89	0.90
BS	0.81	0.97	0.97	1.00	0.91	0.91	0.91	1.00	1.00	0.94
Overall	0.86	0.92	0.94	0.92	0.90	0.91	0.91	0.92	0.91	
SVM= support vector machine; KNN= k-nearest neighbors; DT= decision trees; ANN=artificial neural network; CNN= convolutional neural network; FE= feature engineering										

From table 8.1 and 8.2 I can see that for exact facies classification overall F-1 score, both KNN with feature engineering and decision tree achieve the highest F-1 score, which is 0.66. And for adjacent facies classification results, KNN with feature engineering has obtained 0.94 F-1 score, which is the highest among all machine learning algorithm results. However, other ML algorithms also did a good job in classification. Most of their F-1 scores are over 0.9. Let's take a look at all algorithms' classification performance on each facies individually. I listed the facies and their corresponding abbreviated labels in table 8.3. For facies SS and BS, decision tree has the best score; for facies CSiS, SVM achieves the highest score; for facies FSiS, MS and PS, KNN performs best, and for facies SiSh and D, CNN has a leading score. I also calculate the average F-1 score of all algorithms results of each facies; this number can be a measurement of "how easy" specific facies can be classified. So from easiest to hardest, the facies are BS, CSiS, FSiS, PS, SiSh, WS, D, SS, MS. I can see that facies MS is extremely

difficult to identify since it has a much lower average F-1 score compared to other facies. From a practical point of view, this character is very useful since domain experts will more efficiently arrange their analysis time if they have prior information on which facies is the hardest to identify.

Table 8. 3 Nine classes of lithofacies and corresponding abbreviated labels.

Facies label	1	2	3	4	5
Abbreviated Label	SS	CSiS	FSiS	SiSh	MS
Facies	Nonmarine sandstone	Nonmarine coarse siltstone	Nonmarine fine siltstone	Marine siltstone and shale	Mudstone (limestone)

Facies label	6	7	8	9
Abbreviated Label	WS	D	PS	BS
Facies	Wackestone (limestone)	Dolomite	Packstone-grainstone (limestone)	Phylloidalgal bafflestone (limestone)

Most neural network performances did beat simpler methods like KNN in our test, and the main reason is that our dataset is relatively small, so in most real cases, KNN and SVM can do a better job in classification than deep neural networks. There is a lack of consensus or clear-cut-rules on how small a dataset is; from a practical view, a dataset contains samples less than ten thousand normally is considered as a small dataset.

In table 8.4, I have listed some results from the 2016 SEG ML contest of different algorithms (Hall 2016; Hall and Hall 2017). Boosted trees is an advanced version of decision trees. I have mentioned before that due to the different split of training and testing dataset, it might be unfair to directly compare my results to SEG contest results. However, it is notable that results from SEG contest and my dissertation are similar in some aspects. For example,

boosted trees in SEG contest and decision trees in my dissertation both achieved the highest F-1 score, meanwhile more advanced algorithms like artificial neural network, convolutional neural network haven't performed equally well in both cases. I have analyzed that the main reason of this phenomena is due to the limited size of my dataset. Although none of the F-1 score results in my dissertation and from external studies is over 0.7, the classification results of my work and external studies can be a valuable source for future studies on the similar projects.

Table 8. 4 2016 SEG ML contest results. (https://github.com/seg/2016-ml-contest) Four columns from left to right successively are the name of team, their best F1 score, the algorithm and computer language they used. The first two results are the best results among all participated algorithms. The following results are the best results picked from the same algorithms.

Team	F1 score	Algorithm	Language
LA Team (Mosser, de la Fuente)	0.641	Boosted trees	Python
Ispl (Bestagini, Tuparo, Lipari)	0.640	Boosted trees	Python
CarlosFuerte	0.570	Aritificial neural network	Python
ShiangYong	0.570	Convolutional neural network	Python
StoDIG	0.561	Convolutional neural network	Python
Daghra	0.506	K-nearest neighbors	Python
BrendonHall	0.427	Support vector machine	Python

In the oil and gas industry, machine learning is a relatively new and hot topic. I believe if I can combine our domain knowledge with machine learning techniques, I can enhance, facilitate and automate a lot of processes in the oil and gas industry. In this dissertation, I aim to explore the potential application in rock facies classification from well log data, which would

automate the characterizing process and significantly reduce the manual effort. Since I focus on the test of the feasibility of our algorithms, I did not spend too much time in model parameter tuning which could potentially increase the final F-1 score. However, our results indicate that using machine learning for rock classification is definitely feasible and it has a very promising application.

Our future work could focus on the following areas which could improve the final result. The first one is to apply a deeper CNN model on the same dataset and investigate the relationship between performance and the number of layers of the neural network. The second topic is the feature selection and analysis. It is important to know which features have dominated the impact of the final result. Also, results in this thesis can be potentially improved by cleaning well log signatures for gas-related effects. Another topic is to apply our method across non-continuous sediment depositional settings, such as sequence stratigraphic boundaries and erosional unconformities. I have published all codes on GitHub for future review (https://github.com/weixiongdi/PyFacies).

9. Appendices

Appendix A

SMO (Sequential minimal optimization) algorithm

In the past, I used so many pages, step by step, and made the following problems:

$$\max_{a}\left[\sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i,j=1}^{n} a_i a_j y_i y_j K\left(x_i, x_j\right)\right],$$

$$s.t. C \geq a_i \geq 0, i = 1,2,\ldots,n,$$

$$\sum_{i=1}^{n} a_i y_i = 0\,. \qquad (A-1)$$

In order to facilitate the following description, I will give a code to this question, called "the ultimate problem" and "the ultimate constraint". Now I use the SMO sequence minimum optimization algorithm to solve this "ultimate problem."

Here our solution, in simple terms, is to fix all parameters except a_1 and then find the extreme value on a_1. Is this okay? No, because I have one more question.

$$\sum_{i=1}^{n} a_i y_i = 0\,. \qquad (A-2)$$

In other words, when I fix all parameters except a_1, the value of a_1 is fixed:

$$a_1 y_1 = -\sum_{i=2}^{n} a_i y_i\,. \qquad (A-3)$$

So fixing a parameter is not enough, I have to select two parameters at a time for optimization. Then I select a_1, a_2, and the other variables a_i (i=3, 4,...) are fixed. Let's start solving now and the idea is as follows: simplify the "ultimate constraint". Since I am fixing all the parameters except a_1, a_2, there are:

$$a_1 y_1 + a_2 y_2 = D, \tag{A-4}$$

here D, I use a constant, which is fixed by us. I can use this to represent a_1:

$$a_1 = (D - a_2 y_2)/y_1\,, \tag{A-5}$$

in fact, the value of y is either 1 or -1, so the above formula is equivalent to:

$$a_1 = (D - a_2 y_2) y_1, \tag{A-6}$$

this is the first message I got from simplification. Don't forget that I still have,

$$C \geq a_1 \geq 0, C \geq a_2 \geq 0. \tag{A-7}$$

The above are the two information I get directly. Combine these two information, I can further narrow the range of parameters a_1, a_2: 1. When y_1 and y_2 are different, there is

$$a_1 - a_2 = D \text{ or } a_2 - a_1 = D, C \geq a_i \geq 0, i = 1,2. \tag{A-8}$$

At this time, how do the two parameters a_1 and a_2 take values? I use the following diagram to see visually:

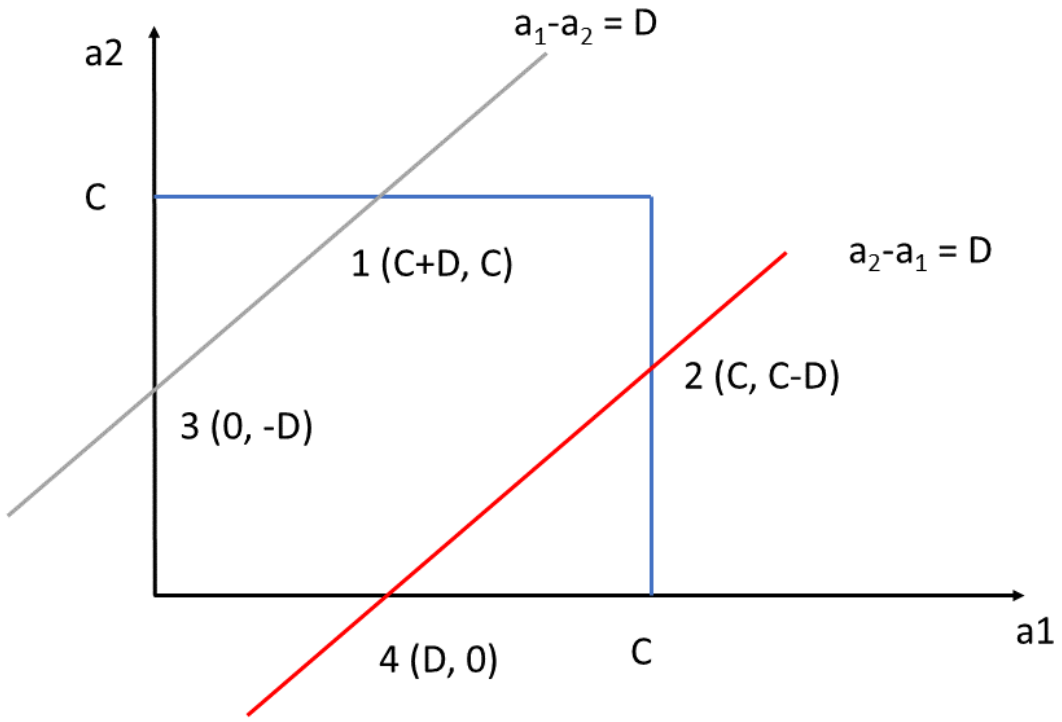

Figure A. 1 SMO illustration.

At this time, the range of a_i (i = 1, 2) must be a purple line or a red line within the square. To sum up, when y_1 and y_2 are different, there is

$$L = \max(0, a_2 - a_1) \le a_2 \le H = \min(C, C + a_2 - a_1)\,. \qquad (A-9)$$

2. When y_1 and y_2 are the same numbers, there is

$$a_1 + a_2 = D \text{ or } a_2 + a_1 = -D, C \ge a_i \ge 0, i = 1,2, \qquad (A-10)$$

in the same way as (1), the range of values of a_2 can be derived.

$$L = \max(0, a_2 + a_1 - C) \le a_2 \le H = \min(C, a_1 + a_2)\,. \qquad (A-11)$$

This is also the range of values for a_1, well, this is the three "extreme constraints" that I get after simplifying the "ultimate constraint." The ultimate problem is this:

$$\max_{a}\left[\sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i,j=1}^{n} a_i a_j y_i y_j K(x_i, x_j)\right], \qquad (A-12)$$

Now let's simplify it. Let's take a_1, a_2 out and make an equivalent transformation to "the ultimate evolution":

$$\max_{a}\left[\sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i,j=1}^{n} a_i a_j y_i y_j K(x_i, x_j)\right]$$

$$= \min\left[\frac{1}{2}\sum_{i,j=1}^{n} a_i a_j y_i\, y_j K(x_i, x_j) - \sum_{i=1}^{n} a_i\right]$$

$$= \min\left[\frac{1}{2}K_{11}a_1^2 + \frac{1}{2}K_{22}a_2^2 + y_1 y_2 K_{12} a_1 a_2 - (a_1 + a_2) + y_1 a_1 \sum_{i=3}^{N} y_i a_i K_{i1} + y_2 a_2 \sum_{i=3}^{N} y_i a_i K_{i2}\right].$$

$$(A-13)$$

This formula is not recommended for derivation, just know. I then continue to simplify, quotes:

$$v_j = \sum_{i=3}^{N} y_i\, a_i K_{ij}, \qquad (A-14)$$

substituting into the above formula, the ultimate problem is reduced to

$$\min\left[\frac{1}{2}K_{11}a_1^2 + \frac{1}{2}K_{22}a_2^2 + y_1 y_2 K_{12} a_1 a_2 - (a_1 + a_2) + y_1 a_1 v_1 + y_2 a_2 v_2\right]$$

$$= J(a_1, a_2). \qquad (A-15)$$

The ultimate constraint" is substituted into the "extreme problem" – solving the "extreme problem". I will first "extremely constrain"

$$a_1 = (D - a_2 y_2) y_1, \qquad (A-16)$$

substituting into the "extreme problem", there are the ultimate problem

$$\mathrm{J}\,(a_2) = \frac{1}{2}K_{11}(D - a_2y_2)^2 + \frac{1}{2}K_{22}a_2^2 + y_2K_{12}(D - a_2y_2)a_2 - (D - a_2y_2)y_1 - a_2$$

$$+(D - a_2y_2)v_1 + y_2a_2v_2, \qquad (A-17)$$

deriving a_2 to make it 0,

$$\frac{\partial J}{\partial a_2} = K_{11}a_2 + K_{22}a_2 - 2K_{12}a_2 - K_{11}Dy_2 + K_{12}Dy_2 + y_1y_2 - 1 - v_1y_2 + y_2v_2 = 0$$

$$\Longrightarrow (K_{11} + K_{22} - 2K_{12})a_2 = y_2(y_2 - y_1 + DK_{11} - DK_{12} + v_1 - v_2), \qquad (A-18)$$

in addition,

$$v_j = \sum_{i=3}^{N} y_i a_i K_{ij} = f(x_j) - \sum_{j=1}^{2} a_j y_j K(x_i, x_j) \; - b$$

$$f(x_j) = \sum_{j=1}^{2} a_j y_j K(x_i, x_j) \; + b. \qquad (A-19)$$

Needless to say, iterative evaluation, given an initial value, and then iteratively updated. Given the initial values of a_2 and a_1 a_{old2}, a_{old1}, there is

$$D = a_{old2} + a_{old1}, \qquad (A-20)$$

substitute for the final solution and get

$$(K_{11} + K_{22} - 2K_{12})a_2^{new,unc} = y_2[(K_{11} + K_{22} - 2K_{12})a_2^{old}y_2 + y_2 - y_1 + f(x_1) - f(x_2)]. \qquad (A-21)$$

What is the unc above a_2? Don't forget that a_2 still needs to satisfy L<= a_2 <= H. I don't consider this range, for the time being, so I use unc to represent this range, and then remove this small tail unc. Make

$$E_j = f(x_i) - y_i, \qquad (A-22)$$

the original is equivalent to

$$(K_{11} + K_{22} - 2K_{12})a_2^{new,unc} = (K_{11} + K_{22} - 2K_{12})a_2^{old} + y_2(E_1 - E_2), \qquad (A-23)$$

iteratively gets:

$$a_2^{new,unc} = a_2^{old} + \frac{y_2(E_1 - E_2)}{(K_{11} + K_{22} - 2K_{12})}. \qquad (A-24)$$

Now remove the little tail unc,

$$a_2^{new} = \begin{cases} H & if\ a_2^{new,unc} > H \\ a_2^{new,unc} & if\ L \le a_2^{new,unc} \le H \\ L & if\ a_2^{new,unc} < L \end{cases}. \qquad (A-25)$$

Appendix B

Karush–Kuhn–Tucker conditions

The solution of *p* =d** is the optimal solution. Introducing KKT conditions is beyond our scope, I only need to know that KKT conditions are met here. Below I will completely abandon *p**, only study *d**

$$\max_{a_i>0} \min_{w,b} F(w,b,a) = d^*, \qquad (B-1)$$

first I fix a, let F minimize w and b, old methods, and w and b:

$$\frac{\partial F}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{n} a_i y_i x_i, \qquad (B-2)$$

$$\frac{\partial F}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} a_i y_i = 0. \qquad (B-3)$$

Then bring it into *F(w,b, a)*,

$$F(w,b,a) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} a_i \left[y_i(W^T x_i + b) - 1\right]$$

$$= \frac{1}{2} W^T W - W^T \sum_{i=1}^{n} a_i y_i x_i - b \sum_{i=1}^{n} a_i y_i + \sum_{i=1}^{n} a_i$$

$$= \frac{1}{2} W^T \sum_{i=1}^{n} a_i y_i x_i - W^T \sum_{i=1}^{n} a_i y_i x_i + \sum_{i=1}^{n} a_i$$

$$= -\frac{1}{2}\left(\sum_{i=1}^{n} a_i y_i x_i\right)^T \sum_{i=1}^{n} a_i y_i x_i + \sum_{i=1}^{n} a_i$$

$$= \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i,j=1}^{n} a_i a_j y_i y_j x_i^T x_j.$$

$$(B-4)$$

Then, I can get the value of a through the following equation

$$\max_a \left(\sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i,j=1}^{n} a_i a_j y_i y_j x_i^T x_j \right)$$

$$s.t.\ a_i \geq 0, i = 1,2,\ldots.,n. \qquad \sum_{i=1}^{n} a_i y_i = 0 \qquad (B-5)$$

10. Bibliography

Abdughani, M., Ren, J., Wu, L., Yang, J. M., & Zhao, J. (2019). Supervised Deep Learning in High Energy Phenomenology: a Mini Review. *Communications in Theoretical Physics, 71*(8), 955-990.

Al-Jamimi, H. A., Al-Azani, S., & Saleh, T. A. (2018). Supervised machine learning techniques in the desulfurization of oil products for environmental protection: A review. *Process Safety and Environmental Protection, 120*, 57-71.

Al-Rawi, H. A. A., Ng, M. A., & Yau, K. L. A. (2015). Application of reinforcement learning to routing in distributed wireless networks: a review. *Artificial Intelligence Review, 43*(3), 381-416.

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*(05), 301-315

Baluja, S., Mittal, V. O., & Sukthankar, R. (2000). Applying machine learning for high-performance named-entity extraction. *Computational Intelligence, 16*(4), 586-595.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory, 1992* (pp. 144-152): ACM

Burger, H. C., Schuler, C. J., & Harmeling, S. (2012). Image denoising: Can plain Neural Networks compete with BM3D? *2012 Ieee Conference on Computer Vision and Pattern Recognition (Cvpr)*, 2392-2399.

Busch, J., Fortney, W., & Berry, L. (1987). Determination of lithology from well logs by statistical analysis. *SPE formation evaluation 2*(04), 412-418.

Chaki, S., Routray, A., & Mohanty, W. K. (2018). Well-Log and Seismic Data Integration for Reservoir Characterization A signal processing and machine-learning perspective. *Ieee Signal Processing Magazine, 35*(2), 72-81.

Chen, C., Takahashi, T., Nakagawa, S., Inoue, T., & Kusumi, I. (2015). Reinforcement learning in depression: A review of computational research. *Neuroscience and Biobehavioral Reviews, 55*, 247-267.

Cohen, M. X. (2008). Neurocomputational mechanisms of reinforcement-guided learning in humans: A review. *Cognitive Affective & Behavioral Neuroscience, 8*(2), 113-125.

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning, 20*(3), 273-297.

Cuddy, S. The application of the mathematics of fuzzy logic to petrophysics. In *SPWLA 38th Annual Logging Symposium, 1997*: Society of Petrophysicists and Well-Log Analysts

Delfiner, P., Peyret, O., & Serra, O. J. S. f. e. (1987). Automatic determination of lithology from well logs. *SPE formation evaluation 2*(03), 303-310.

Dong, C., Loy, C. C., & Tang, X. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision, 2016* (pp. 391-407): Springer

Dubois, M., Bohling, G., & Chakrabarti, S. (2004). Comparison of rock facies classification using three statistically based classifiers. Technical Report 2004-64.

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

Fayyad, U. M. (1996). Data mining and knowledge discovery: Making sense out of data. *Ieee Expert-Intelligent Systems & Their Applications, 11*(5), 20-25.

Fernandez-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014). Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research, 15*, 3133-3181.

Giannakis, I., Giannopoulos, A., & Warren, C. (2019). A Machine Learning-Based Fast-Forward Solver for Ground Penetrating Radar With Application to Full-Waveform Inversion. *Ieee Transactions on Geoscience and Remote Sensing, 57*(7), 4417-4426.

Gill, D., Shomrony, A., & Fligelman, H. J. A. B. (1993). Numerical zonation of log suites and logfacies recognition by multivariate clustering. *AAPG Bulletin, 77*(10), 1781-1791.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*: MIT press.

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.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*(5786), 504-507.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks, 2*(5), 359-366, doi:Doi 10.1016/0893-6080(89)90020-8.

Hubner, D., Verhoeven, T., Muller, K. R., Kindermans, P. J., & Tangermann, M. (2018). Unsupervised Learning for Brain-Computer Interfaces Based on Event-Related Potentials: Review and Online Comparison. *Ieee Computational Intelligence Magazine, 13*(2), 66-77.

Jaafra, Y., Laurent, J. L., Deruyver, A., & Naceur, M. S. (2019). Reinforcement learning for neural architecture search: A review. *Image and Vision Computing, 89*, 57-66.

Jahoda, A., Trower, P., Pert, C., & Finn, D. (2001). Contingent reinforcement or defending the self? A review of evolving models of aggression in people with mild learning disabilities. *British Journal of Medical Psychology, 74*, 305-321.

Jia, Y. N., & Ma, J. W. (2017). What can machine learning do for seismic data processing? An interpolation application. *Geophysics, 82*(3), V163-V177.

Kapur, L., Lake, L. W., Sepehrnoori, K., Herrick, D. C., & Kalkomey, C. T. Facies prediction from core and log data using artificial neural network technology. In *SPWLA 39th Annual Logging Symposium, 1998*: Society of Petrophysicists and Well-Log Analysts

Keller, J. M., Gray, M. R., & Givens, J. A. (1985). A Fuzzy K-Nearest Neighbor Algorithm. *Ieee Transactions on Systems Man and Cybernetics, 15*(4), 580-585.

Kim, M., Lee, D. G., & Shin, H. (2019). Semi-supervised learning for hierarchically structured networks. *Pattern Recognition, 95*, 191-200.

Kovachki, N. B., & Stuart, A. M. (2019). Ensemble Kalman inversion: a derivative-free technique for machine learning tasks. *Inverse Problems, 35*(9), 12-21.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the Acm, 60*(6), 84-90.

Langkvist, M., Karlsson, L., & Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters, 42*, 11-24.

Lawrence, S., Giles, C. L., Tsoi, A. C., & Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *Ieee Transactions on Neural Networks, 8*(1), 98-113.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436-444, doi:10.1038/nature14539.

Leuenberger, M., & Kanevski, M. (2015). Extreme Learning Machines for spatial environmental data. *Computers & Geosciences, 85*, 64-73.

Li, Z. F., Meier, M. A., Hauksson, E., Zhan, Z. W., & Andrews, J. (2018). Machine Learning Seismic Wave Discrimination: Application to Earthquake Early Warning. *Geophysical Research Letters, 45*(10), 4773-4779.

Liu, J., Timsina, P., & El-Gayar, O. (2018). A comparative analysis of semi-supervised learning: The case of article selection for medical systematic reviews. *Information Systems Frontiers, 20*(2), 195-207.

Malfante, M., Dalla Mura, M., Metaxian, J. P., Mars, J. I., Macedo, O., & Inza, A. (2018). Machine Learning for Volcano-Seismic Signals Challenges and perspectives. *Ieee Signal Processing Magazine, 35*(2), 20-30.

Mammeri, Z. (2019). Reinforcement Learning Based Routing in Networks: Review and Classification of Approaches. *Ieee Access, 7*, 55916-55950.

Mao, X. K., Yang, H., Huang, S. B., Liu, Y., & Li, R. S. (2019). Extractive summarization using supervised and unsupervised learning. *Expert Systems with Applications, 133*, 173-181.

Mason, K., & Grijalva, S. (2019). A review of reinforcement learning for autonomous building energy management. *Computers & Electrical Engineering, 78*, 300-312.

Mcculloch, W. S., & Pitts, W. (1990). A Logical Calculus of the Ideas Immanent in Nervous Activity (Reprinted from Bulletin of Mathematical Biophysics, Vol 5, Pg 115-133, 1943). *Bulletin of Mathematical Biology, 52*(1-2), 99-115.

Mikołajczyk, A., & Grochowski, M. Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW), 2018* (pp. 117-122): IEEE

Miller, C., Nagy, Z., & Schlueter, A. (2018). A review of unsupervised statistical learning and visual analytics techniques applied to performance analysis of non-residential buildings. *Renewable & Sustainable Energy Reviews, 81*, 1365-1377.

Moore, D. E., & Liou, J. G. (1979). Mineral Chemistry of Some Franciscan Blueschist Facies Meta-Sedimentary Rocks from the Diablo Range, California - Summary. *Geological Society of America Bulletin, 90*(12), 1089-1091.

Mora, P. (1989). Inversion = Migration + Tomography. *Geophysics, 54*(12), 1575-1586.

Mudur, S. P., & Koparkar, P. A. (1984). Interval-Methods for Processing Geometric Objects. *Ieee Computer Graphics and Applications, 4*(2), 7-17.

Nishitsuji, Y., & Exley, R. (2019). Elastic impedance based facies classification using support vector machine and deep learning. *Geophysical Prospecting, 67*(4), 1040-1054.

Ray, A., & Myer, D. (2019). Bayesian geophysical inversion with trans-dimensional Gaussian process machine learning. *Geophysical Journal International, 217*(3), 1706-1726.

Reynen, A., & Audet, P. (2017). Supervised machine learning on a network scale: application to seismic event classification and detection. *Geophysical Journal International, 210*(3), 1394-1409.

Rogers, S. J., Fang, J., Karr, C., & Stanley, D. J. A. b. (1992). Determination of lithology from well logs using a neural network (1). *76*(5), 731-739.

Rout, J. K., Dalmia, A., Choo, K. K. R., Bakshi, S., & Jena, S. K. (2017). Revisiting Semi-Supervised Learning for Online Deceptive Review Detection. *Ieee Access, 5*, 1319-1327.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling, 5*(3), 1.

Safavian, S. R., & Landgrebe, D. (1991). A Survey of Decision Tree Classifier Methodology. *Ieee Transactions on Systems Man and Cybernetics, 21*(3), 660-674.

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010* (pp. 92-101): Springer.

Schwenker, F., & Trentin, E. (2014). Pattern classification and clustering: A review of partially supervised learning approaches. *Pattern Recognition Letters, 37*, 4-14.

Shannon, C. E. (1997). The mathematical theory of communication (Reprinted). *M D Computing, 14*(4), 306-317.

Sun, J., Li, H., & Xu, Z. Deep ADMM-Net for compressive sensing MRI. In *Advances in neural information processing systems, 2016* (pp. 10-18)

Suykens, J. A. K., & Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural Processing Letters, 9*(3), 293-300.

Teng, F. (2015). Application of Kernel Based Machine Learning to the Inversion Problem of Photospheric Magnetic Fields. *Solar Physics, 290*(10), 2693-2708.

Wang, G. (2016). A Perspective on Deep Imaging. *Ieee Access, 4*, 8914-8924, doi:10.1109/Access.2016.2624938.

Wei, Z., Hu, H., Zhou, H.-w., Lau, A. J. P., & Geophysics, A. Characterizing Rock Facies Using Machine Learning Algorithm Based on a Convolutional Neural Network and Data Padding Strategy. 1-13.

Wei, Z., Hu, H., Zhou, H. W., & Lau, A. (2019). Characterizing Rock Facies Using Machine Learning Algorithm Based on a Convolutional Neural Network and Data Padding Strategy. *Pure and Applied Geophysics, 176*(8), 3593-3605.

Werbos, P. J. (1994). The roots of backpropagation: from ordered derivatives to neural networks and political forecasting (Vol. 1): John Wiley & Sons.

Wolf, M., & Pelissier-Combescure, J. FACIOLOG-automatic electrofacies determination. In *SPWLA 23rd Annual Logging Symposium, 1982*: Society of Petrophysicists and Well-Log Analysts

Wrona, T., Pan, I., Gawthorpe, R. L., & Fossen, H. (2018). Seismic facies analysis using machine learning. *Geophysics, 83*(5), O83-O95.

Würfl, T., Ghesu, F. C., Christlein, V., & Maier, A. Deep learning computed tomography. In *International conference on medical image computing and computer-assisted intervention, 2016* (pp. 432-440): Springer

Xie, J., Xu, L., & Chen, E. Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems, 2012* (pp. 341-349)

Yau, K. L. A., Komisarczuk, P., & Teal, P. D. (2012). Reinforcement learning for context awareness and intelligence in wireless networks: Review, new features and open issues. *Journal of Network and Computer Applications, 35*(1), 253-267.

Ye, Q., Zhang, Z. Q., & Law, R. (2009). Sentiment classification of online reviews to travel destinations by supervised machine learning approaches. *Expert Systems with Applications, 36*(3), 6527-6535.

Youn, E., & Jeong, M. K. (2009). Class dependent feature scaling method using naive Bayes classifier for text datamining. *Pattern Recognition Letters, 30*(5), 477-485.

Zeng, X. J., Guo, W. P., Yang, K. C., & Xia, M. (2018). Noise reduction and retrieval by modified lidar inversion method combines joint retrieval method and machine learning. *Applied Physics B-Lasers and Optics, 124*(12).

Zhang, D. X., Han, X. Q., & Deng, C. Y. (2018). Review on the Research and Practice of Deep Learning and Reinforcement Learning in Smart Grids. *Csee Journal of Power and Energy Systems, 4*(3), 362-370.

Zhou, H. W. (2006). Multiscale deformable-layer tomography. *Geophysics, 71*(3), R11-R19.