FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme / specialisation: Petroleum Geosciences Engineering	The spring semester, 2023 Open
Author: Aigul Taimour Alvi	
Supervisor at UiS: Nestor Cardozo Co-supervisor: External supervisor(s): Lothar Schulte (SLB)	
Thesis title: Machine learning based seismic classification for facies prediction	
Credits (ECTS): 30	
Keywords: Machine learning, facies, synthetic models, 2D facies prediction, 3D facies prediction, seismic attributes, seismic inversion	Pages: 112 + appendix: 31 Stavanger, *14.06.2023*

Machine learning based seismic classification for facies prediction

by

Aigul Taimour Alvi

MSc Thesis

Stavanger, June 2023

University of Stavanger

Faculty of Science and Technology

Department of Energy Resources

Acknowledgements

I would like to express my deepest gratitude to my supervisor Professor Nestor Cardozo for his unconditional support, advise and coordination throughout writing the thesis. Special thanks to my external supervisor Doctor Lothar Schulte for his encouragement, patience and guidance from generating ideas to performing calculations and discussing results.

I would also like to thank my family for supporting and believing in me along this journey.

Abstract

This thesis explores the performance of machine learning (ML) methods for predicting facies from seismic attributes for 2D and 3D datasets. It focuses on building, training, and testing four supervised methods: Logistic Regression, Support Vector Machines, K-Nearest Neighbors, and Random Forest; and one deep learning method: Neural Network with two hidden layers. A realistic synthetic facies model with complex depositional systems, and a synthetic seismic cube from the facies model are used for the comparison of facies prediction performed by the ML approach with the ground-truth facies distribution. This comparison makes it possible to validate the ML models' prediction based on wells and seismic. In addition, the research evaluates the role of the number of wells and their locations, the impact of seismic data frequency, and the effect of using various seismic attributes. The most important features for facies prediction are seismic inversion and relative acoustic impedance. Instantaneous frequency and envelope have little effect on the accuracy of the ML prediction. Incorporating information about the lateral geometry of the facies in the reservoir also improves the accuracy of the ML prediction.

Table of Contents

List of figures

List of tables

1. Introduction, Objectives, and Thesis Structure

1.1 Introduction

Facies classification is the process of assigning a specific rock type to a particular rock sample based on the measured features [1]. These features include fossil content, mineralogical composition, sedimentary structures, and texture description [2]. Knowledge about the facies distribution in a reservoir is critical in reservoir characterization, exploration, and reservoir simulation because it can indicate petrophysical characteristics, porosity distribution, and consequently, permeability values [3]. The most reliable and direct source of information about facies in the reservoir is core samples from wells. However, core extraction is expensive and cores not always and not everywhere can be obtained. Moreover, a conventional approach to assigning facies manually from core samples is time-consuming [1]. For this reason, alternative ways for predicting facies from indirect sources that can reduce costs without sacrificing the quality of facies classification are necessary.

Three-dimensional seismic data is one of the most valuable sources of information about subsurface structure. Specific seismic attributes derived from seismic data highlight specific information hidden in the seismic data that can help to identify depositional environments (or facies distribution). For this reason, seismic attributes, which have been developed since the 1990s, are currently widely integrated into many facies analyses. There are many seismic attributes now available for various purposes, from prospect identification to detection and characterization of faults [4]. In this thesis, seismic attributes such as relative acoustic impedance, instantaneous frequency, and envelope, together with seismic inversion, which can potentially highlight facies distributions in the reservoir, are utilized for facies classification. However, the large size of the data requires the implementation of automatic methods that can handle facies classification and are faster than the current manual interpretation methods.

The development of artificial intelligence (AI) and its application to a variety of problems over the last decades demonstrate the effectiveness of this method in tasks that include large amount of data. Machine Learning (ML) and Deep Learning (DL), as parts of AI, have aroused the interest of many geologists because of their ability to solve geological problems with large data in a

relatively short amount of time. This thesis investigates the ability of the ML and DL methods for facies prediction from seismic attributes.

A complete validation of the machine learning approach for facies prediction without a ground truth facies distribution in the reservoir is impossible. For this reason, a realistic synthetic facies cube, from which we precisely know the facies distribution, is utilized to evaluate the effectiveness of the ML approach. In addition, the use of the synthetic seismic cube can help to evaluate the role of seismic data derived from different frequencies, various wells' locations and their number, and the impact of noise. The synthetic model involves realistic depositional environments, normal faults, and folds. Most of the work, including the ML, is done by using Python, and seismic attributes extraction from seismic data is performed in Petrel.

1.2 Objectives

The main purpose of this study is to evaluate the performance of ML for facies prediction from seismic attributes.

The specific goals to be covered in this thesis are:

- Building, training, and testing four supervised and one unsupervised ML methods on seismic attributes for facies classification.
- Evaluating the impact of the number of wells and their location on facies prediction.
- Evaluating and comparing the performance of the ML models for facies prediction when using seismic attributes derived from different frequencies.
- Estimating the role of seismic data contaminated with spectral noise.
- Evaluating the role of additional features, such as the facies lateral geometry, for ML facies prediction.

1.3 Thesis structure

This thesis contains 9 chapters. Chapter 1 describes the dataset. Chapter 2 explains the theoretical background of the above-mentioned ML methods: Logistic Regression (LR), K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Random Forest (RF), and Neural Networks (NN). Chapter 3 includes information about the software tools and Python libraries used

for the analysis. Chapters 4 and 5 describe the workflow which is divided in two main parts. Chapter 6 presents the results of the facies prediction. Finally, the discussion, future work, and conclusions are presented in Chapters 7, 8 and 9 respectively.

2. Dataset description

This chapter describes the dataset used in this thesis, which comprises a realistic facies model and, derived from it, seismic cubes and seismic attributes. All these data were provided by SLB. A detailed description of the dataset, including depositional environment, geological structure, and a short description of the facies and seismic cubes generation, are presented in the following sections.

2.1 3D facies cube and its sections

The synthetic facies cube provided for this study covers an area of 39.5 km2. The vertical range of the cube is presented in two-way travel time (TWT) and ranges from -2000 ms to -2700 ms (Figure 1). To simplify the study, we assume that the time and depth domain are equivalent. The Z axis is either two-way travel time or depth in meters. Therefore, a constant VP velocity of 2000m/s is assumed.

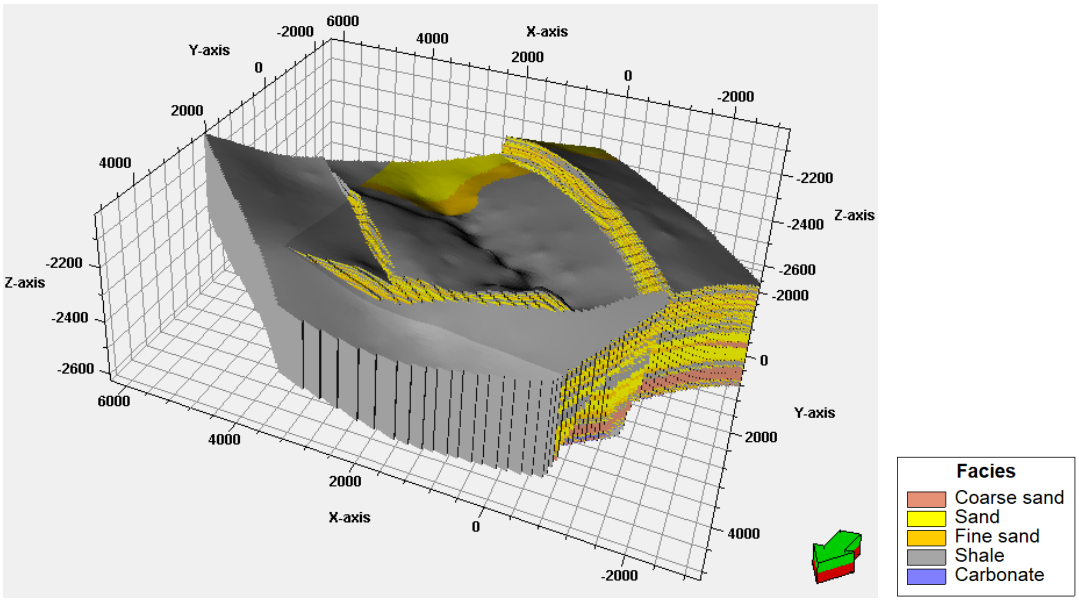

Figure 1. 3D facies cube. Axes are in meters

The facies cube includes several types of depositional environments, such as meandering-deltaic and shoreline systems. The upper part of the reservoir consists of meandering-deltaic

deposits with a series of interbedded fine and medium-grained sandstones (FS, S), shales (Sh), and thin layers of coarse sandstones (CS). The sediments of the meandering channels and delta are unconformable and erosive (Figure 2).

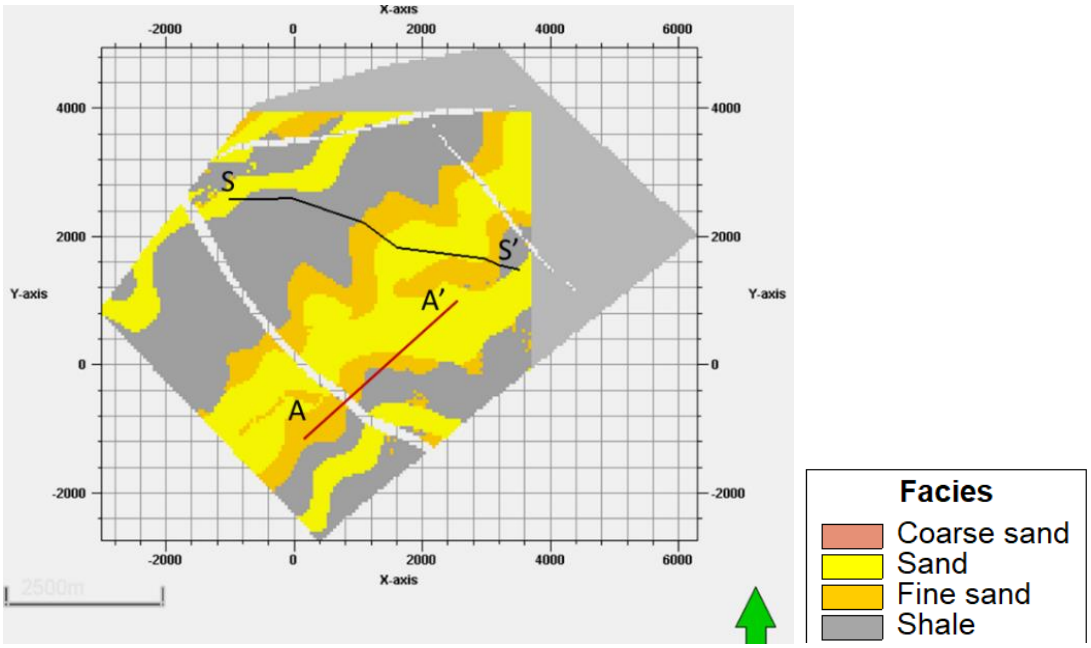

Figure 2. Depth slice through the facies model showing channel deposits. Axes are in meters. The black line S-S' is a navigation of the section in Figure 4. The black line A-A' is a navigation of the section in Figure 5.

The lower part of the reservoir is a shoreline depositional environment that is represented by thick and conformable layers of coarse, medium, and fine-grained sandstone and shale (Figure 3). The base of the lower part of the reservoir consists of interbedded thin layers of carbonates, shales, and sandstones.

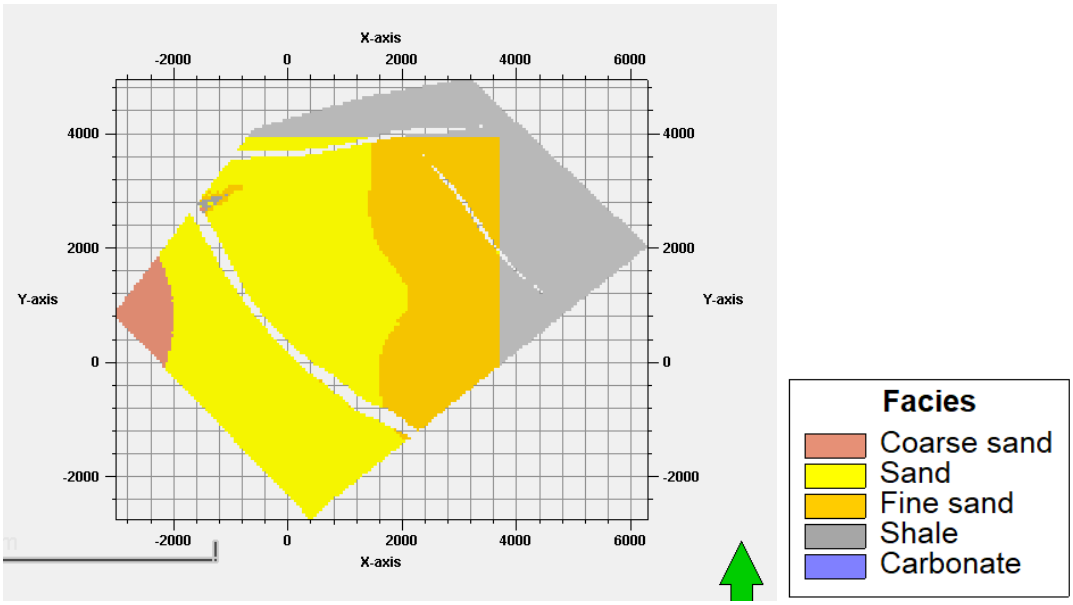

Figure 3. Depth slice through the facies model showing shoreline deposits. Axes are in meters.

Figure 4 shows an E-W cross-section S-S' through the facies cube, including the above-mentioned deposits and the separation of the lower and upper zones of the reservoir.

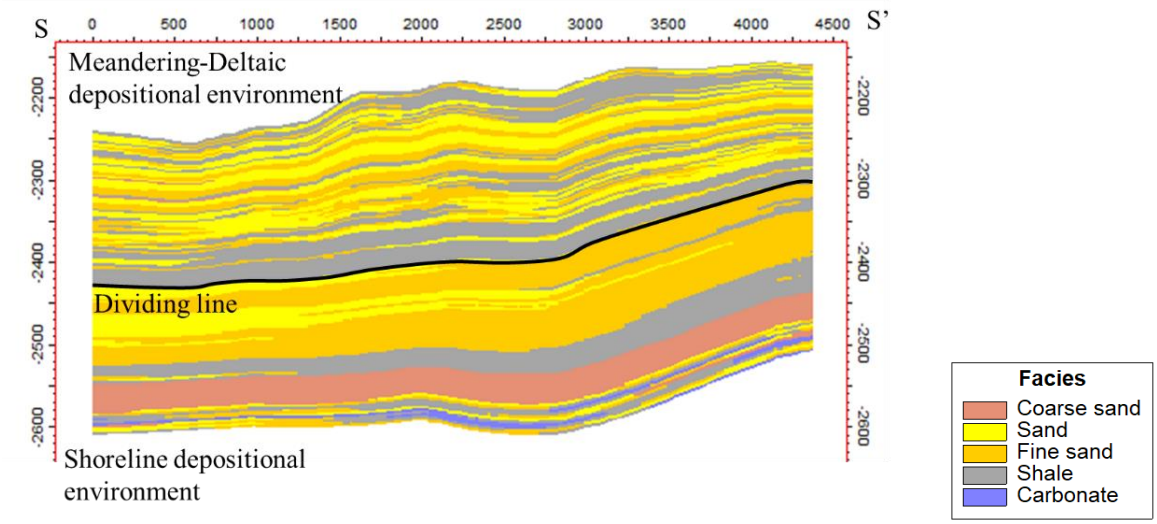

Figure 4. Cross-section S-S' through the facies model. A black line divides the reservoir into two zones: the upper zone and the lower zone. The navigation of the section is shown in Fig. 2. Axes are in meters.

The area underwent extension which resulted in the formation of three normal faults. Two of these faults have a NW direction, and one fault is oblique to them with a NE direction. The faults divide the area into three segments. The faults are post-depositional, meaning that the extension took place after the deposition of the sediments. The vertical section A-A' shown in Figure 5 is offset entirely by the largest NW normal fault which caused upward displacement of the footwall on the SW side and downward displacement of the hanging wall on the NE side.

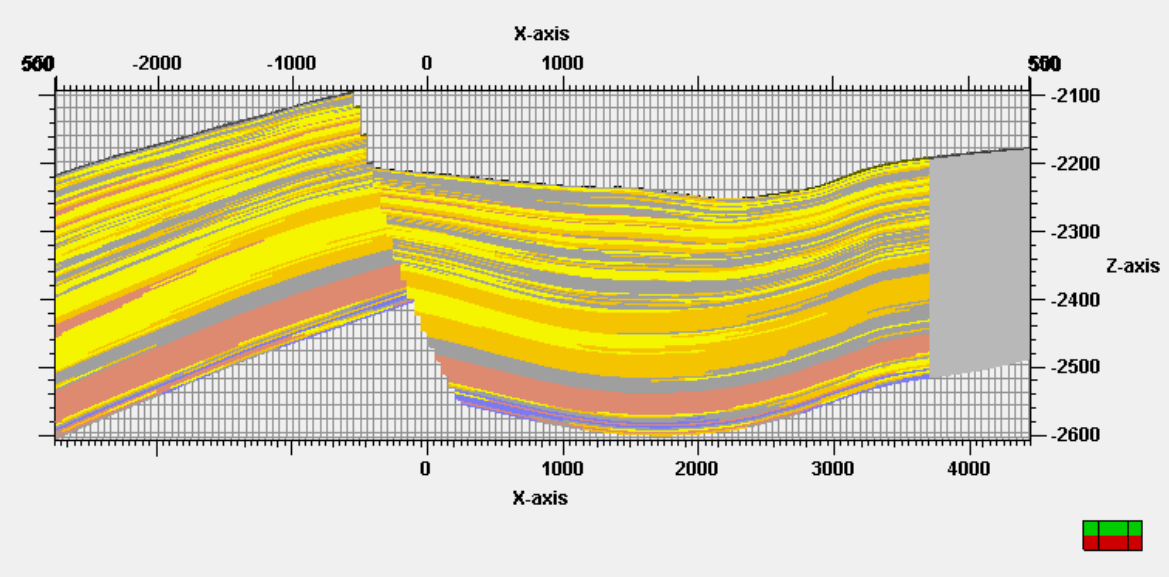

Figure 5. Cross-section A-A' through the facies cube and across a NW-SE normal fault. Axes are in meters. The navigation of the section is shown in Fig. 2 as the red line A-A'.

2.2 The description of the dataset generation and seismic attributes

As mentioned earlier, the three-dimensional facies model, the seismic cube, and the seismic attributes were provided by SLB for this study. A short description of the facies and seismic cubes generation is given below.

First, the facies model is stored in a 3D grid which was populated stochastically with sonic, density, and porosity properties whose values were generated using geostatistical tools. The range of values for each property and for each facies were taken from the literature. From the simulated sonic and density properties, the acoustic impedance was calculated. The synthetic facies model, as well as the impedance cube, are regarded as ground-truth facies and acoustic impedance. Thus,

we can compare and verify the predicted values from the ML methods with the ground-truth values of the synthetic model.

When it comes to the seismic cube, the process started with the acoustic impedance property, which was converted into a 3D SEGY cube. Then, the 3D seismic cube was derived from the acoustic impedance 3D cube using a Ricker wavelet with a frequency of 25 Hz. Moreover, in this research, I explored the role of various seismic frequency ranges on ML facies prediction performance. To do this, Ormsby wavelets with a set of different frequency ranges 10-60 Hz, 10-80 Hz, and 10-100 Hz, were used for the modeling of additional seismic cubes (Figure 6).

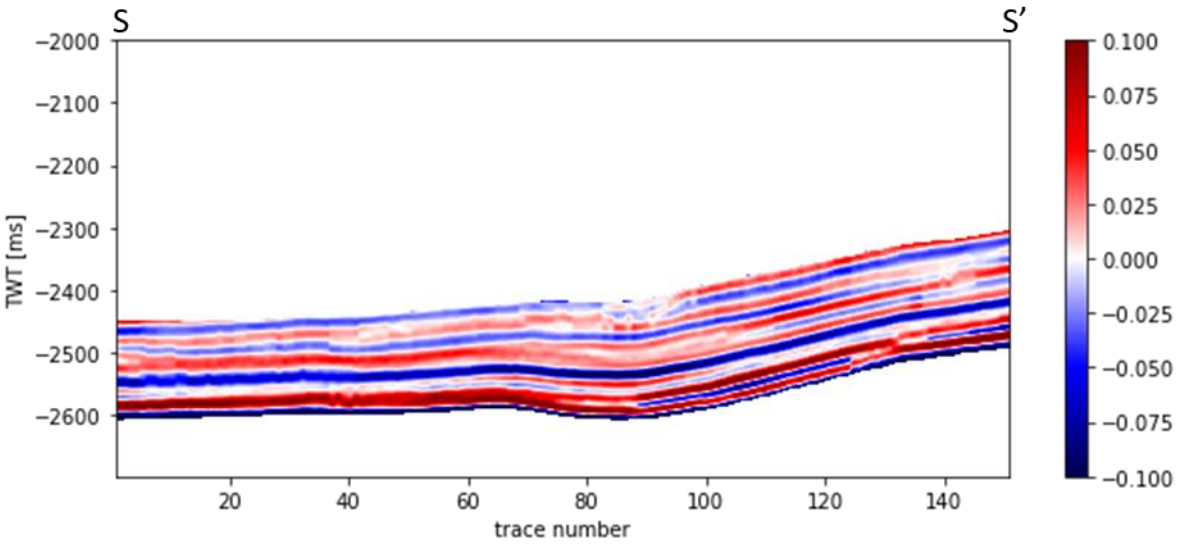

Figure 6. Time-section through the seismic cube. This is the lower part of the reservoir along section S-S´ in Figure 2.

However, seismic data give primary information about the subsurface structure rather than revealing facies distribution in reservoirs. Based on amplitudes, seismic datasets are used for mapping subsurface stratigraphic and structural features. In contrast, seismic attributes obtained from seismic data can help identify characteristics of prospects such as depositional environments, sequence boundaries and unconformities.

There are multiple seismic attributes available, and they can be divided into two groups: geometrical and physical attributes [4, 5]. The geometrical attributes, for example, variance and edge evidence, can be utilized for the interpretation of the seismic data and the mapping of features such as sequence boundaries, discontinuities, and faults. Physical seismic attributes, such as

spectral decomposition, root mean square amplitude (RMS), and instantaneous phase, are aimed at highlighting the hydrocarbon present in the reservoir and identifying coarse-grained facies [4, 6].

Facies delineation can be obtained from physical attributes. One of the physical properties is the relative acoustic impedance (Rel AI) which is calculated by integration of the seismic trace. This seismic attribute is typically used for the indication of sequence boundaries, lithology, and hydrocarbons. Extraction of the relative acoustic impedance from seismic is a computationally inexpensive and straightforward process, unlike the building of seismic inversion. Low values of relative acoustic impedance can be associated with sandy intervals, while high values are related to shales and sequence boundaries [5]. A time section through the relative acoustic impedance cube is shown in Figure 7.

Other seismic attributes used for facies classification are envelope and instantaneous frequency. Sweetness is aimed at highlighting coarse-grained sand intervals. This attribute is defined as the ratio of the trace envelope and the square root of the average frequency. According to [7], the envelope is a powerful attribute for detecting channel deposits. Instantaneous frequency is defined as the time rate of the instantaneous phase change and is usually used for detecting thin facies

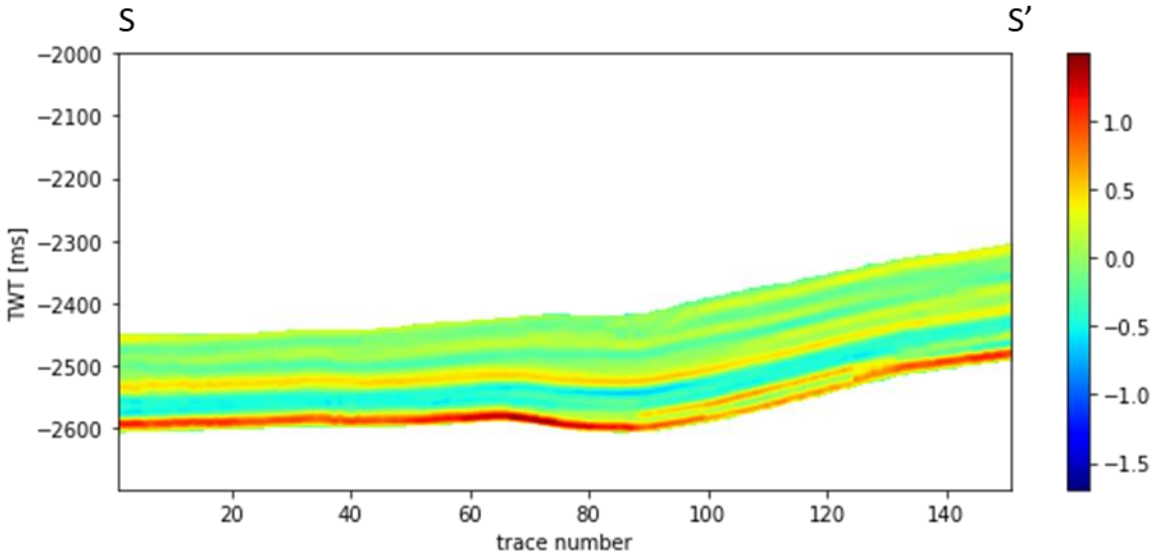

Figure 7. Time section through the relative acoustic impedance cube. This is the lower part of the reservoir along section S-S´ in Figure 2.

Seismic inversion is aimed at extracting the acoustic impedance from seismic data. However, this process is not straightforward and non-linear, thus making the inversion computationally expensive. If computed properly, seismic inversion is a robust technique for facies identification. A time section through the seismic inversion is illustrated in Figure 8.

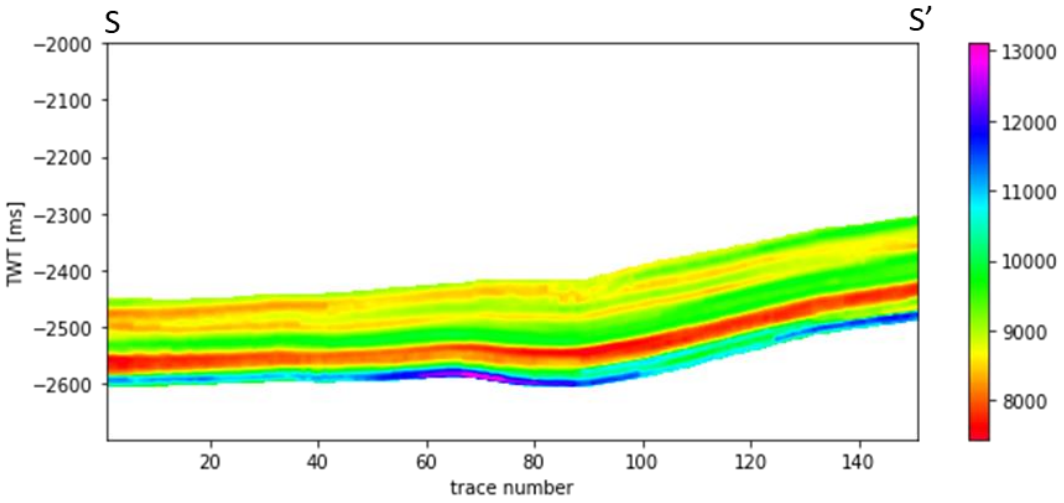

Figure 8. Time section through the seismic inversion cube. This is the lower part of the reservoir along section S-S´ in Figure 2.

Finally, seismic data contaminated with noise can have a negative impact on facies prediction. To evaluate the role of noise, the seismic data with spectral noise was provided for the analysis (Figure 9).

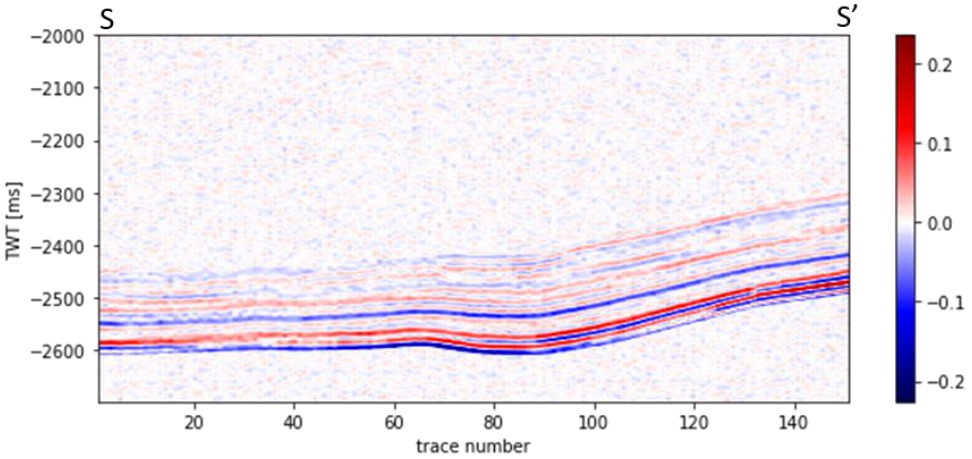

Figure 9. Time section through the seismic cube with spectral noise. This is the lower part of the reservoir along section S-S´ in Figure 2.

The description of how these data were used for training ML models, their verification, and facies prediction is presented in Chapter 4.

3 Machine Learning Theoretical Background

3.1 Supervised ML methods (binary, multi-class, regression)

Supervised machine learning is the learning paradigm (algorithm) that processes the training dataset consisting of the observed data (or input data) and the dependent variable (or output data) for every record [8]. The learning procedure uses these data and builds a model that identifies the underlying relationship between the observed data and the dependent variable. Trying to minimize the difference between input and output in the training dataset, the model optimizes its parameters. Finally, the model with the upgraded parameters predicts the output data with some degree of uncertainty for any newly observed data [9].

The dependent variable can be represented as a continuous numerical or categorical value. A supervised machine learning algorithm uses regression or classification techniques, respectively. Categorical means that there is a certain number of outputs (or discrete variables). Classification means that the variables are classified into one of two (binary classification) or more (multi-class classification) classes. An example of binary classification can be spam detection when the model can tell us whether an email is a spam or not. A good example of a multi-class task is identifying the facies distribution based on well logs. Classifiers create a boundary that divides the region into areas equal to the number of classes. The boundary line represents the equal probability between classes. For binary case classification, there is only one boundary line (Figure 10a). An example of multi-class classification is shown in Figure 10b, where each boundary classifier distinguishes a single class from the remaining data.

In contrast, a regression technique predicts the output having a continuous nature (Figure 10c). This method tries to approximate the function f for an input data x that generates the output value y minimizing the error between the model and the data. A good example is predicting plane ticket prices based on the season and destination.

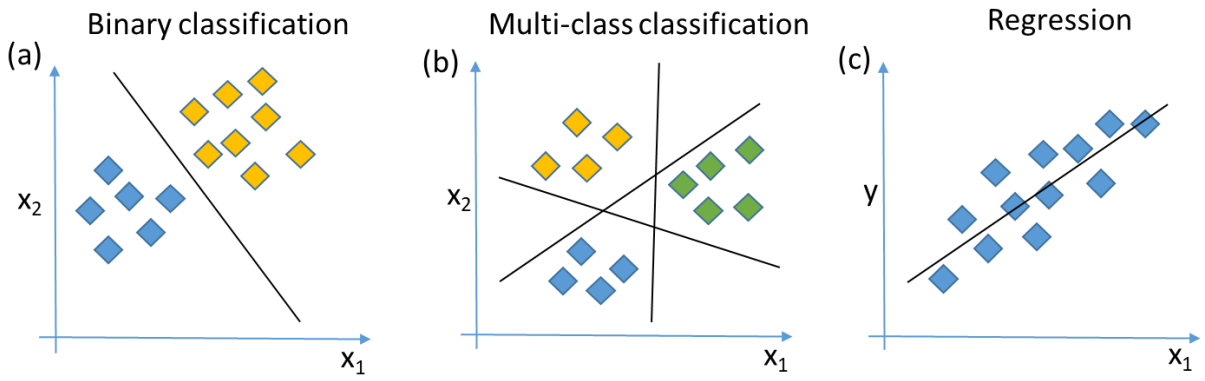

Figure 10. Supervised ML types of problems: (a) Binary classification model where each combination of x_1 and x_2 gives the yellow or blue target value. (b) Multi-class classification model where every combination of x_1 and x_2 gives yellow, blue, or green classes. (c) Regression model where the regression line predicts the target value y from input values x_1 (modified from [10]).

Facies prediction from seismic attributes is a multi-class classification problem because we are dealing with five facies classes: coarse sand (CS), sand (S), fine sand (FS), shale (Sh), and carbonate (C).

There are multiple methods used for multi-classification problems: Classification trees, Random Forests, K nearest-neighbor, Logistic Regression, Support vector machine, etc [11]. The decision of which ML algorithm to choose depends on several factors. First, as discussed earlier, if the classification is based on known classes, the analysis should utilize supervised ML algorithms. Secondly, the size of the training dataset, its quality, and whether the data is structured or unstructured provides insight into which methods can give more reliable results. Poor-quality, inadequate and unprocessed data will lead to poor training of supervised methods. Thirdly, the number of features used for training can directly affect the outcome. Once the number of features is finalized and approved, the choice of machine learning techniques should start from the simplest models, such as Linear Regression or Logistic Regression, which take less time for training but also are less flexible. Complex models, in contrast, will take more time to learn, however, they can be a good investment for the accuracy of the output. The final choice of ML algorithms depends on finding a balance between simplicity and flexibility, in other words, between bias and variance. The definition and importance of these two parameters is discussed in section 3.3.2.

3.1.1 Logistic Regression

The name 'regression' in a Logistic Regression model can be misleading because, under the term regression, it is assumed that the dependent variable is continuous in nature. Logistic Regression is a statistical classification model that is used to predict categorical or binary variables. The term Logistic refers to 'log odds', the modeled probability ratio. The model is similar to a linear regression model; however, it estimates the probability of the occurring event [12]. Thus, the dependent predicted variable is bounded between 0 and 1. To accomplish this, Logistic Regression uses a sigmoidal function. Mathematically, the sigmoidal function can be expressed as [12]

$$f(x) = \frac{1}{1+ e^{-x}} \tag{1}$$

where e (or epsilon) is a base of the natural logarithms or 'Euler's number'.
This formula can be rewritten to express the probability of the outcome Y given the knowledge of the dependent variable X using the logistic function [11].

$$P(Y|X) = \frac{e^{\beta_0+ \beta_1 x}}{1+ e^{\beta_0+ \beta_1 x}} \tag{2}$$

where β_0 and β_1 are coefficients that are used to fit a regression line in a ln-ln space.
Rewriting and transforming formula (2) as the inverse of the logistic function, which is called *logit*, allows determining the coefficients β_0 and β_1 [11].

$$logit\big(P(Y|X)\big) = \ln\left(\frac{P(Y|X)}{1-P(Y|X)}\right) = \beta_0 + \beta_1 X \tag{3}$$

The logistic curve is non-linear (Figure 11), and the used logit transform (Equation 3) gives a linear regression, where the probability of success Y for a given X (P(Y|X)) can be calculated [11]. As input, the logistic function takes the values (β_0 + β_1X) and gives the output as the probability of Y given X (P(Y|X)). An example of the logistic function between the X interval -4 to 4 is shown in Figure 11.

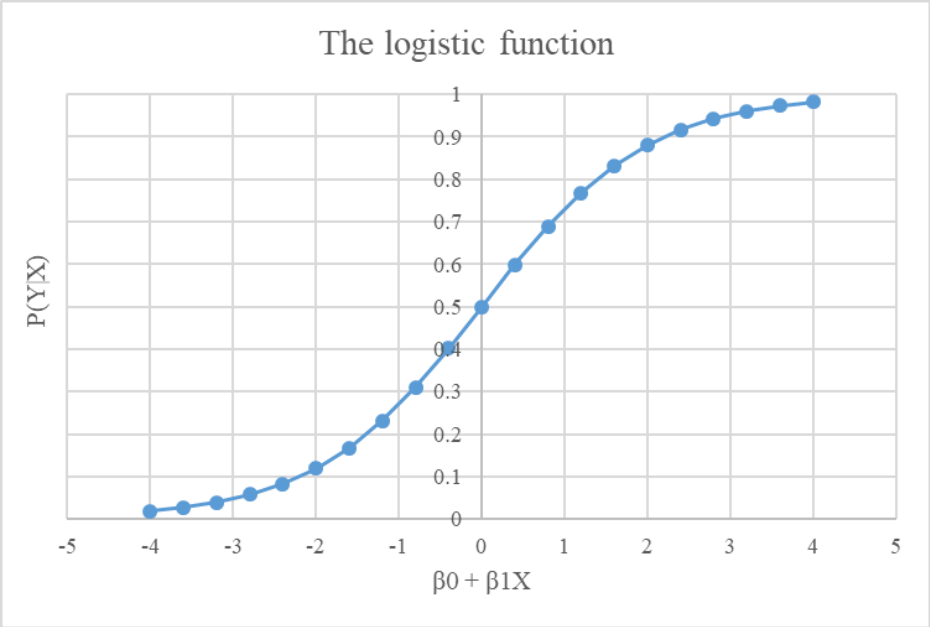

Figure 11. Logistic function for data between -4 and 4.

3.1.2 Support Vector Machines

Support Vector Machines (SVM) have become well-known over the past thirty years after their introduction by Cortez and Vapnik. Originally, SVM was not introduced for multiclass classification. However, further development made it possible to implement this method for more than two groups of outcomes [13]. The purpose of the SVM algorithm is to identify the optimal line, hyperplane, or plane for 1D, 2D, or 3D space, respectively. This boundary splits a dataset into two classes. For an easier understanding of the principles of the SVM algorithm, we will consider a binary classification as shown in Figure 12.

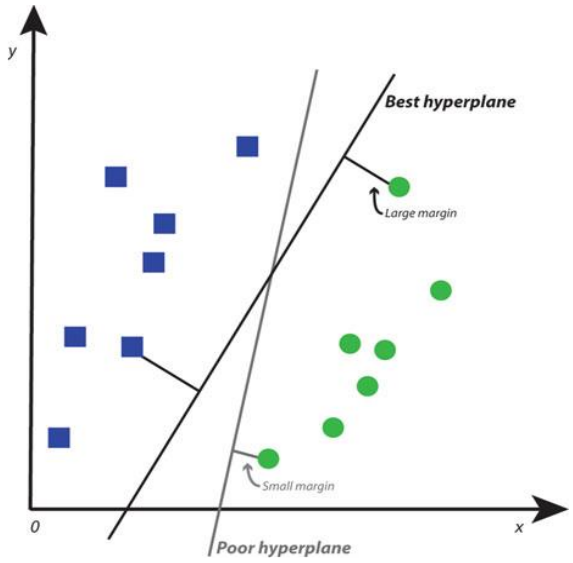

Figure 12. In two-dimensions, the hyperplane is a line with the greatest distance to the nearest element of each class [9].

SVM relies on data points from two data classes that are located closer to each other. Drawing a line through the closest data point from each side gives two support vectors. Passing a classification line that maximizes the distance between the two support vectors and separates points on each side reduces the upper limit of the error [14]. This is called a hard maximal margin classification, as shown in Figure 13(a). However, there might be a case when it is not possible to divide data points completely by a line as shown in Figure 13(b). In this case, SVM introduces slack variables that concede inaccuracy in the classification. This is called a soft margin classification. Unlike a hard margin classifier, the soft margin approach misclassifies some data points close to the boundary while separating most of the data points correctly

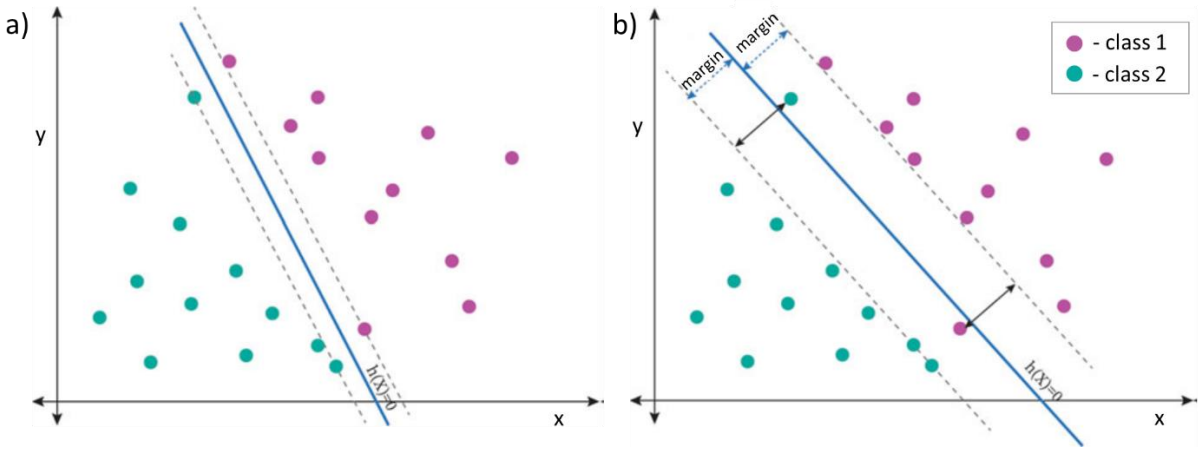

Figure 13. The illustration of a hard (a) and soft (b) margin classifier for a linear SVM [9].

In addition, the SVM algorithm operates with a kernel method in a situation when data points cannot be separated linearly by a hyperplane and when the soft margin classifier misclassifies the data. The kernel is an additional use for the SVM that makes it possible to model nonlinear and high-dimensional models by adding extra dimensions to nonlinear data, therefore, converting them to linear data. The SVM kernel is applied to map a low-dimensional dataset into a higher-dimensional space. By introducing additional dimensions, the SVM model achieves better scalability and accuracy by classifying nonlinear datasets with sophisticated boundaries [9]. Different types of kernels can be utilized, for example, linear, polynomial, and radial. The selection of a particular kernel depends on the dataset. This variety in the kernel makes it possible to implement the SVM model in different fields, such as chemistry, geology, weather forecasting, etc.

3.1.2 K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a non-parametric supervised model used for classification, regression, and clustering. The algorithm classifies data points based on their closest neighboring instances. New data points are assigned to one of the existing labeled samples that are dominant in the locality. The accuracy of the classification can be affected by the number of labeled instances. For example, when the new data point is surrounded by equidistant samples that belong to two classes, a new instance can be misclassified. In contrast, the classification accuracy

increases if the unknown data point is located close to samples with one class. For this reason, the KNN algorithm uses several labeled samples that must be considered during the classification, so-called, k-nearest neighbors. The one-nearest neighbor is applied when only one labeled data point is used for classifying a new sample, while the four-nearest neighbors take into consideration the four closest labeled samples (Figure 14).

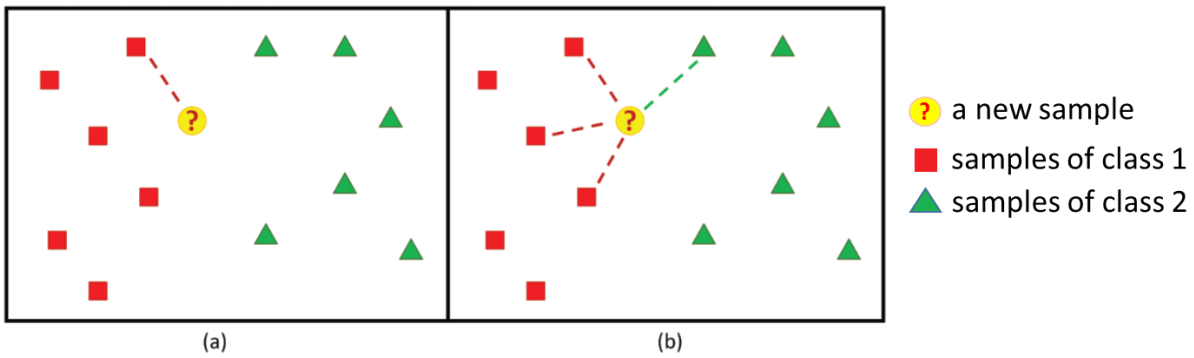

Figure 14. Principle of the k-NN decision rule. The 1-NN rule (a): a new sample is labeled by using only one labeled instance. The 4-NN rule (b): a new sample is labeled by using four labeled instances [15].

The distance between new data points and existing labeled samples of a particular class is extremely important. KNN operates with an Euclidean distance to find the closest instances. The smaller the distance, the higher the likelihood that unknown samples would be classified into the same class as its closest known labeled data [15].

The KNN method is computationally expensive due to the calculation of the Euclidean distance between the new and labeled instances. The larger the dataset, the more distances must be calculated.

3.1.3 Random Forest

Random Forest (RF), introduced by Leo Breiman, is a decision tree-based ensemble model that works well for regression and classification problems [16]. The RF combines the output of multiple decision trees to produce a result. A principal decision tree example is shown in Figure 15.

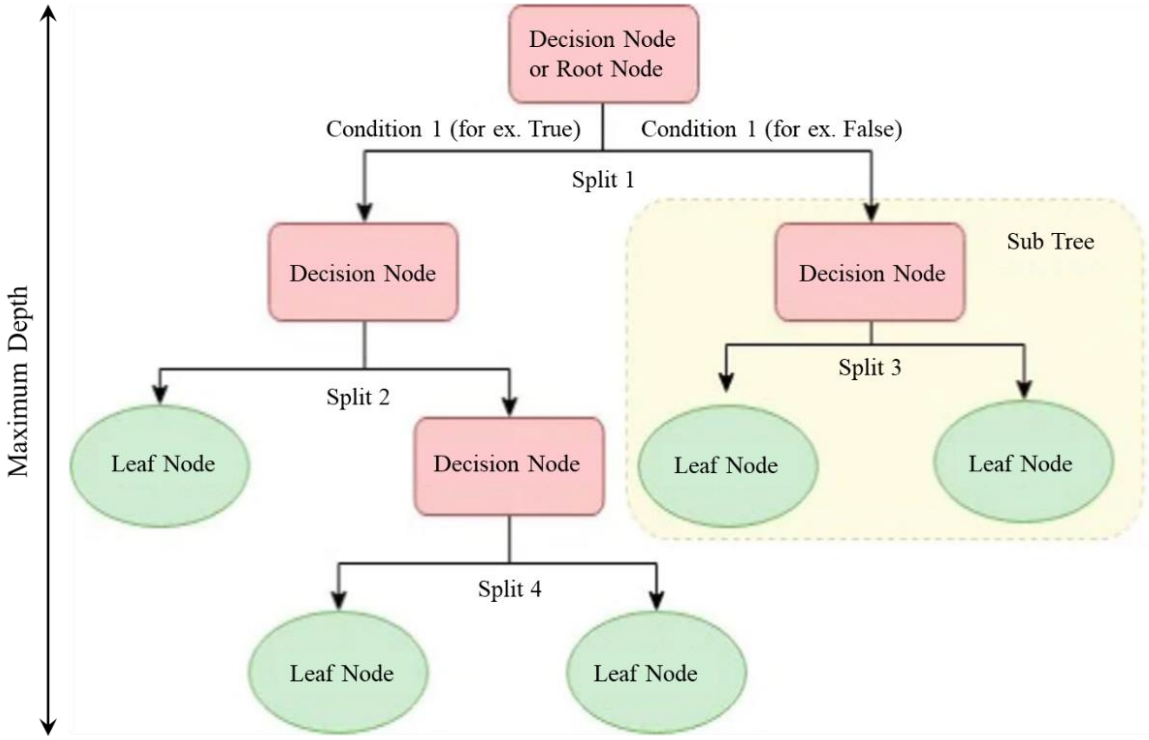

Figure 15. Principal structure of Decision Tree (modified from [17]).

Each decision tree takes some random number of bootstrap samples from the training data so that the number of features and rows in a sample is less than in the initial dataset. Some features and rows can be used again in another decision tree. Several decision trees are created from bootstrap samples, and at each node of every tree, the best split is decided based on the provided features. In the final stage, the output produced by every decision tree is aggregated and the result is chosen by a majority vote [11]. The schematic structure of the Random Forest algorithm is shown in Figure 16.

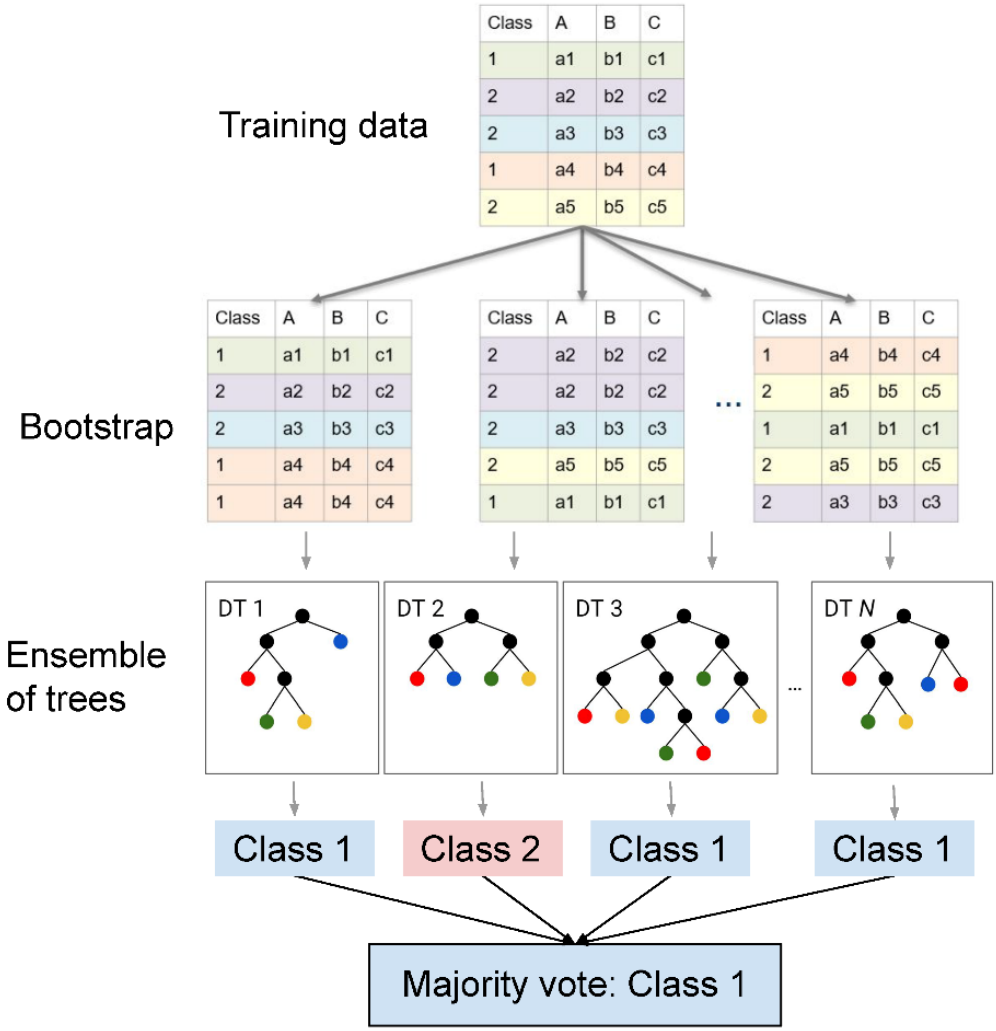

Figure 16. Schematic structure of the RF algorithm [18].

Random Forest is an effective method for classification prediction. Despite the number of decision trees used in RF, the model usually does not overfit [19]. The method is fast and performs a high-accuracy prediction.

3.2 Neural Networks

Neural Networks (NN), or Artificial Neural Networks (ANN) are a part of machine learning and a core of Deep Learning methods. They are highly efficient algorithms for

classification that can learn from training data and increase the model performance over time. This technique does not have information about how to solve the problem but can find the best solution, unlike the supervised ML. In several areas, such as speech recognition and language processing, the ANN proved to surpass other classification approaches. However, it was only possible to implement the NN after the introduction of Deep Learning. Initially, the idea of developing artificial neural networks was designed to imitate the biological nervous system. The human brain is composed of cells called neurons which are connected to other neurons by axons. A synapse defines a connectivity strength between neurons. The brain is composed of billions of neurons and uses the changing synapse signals to learn new activities. A single neutron can make the simple task of receiving and responding to a coming activation signal and transmitting it to another receptor neuron [20]. A complex interconnection of neurons allows the system to perform complex tasks.

The same principle underlies the idea of the Artificial Neural Network. It consists of computing nodes connected with each other through direct links. The synapse is represented by the weights of those links. The main goal of the ANN is to modify weights in a such way that they can reproduce the input and output data. To better understand the concept of the ANN, an example of a one-layer network is shown in Figure 17.

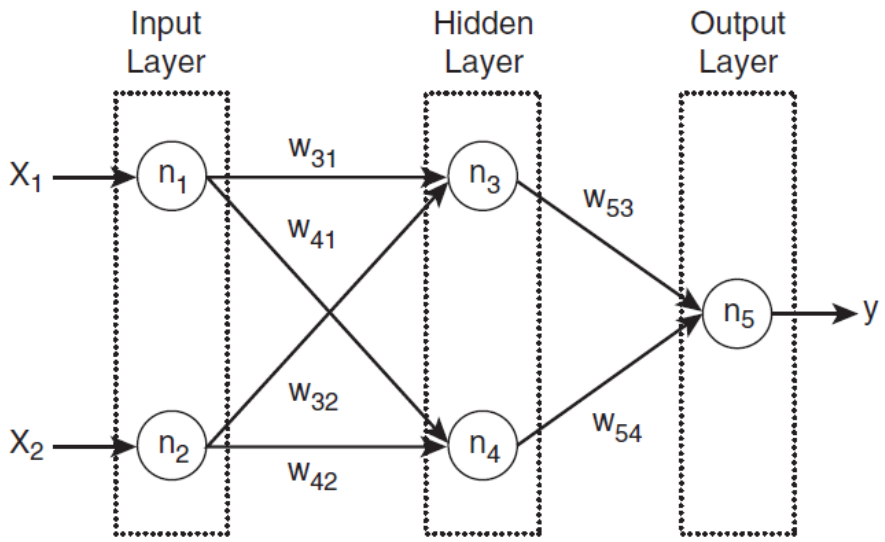

Figure 17. One-layer Neural Network.

The first input layer is used for adding inputs from attributes, and every numerical or categorical attribute is assigned to each node (n_1, n_2). Then a specific weight is applied to every node (w_{31}, w_{41}, w_{32}, w_{42}) and they are fed into a hidden layer that is comprised of hidden nodes (n_3, n_4). Every hidden node utilizes an activation function that takes a decision of whether a neuron will be activated or not, and then produces an activation value that is passed to the next output layer. These values are again multiplied by a new set of weights (w_{53}, w_{54}). In the output layer, activation values are processed, and the output values are predicted [20].

If we consider one *i* node at the *n* layer, then the activation values *a* will be [20]:

$$a_i^n = f(\sum_j weight_{ij}^n \cdot a_j^{n-1} + bias_i^n) \quad (4)$$

where a_i^n is the activation function of node *i* at layer *n*; $weight_{ij}^n$ is the weight between nodes *j* and *i* in the layers *n* and *n-1*, respectively; and $bias_i^n$ is the bias at the node *i* .

Different types of activation functions are used in multi-layer NN. An activation value is generated in every node and is represented as an activation function calculated from neutrons in the previous layer. There are several activation functions, such as sigmoid, linear, and hyperbolic, which can be utilized when customizing the NN model. When training the NN for facies prediction, I used sigmoid and softmax activation functions for the first and second hidden layer, respectively. The reason for using these activation functions is that they are usually implemented for multi-class classification, as well as they are relatively simple and reliable.

The one-layer neural network utilizes one hidden layer in which relatively simple features are captured from the input attributes. Including the additional hidden layer in the NN, this makes it possible to combine two hidden layers and produce more complex features. This ability of the ANN makes it a powerful method for classification prediction compared with other approaches. In this thesis, I used a neural network with two hidden layers.

After the output is calculated, the difference between the input value and the predicted value is computed. The sum of square differences between the last two values is defined as a cost function. The cost function can be minimized during the training process. If the cost function is large, then weights in the NN should be updated by utilizing optimizers. This process is called backpropagation. Once weights are updated, the new cost function is calculated. The formula for calculating the updated weight is shown below [20].

$$weight_{new}^{n} = weight_{old}^{n} - \eta \frac{\partial L}{\partial weight_{old}^{n}} \quad (5)$$

where $weight_{new}^{n}$ is an updated weight in the layer *n*; $weight_{old}^{n}$ is a previous weight; η is a learning rate; $\frac{\partial L}{\partial weight_{old}^{n}}$ is a derivative of the loss function with respect to the previous weight. The one cycle of forward and backpropagation is called epoch. This process is iterative until the cost function is minimized.

3.3 Optimizing the Machine Learning Model

3.3.1 Hyperparameters and hyperparameter tuning

In machine learning algorithms, there are two types of variables: model parameters and hyperparameters. The model parameters are the parameters that a ML model tunes according to the provided training dataset, such as weights in neural networks, while hyperparameters are high-level parameters of ML techniques that are set before the start of the model training [20] and they are not a part of the final model. Hyperparameters are one of the most crucial parts of producing effective machine-learning models. It is important to understand how hyperparameters can affect a model's performance before training the model.

There are several hyperparameter optimization approaches implemented in the scikit-learn library: Random Search, Grid Search, Bayesian Optimization, Gradient Descent, etc. The first two strategies are the most used for hyperparameter tuning. In this thesis, Random Search and Grid Search are applied as hyperparameter optimization techniques.

Grid Search is a powerful approach to identify the optimal set of hyperparameters for a given model. In this case, Grid Search tries all possible combinations of the passed hyperparameters. This approach certainly will find the best hyperparameters, however, it requires large computational resources and time. Random Search requires less computational time and can be utilized for a large dataset. This approach can be more efficient in high-dimensional space when the model has a variety of hyperparameters and some of them are more important than others [22].

In this study, the hyperparameter optimization was applied to the model that showed the best classification performance which is Random Forest Classifier. A set of hyperparameters of the RF Classifier is shown in Table 1.

Table 1. The main hyperparameters of the Random Forest Classifier model.

Hyperparameter	Description
n_estimators	The number of trees inside the RF Classifier
max_features	The parameter that looks for the number of features to achieve the best split
max_depth	The maximum height of the trees inside the model
min_sample_split	The minimum number of samples in the internal node
min_sample_leaf	The minimum number of samples that a node holds after split
criterion	The function that defines the quality of a split

N_estimators control the number of trees in the model. The right number of n_estimators improves the prediction accuracy of a training dataset, however, it may lead to overfitting and increasing the model's complexity. By default, the parameter n_estimators is equal to 100. The parameter max_features sets a limit to the number of features in every tree to predict the target variable. By default, the parameter max_features is an 'sqrt' which means that the number of features used for splitting is equal to a square root of the number of features. The maximum number of splits in every tree (max_depth) should be properly chosen to avoid overfitting and underfitting (see paragraph 3.3.2). By default, the max_depth is None. The min_sample_split shows the minimum number of samples in the internal node. A small number of samples restricts the tree and can result in overfitting of the model. The default value is equal to 2. The min_sample_leaf represents the minimum number of samples that a node holds after the split and a sufficient number of leaves reduces the risk of overfitting. The default number is equal to 1. As explained in Table 1, the hyperparameter criterion measures the split quality, and the default value used in the thesis is 'gini' which means the optimum split is based on a gini impurity criteria is calculated as follows [9]:

$$Gini = 1 - \sum_{i=1}^{j} P(i)^2 \quad (6)$$

where *j* is the number of classes, and *P* is the probability of each data point of class *i*.

Since hyperparameter optimization relies on a training dataset and considers multiple combinations of it to evaluate the best model's performance, the process can lead to overfitting. Below we explain model overfitting or underfitting.

3.3.2 Overfitting and Underfitting

Overfitting is an essential problem of the supervised machine-learning techniques and takes place when the model shows a low error for the training data but has a poor performance for the testing dataset. In addition to the reasons mentioned earlier for overfitting, a noisy dataset and insufficient training dataset size may cause this problem [23]. The complex nonlinear and nonparametric models with large flexibility usually tend to be overfitted. Overfitting is characterized by a high variance and low bias (Figure 18). These terms are critical for analyzing any model performance.

Bias is a term describing the difference between the average of the model's prediction and the actual value of the dataset. Variance represents the amount of variation in the model prediction for different training datasets.

Underfitting takes place when the model is oversimplified and unable to represent the relationship between the attributes and targets. Underfitting is the opposite of overfitting. Underfitting leads to a high error during the training and testing process, and it usually takes place in a small dataset. Collecting additional data, using more complex models, and choosing proper features can tackle the problem. Underfitting is associated with low variance and high bias and is shown in Figure 18 [24].

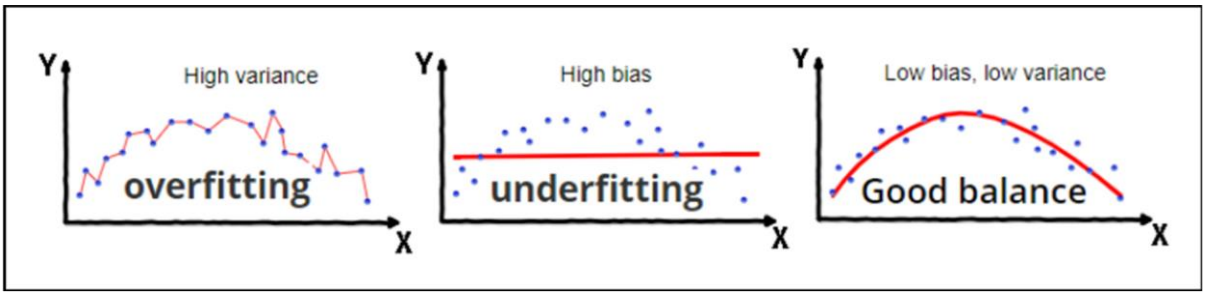

Figure 18. Examples of overfitting (a), underfitting (b), and a good balance between the data and model (c). Blue points represent training data, and the red line is the model [24]

It is essential to achieve a correct balance between variance and bias because an increase in variance results in a decrease in bias and vice versa. The model that has low bias and low variance produces more accurate predictions and, consequently, reduces the total error. The bias and variance trade-off are shown in Figure 19 [25].

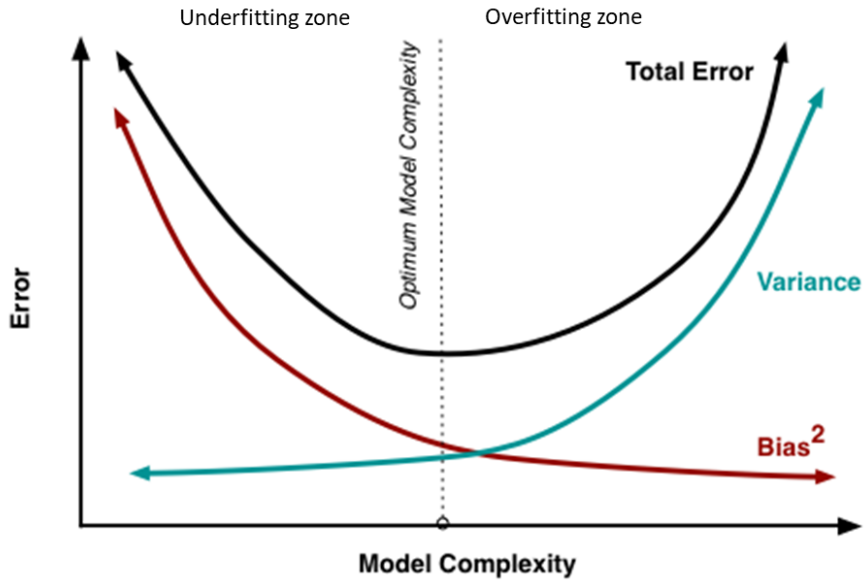

Figure 19. Bias and variance trade-off for the case of underfitting and overfitting (modified after [25])

The total error is the sum of variance and bias and can be expressed as:

$$Total\ error = Variance + Bias^2 + Irreducible\ Error \quad (7)$$

where the Irreducible Error is the error that represents the noise of the dataset.

3.3.3 Cross-validation

Cross-validation is a popular method for evaluating ML models and testing their performance. The algorithm is based on randomly partitioning the dataset into a *K* number of equally sized subsets, training a model on a K-1 number of subsets, and testing it on the remaining subset. Cross-validation is an iterative method and repeats the workflow for the rest of the dataset. For a better understanding, we will use a dataset that is split into three samples (S_1, S_2, S_3). During the first iteration, samples S_2 and S_3 are used to train a model, and sample S_1 is utilized for testing. The model error E_1 is calculated for the first sprint. Similarly, the model is trained for samples S_1 and S_3, tested for the S_2 subset, and E_2 error is computed. Ultimately, samples S_1, S_2, and S3 are used for training and testing. The value of the overall error is the average of the three obtained errors E_1, E_2, and E_3. This example is called three-fold cross-validation (Figure 20).

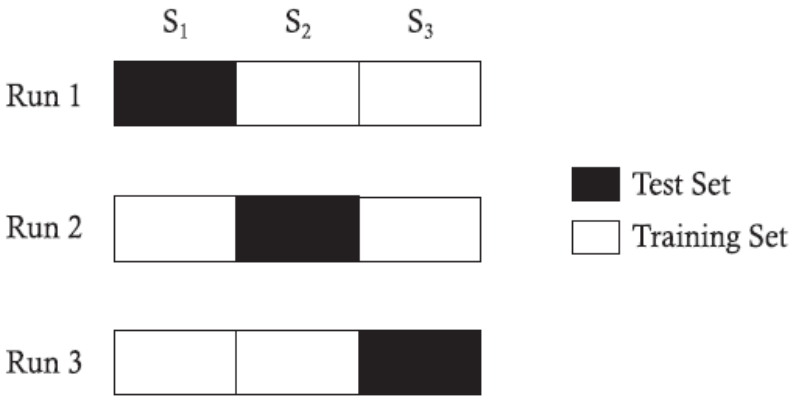

Figure 20. Three-folded cross-validation [20].

It is essential to choose a proper number of folds in cross-validation because a small value can lead to increasing bias and generalization error. On the other side, a high number of K-folds may result in decreasing bias but increasing variance. In general, the number of K-folds varies between three and ten and depends on the size of the dataset. In this thesis, cross-validation is used when tuning hyperparameters for the Random Search and Grid Search models. However, because

the training of the ML models is done by using three to seven wells comprising approximately 2-5% of all data, the use of cross-validation on training and validation can be misleading and increase bias. For this reason, the models' performance is evaluated by comparing the f1-scores of the testing set. An explanation of the F1-score and other performance evaluation parameters is provided below.

3.4 Evaluation performance for classification

The evaluation method is crucial in estimating the classification performance of a model when executing training and testing. In classification, the performance is presented as a comparison between the predicted class and true class. The summary of the comparison is a matrix called a confusion matrix or contingency table [26]. This matrix shows which classes are misclassified by a model.

The concept of the confusion matrix is the same for binary and multi-class problems. To better understand this matrix, we will consider a binary classification with a positive P class and a negative N class. The trained model takes unknown samples to predict one of the two actual classes. The illustration of the confusion matrix is shown in Figure 21.

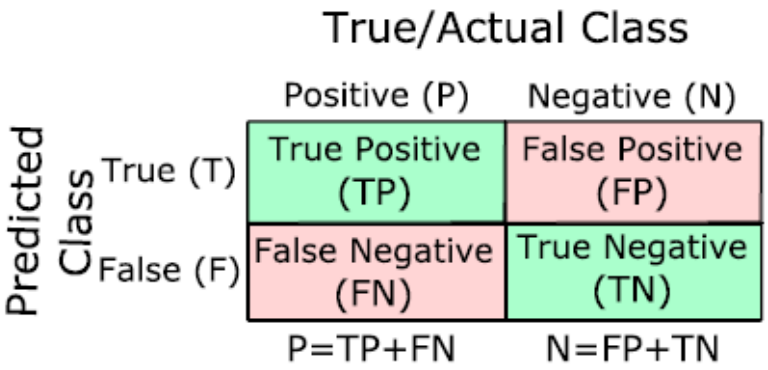

Figure 21. A confusion matrix for binary classification [26].

This contingency table is utilized to calculate the main classification metrics. Accuracy is one of the most common parameters for the evaluation of classification performance, and it is the fraction of correctly classified samples and the total number of samples.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (8)$$

where TP is the number of properly identified positive class samples, TN is the number of correctly classified negative samples; FP and FN are the numbers of incorrectly classified positive and negative samples, respectively.

Precision is the ratio between the number of correctly classified positive values to the total number of samples classified as positive.

$$Precision = \frac{TP}{TP+FP} \quad (9)$$

The recall is a parameter that is measured as a proportion of instances classified as positive and a sum of true and false negative samples (the total number of positive samples).

$$Recall = \frac{TP}{TP+FN} \quad (10)$$

F1-score is the harmonic mean between the recall and precision values.

$$F1\ score = \frac{2(Precision \cdot Recall)}{Precision + Recall} \quad (11)$$

Some of the above-mentioned metrics are vulnerable if the dataset is imbalanced which means that instances of some classes exceed the number of other classes. If the dataset is imbalanced, accuracy is not the correct metric for classification performance. Using the accuracy metric for this case can lead to a misleading interpretation of the prediction. In this case, it is recommended to use weighted parameters, such as F1-score, which give to the less presented facies higher weight. In this thesis, the F1-score is used for evaluating the ML models “accuracy”.

3.4 Software Tools and Libraries

In this section, an overview of software tools, their libraries, and modules is described. It is important to use stable and compatible packages to avoid any possible conflicts and errors while running code.

All calculations in this thesis are done in the operating system Windows 10. Visual Studio Code (VS Code) made by Microsoft is used as a code editor and can be utilized with many programming languages. VS Code is a very popular tool for running, coding, and debugging. It is also an open-source and free project for students, and regular stable updates make it possible to utilize this tool with the latest version of programming languages, such as Python. In this study, the VS Code 1.78 version is used.

Python is a powerful open-source programming language. Being based on object-oriented paradigm concepts, the language offers many different libraries and modules that are extremely popular among data scientists. The reason for Python's popularity is the simplicity of coding and easily understandable syntax. In this thesis, Python, with a stable version of 3.9.7, is used. The Python tool can import a large variety of data science-related libraries that can help to perform ML applications and tasks. These libraries are Pandas, Matplotlib, NumPy, Scikit Learn, TensorFlow, etc. An overview of the libraries is shown in Table 2.

Table 2. An overview of libraries used in Python and their versions.

Library	Version
Matplotlib	3.4.3
NumPy	1.22.4
Scikit Learn	0.24.2
TensorFlow	2.12.0
Pandas	1.3.4
Plotly	5.5.0
Seaborn	0.11.2

NumPy or Numerical Python is a library that aims to work with array objects and operates with many mathematical functions. In this thesis, NumPy is used to perform calculations on one-dimensional and two-dimensional arrays. For visualization of these arrays, I used several libraries

such as Matplolib, Plotly, and Seaborn. In addition to arrays, the DataFrame and Series structures were used in this thesis. Series is a one-dimensional array with homogeneous data, while DataFrame is a two-dimensional structure with heterogeneous data. The DataFrame and Pandas are like the table and Excel program, respectively.

Scikit Learn is a Data Science and Machine Learning library containing different algorithms for regression, classification, and clustering, among others. This library has many tools for optimizing and improving supervised ML models. TensorFlow is used for the Deep Learning and Neural Networks methods in this thesis [27].

In addition to the above mentioned libraries, I used several specific tools for loading and handling SEG-Y files that store geophysical data (seismic and its attributes). These libraries are developed by Equinor (Table 3).

Table 3. An overview of the specific Python libraries for SEG-Y data and their versions.

Library	Version
Segyio	1.9.10
Segysak	0.3.4

3.5 Standardization

Some machine learning methods are sensitive to the feature scale, and this can affect the models' performance if, for example, the models are distance-based such as KNN and SVM, where the classification is performed by measuring the distances between new and labeled instances. As a result, some features with larger scales can dominate over others [28]. By bringing all features to a similar scale, it is possible to assure all features contribute to the facies classification equally. There are different techniques for feature scaling, and in this study, the method called standardization is used. Standardization is a feature scaling approach that transforms data in such a way that their distribution has mean and standard deviation equal to zero and one, respectively. Equation 12 shows how to scale feature values by standardization.

$$X_{standard} = \frac{X - \mu}{\sigma} \tag{12}$$

where $X_{standard}$ is the standardized value of X from the feature dataset, μ is the mean of the feature dataset, and σ is the standard deviation of the feature dataset.

As discussed in Chapter 5 ‘Data Analysis’, features such as Seismic, Seismic Inversion, Relative Acoustic Impedance, Instantaneous Frequency, Envelope, and Geological Time, have a different scale and can affect the ML model performance. For this reason, the standardization was applied to all features used in this thesis.

4 Methodology

This chapter describes the methodology, which comprises three parts. The first part gives information about dataset preparation. The second part describes the workflow of what type of analysis was performed for the given dataset. Finally, the third part describes the machine learning workflow implemented for the prepared data.

4.1 Dataset preparation and analysis workflow

The composite subsurface geometry of a reservoir, together with the presence of various deposits, makes the facies prediction more challenging. For this reason, several 2D vertical sections and one 3D cube with various reservoir geometries are used for facies classification. I started with a facies section that is divided into the lower and upper zones, and these zones were analyzed separately. This is because, in the lower zone, facies are homogeneous and consistent laterally, while in the upper zone, facies are thin and inconsistent. In addition, I used a section that contains a normal fault that divides the reservoir into two parts. Moreover, to test the ML benefits and challenges, I used a three-dimensional cube also containing a normal fault. The navigation of the two sections and the 3D cube are shown in Figure 22.

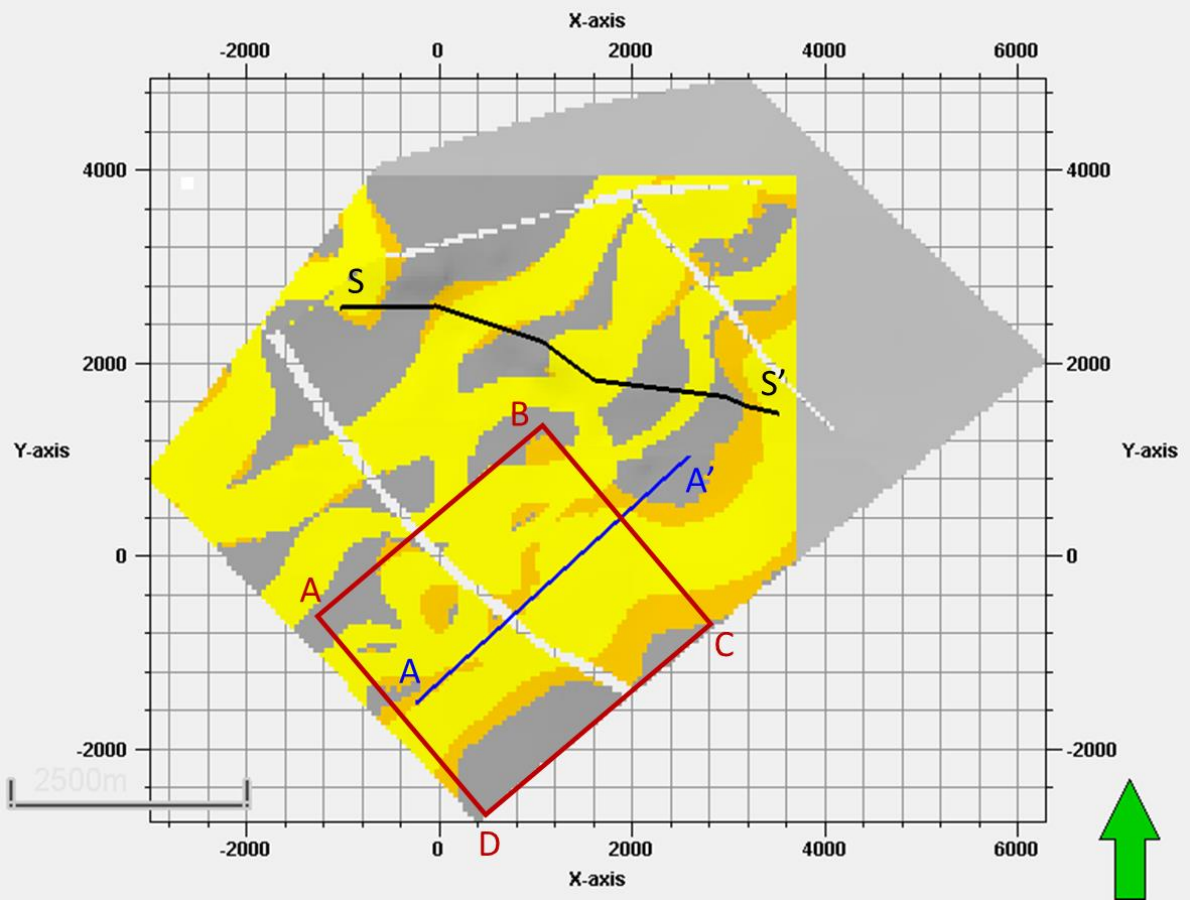

Figure 22. Depth slice through the Facies cube with the navigation of the sections and the 3D cube used in this analysis. The black line S-S' is the navigation of the section with the lower and upper zones. The blue line A-A' is a navigation of the section with the normal fault. The red square ABCD is a navigation of the 3D sub-cube used for the NN analysis.

In total, there are four cases considered in this study: Cases 1 and 2 are the lower and upper reservoir zones in section S-S', respectively. Case 3 is the lower reservoir zone in section A-A'. Case 4 is the lower reservoir zone in the facies sub-cube. Cases 1 to 4 are shown in Figure 23.

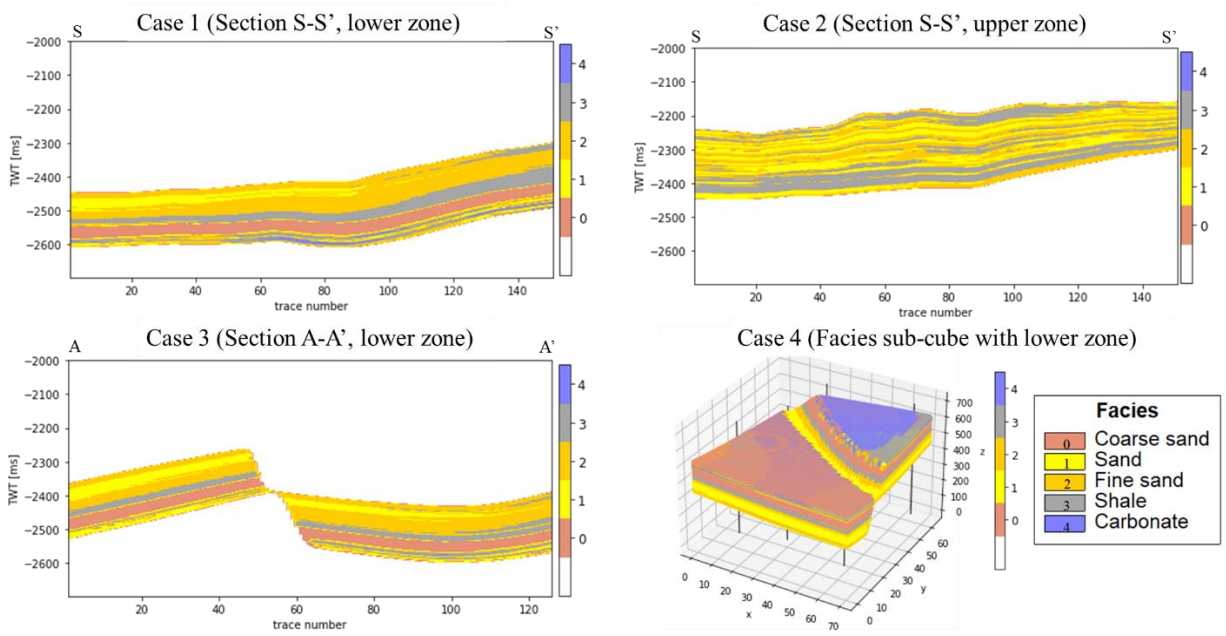

Figure 23. Four cases considered in this study. Case 1 is section S-S' with the lower reservoir zone. Case 2 is section S-S' with the upper reservoir zone. Case 3 is section A-A' with the lower reservoir zone offset by a normal fault. Case 4 is the lower reservoir zone in the facies sub-cube.

The use of different composite geometries helps to understand whether machine-learning methods trained on a few wells can predict facies classification accurately on a 2D or 3D dataset.

The facies classification prediction is based on seismic data, its attributes and seismic inversion. The analysis starts with the application of ML to the 2D sections, namely for the lower and upper zones. I started with the following data: seismic, relative acoustic impedance, and seismic inversion. These data were used as features and the facies as labels.

In order to apply the ML models for facies classification, I extracted from each case several traces which were further utilized as synthetic well-logs. The training and validation process was performed for these synthetic well-logs, and the ML testing for the cross-section and the subcube. Since it is possible to extract any number of well traces, I investigated the role of the number and location of traces (or wells) on the facies prediction performance. The minimum acceptable number of traces (or wells) was selected from the condition that the ML model performance should exceed a threshold of 75%.

Following that, I evaluated the role of various frequencies, such as Ricker wavelet (25 Hz) and three Ormsby frequencies (10-60, 10-80, 10-100 Hz), which were used for synthetic seismic

processing. The use of different frequency ranges can increase or decrease the ML models performance in thick or thin-layered reservoirs.

In the real world, seismic data acquisition is usually accompanied by noise coming from several sources such as acquisition, processing, cultural noise, weather, etc. For this reason, in this study, I checked if ML can handle noisy data and how noise can affect the facies classification.

After that, additional seismic attributes, such as Envelope and Instantaneous Frequency, were incorporated into the analysis. As mentioned earlier, seismic inversion can be computationally expensive and is not as straightforward as, for example, seismic attributes, such as relative acoustic impedance or complex attributes. So, I estimated the feasibility of using seismic attributes and compared their performance with the seismic data, and seismic inversion.

Finally, incorporating extra information about the geological structure of the reservoir can improve the facies prediction. These data can be obtained from different methods. One of the techniques was described and implemented in [29] where seismic horizons or sequence boundaries are used. In this study, the geological time property was considered as another type of data. This property was incorporated in the facies prediction as an additional feature, and it was used for the lower and upper zones. The overview of the described workflow is shown in Figure 24.

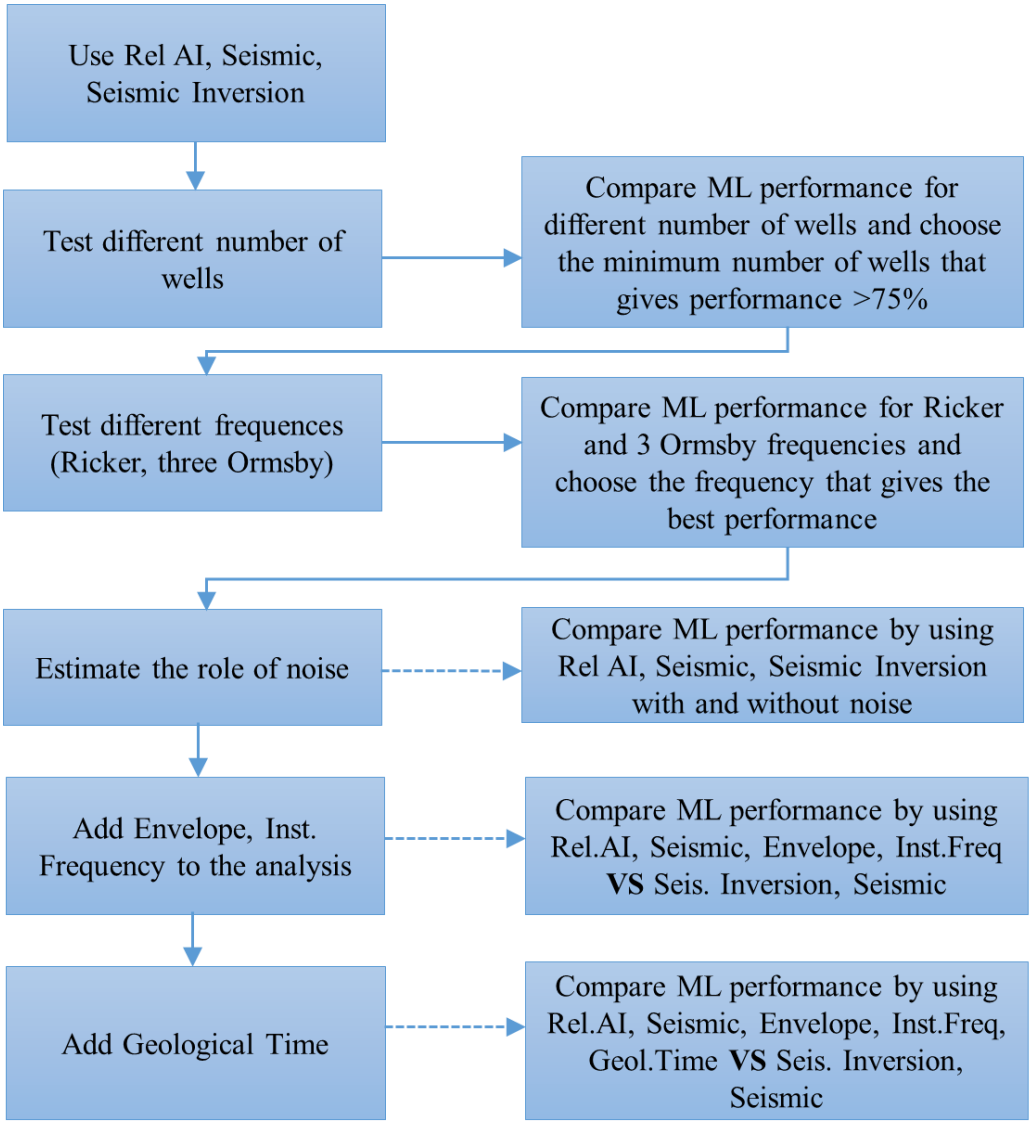

Figure 24. The overview of the performed analysis for the lower and upper zones of the reservoir in cases 1 and 2.

As the calculations for the lower zone and the upper zone are completed along section S-S', and the minimum number of synthetic well-logs with the appropriate frequency was determined for both zones of the reservoir, the research continued by taking into consideration section A-A' with the lower zone offset by a normal fault, and the 3D small cube with the lower zone. For these two cases, all mentioned seismic attributes, together with seismic inversion, were used for facies classification. Geological time was not used in these cases. The following section explains the general machine learning workflow.

4.2 Machine learning workflow

The general workflow carried out in this thesis is shown in Figure 25. The workflow is designed to accomplish the aim of the study, which is to explore the ML opportunities for facies classification based on seismic data, seismic attributes, and seismic inversion.

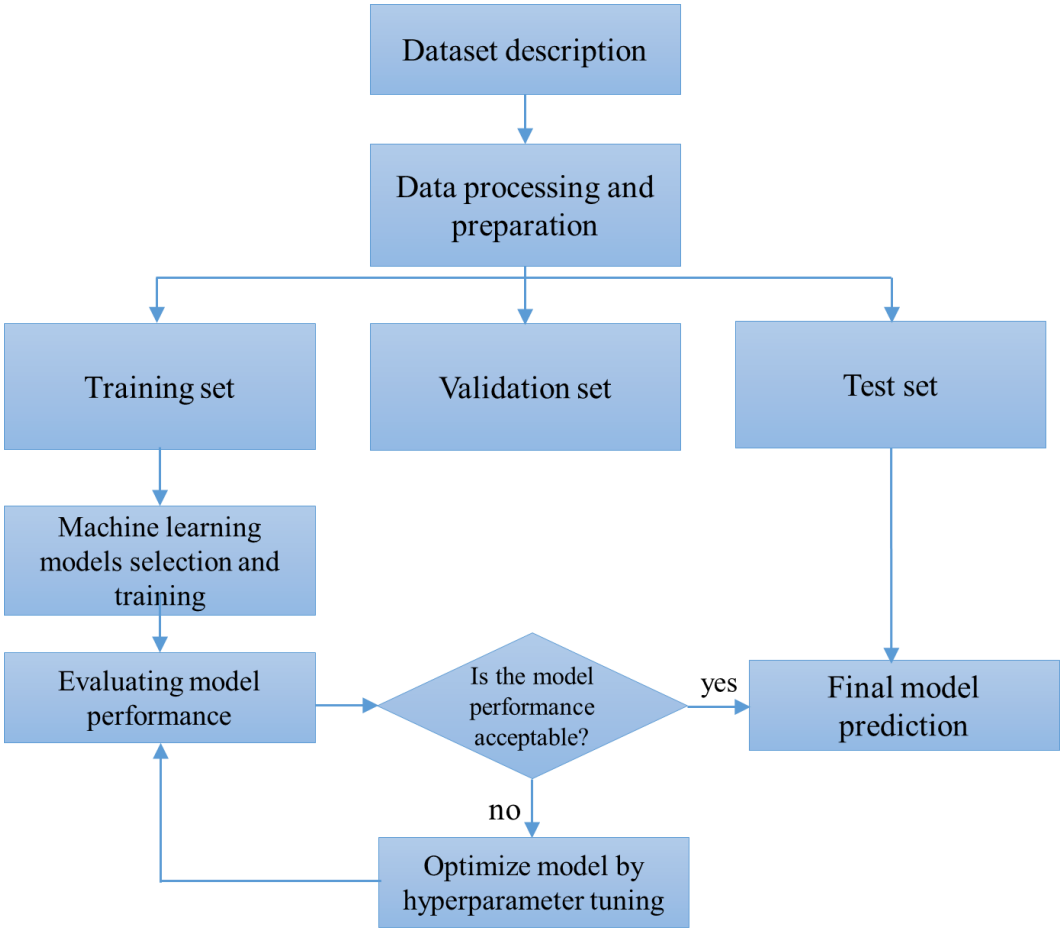

Figure 25. Machine learning workflow of this study.

Once the processing and preparation of the data were completed, the data were divided into the training, validation, and testing subsets. The extracted well-logs from the sections of facies and features were used as training and validation subsets, and their proportion was selected as 80/20. So, for example, for three wells used for building and training the ML models, 80% of data were used for training and 20% for validation. When various machine learning models were built and trained, the rest of the facies traces in the section were used for testing the ML models.

Having a synthetic model and, consequently, the ground-truth values of facies, makes it possible to evaluate the prediction accuracy. The comparison of the facies prediction was made by utilizing a difference map that subtracts the predicted facies values from the actual values. If the predicted value is the same as the actual one, the result is true; otherwise, the result is false.

The workflow, shown in Figure 24, was implemented for the lower and upper zones (for cases 1 and 2). The main goal of this part is to evaluate the role of the number of wells, frequency ranges, spectral noise, and additional features. For this reason, the analysis is done for baseline models, such as the LR, KNN, SVM, RF, and NN, which means that the hyperparameter tuning are not applied for these scenarios. For these parts, the evaluation is performed by using the overall F1-score of the ML models and F1-score for each facies. After the analysis is completed, and the number of wells, frequency range, and the role of using additional features is determined, the ML process for facies prediction is applied to the other two cases, the section A-A' with the lower zone (case 3), and the seismic cube with the lower zone (case 4). In these regions, I performed the models' optimization by utilizing hyperparameter tuning and using all given features together.

5 Data Analysis and Processing

Data analysis and processing transforms raw input data into a usable form, highlighting valuable information, and identifying hidden patterns to make them more informative. This step is an essential part of the Machine Learning workflow because the processed data are then used for building the ML models. Clean data also improves the ML models' performance by reducing bias. In this chapter, the exploratory analysis of the input facies and seismic features is given.

5.1 Exploratory facies analysis

The facies dataset of cases 1, 3 and 4 (lower reservoir zone) consists of five facies: coarse sand (CS), sand (S), fine sand (FS), shale (Sh), and carbonate (C), while the datasets of case 2 (upper reservoir zone) consists of four facies: coarse sand (CS), sand (S), fine sand (FS) and shale (Sh). A detailed description of the facies and their distribution is given in Table 4. In addition, histograms showing the facies distribution, based on the data from Table 4, are shown in Figure 26.

Table 4. Summary of the facies presence in cases 1 to 4.

Facies	Label	Code	The presence of facies in study cases (case), %			
			Case 1	Case 2	Case 3	Case 4
Coarse Sand	CS	0	19.2	0.6	21.4	23.7
Sand	S	1	12.8	29.6	20.3	26.7
Fine Sand	FS	2	42.1	35.7	38.4	28.1
Shale	Sh	3	23.7	34.1	18.9	17.4
Carbonate	C	4	2.3	0	1.0	4.1

In addition, histograms showing the facies distribution, based on the data from Table 4, are shown in Figure 26.

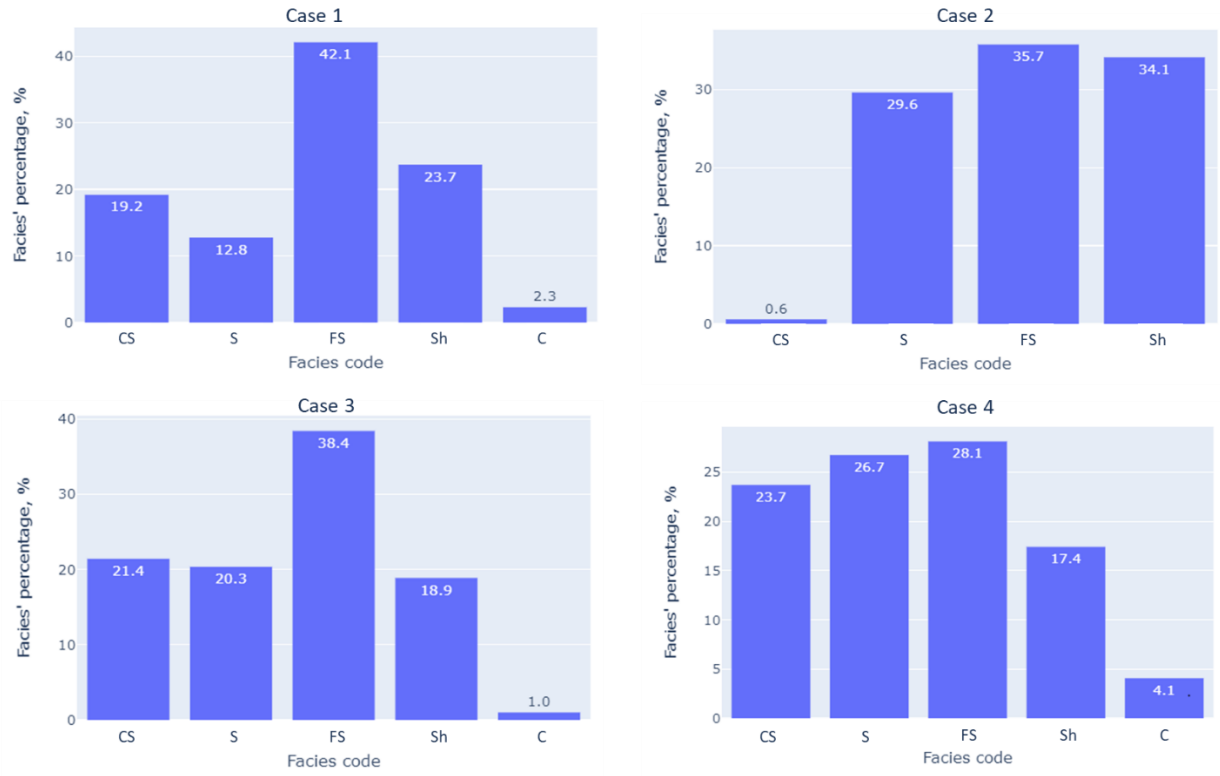

Figure 26. Distribution of facies presence in cases 1, 2, 3 and 4.

Cases 1, 3 and 4 contain the lower part of the reservoir, while case 2 has the upper part of the reservoir. As mentioned in Chapter 2 (Dataset description), the facies of the lower part are thicker and unconformable in comparison to the facies from the upper part.

Cases 1, 3, and 4 have a similar distribution, with predominance of fine sand (FS) which is 42.1%, 38.4% and 28.1%, respectively. The presence of coarse sand (CS) and shale (Sh) in these cases is also almost the same and is approximately 20%, while the carbonates are represented only by thin layers at the base of the section, and their proportion is around 1-4%.

An interfingering of thin layers of fine sand (FS), sand (S), and shale (Sh) mainly characterizes case 2. The presence of coarse sand does not exceed 1%.

The facies distribution from these four cases shows that the dataset is imbalanced, meaning that some facies are predominant compared to others which can affect the training process. Dealing with imbalanced data is an essential part of data processing, as it can decrease bias and improve ML model predictability.

Different approaches are developed for addressing the imbalance dataset problem, and the most popular are oversampling and undersampling [30]. Undersampling is a technique that identifies the minority class and duplicates examples, while oversampling removes examples from the majority class. In addition, some machine-learning methods, such as Balanced Random Forest Classifiers, can handle highly imbalanced datasets because the model considers class weights making the methods cost-sensitive. Moreover, the majority class is down-sampled, thus the decision trees are built and trained on a balanced dataset [30].

5.2 Exploratory features analysis

Along with the facies dataset, the information about features and their distribution is also essential when deciding about feature engineering and implementing optimal machine learning methods. For example, features skewness, i.e., the asymmetry of the distribution, can result in facies misclassification. In addition, skewness gives insight into the presence of outliers and more importantly, the shift of the mean value which is quite essential for prediction performance. Outliers are the data that differ significantly from the rest of the dataset and usually are caused by the human factor, recording errors, etc [30]. The following plots were constructed for better visualization of seismic attributes for every zone.

5.2.1 Case 1

For the first zone, seismic, seismic inversion, relative acoustic impedance, instantaneous frequency, envelope, and geological time, are considered for facies classification. Their distribution is shown in Figure 27.

The distributions of seismic inversion and envelope are skewed to the right, while seismic is skewed to the left. Relative acoustic impedance and instantaneous frequency are normally distributed with minor skewness. The geological time distribution is uniform. As mentioned earlier, the skewness can cause misclassification while using ML models based on Euclidean distances.

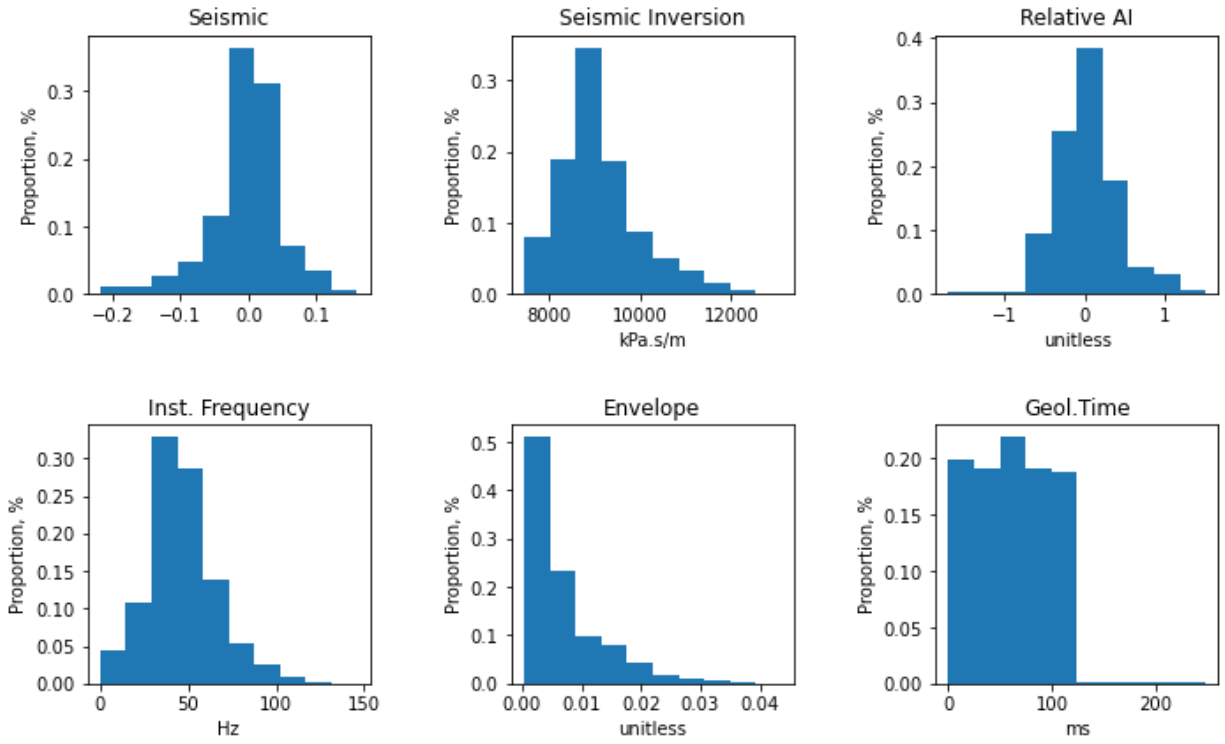

Figure 27. Distribution of seismic data and seismic attributes in case 1.

In addition, to detect inadequate values in the dataset via histograms, the boxplots of all features for case 1 are presented in Figure 28. The red line inside the box shows the data median, and the height of the black box shows the range of 50% of the data. Two horizontal lines outside the box indicate the range of 95% of the data, while the black points outside the lines detect outliers [31].

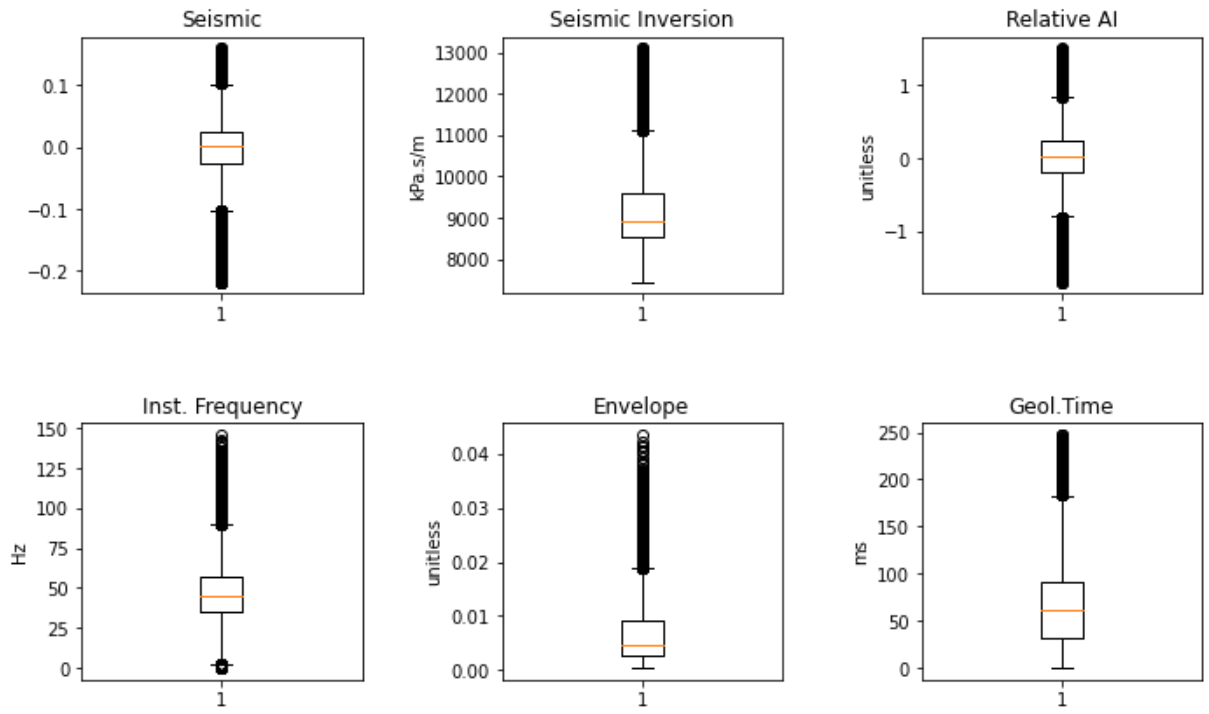

Figure 28. Boxplots of seismic data and its attributes used for facies classification in case 1.

The boxplots show that all values of the features do not go beyond their physical boundaries.

5.2.2 Case 2

For case 2, the same features as for case 1 were used for facies classification: seismic, seismic inversion, relative acoustic impedance, instantaneous frequency, envelope, and geological time. The distribution of these features is shown in Figure 29.

In this area, the distribution of seismic, relative AI, and instantaneous frequency is normal, while seismic inversion and envelope have slightly skewed to the right distribution.

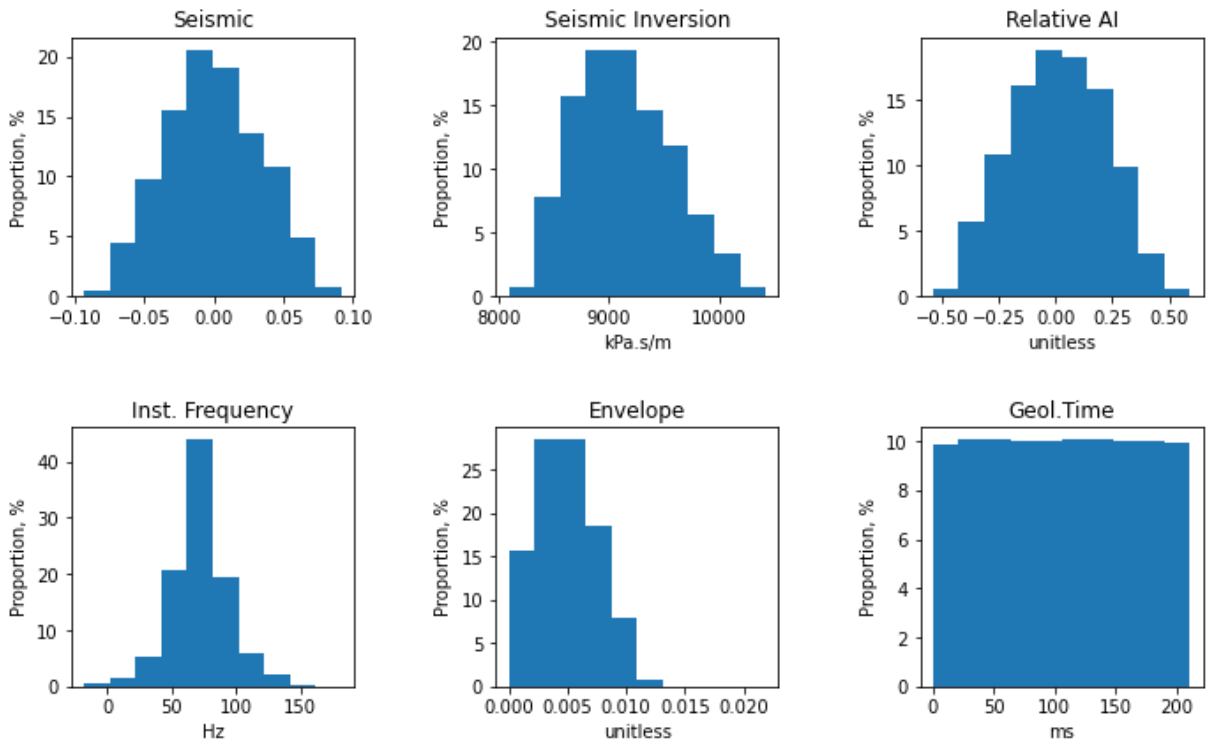

Figure 29. Distribution of seismic data and seismic attributes features in case 2.

Figure 30 shows the boxplots of the features for case 2. The features, such as seismic, relative AI and geological time, do not have outliers. At the same time, the outliers in seismic inversion, instantaneous frequency and envelope do not exceed their physical boundaries.

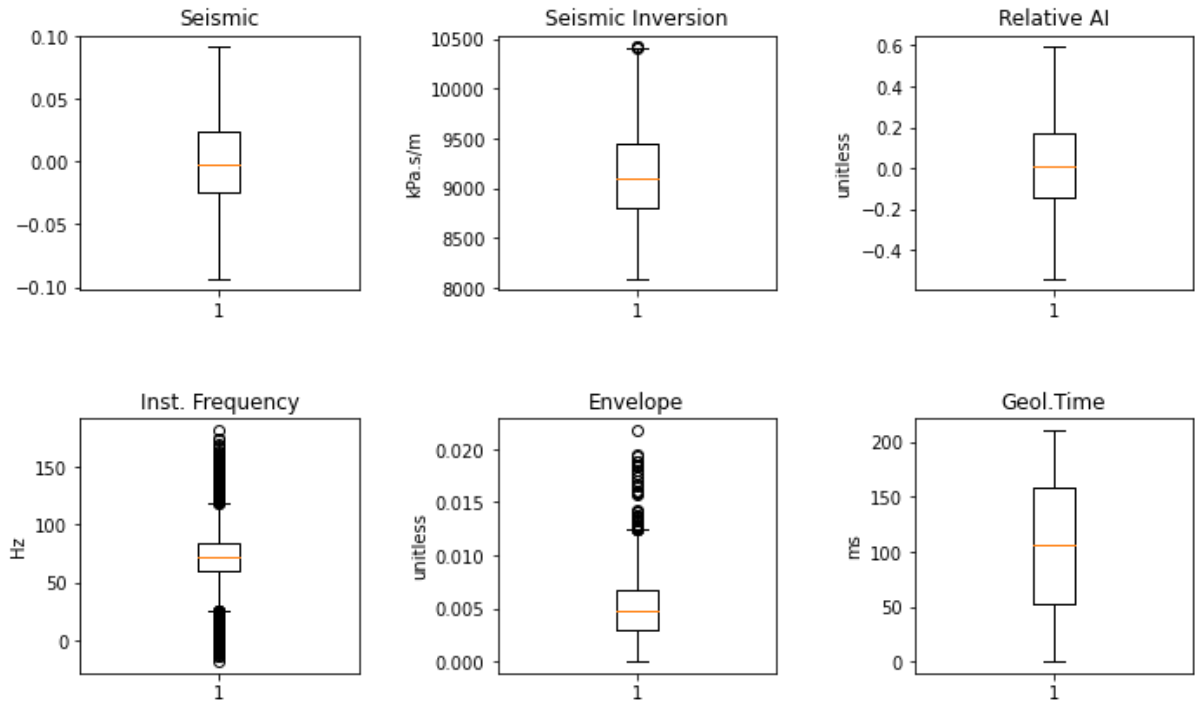

Figure 30. Boxplots of seismic and its attributes used for facies classification in case 2.

5.2.3 Case 3

Facies classification of case 3 is performed by using five features together: seismic inversion, relative acoustic impedance, instantaneous frequency, and envelope. The distribution of these features is shown in Figure 31. The geological time attribute is not available for this case. Most of the features have skewed to the right distribution except for seismic and instantaneous frequency.

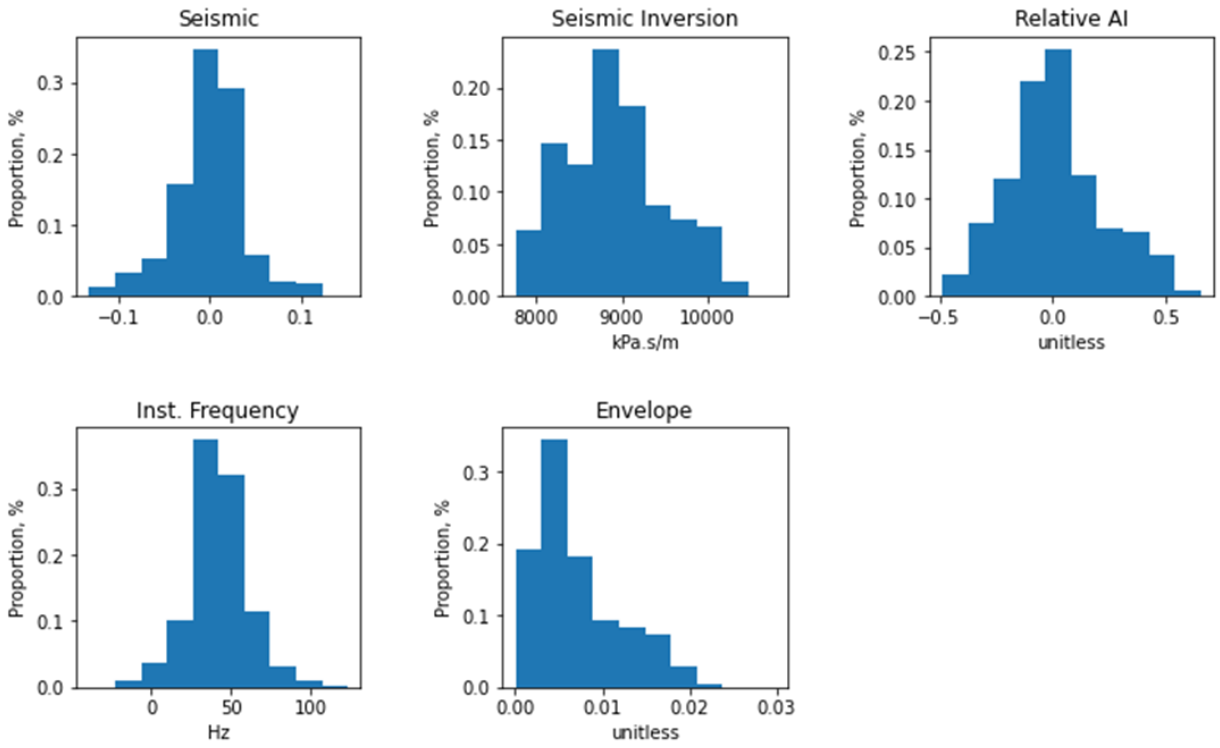

Figure 31. Distribution of seismic data and seismic attributes features in case 3.

A statistical summary of the features in case 3 given as boxplots shows that all features have outliers, but their values are within the physical boundaries (Figure 32).

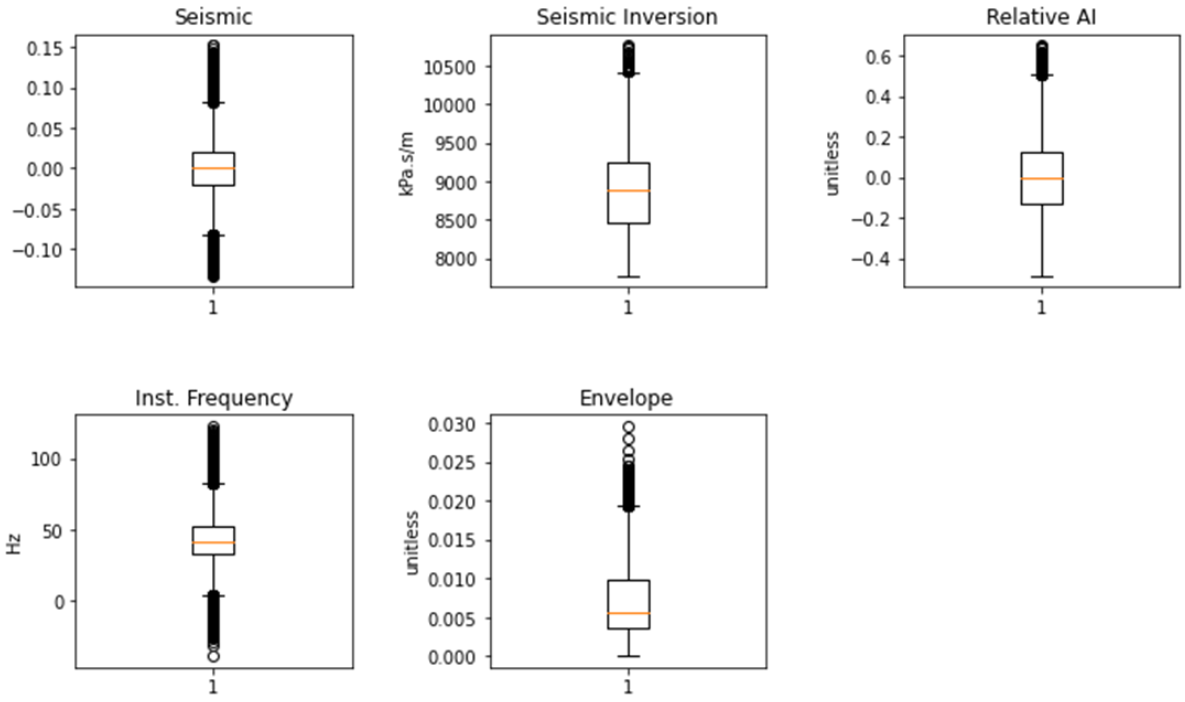

Figure 32. Boxplots of seismic and its attributes used for facies classification in case 3.

5.2.4 Case 4

Facies classification of case 4 is also made by utilizing five features together: seismic inversion, relative acoustic impedance, instantaneous frequency, and envelope. The distribution of these features is shown in Figure 33. Almost all features are skewed either to the left or right.

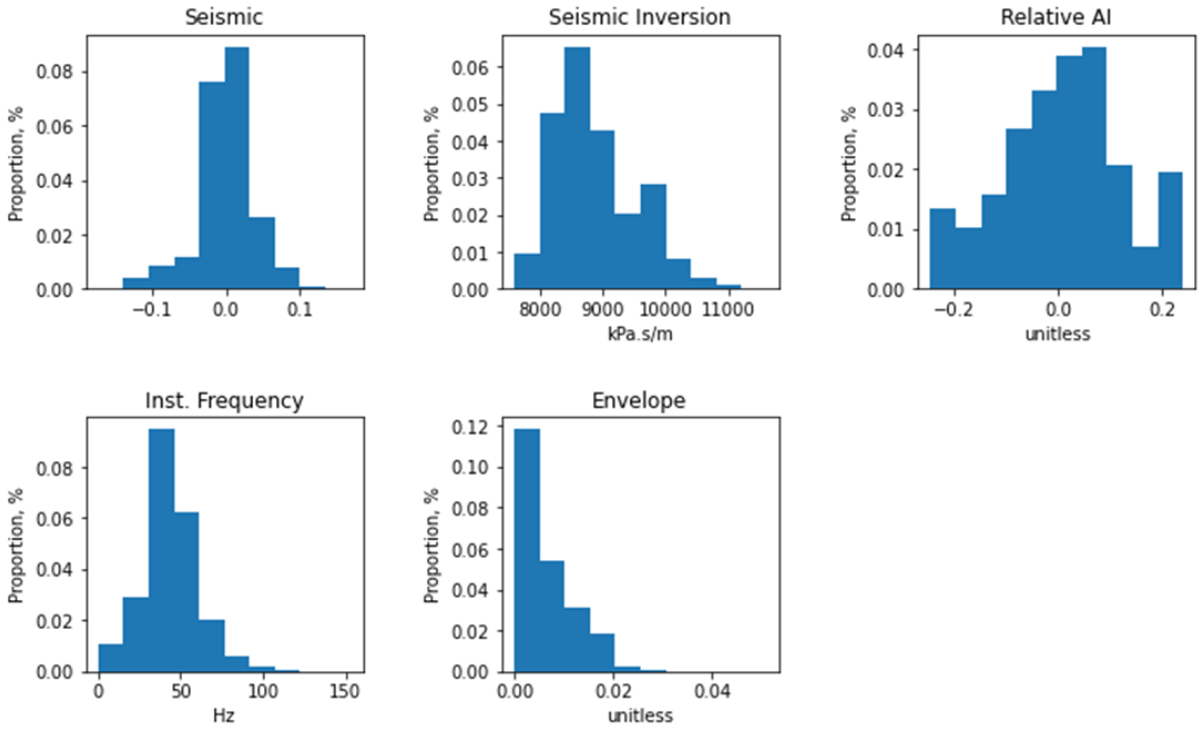

Figure 33. Distribution of seismic data and seismic attributes features in case 4.

The data analysis shows that the features used for facies classification have various ranges and measurement units. For example, the seismic range is between -0.2 and 0.2, while the seismic inversion range varies between 7500 and 11700. As mentioned earlier, different ML models are sensitive to the features' scale. Scaling can affect the models' performance if, for example, some features with a larger scale dominate over other features with a lower scale [28]. So, to avoid misclassification of facies, standardization was applied to all features. The boxplots show that some features have outliers, however, all of them are within their physical boundaries. The absence of the outliers that are outside the physical boundaries is explained by the use of synthetic dataset. However, in the real case, the statistical analysis might identify values that are beyond boundaries which can decrease the ML models performance.

Results

In this chapter, the results of facies classification from seismic, its attributes, and seismic inversion, for the four cases described before, are presented.

5.3 Results for case 1

As mentioned in the Methodology chapter, the analysis started with using three features: seismic, relative acoustic impedance, and seismic inversion. As mentioned earlier, the main goal of this part is to evaluate the role of the number of wells, frequency ranges, spectral noise, and additional features such as geologic time. For this reason, the analysis is done for the baseline models LR, KNN, SVM, RF, and NN, which means that hyperparameter tuning is not applied to these scenarios.

5.3.1 Baseline models overview

Before performing the analysis of the minimum number of wells necessary for facies classification, five baseline machine-learning models were built and tested for a random number of wells in order to choose the ML model that gives the highest global average F1-score. This model was later used for testing different numbers of wells. Figure 34 shows a comparison of five baseline models. Four out of five baseline models demonstrate almost the same F1-score; however, the best performance is observed for the Random Forest Classifier (RF). So, this model was further utilized as the main model when estimating the impact of the number of wells on facies classification. Logistic Regression is the least flexible method and has the lowest overall performance.

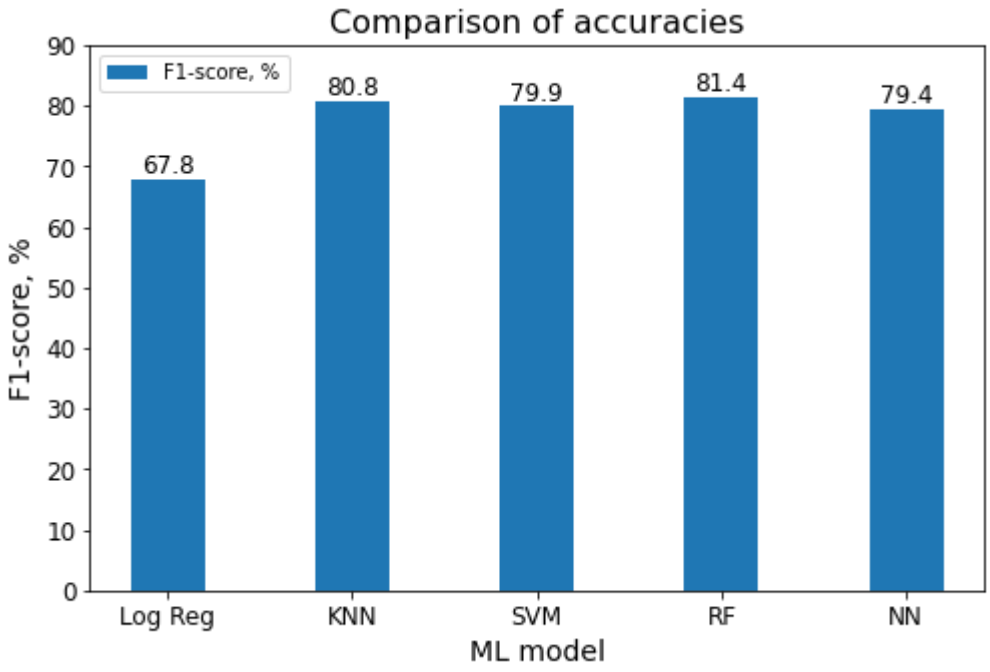

Figure 34. The overall F1-score of the baseline models for case 1.

5.3.2 Evaluating the impact of the number of wells

The estimation of the role of well numbers on facies prediction started for 30 wells and continued for 15, 10, 8, 6, 5, 4, 3, 2, and 1 well. It is assumed that the higher the number of wells used for building and training the model, the better the model's performance. As the Random Forest Classifier shows the best performance among the used ML models, this model was utilized for the analysis. The imbalanced proportion of the facies in the training and validation sets was overcome by incorporating class weighting which changes the weight of each class and assigns a higher weight to undersampled data.

As mentioned earlier, the minimum acceptable number of wells must give a threshold accuracy (overall F1-score) of 75%. Case 1 consists of 151 traces, so for 30 wells, every fifth or sixth trace is used as a well, and for 15 wells, every tenth or eleventh trace is used as a well. The model building, validation, and testing are based on three features: relative acoustic impedance, seismic, and seismic inversion. Figure 35 shows that the overall F1-score of at least 75% for the test set can be achieved by using only three wells.

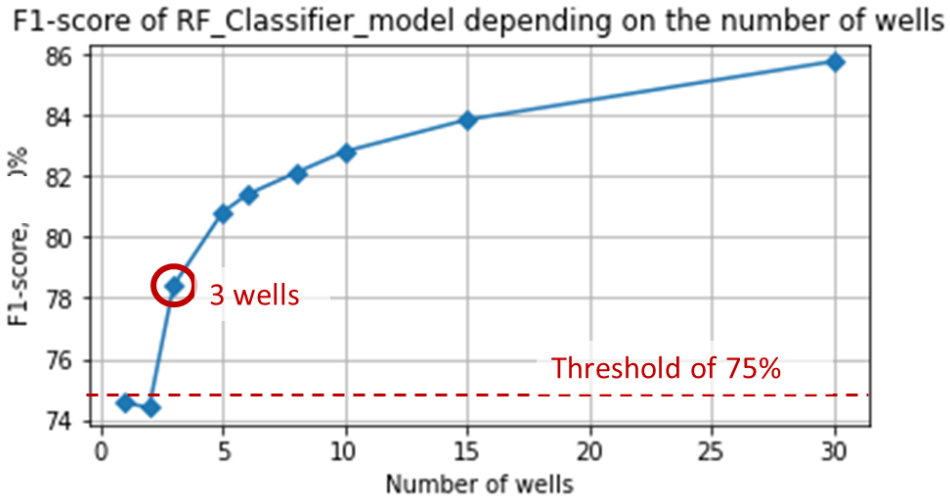

Figure 35. The impact of the number of wells on the model's F1-score for case 1.

The comparison of the model's performance, namely the facies prediction section and the difference map for 15, 5, and 3 wells, is shown in Figure 36. The results show that the number and location of the wells influence the accuracy, especially in the thin layers of the reservoir.

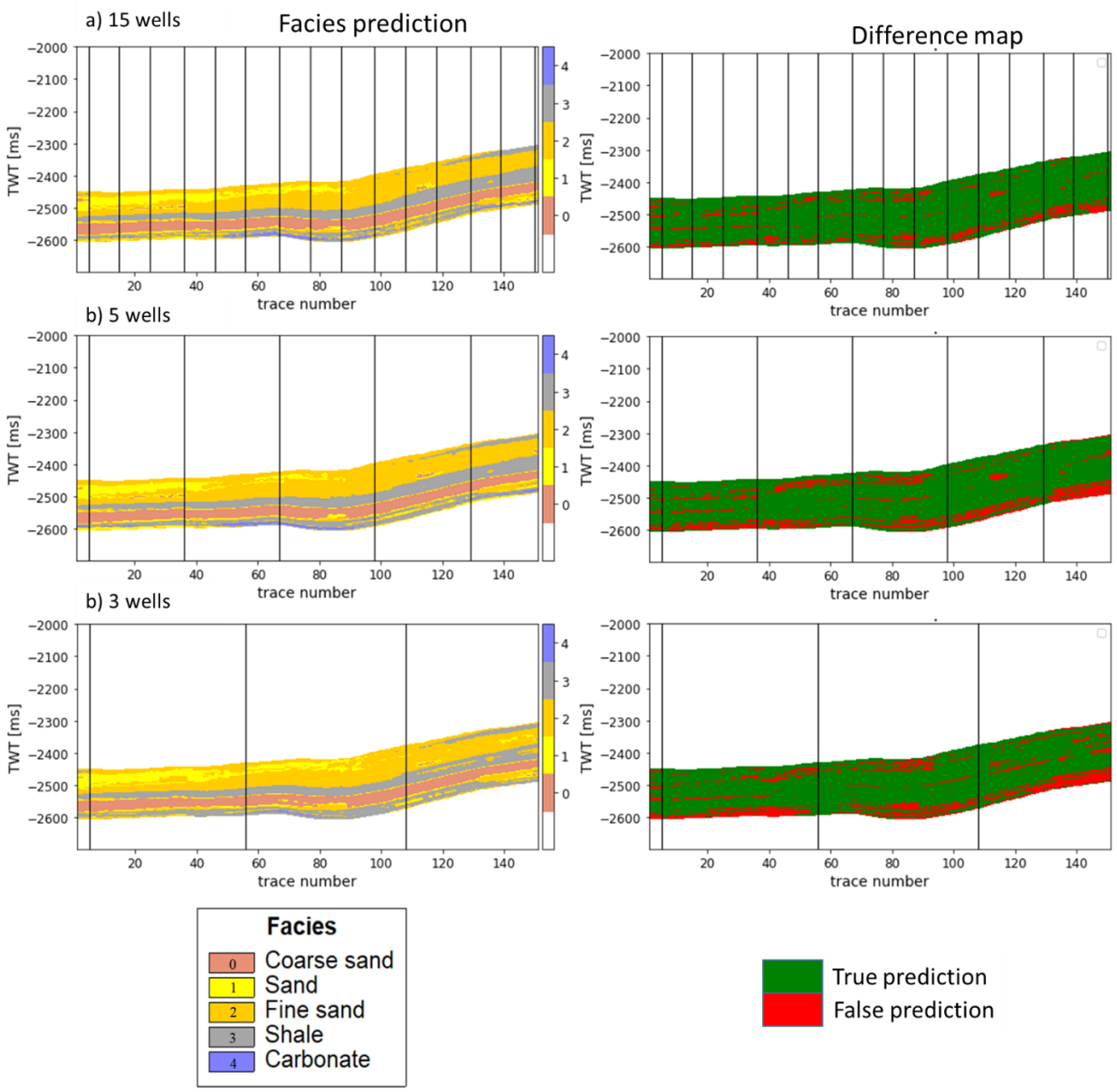

Figure 36. The facies prediction and difference map for case 1 when using: a) 15 wells, b) 5 wells, c) 3 wells. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right.

5.3.3 Evaluating the role of the seismic frequency range.

The performance of facies classification depends on seismic data derived using various frequencies: Ricker (25 Hz) and three Ormsby (10-60 Hz, 10-80 Hz, 10-100 Hz) wavelets. A comparison of the overall F1-score of the Random Forest Classifier model when using seismic features with different frequencies is shown in Figure 37. The comparison shows that the best overall accuracy (F1-score) is achieved when using seismic features derived with an Ormsby filter with a frequency range of 10-60 Hz.

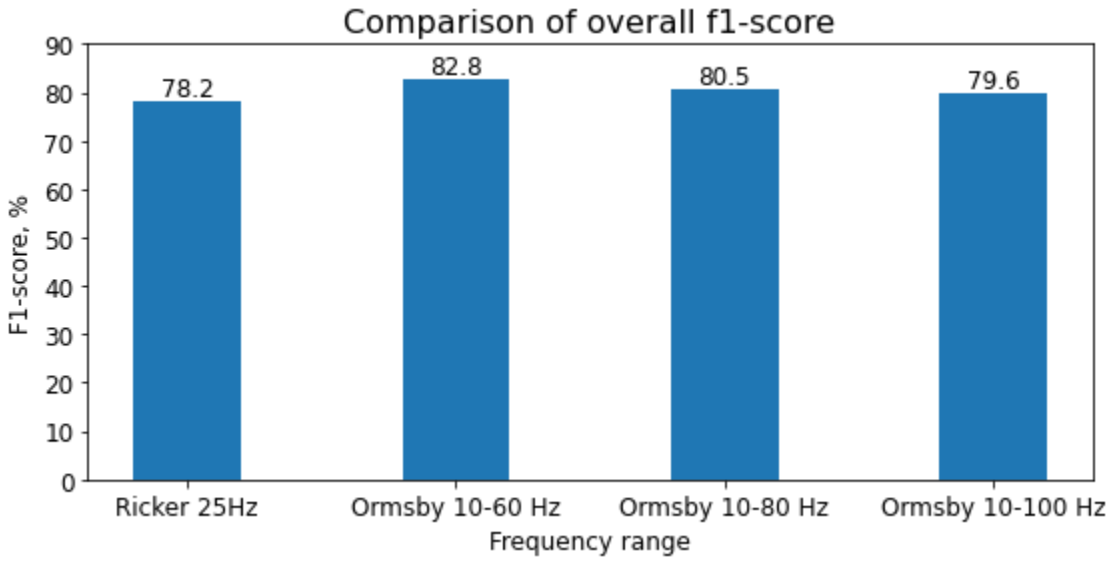

Figure 37. A comparison of the overall F1-score of facies prediction for the RF model when using seismic features with different frequencies for case 1.

The comparison shows that the best overall accuracy (F1-score) is achieved when using seismic data derived with using Ormsby filter with a frequency range of 10-60 Hz.

To identify what impact various frequencies have on each of the facies, the comparison of the F1-score of each facies by the RF Classifier is presented in Figure 38. The highest F1-score is achieved from seismic data (relative acoustic impedance, seismic, and seismic inversion) calculated by using the Ormsby filter with a frequency of 10-60 Hz for Coarse Sand, Fine Sand, and Shale, thereby following the trend of the overall f1-score. However, for the least presented facies, such as Sand and Carbonates, the highest F1-score is obtained for the Ricker wavelet with 25 Hz. The difference in the performance of these two facies for the four frequencies is quite

significant. In general, there is no direct relationship between increasing frequency and the facies prediction for thin layers.

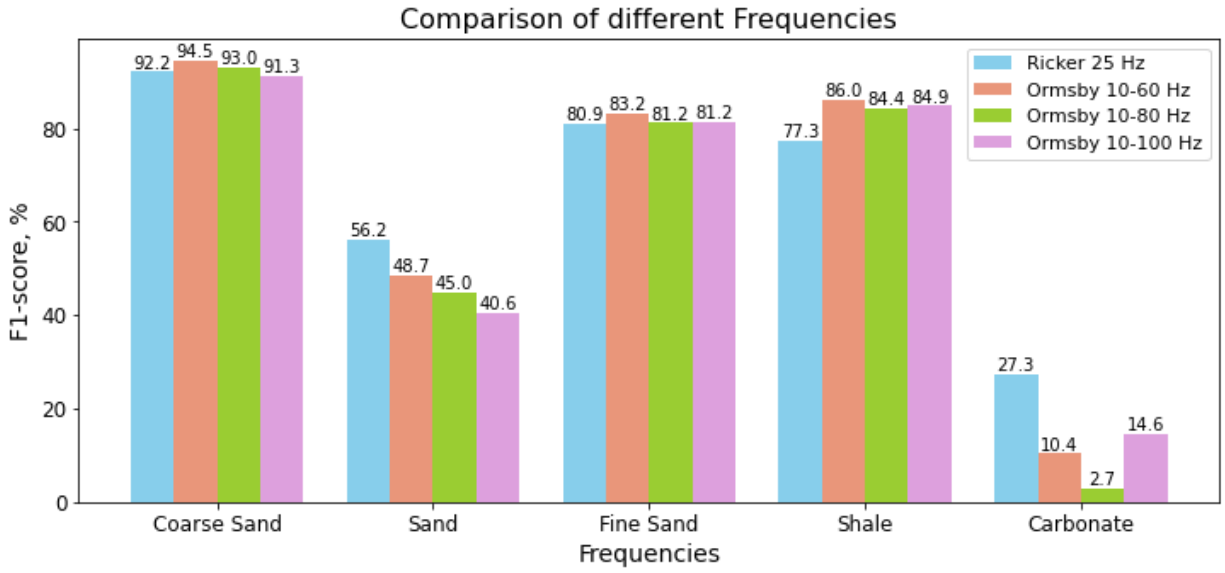

Figure 38. The F1-score for each of the facies from case 1 that were predicted from seismic data derived by using the following frequencies: Ricker (25Hz), and three Ormsby (10-60Hz, 10-80Hz, 10-100Hz) wavelets.

The 2D section of facies prediction and the difference map for the tested frequencies is shown in Figure 39.

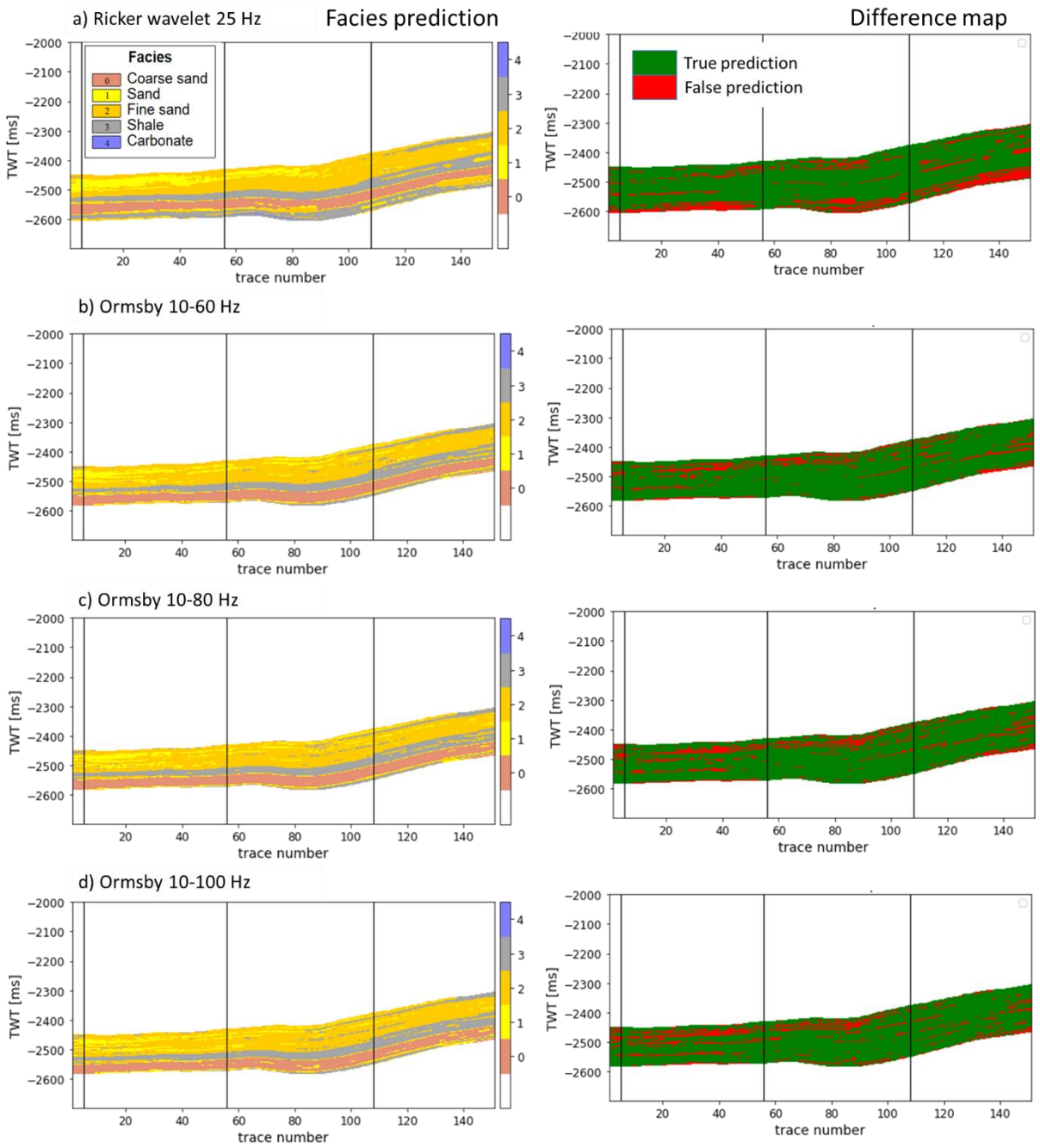

Figure 39. The facies prediction and difference map of case 1 when using seismic features derived from: a) Ricker wavelet 25 Hz, b) Ormsby filter with frequency range 10-60 Hz, c) Ormsby filter with frequency range 10-80 Hz, d) Ormsby filter with frequency range 10-100 Hz. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right side.

5.3.4 Evaluating the role of using Spectral Decomposition

Including an additional feature such as spectral decomposition can improve the prediction accuracy for thin layers such as carbonates. To confirm this, three spectral decompositions (30 Hz, 60 Hz, 90 Hz) are used as additional features for facies prediction, together with the features (seismic, relative acoustic impedance and seismic inversion) derived with the 25 Hz Ricker wavelet. The results are shown in Figure 40.

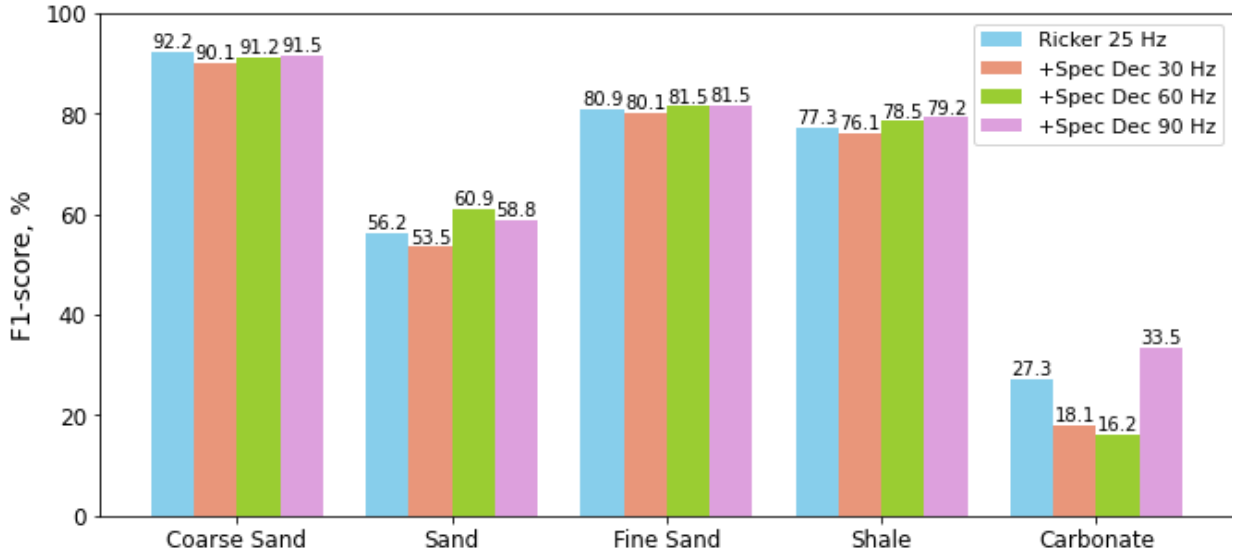

Figure 40. F1-score of each facies in case 1, as predicted from seismic data derived by using the 25 Hz Ricker wavelet, and spectral decomposition (30 Hz, 60 Hz, 90 Hz) as additional features added one at a time.

The usage of spectral decomposition of 30 Hz, 60 Hz, and 90 Hz has almost no effect on the F1-score for coarse sand, fine sand, and shale. However, the spectral decomposition of 90 Hz and 60 Hz slightly improves the facies prediction of carbonates and shale, respectively.

As demonstrated in the Data Analysis chapter, the prevailing facies in case 1 are fine sand, shale, and coarse sand, while carbonates and sand are less common. Accordingly, the lowest performance is gained for carbonates deposits. A similar performance is observed for all ML methods used in this thesis and can be explained by the lack of data used for training the models since the thin layer of carbonates is penetrated only by one out of three wells. In contrast, the F1-score for coarse sand, fine sand, and shale is more than 76%. The facies 2D section of case 1 and the difference map for Spectral Decomposition 30 Hz, 60 Hz, and 90 Hz are shown in Figure 41

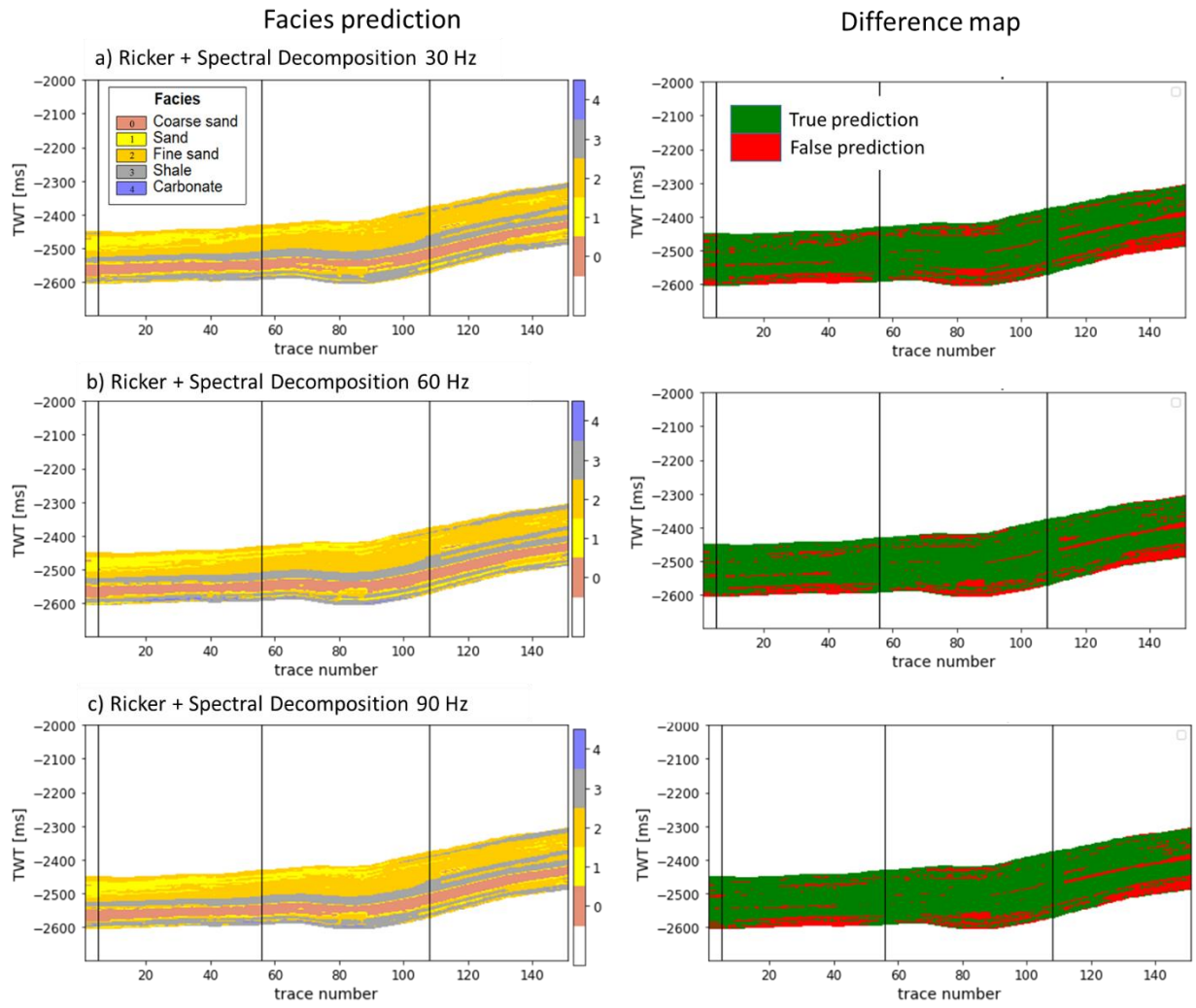

Figure 41. The facies prediction and difference map of case 1 when using the additional feature: a) Spectral Decomposition 30 Hz, b) Spectral Decomposition 60 Hz, c) Spectral Decomposition 90 Hz. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right side.

5.3.5 Evaluating the impact of noise for case 1

Figure 42 shows the overall F1-score of facies classification for case 1, calculated by the Random Forest Classifier model, from data with and without spectral noise.

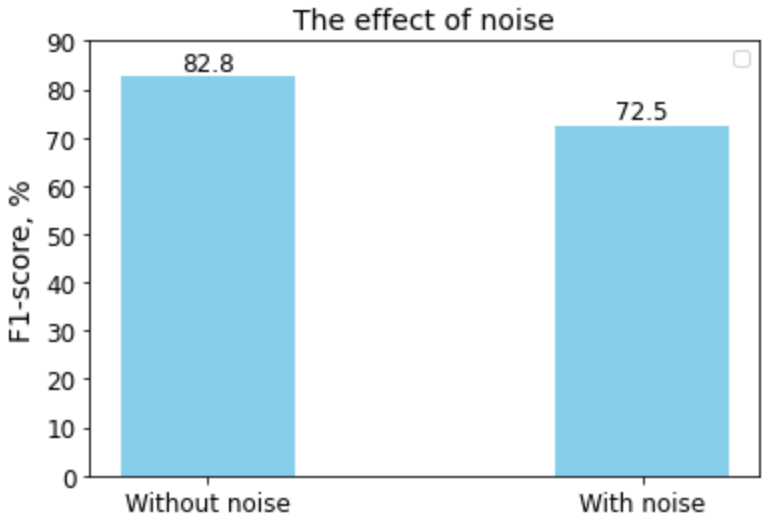

Figure 42. F1-score comparison for facies prediction when using data with and without spectral noise for case 1.

Features contaminated with spectral noise give overall accuracy (F1-score) of 72.5%, while the performance for the facies predicted from seismic attributes and seismic inversion without noise, is 82.8%. In general, the results of the ML prediction are expected. The comparison of the facies derived from features with and without noise is given in Figure 43.

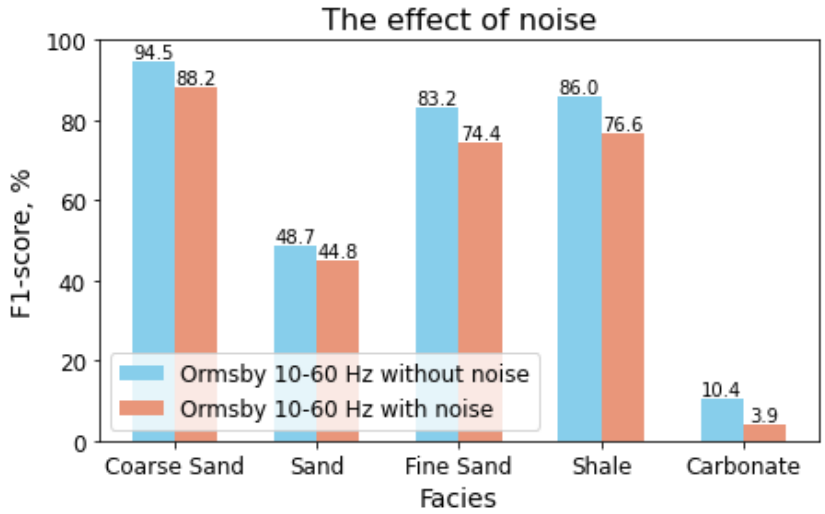

Figure 43. The F1-score of facies prediction from data with and without noise for case 1. Seismic data was constructed using an Ormsby 10-60 Hz wavelet.

Spectral noise negatively affects the prediction of every facies. So, when dealing with noisy data, one should expect a decrease in the efficiency of facies prediction. The 2D facies section and difference map for seismic data without and with noise are shown in Figure 44.

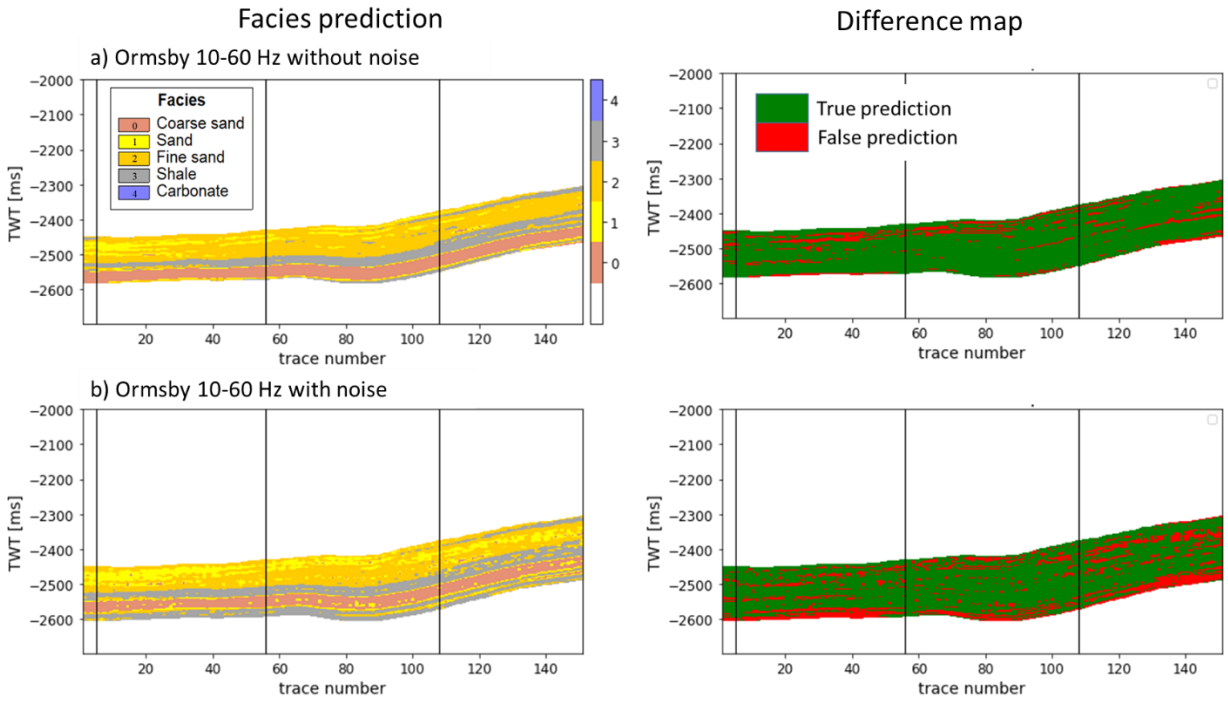

Figure 44. The facies prediction and difference map of case 1 when using an Orsmby 10-60 Hz wavelet, and features: a) without noise, b) with noise. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right.

5.3.6 Exploration and comparison of additional features

5.3.6.1 Instantaneous frequency, envelope

As discussed in the previous section, seismic inversion is a computationally expensive and not straightforward method in contrast to seismic attributes. The analysis of the facies prediction from the ML model based on seismic inversion together with seismic on one hand, and seismic attributes (relative acoustic impedance, envelope, and instantaneous frequency) and seismic on the other hand, is shown in Figure 45. All attributes are derived from the Ormsby filter with a frequency range of 10-60 Hz.

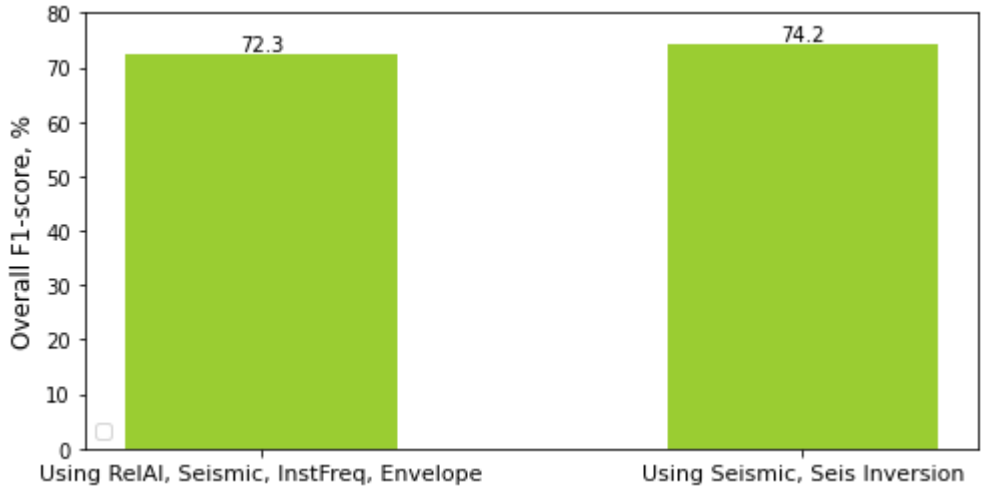

Figure 45. The overall F1-score comparison for facies prediction of case 1 when using seismic and seismic inversion versus relative acoustic impedance, seismic, instantaneous frequency, and envelope.

The facies performance is slightly better when using seismic inversion and seismic than when using seismic attributes and seismic. However, the difference is less than 2%. Figure 46 shows the comparison of the F1-score for each facies.

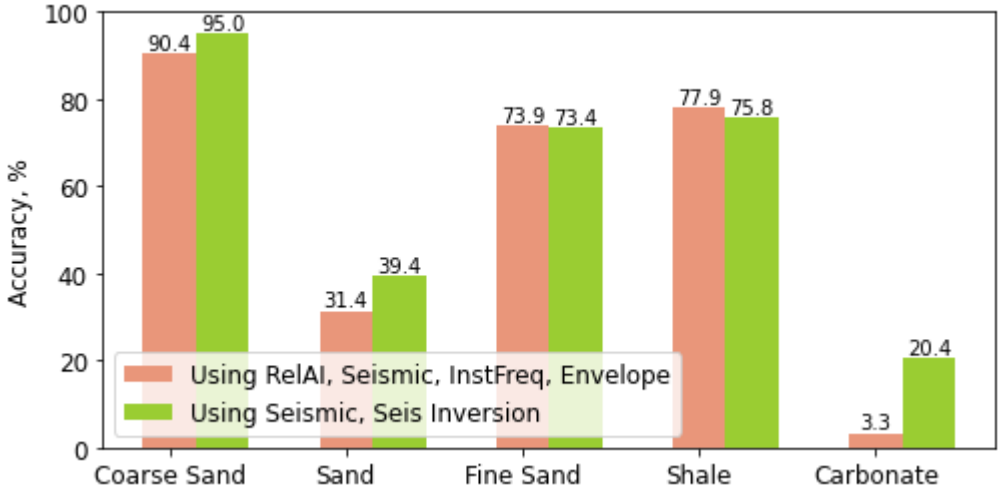

Figure 46. The F1-score of facies prediction for case 1 from seismic and seismic inversion on the one hand, and relative acoustic impedance, seismic, instantaneous frequency, and envelope on the other hand.

The most significant difference, of 17.1% in facies prediction, takes place for carbonates. The F1-score is only 3.3% for seismic and seismic attributes, and 20.4% for seismic and seismic inversion. The F1-score for other facies is almost the same, with the exception of sand prediction, where the prediction by seismic inversion is higher by 8%. Seismic inversion is a critical feature to correctly identify the carbonates.

So, the main conclusion is that it is possible to predict facies using seismic attributes. However, seismic inversion is the most important feature for increasing the performance of the prediction. Figure 47 shows the comparison of feature importance for the RF Classifier model. The bar plot shows that seismic inversion and relative acoustic impedance are the features that contribute more to the accuracy of the prediction, with scores of 68% and 59%, respectively. Envelope and instantaneous frequency contribute to facies prediction almost equally and are around 20%.

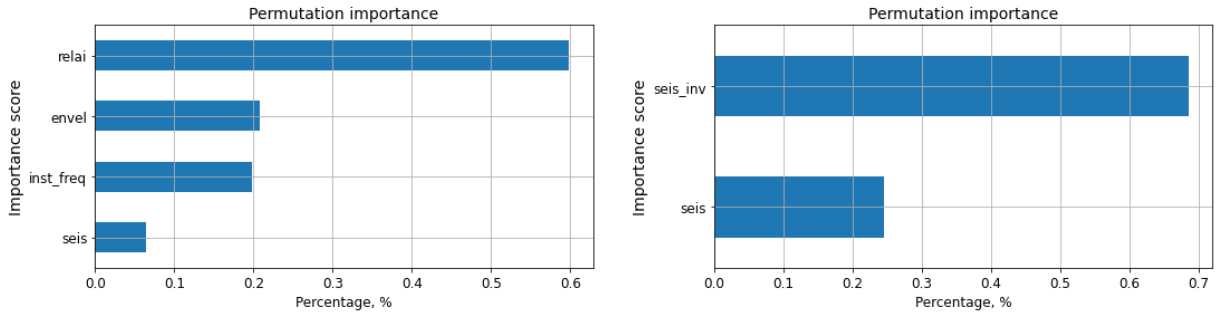

Figure 47. Features importance for facies prediction of case 1, for seismic and seismic attributes (left), or seismic and seismic inversion (right) features and the RF model.

5.3.6.2 Geological Time

Incorporating information about geological time as an additional feature improves the facies prediction. The comparison of the F1-score for the Random Forest Classifier shows that using geological time together with seismic attributes gives almost the same accuracy as using seismic inversion (Figure 48). The use of geological time improves the prediction accuracy of sand, fine sand, and shale. However, for the thinnest layer of carbonates, the accuracy does not outperform the F1-score when using seismic inversion which is equal to 20.4%.

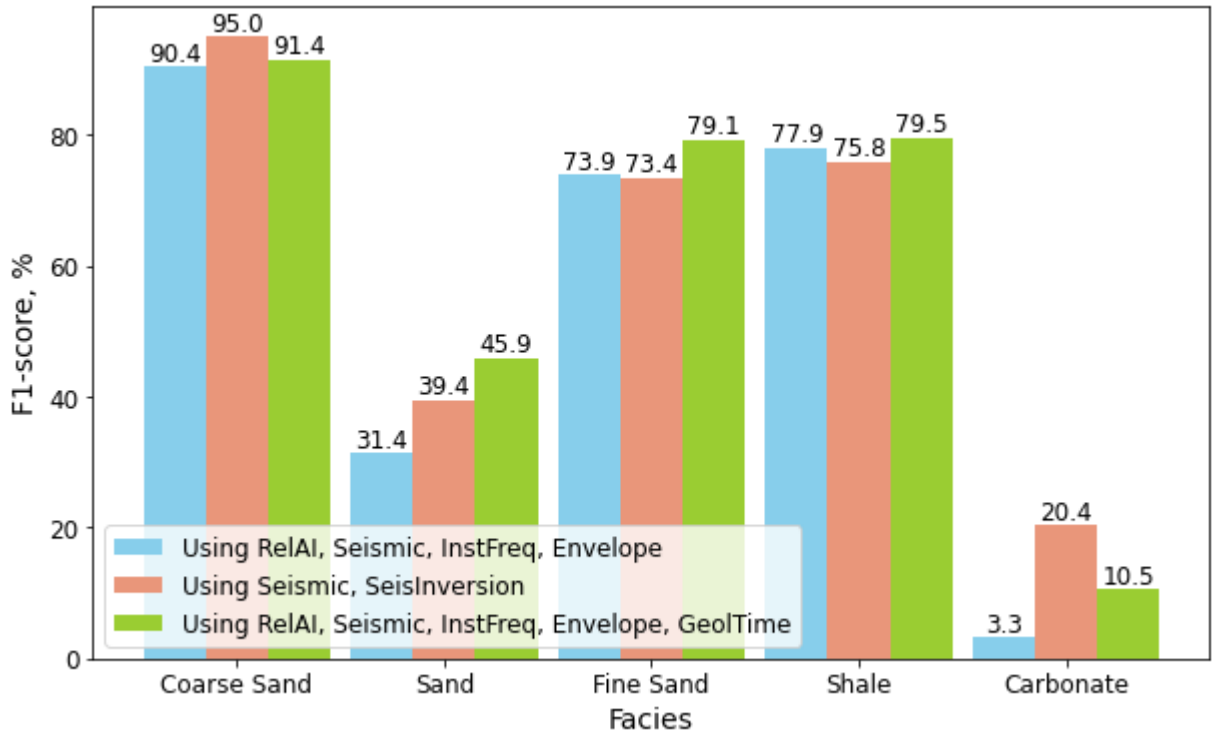

Figure 48. Comparison of the F1-score of RF classifier model for facies prediction of case 1 based on seismic and seismic attributes (blue), seismic and seismic inversion (brown), and seismic, seismic attributes, and geological time (green).

The comparison of the feature importance for these cases shows that geological time plays an important role in improving the accuracy of the facies classification. The feature importance for the Random Forest classifier with seismic, seismic attributes, and geological time is shown in Figure 49. The geological time contributes 23.5% to the accuracy of the facies prediction because it gives extra information about the geometry of the reservoir. Figure 50 shows the 2D section of facies prediction and the difference map, for RF models with seismic and seismic inversion (a), and seismic, seismic attributes, and geological time (b).

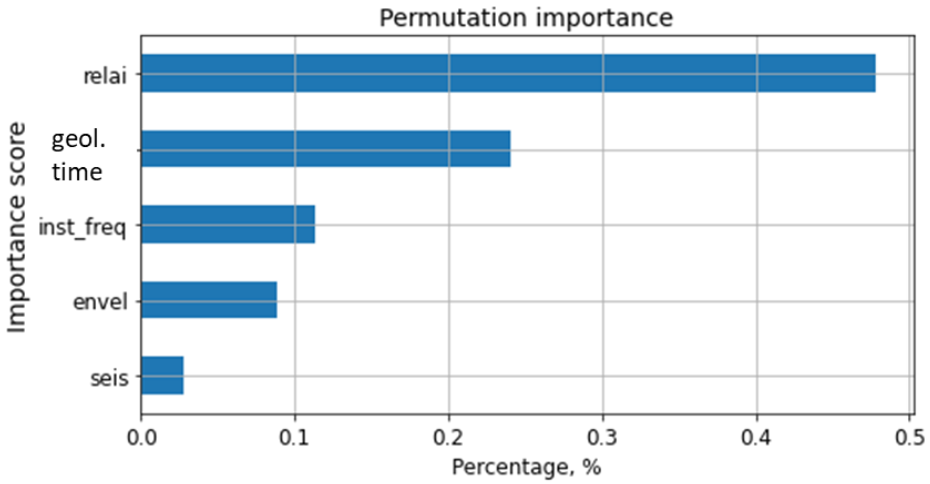

Figure 49. The permutation importance of RF model for facies classification of case 1, and seismic, seismic attributes, and geological time as features.

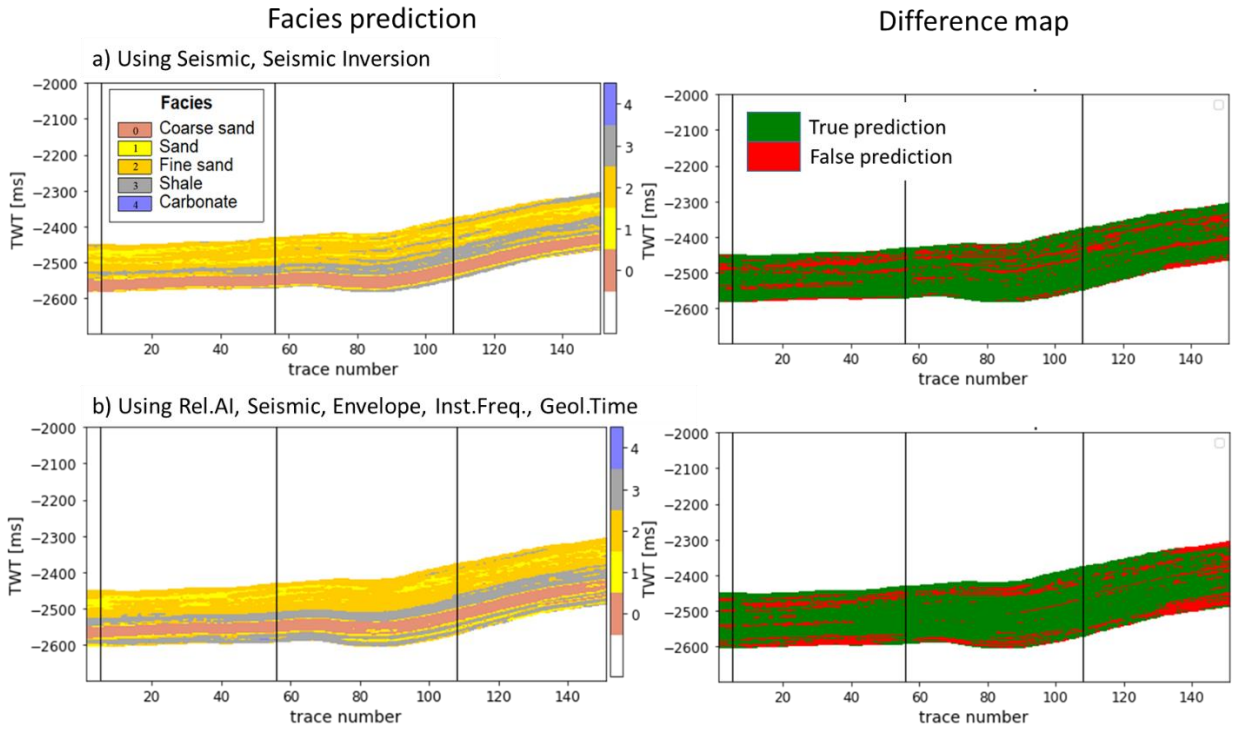

Figure 50. The facies prediction and difference map of RF model for case 1 when using the features: a) seismic and seismic inversion, and b) seismic, seismic attributes, and geological time. The wells are shown as black lines. The facies prediction is on the left side, and the difference between the true facies and predicted facies is on the right.

5.4 Results for case 2

For case 2, the same analysis is performed as for case 1: evaluating the impact of using different numbers of wells, frequency ranges, and spectral noise, and comparing the facies prediction from seismic inversion and seismic attributes. The analysis is done using the baseline ML model that gives the best accuracy among other models for the random number of wells. As for case 1, hyperparameter tuning of the ML models is not performed.

5.4.1 Baseline models overview

In this part, five baseline ML models are built and tested for a random number of wells: LR, KNN, SVM, RF, and NN. The model with the best performance is further utilized for evaluating the impact of various numbers of wells. Figure 51 shows that the best overall F1-score of 66.1% is achieved by the Random Forest Classifier. The SVM has almost the same accuracy, but the RF is used as the base ML model for further analysis.

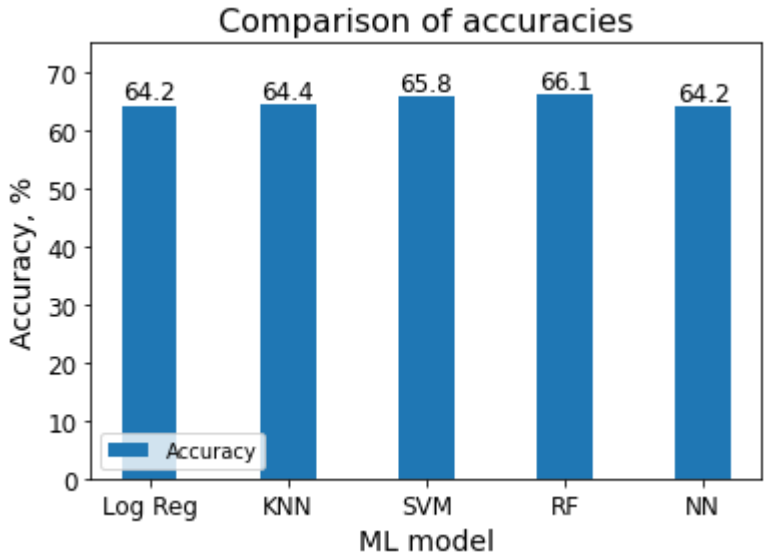

Figure 51. The overall F1-score of the baseline models for Case 2.

5.4.2 Evaluating the impact of the number of wells

The role of the number of wells on facies prediction is estimated for 30, 15, 10, 8, 6, 5, 4, 3, 2, and 1 well. The analysis shows that the number of wells has an impact on the overall accuracy of the facies prediction. However, in this case, the threshold accuracy of 75% is not achieved for any number of wells (Figure 52). This means that when dealing with heterogeneous, thin facies deposits such as those of the upper reservoir zone, one should expect lower facies prediction accuracy than with homogeneous, thick facies deposits, as those of the lower reservoir zone in case 1.

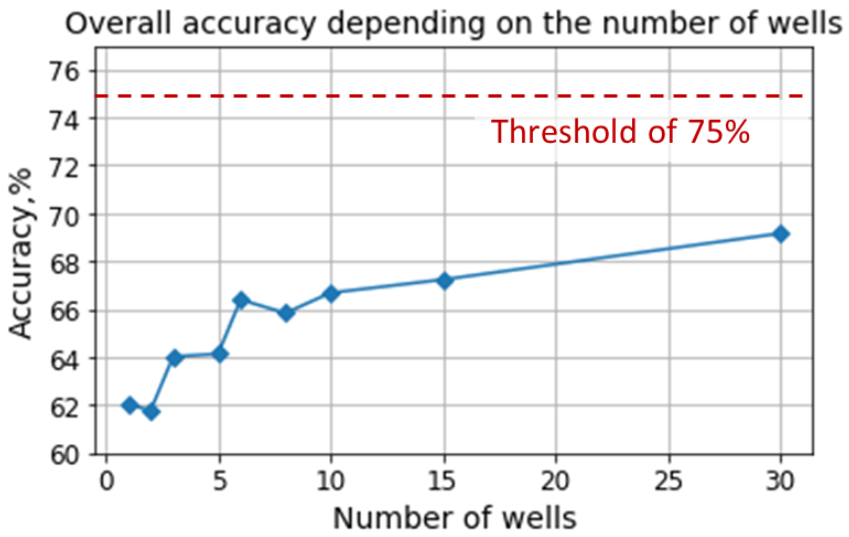

Figure 52. The overall accuracy (F1-score) of facies prediction of case 2 by the RF model with different number of wells.

It is assumed that the drilled wells penetrate the entire thickness of the reservoir, including the upper and lower zones. The necessary number of wells for the lower reservoir zone in case 1 was determined to be three (section 5.1.2). Therefore, for the upper reservoir zone and case 2, the same number of wells is used in further analyses. Figure 53 shows the 2D section with facies prediction and difference map for 15, 6, and 3 wells. The least accurate facies prediction is located on the right (eastern) side and top left (western) side of the reservoir (Figure 53).

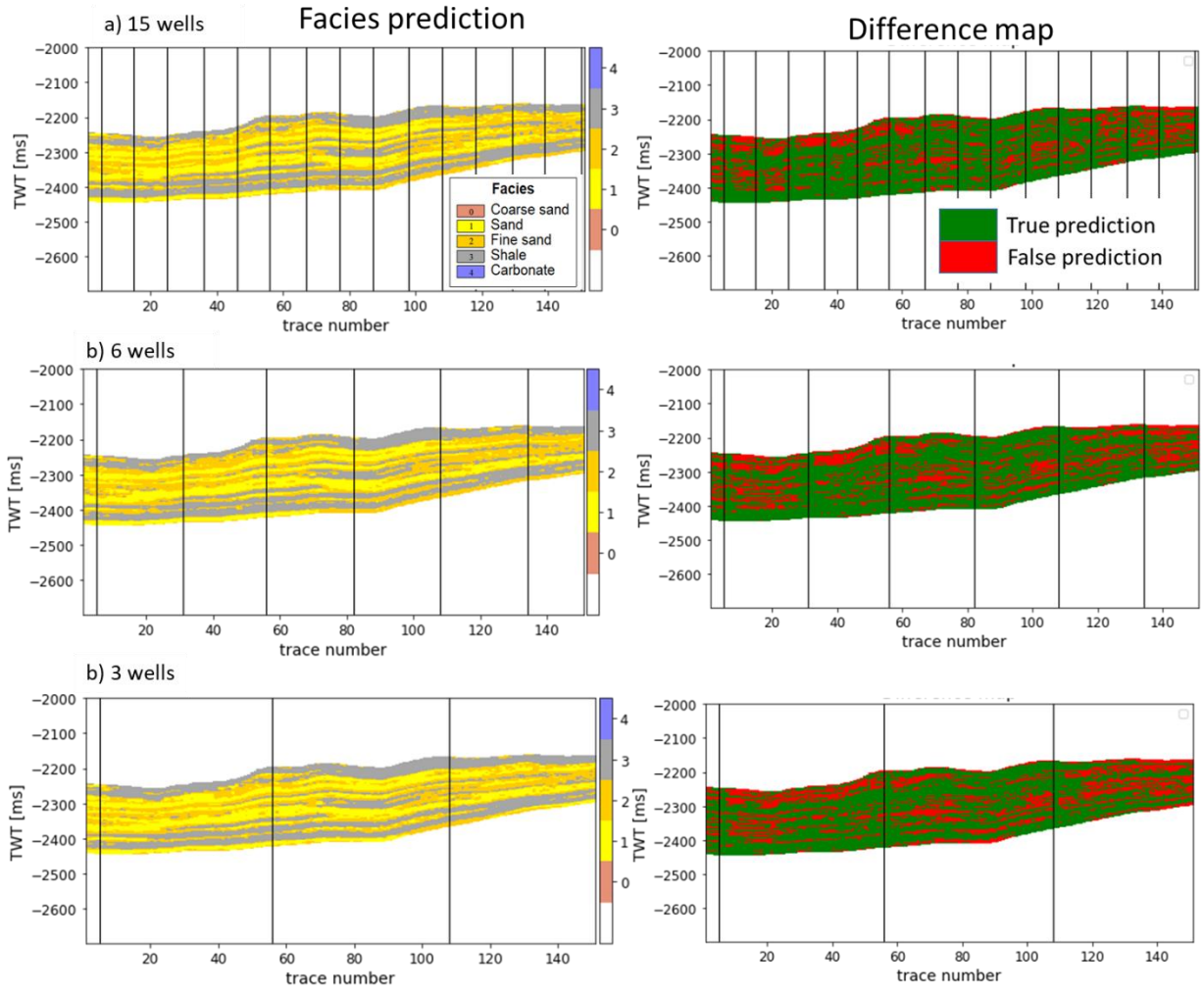

Figure 53. The facies prediction and difference map of case 2 when using: a) 15 wells, b) 6 wells, c) 3 wells. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right.

5.4.3 Evaluating the role of seismic frequency range.

The facies prediction performance depends not only on the features used but also on how these features were derived. The evaluation of the impact of seismic inversion and seismic attributes obtained from four different frequencies, Ricker (25 Hz) and three Ormsby (10-60 Hz, 10-80 Hz, 10-100 Hz) wavelets, is shown in Figure 54. This figure shows that the best overall accuracy is achieved when using features derived with the Ormsby filter with a frequency range of 10-100 Hz.

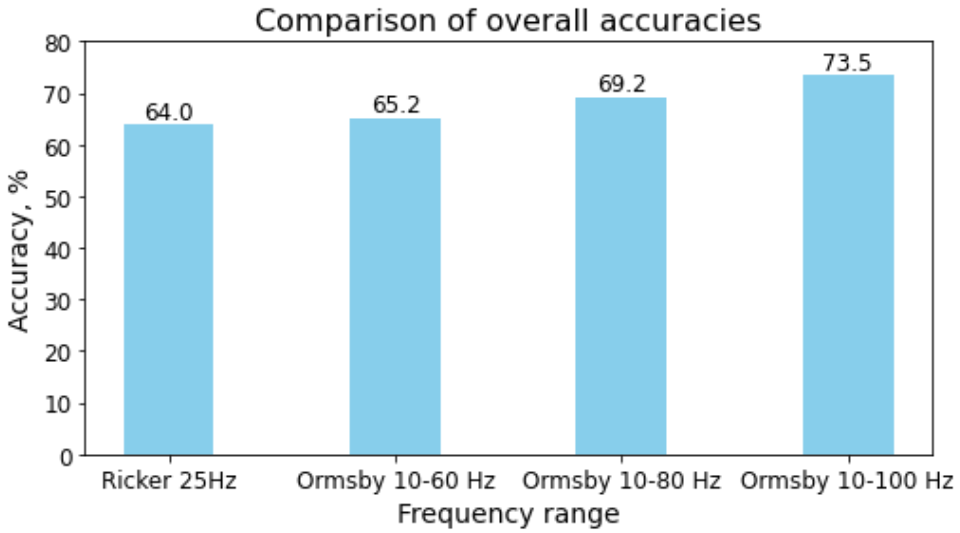

Figure 54. A comparison of the overall accuracy (F1-score) of facies prediction of case 2 for the RF model when using features with different frequencies.

To identify what impact different frequencies have on each of the facies, the comparison of the F1-score for each facies for the RF Classifier is presented in Figure 55. The highest F1-score is achieved from seismic data (relative acoustic impedance, seismic, and seismic inversion) calculated by using the Ormsby filter with a frequency of 10-100 Hz for all facies: coarse sand, fine sand, sand, and shale. Moreover, the bar chart shows that the higher the frequency used for the features, the better the facies prediction performance, with an insignificant difference for the coarse sand. So, for reservoirs containing thin facies, the higher frequencies give more robust results.

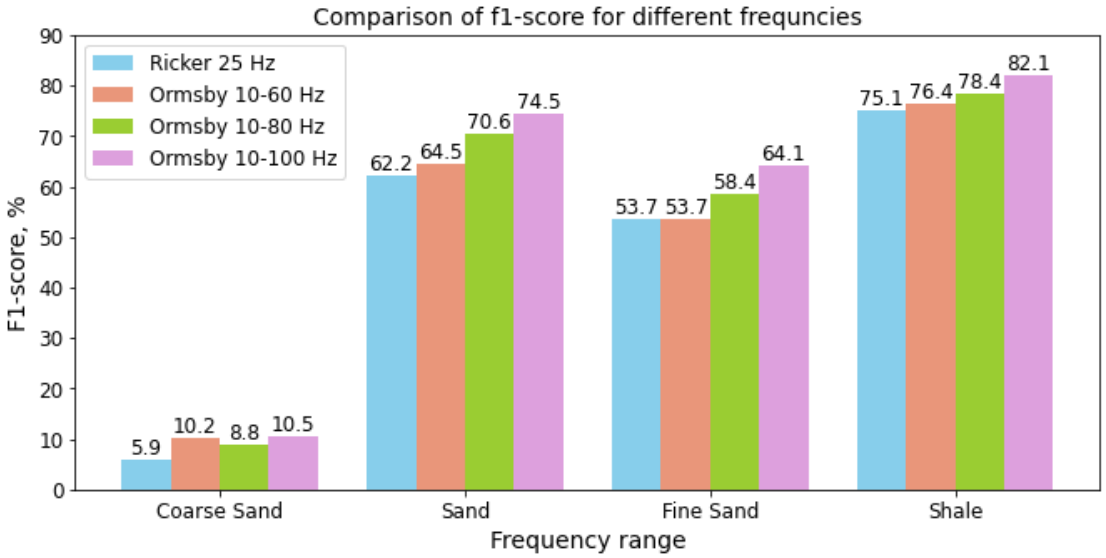

Figure 55. The F1-score for each of the facies of case 2 that were predicted by the RF model from seismic data derived by using the following frequencies: Ricker (25Hz), and three Ormsby (10-60Hz, 10-80Hz, 10-100Hz).

The 2D section of facies prediction and the difference map for the different tested frequencies is shown in Figure 56.

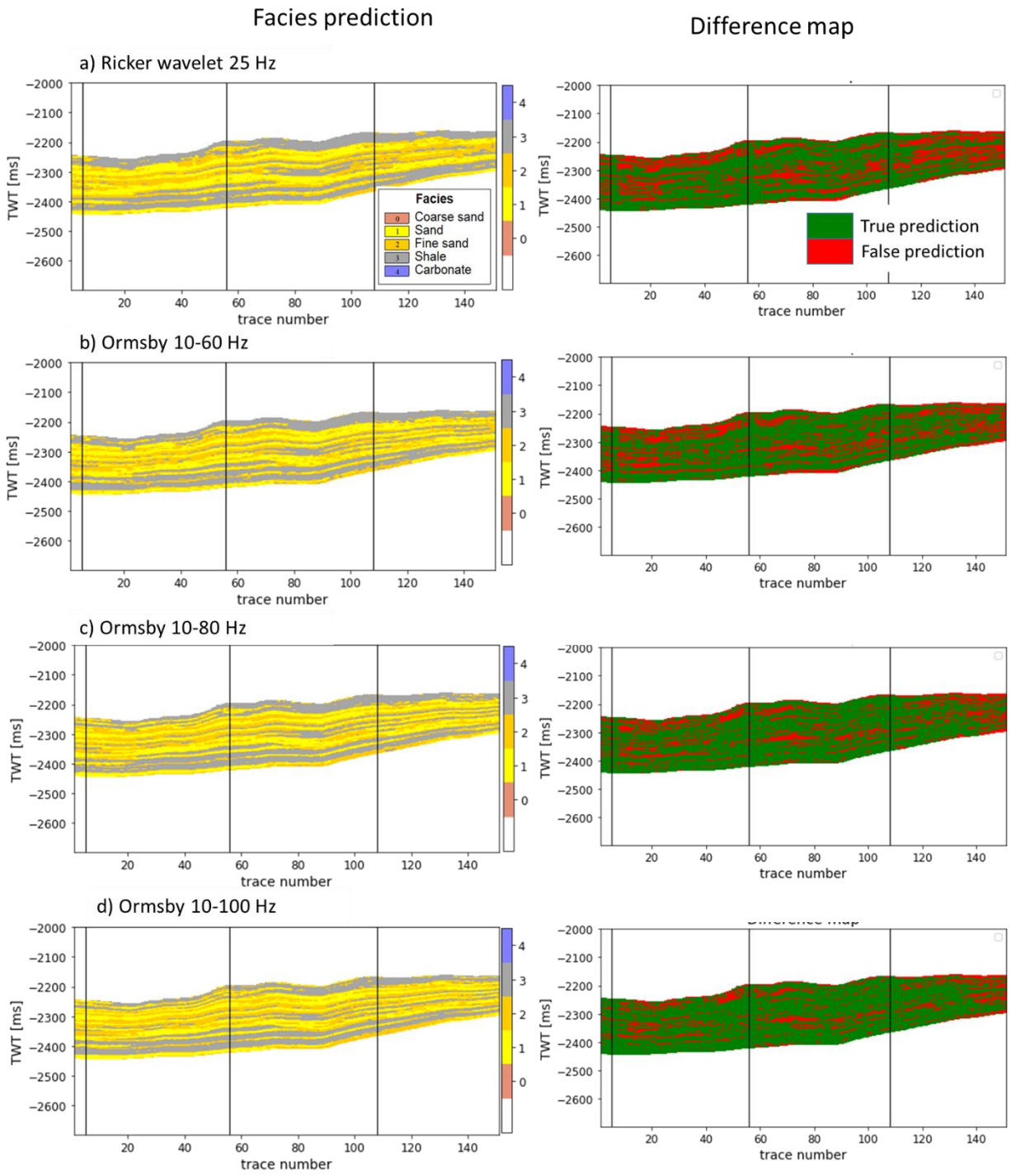

Figure 56. The facies prediction and difference map of case 2 when using features derived from: a) Ricker wavelet 25 Hz, b) Ormsby filter with frequency range 10-60 Hz, c) Ormsby filter with frequency range 10-80 Hz, and d) Ormsby filter with frequency range 10-100 Hz. The wells are shown as black lines. The facies prediction by RF model is on the left side, and the difference between the true facies and predicted facies is on the right.

5.4.4 Evaluating the role of using Spectral Decomposition

Incorporating spectral decomposition as an additional feature can also increase the ML model's performance for heterogeneous reservoirs. For case 2, three spectral decompositions (30 Hz, 60 Hz, and 90 Hz) are used additionally, together with seismic, relative acoustic impedance, and seismic inversion derived with the use of Ricker wavelet. Figure 57 shows the comparison of the overall accuracy of the RF model when using spectral decomposition.

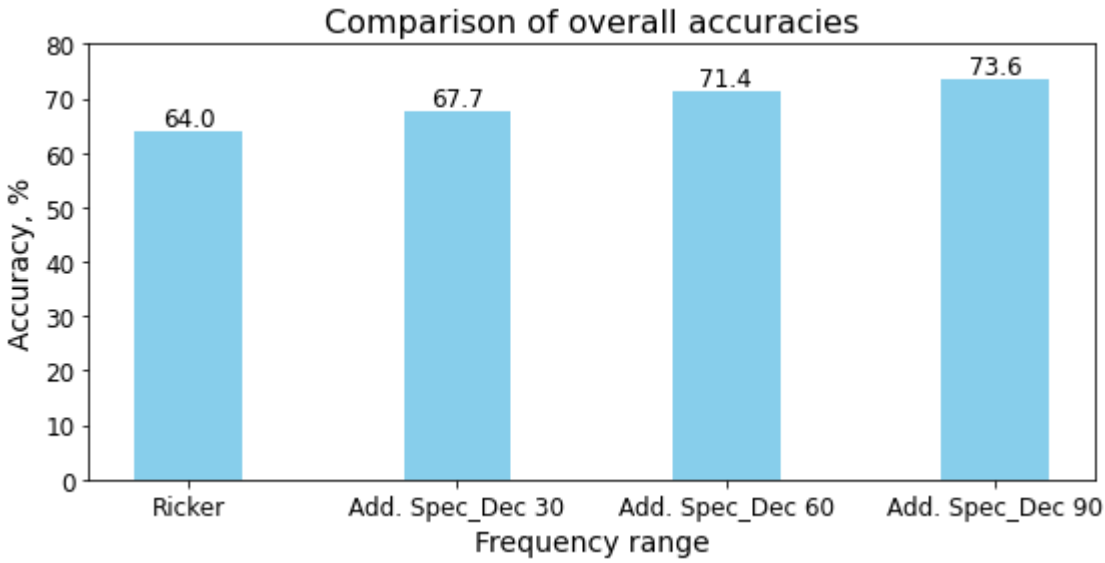

Figure 57. A comparison of the overall accuracy (global F1-score) of facies prediction in case 2 for the RF model when using spectral decomposition (30 Hz, 60 Hz, 90 Hz) in addition to seismic, seismic inversion, and relative acoustic impedance derived using a Ricker wavelet with 25 Hz.

The best overall accuracy of 73.6% is achieved when using a spectral decomposition of 90 Hz. To check the assumption that spectral decomposition improves the predictability of thin-layered facies, the comparison of the F1-score for each facies is shown in Figure 58.

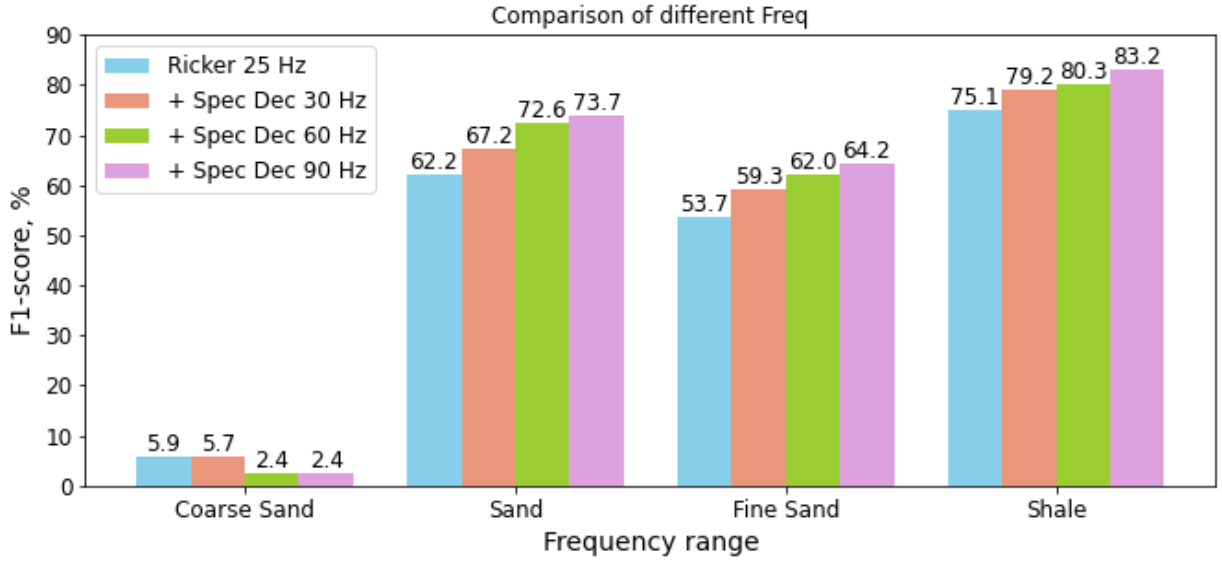

Figure 58. The F1-score for each of the facies of case 2 that were predicted by the RF model from seismic data derived by using the Ricker 25 Hz wavelet and additional feature spectral decomposition (30 Hz, 60 Hz, and 90 Hz).

The usage of spectral decomposition of 30 Hz, 60 Hz, and 90 Hz improves the F1-score for sand, fine sand, and shale. The higher the frequency of spectral decomposition, the better the facies prediction performance, except for coarse sand, for which the best F1-score is achieved when using spectral decomposition of 30 Hz. The lack of data in the coarse sand in case 2 explains this. The predicted facies 2D section and the difference map for spectral decomposition 30 Hz, 60 Hz, and 90 Hz is shown in Figure 59.

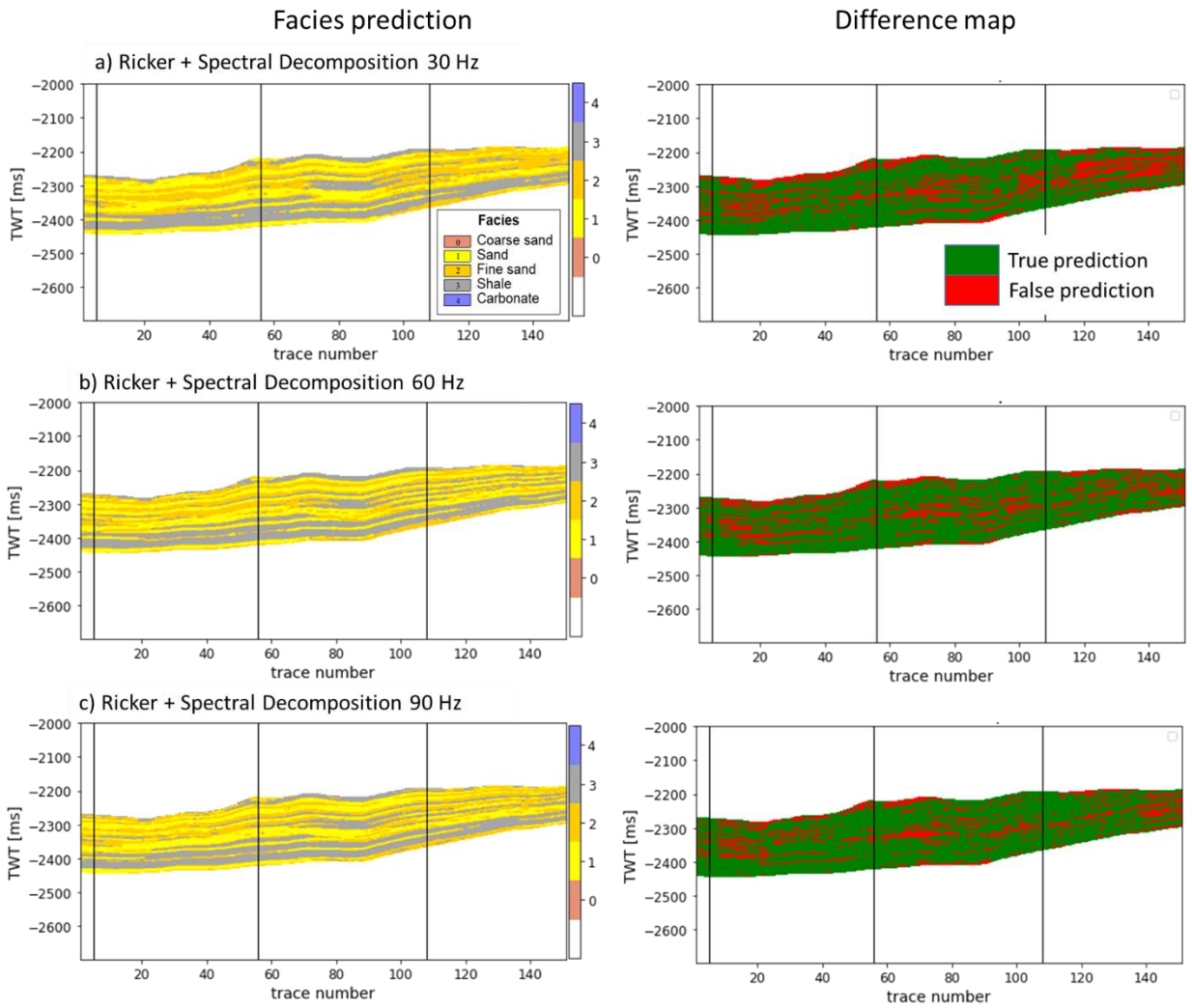

Figure 59. The facies prediction and difference map of case 2 when using the additional feature: a) spectral decomposition 30 Hz, b) spectral decomposition 60 Hz, and c) spectral decomposition 90 Hz. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right.

5.4.5 Evaluating the impact of noise

Figure 60 shows the overall F1-score (accuracy) of facies classification for case 2, calculated by the Random Forest classifier model, from data derived with the use of an Ormsby filter with a frequency range of 10-100 Hz, with and without spectral noise.

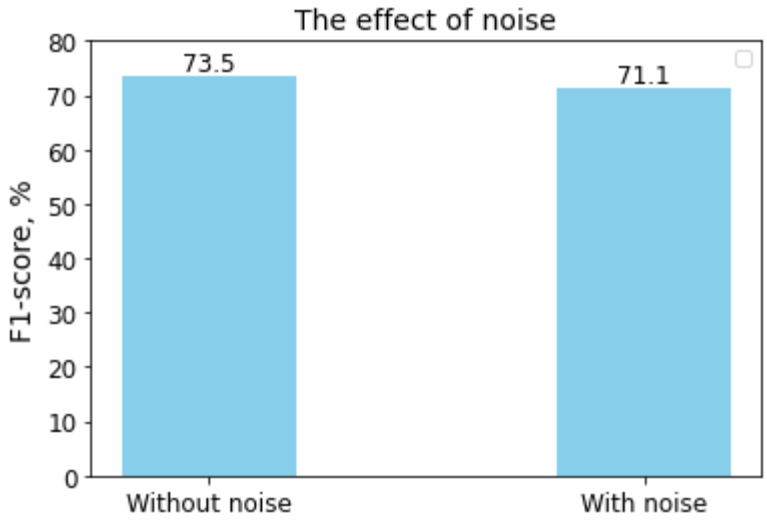

Figure 60. The overall accuracy (global F1-score) comparison for RF facies prediction when using data with and without spectral noise for case 2.

Features contaminated with spectral noise give an overall accuracy of 71.1%, while the performance for the facies predicted from seismic attributes, inversion without noise, is 73.5%. The comparison of each facies derived from features with and without noise is given in Figure 61. Spectral noise decreases the facies prediction performance. For example, the RF model fails when predicting coarse sand on noisy data. However, the difference for other facies is not critical. So, when dealing with noisy data, one should expect a slight decrease in the efficiency of facies prediction. The 2D facies section and difference map are shown in Figure 62.

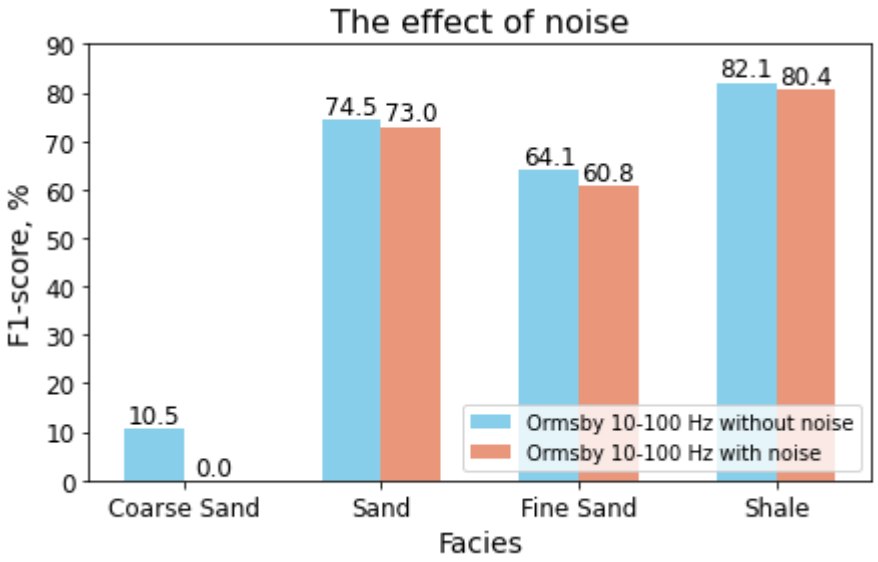

Figure 61. The F1-score of RF facies prediction from data with and without noise for case 2.

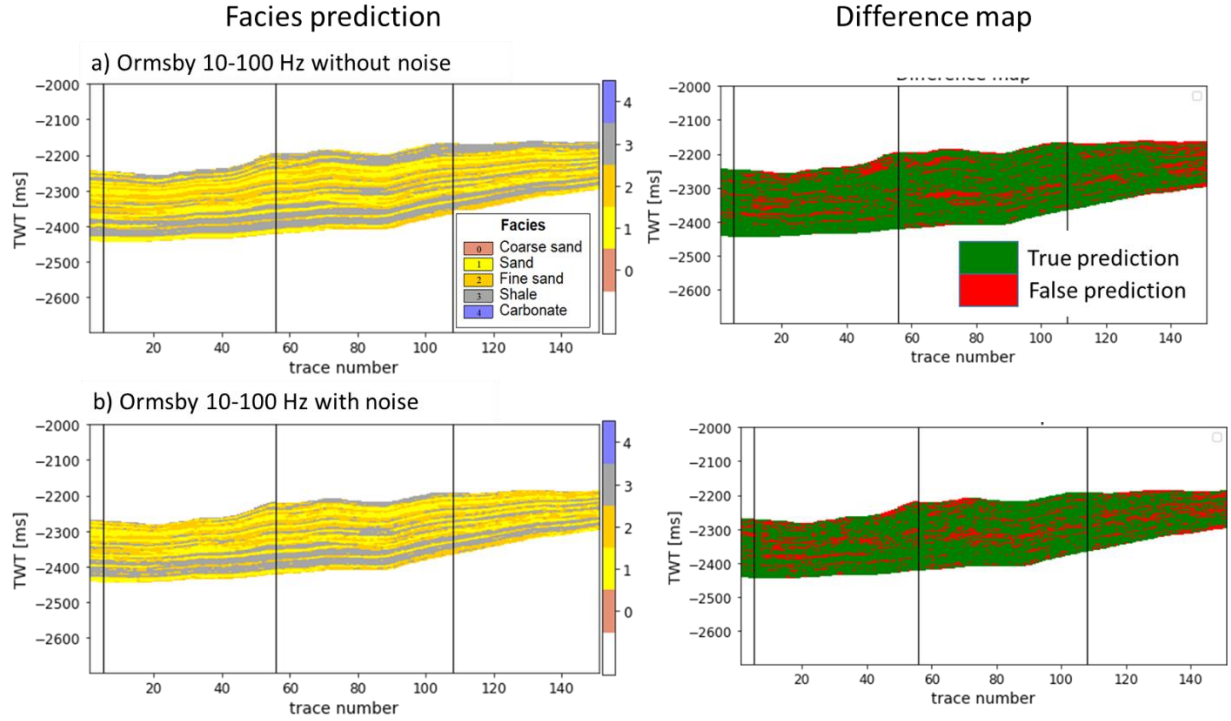

Figure 62. The facies prediction and difference map of case 2 when using seismic features: : a) without noise, and b) with noise. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right.

5.4.6 Exploration and comparison of additional features

5.4.6.1 Instantaneous Frequency, Envelope, and Geological Time

A comparison of facies performance for the RF classifier when using only seismic and seismic attributes (relative acoustic impedance, envelope, and instantaneous frequency), versus seismic and seismic inversion, versus seismic, seismic attributes, and geological time, is shown in Figure 63. All features are derived with the use of an Ormsby filter with a frequency range of 10-100 Hz.

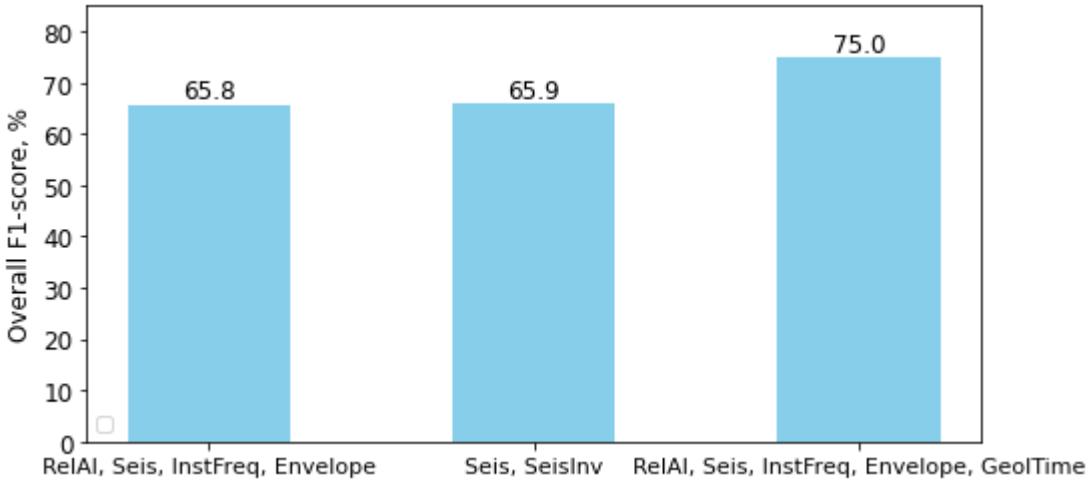

Figure 63. The overall accuracy (global F1-score) for RF facies prediction of case 2 when using seismic and seismic attributes, seismic and seismic inversion, and seismic, seismic attributes, and geological time.

Figure 63 shows that using only seismic and seismic attributes for facies prediction gives almost the same result than using seismic and seismic inversion, namely 65.8%. Incorporating geological time improves the overall accuracy by 9.2%. So, with the use of seismic attributes together with geological time, the facies prediction is equal to the threshold accuracy of 75%. Figure 64 shows the comparison of the F1-score for each facies.

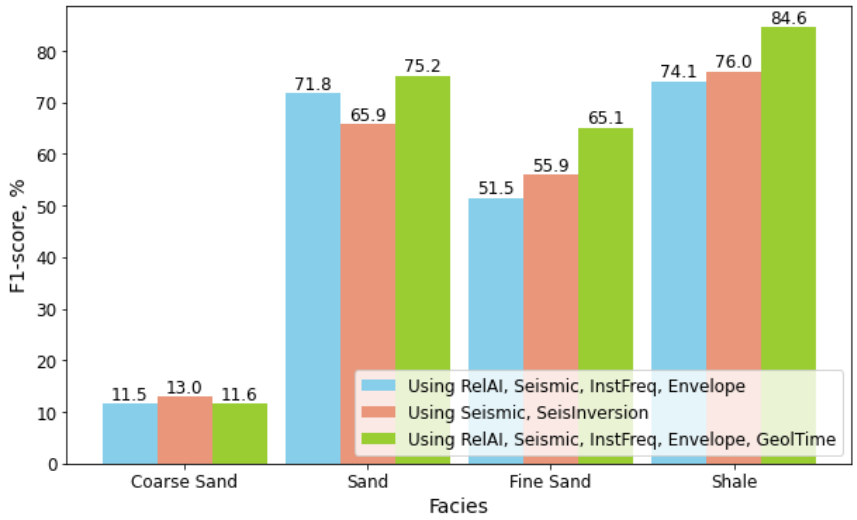

Figure 64. The F1-score of RF facies prediction of case 2 from seismic and seismic attributes (blue), versus seismic and seismic inversion (brown), versus seismic, seismic attributes, and geological time (green).

The use of seismic attributes together with geological time gives the best performance for sand, fine sand, and shale with 75.2%, 65.1%, and 84.6%, respectively. However, the highest performance for coarse sand is gained when using only seismic inversion.

So, the main conclusion in case 2 is that it is possible to predict facies using seismic attributes. However, incorporating additional information about the geometry of the reservoir (geological time) increases the facies prediction. Figure 65 shows the comparison of feature importance for the RF Classifier model.

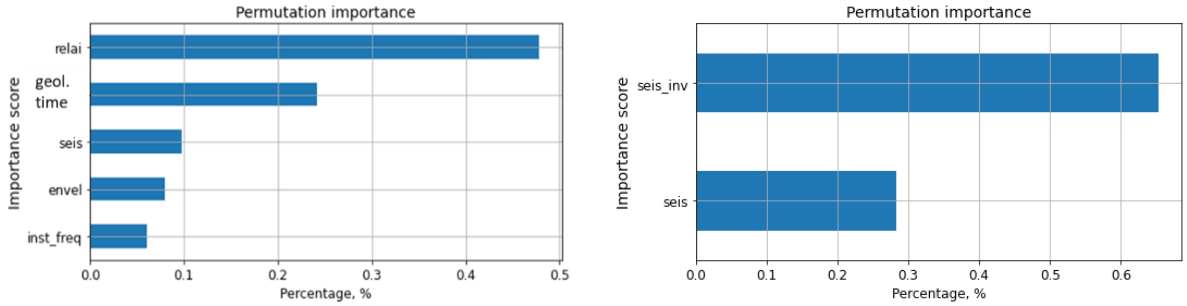

Figure 65. Feature importance for RF facies prediction of case 2 from seismic and seismic attributes (left), and seismic and seismic inversion (right).

Figure 65 shows that seismic inversion and relative acoustic impedance are the features that contribute more to the accurate prediction, with values of 65% and 48%, respectively. Envelope and instantaneous frequency contribute to facies prediction less than 10%. The geological time contribution to the facies prediction is around 24%. Figure 66 shows a 2D section of case 2 with facies prediction and difference map when using seismic and seismic inversion on one hand, and seismic, seismic attributes, and geological time on the other hand.

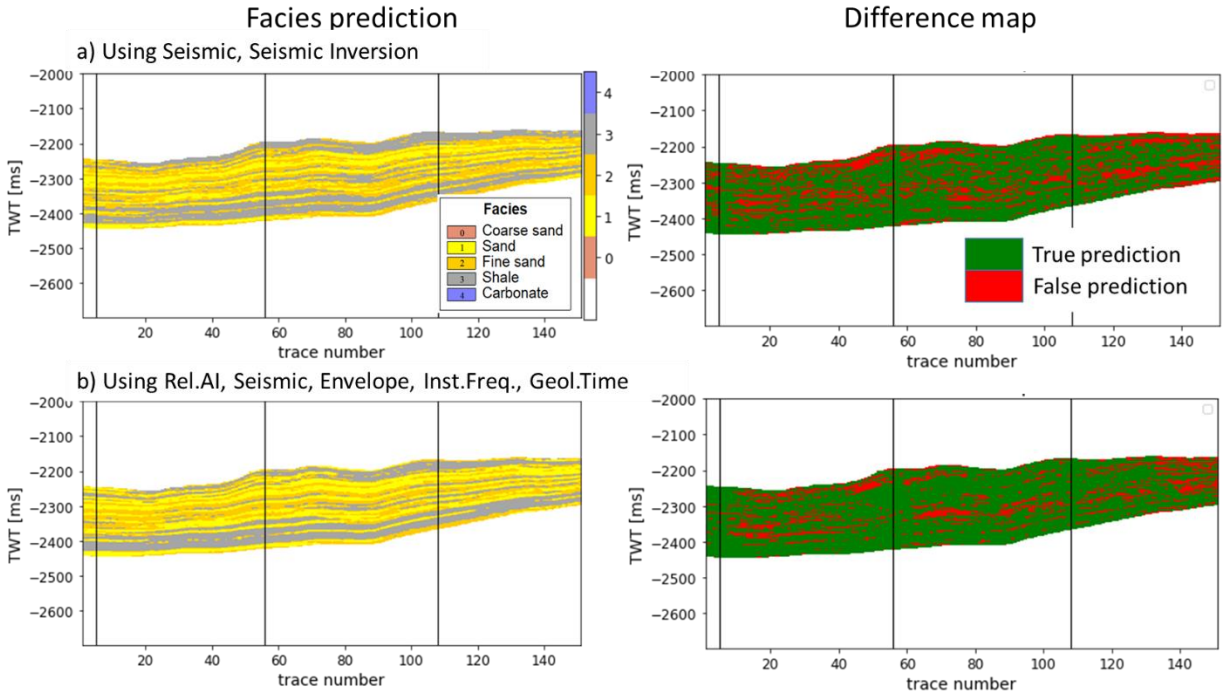

Figure 66. Facies prediction and difference map of case 2 when using the features: a) seismic and seismic inversion, and b) seismic, relative acoustic impedance, envelope, instantaneous frequency, and geological time. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right.

5.5 Results of case 3

Case 3 belongs to the lower zone of the reservoir and is entirely crossed by a normal fault. From the analysis of case 1 which also belongs to the lower part of the reservoir, we justified the necessary number of wells (3 wells) for achieving the threshold accuracy of 75%. Apart from that,

the best performance of facies prediction was achieved when using seismic data derived with Ormsby frequency 10-60 Hz. So, this information is utilized when applying ML models for facies prediction in case 3 from five features: seismic, relative acoustic impedance, envelope, instantaneous frequency, and seismic inversion. Geological time is not available for this zone.

5.5.1 Baseline models of case 3

In this part, five baseline ML models are built and tested for three wells. These ML models are LR, KNN, SVM, RF, and NN (Figure 67). The location of the wells is random with only one condition, that wells should not cross the fault. Hyperparameter tuning is applied to the model with the highest overall accuracy of facies prediction.

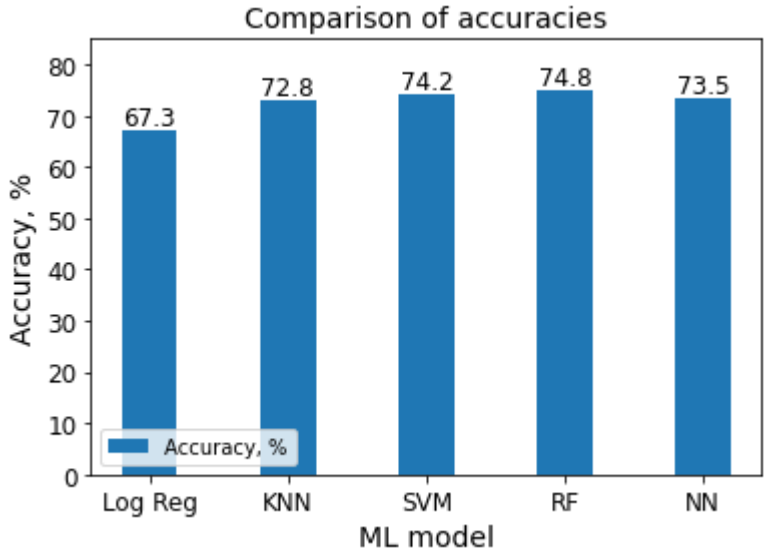

Figure 67. The overall accuracy (global F1-score) of the baseline models for case 3.

The best overall accuracy of 74.8% is achieved for the Random Forest Classifier. Other ML models such as KNN, SVM, and NN showed slightly lower accuracy, from 74.2% to 72.8%. The lowest accuracy of 67.3% was obtained for the least flexible Logistic Regression model. Figure 68 shows the F1-score of each facies predicted by the five ML models.

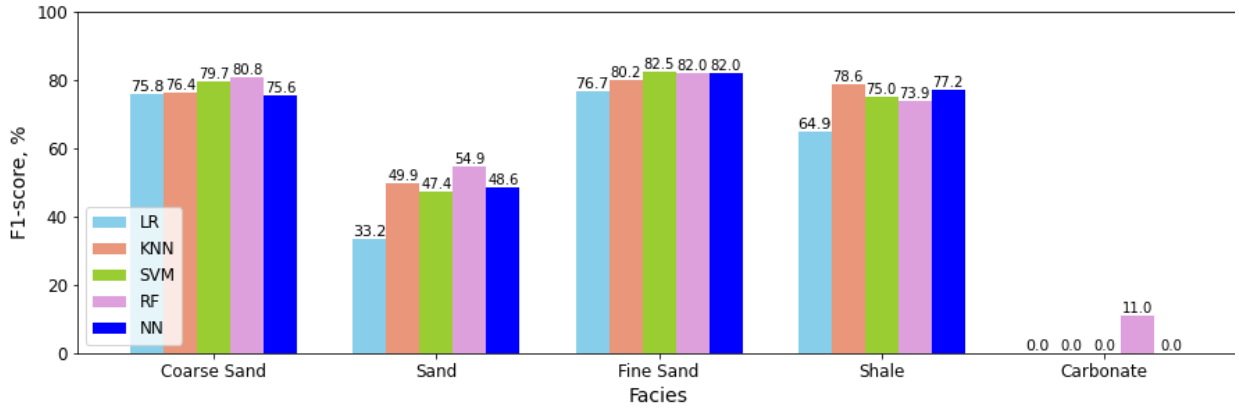

Figure 68. Comparison of the F1-score of each facies by the ML models LR, KNN, SVM, RF, and NN for case 3.

Figure 68 shows that four out of the five ML models failed to predict the least present carbonate facies, while only the Random Forest Classifier managed to predict this facies. The F1-score for fine sand is almost the same for all five models, except for Logistic Regression, whose performance is slightly lower and equal to 76.7%. The RF model has the highest performance when predicting coarse sand and sand and is equal to 80.8% and 54.9%, respectively. The KNN model predicts better the shales.

5.5.2 Model optimization

As the RF model gives the best overall accuracy and manages to predict the least presented facies, hyperparameter tuning is applied to this model to improve its performance. Hyperparameter tuning is done by applying Random Search and Grid Search methods. Due to time constraints, other hyperparameter tuning approaches, such as Bayesian Optimization, were not used.

Hyperparameter tuning for the RF model starts with the Random Search that is executed based on the relevant hyperparameters of the baseline RF model (Table 5).

Table 5. The hyperparameter range and optimal values of the Random Search for the Random Forest Classifier model for case 3.

Hyperparameter	Values	Optimal values
n_estimators	200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000	400
max_features	'auto', 'sqrt', 'log2'	‘auto’
max_depth	10, 120, 230, 340, 450, 560, 670, 780, 890, 1000	890
min_samples_split	1, 3, 4, 5, 7, 9	4
min_samples_leaf	1, 2, 4, 6, 8	1
criterion	'entropy', 'gini'	‘entropy’

The optimal parameters for Random Search are achieved by executing 100 iterations and 3 k-folds cross-validation of the training dataset. This resulted in the improvement of the overall accuracy of the validation set of the RF model by 2% and equal to 88%. The overall accuracy for the testing set increased by 0.5%.

Based on the values of the optimal hyperparameters calculated by the Random Search, the range of values for the Grid Search are chosen (Table 6).

Table 6. The hyperparameter range and optimal values of the Grid Search for the Random Forest Classifier model for case3.

Hyperparameter	Values	Optimal values
n_estimators	200, 300, 400, 500, 600	300
max_features	'auto'	‘auto’
max_depth	890	890
min_samples_split	2, 3, 4, 5, 6	4
min_samples_leaf	1, 3, 4	1
criterion	'entropy'	‘entropy’

In total, the Grid Search executed 100 iterations with 10 k-folds cross-validation for the training set. As a result, the overall accuracy for the validation dataset is increased by 2% and is equal to 90%. The accuracy of the testing set is improved by 0.3% compared with the accuracy after implementing the Grid Search. Figure 69 shows the comparison of the overall accuracies for

the testing set without hyperparameter tuning, and with hyperparameter tuning using Random Search and Grid Search.

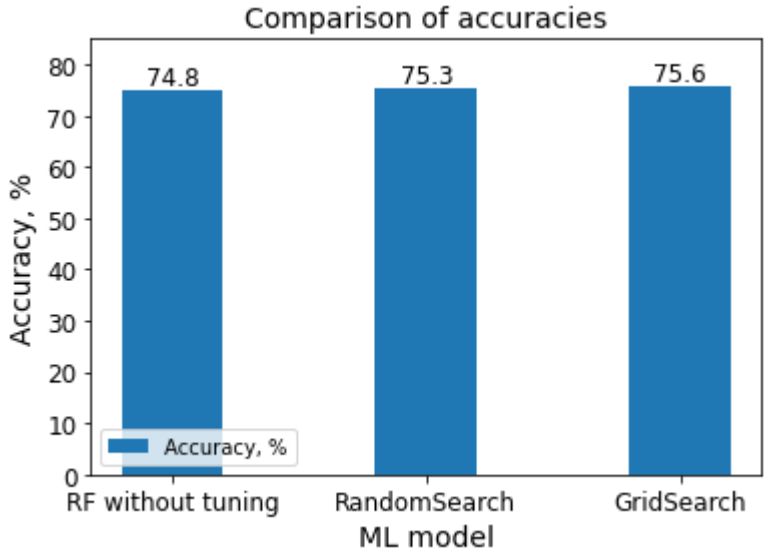

Figure 69. Comparison of the overall accuracies for the RF model for the test set of case 3 without hyperparameter tuning, and after hyperparameter tuning using Random Search and Grid Search.

Figure 70 shows the facies prediction and the difference map by using the RF model with Grid Search hyperparameter tuning.

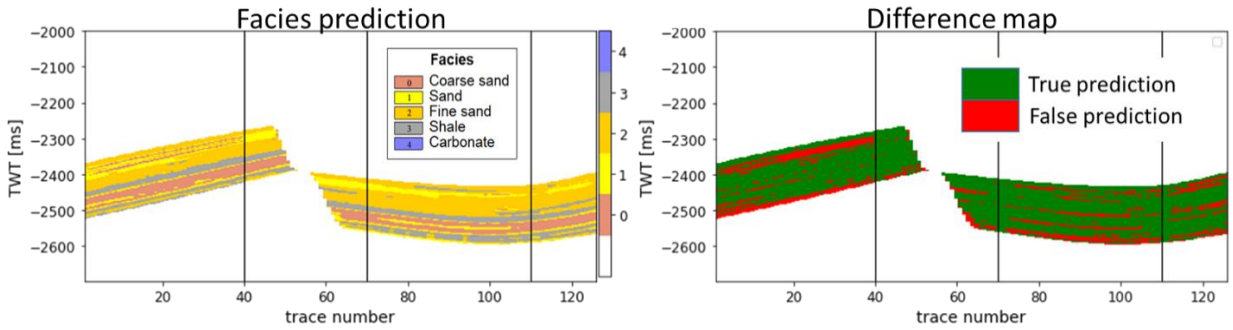

Figure 70. The facies prediction and difference map of case 3 after implementing hyperparameter tuning Grid Search. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right

The difference map shows that the RF model managed to predict well the conformable and thick facies deposits, even though these deposits are offset by the normal fault. However, in the transition areas where one facies is replaced by another and consequently are characterized by thin and interbedded deposits, the ML model's performance is much lower. This is the case for all zones and cases studied in this thesis. The main reason for lower performance in thinner deposits is the difference in vertical resolution between seismic data and the facies deposits. Apart from that, the most correct facies prediction is in areas close to the well locations.

5.6 Results of case 4

Case 4 is a 3D sub-cube and belongs to the lower zone of the reservoir and is entirely crossed by a normal fault as Case 3. From the analysis of Case 1, which also belongs to the lower zone of the reservoir, the optimal wavelet for deriving seismic data was the Ormsby filter with a frequency range of 10-60 Hz. However, the number of wells and their location are chosen manually for Case 4, which is 7 wells. As in Case 3, the prediction is made by using five features: seismic inversion, seismic, relative acoustic impedance, envelope, and instantaneous frequency. Geological time is not available for this zone.

5.6.1 Baseline models for case 4

In this part, five baseline ML models are built and tested for seven wells. These ML models are LR, KNN, SVM, RF, and NN (Figure 71). The location of the wells is random, with the only condition that wells should not cross the fault. Hyperparameter tuning is applied to the model with the highest overall accuracy (global F1-score) of facies prediction.

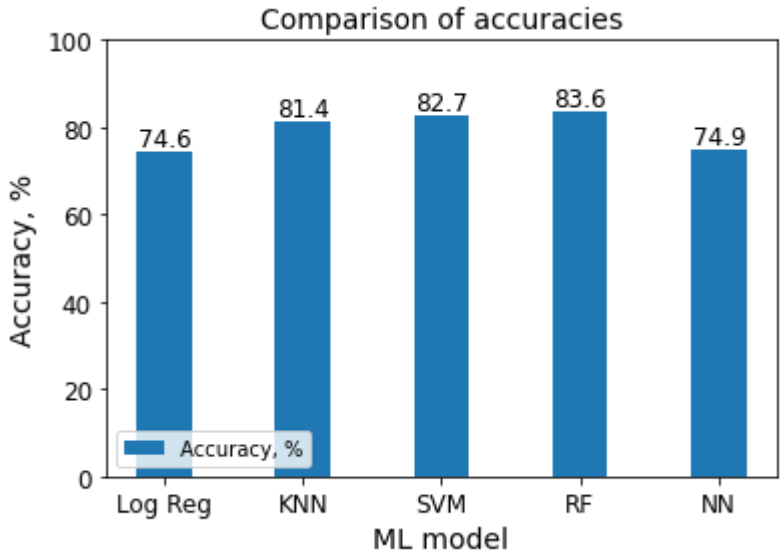

Figure 71. The accuracy (global F1-score) of the baseline models for case 4.

The best overall accuracy of 83.6% is achieved for the Random Forest classifier. Other ML models, such as KNN and SVM, showed slightly lower accuracy, 81.4% and 82.7%, respectively. The lowest accuracy of 74.6% was obtained for the least flexible Logistic Regression model. Figure 72 shows the F1-score of each facies predicted by the five ML models.

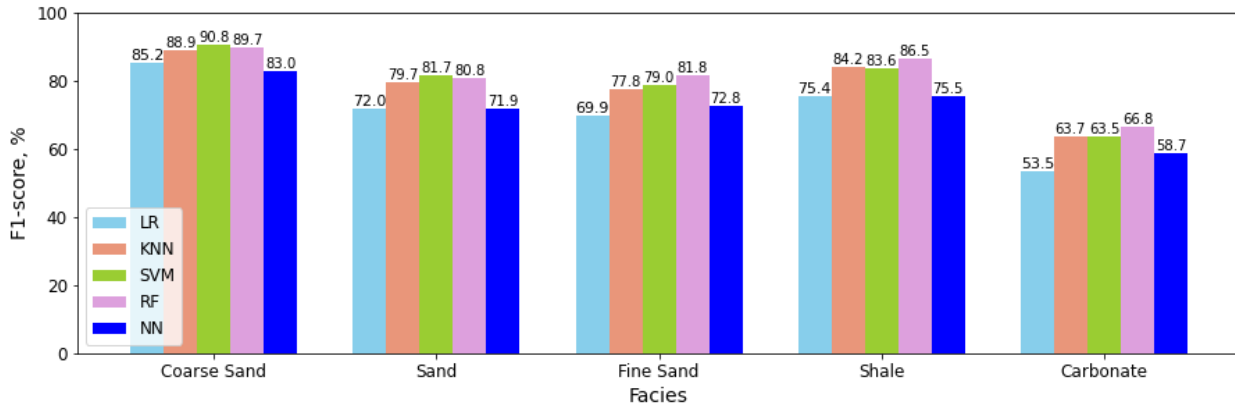

Figure 72. F1-score of each facies in case 4 for the ML models LR, KNN, SVM, RF, and NN.

Figure 72 shows that all ML models managed to predict the least present carbonate facies. The highest F1-score for fine sand, carbonate, and shale is achieved by the RF model; while for coarse sand and sand, the SVM model shows the best performance.

The feature importance of the RF model is shown in Figure 73. Seismic Inversion contributes more than 65% to the accuracy of facies prediction, while relative acoustic impedance only 12%. The impact of envelope and instantaneous frequency is less than 10%.

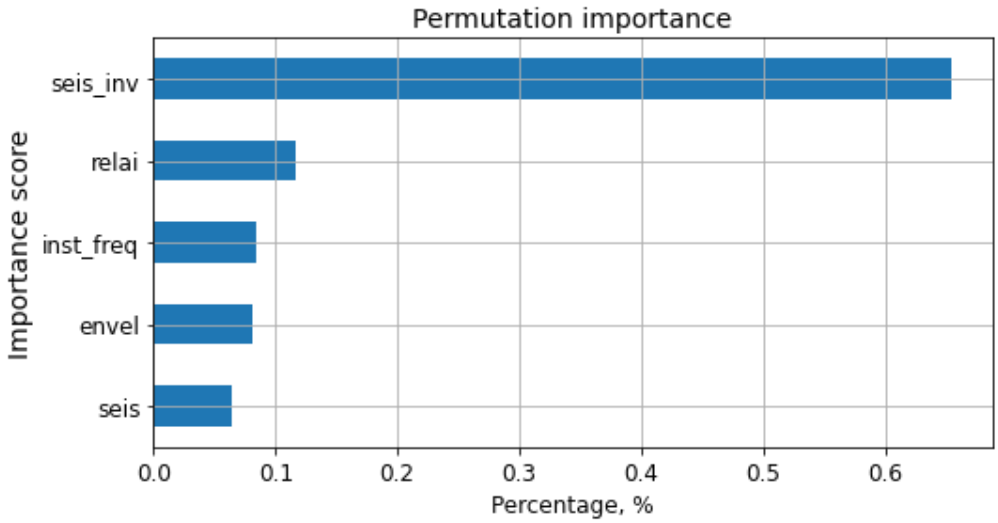

Figure 73. Feature importance of the RF model for Case 4.

A 3D cube with the RF model facies prediction and difference map is shown in Figure 74. A 2D section of the facies prediction and difference map gives a better visualization of the performance of the model (Figure 75). The RF model manages to predict consistent and thick layered facies. However, in the transition areas that are characterized by thin and interbedded facies, the prediction is less reliable.

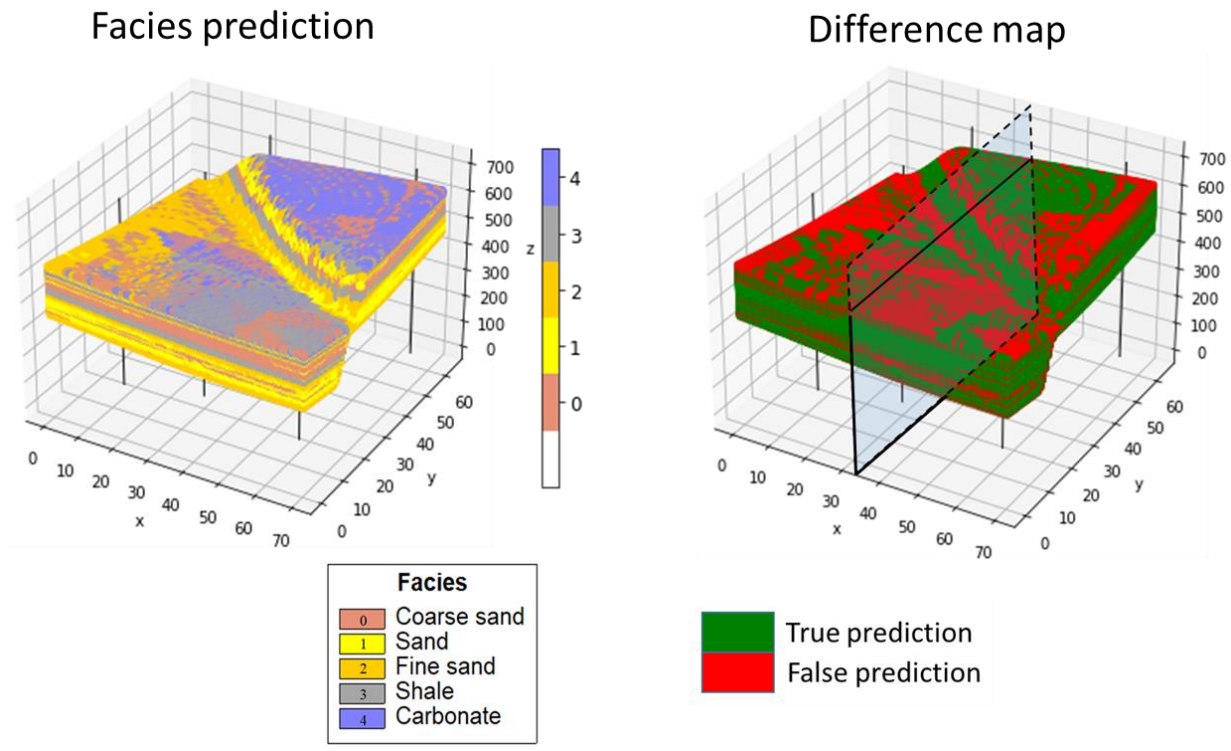

Figure 74. Facies prediction and difference map for the 3D sub-cube of case 4. The wells are shown as black lines. The facies prediction by the RF model is on the left side, and the difference between the true facies and predicted facies is on the right. Transparent rectangle shows the location of cross-section in Figure 75.

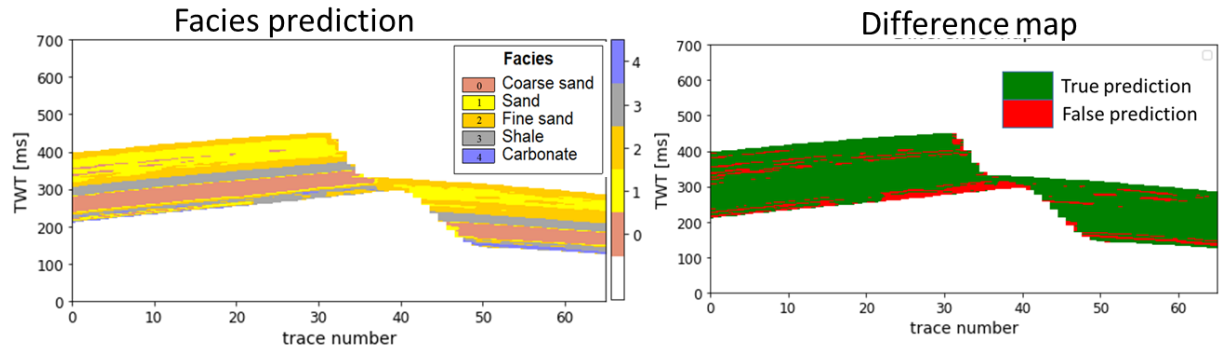

Figure 75. Cross-section of the 3D facies prediction and difference map in Figure 74.

5.6.2 Model optimization for case 4

As the RF gives the best overall performance, hyperparameter tuning is applied to the model to improve its performance. Hyperparameter tuning is done by applying Random Search and Grid Search methods as for case 3. The best hyperparameters from Random Search are shown in Table 7.

Table 7. The hyperparameter range and optimal values of the Random Search for the Random Forest Classifier model of case 4.

Hyperparameter	Values	Optimal values
n_estimators	200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000	1000
max_features	'auto', 'sqrt', 'log2'	‘log2’
max_depth	10, 120, 230, 340, 450, 560, 670, 780, 890, 1000	1000
min_samples_split	1, 3, 4, 5, 7, 9	4
min_samples_leaf	1, 2, 4, 6, 8	1
criterion	'entropy', 'gini'	‘gini’

As for case 3, the optimal parameters for Random Search are achieved by executing 100 iterations and 3 k-folds cross-validation of the training dataset. This resulted in the improvement of the overall F1-score of the validation set of the RF model by 0.2% and equal to 87.9%. The overall accuracy for the testing set is not improved and is equal to 83.3%.

Grid Search was applied to improve the F1-score of the RF model after using Random Search. The hyperparameters of the Grid Search are shown in Table 8.

Table 8. The hyperparameter range and optimal values of the Grid Search for the Random Forest Classifier model of case 4.

Hyperparameter	Values	Optimal values
n_estimators	800, 900, 1000, 1100, 1200	300
max_features	‘log2’	‘log2’
max_depth	1000	1000
min_samples_split	2, 3, 4, 5, 6	4
min_samples_leaf	1, 3, 4	1
criterion	'gini'	‘gini’

In total, the Grid Search executed 100 iterations with 10 k-folds cross-validation for the training set. As a result, the overall F1-score was not improved. Figure 76 shows the comparison of the overall accuracies for the testing set without hyperparameter tuning, and with hyperparameter tuning using Random Search and Grid Search.

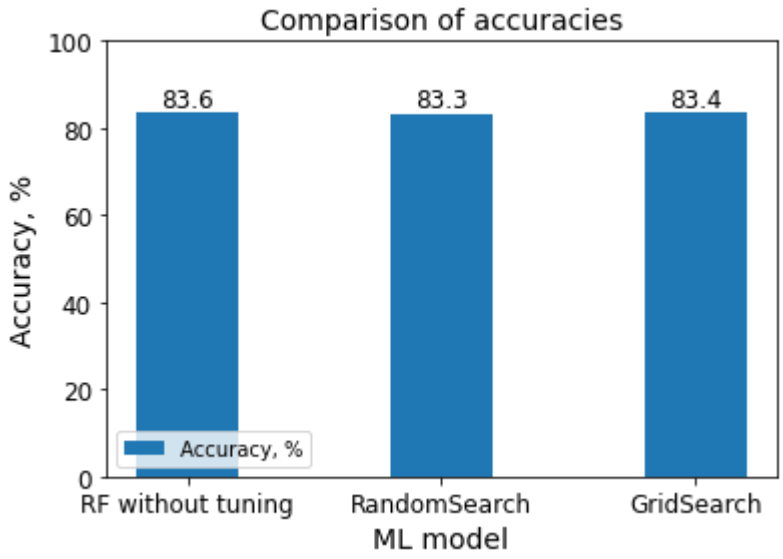

Figure 76. Comparison of the overall F1-score of the RF model applied to the test set in case 4 without hyperparameter tuning, and after implementing hyperparameter tuning with Random Search and Grid Search.

As a result, the use of hyperparameter tuning for the Random Forest model did not improve the facies prediction. There can be two reasons: the default parameters are better than customized hyperparameters, and tuning hyperparameters on a limited training and validation sets does not affect the model's performance on a large test set. The default hyperparameters are shown in Table 9.

Table 9. The default hyperparameters of the RF model for case 4.

Hyperparameter	Default values
n_estimators	100
max_features	'sqrt
max_depth	None
min_samples_split	2
min_samples_leaf	1
criterion	'gini'

6 Discussion

In this study, a performance comparison of four supervised machine-learning models (Logistic Regression, K-Nearest Neighbors, Support Vector Machines, and Random Forest) and a deep-learning model (Neural Networks), is performed for facies prediction based on a realistic synthetic dataset consisting of a geological model, and forward modeled seismic. In general, the best performance is achieved by implementing a decision tree-based method, the Random Forest method, which outperforms all other ML models, including the more powerful Neural Networks method. The reason behind this is that the Neural Networks performance depends on the amount of data in the training dataset. The more data and features are available for training the model, the more robust and higher the performance of the NN is. In our study, the training processes are based on three wells for two-dimensional cases, and seven wells for a three-dimensional sub-cube, which correspond to about 2-5% of the total dataset. For this reason, the ML models' performance is based on their ability to predict facies on a limited training set.

The Logistic Regression showed the least accurate performance compared to the other ML models for all cases. This is because this model is less flexible, and the actual relationship between features and facies is not linear. In addition, LR usually works well for a relatively balanced dataset, which is not the case in this study.

Generally, the SVM and KNN models demonstrated almost the same performance as the RF for all cases. However, the accuracy of the KNN model is slightly better for the least present facies (carbonates in cases 1, 3, 4, and coarse sand in Case 2), compared to the SVM performance which, in some realizations, failed to predict them. In contrast, the Random Forest method produced the highest performance when classifying the least present facies, such as carbonates in cases 1, 3, and 4, and coarse sand in Case 2. This means that the RF method is more robust and suitable when dealing with imbalanced datasets.

However, in the transition areas where facies laterally change to other facies and they are characterized by thin and interbedded deposits, the ML model's performance is much lower. This

is true for all the cases studied in this thesis. The main reason for the lower ML performance in thinner deposits is the difference in vertical resolution between the seismic data and the facies deposits. It is a common problem for seismic data to have worse resolution with depth because of increasing velocity and decreasing frequency. The increase in velocity is caused by rock compaction, while the drop in frequency is explained by seismic data attenuation [32]. As a result, the seismic wavelength, which is the velocity divided by frequency, also increases with depth, leading to the loss of vertical resolution. To evaluate this problem, the role of various frequencies used for deriving the seismic data, was evaluated.

The comparison of facies prediction from seismic attributes derived from Ricker wavelet with a frequency of 25 Hz and three Ormsby filters with a frequency range of 10-60 Hz, 10-80 Hz, and 10-100 Hz showed that the overall performance of ML facies prediction depends on the depositional environment and frequencies. So, the use of the Ormsby filter with a frequency range of 10-60 Hz showed the best overall performance of facies prediction for the lower part of the reservoir, which is represented by a shallow marine environment with consistent and thick layers of facies. In contrast, for the upper part of the reservoir with interbedded, thin, and inconsistent facies, the best overall accuracy is achieved by using seismic features derived from the Ormsby filter with a frequency range of 10-100 Hz. Moreover, for the heterogeneous facies deposits from the upper part of the reservoir, the accuracy of each facies increases with the increase in frequency values. However, this is not true for the lower part of the reservoir. This means that when dealing with real data, it is important to use seismic data derived from various frequency ranges, especially if dealing with complex depositional environments.

Incorporating additional features such as spectral decomposition with frequencies 30 Hz, 60 Hz, and 90 Hz, also influences the facies prediction depending on the facies depositional characteristics. In the upper zone of the reservoir, the higher the frequency of the spectral decomposition, the better the facies prediction accuracy. This confirms the previous statement that when dealing with thin bedded, non-conformable facies deposits, it is essential to perform the analysis from the seismic data derived from higher frequencies. However, for the lower zone of the reservoir with thick and conformable deposits, the use of higher frequencies does not always lead to a better facies prediction. ML is not an automatic procedure, but a geologist is needed to

divide the volume of investigation into smaller geologically meaningful domains (e.g., lower and upper reservoir zones), for which different ML methods can be designed to get the best prediction. This is the case for any geological feature to be predicted, sedimentary facies or geological structures such as faults.

In addition, the number of wells and their location has an impact on the ML facies prediction because its performance is based on the amount and quality of training data. However, the prediction also depends on the facies distribution in the reservoir. For thicker and more conformable facies, the threshold accuracy of 75% is achieved when using three wells for training the ML model. Moreover, the use of only one well gives a slightly lower accuracy of 74.8%. In contrast, the analysis of thin and less conformable facies from case 2 shows that even when utilizing thirty wells for training the ML model, the threshold accuracy is not achieved, and the maximum accuracy is just 69.2%. In this case, to improve the ML models performance, it is recommended to incorporate additional features to the analysis. In addition, the accuracy of facies prediction is higher in areas in the vicinity of the wells (Figure 77).

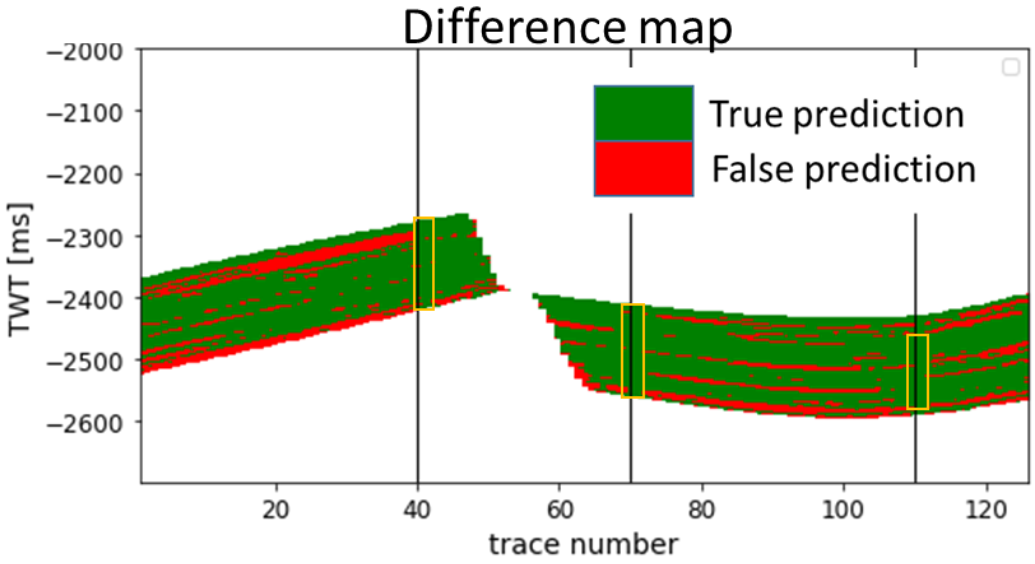

Figure 77. Difference map between the ground truth facies distribution and facies prediction by ML in case 2. The yellow rectangles show the areas close to the location of the wells, where the facies prediction is better compared with the facies prediction at a larger distance from the wells.

.

As one moves away from the location of the well, the accuracy of the facies prediction decreases laterally along the stratum. This highlights the importance of the placement of the wells. The ideal scenario is when the wells intersect all the facies in the reservoir. However, in the real world, the number of wells is limited – because drilling wells is expensive and requires proper justification. For this reason, it is important to include in the analysis as many wells as possible corresponding (or related) to the analyzed reservoir.

Including the additional seismic attributes in the analysis improves the accuracy of the facies prediction to varying degrees. Seismic inversion has the highest positive impact on facies prediction, followed by relative acoustic impedance. Instantaneous frequency and envelope are less important for correct facies prediction. Providing the ML models with information about the lateral geometry of the reservoir, e.g., geological time as an additional feature, improves the facies prediction.

One of the goals of this thesis was to assess the impact of seismic attributes versus seismic inversion on the facies prediction. The thesis shows that the use of only seismic attributes (relative acoustic impedance, instantaneous frequency, and envelope) gives almost the same prediction accuracy as when utilizing seismic inversion. This conclusion is very important because seismic inversion and the process of deriving acoustic impedance from seismic data is computationally expensive and not as straightforward as deriving seismic attributes (e.g., relative acoustic impedance) from seismic data. There are other seismic attributes that can potentially highlight the facies distribution in the reservoir, however, due to time constraints, only a few relevant seismic attributes are considered.

This study incorporates the facies prediction on 1D, 2D, and 3D datasets. For the 3D cube, I consider it as a set of parallel 2D sections. The prediction process is also based on wells, and as the facies classification is performed for one 2D section, the algorithm continues the calculations for the next 2D section and so on until completing all sections. Thus, the ML models do not consider previous predictors. In other words, the selected approach is a strictly single-trace process which means that the results of neighboring traces do not influence the result of the trace under

consideration. In a real scenario, this approach of 3D facies prediction can cause inconsistent predicted values along the x- and y-axis. To avoid or minimize possible discrepancies in facies prediction in a 3D dataset, it is recommended to provide the ML model with information about the horizontal spatial change along the x- and y-axis. This can be done by taking the average value of each feature within a specific area surrounding every point and adding this value as a new feature. So, facies prediction is further based on the average value of a feature within a specific area. Because of time limitations, the averaging of the closest values in each feature was not performed in this study.

7 Future work

As a further development, it is recommended to apply the results obtained in this study to a real case scenario. This is because in the synthetic model, the wells used for training the ML models are traces and have the same domain in TWT as seismic data, and resolution, which is not the case in real scenarios. In a real scenario, one would need to tie the well to the seismic data, and upscale the well logs to the seismic. Moreover, including the investigation of different filter options to address the noise problem is also desirable. In addition, incorporating additional relevant seismic attributes into the analysis might improve the prediction of facies. Exploring other machine-learning methods, such as XGBoost, and K-means Clustering, might be advantageous. In this thesis, four supervised models and one deep-learning method were tested, and hyperparameter tuning (Random Search and Grid Search) was applied to the baseline model with the best performance. Implementing the Bayesian Optimization approach to the Neural Networks method can also improve the performance of the facies prediction. Finally, testing the implementation of the best average window for facies prediction in a 3D cube can be interesting as a future work.

8 Conclusions

This thesis proves that machine-learning methods are a powerful tool for predicting facies from seismic attributes and can be used in 2D and 3D datasets. Built and trained ML models, together with developed algorithms for visualization and validation of results, will help to implement the same methodology in other scenarios. The research shows that a complex reservoir with various depositional environments requires the use of seismic data derived from different frequencies, depending on the thickness and consistency of facies, to obtain better performance. The use of relevant seismic attributes provides almost the same performance as when using seismic inversion and incorporating additional information about the geological time (relative depth) can even outperform the prediction from seismic inversion. The Random Forest model showed the best performance and seemed to be a robust method among others. However, it is recommended to develop unsupervised ML methods since they can handle a bigger dataset and have a good performance if applied properly.

References

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

2. Wang, G., Carr, T.R., (2012). Marcellus Shale Lithofacies Prediction by Multiclass Neural Network Classification in the Appalachian Basin. Math Geosci 44, 975–1004. https://doi.org/10.1007/s11004-012-9421-6.

3. Rahimi M., Riahi, M.A. (2022). Reservoir facies classification based on random forest and geostatistics methods in an offshore oilfield. Journal of Applied Geophysics. vol. 201. doi:104640. 10.1016/j.jappgeo.2022.104640

4. Koson, S., Chenrai, P., & Choowong, M. (2021). Seismic Attributes and Their Applications in Seismic Geomorphology. Bulletin of Earth Sciences of Thailand, vol. 6(1), 1–9. Retrieved from https://ph01.tci-thaijo.org/index.php/bestjournal/article/view/246597.

5. Emujakporue, G.O., Enyenihi, E.E., (2020). Identification of seismic attributes for hydrocarbon prospecting of Akos field, Niger Delta, Nigeria. SN Appl. Sci. 2, 910. https://doi.org/10.1007/s42452-020-2570-1.

6. Ginting H., Firmansyah M., Ariansyah M., Aswad S., Siregar D., (2019). Reservoir characterization using acoustic impedance seismic inversion method and seismic attribute in the “RST” field of the Taranaki Basin, Indonesian Petroleum Association 43rd Annual Convention & Exhibition. Indonesia. 10.29118/IPA19.SG.41.

7. Hart BS (2008) Channel detection in 3-D seismic data using sweetness. AAPG Bulletin 92, pp. 733–742.

8. Jo, T. (2021) Machine learning foundations: Supervised, unsupervised, and advanced learning. Cham: Springer Nature.

9. El Morr, C., Jammal, M., Ali-Hassan, H., El-Hallak, W. (2022). Introduction to Machine Learning. In: Machine Learning for Practical Decision Making. International Series in Operations Research & Management Science, vol 334. Springer, Cham. https://doi.org/10.1007/978-3-031-16990-8_1.

10. Emerson N., Vijayakumari P., (2017). FPGA Based Real Time Classification with Support Vector Machine. 2017 International Conference on Intelligent Computing and Control (I2C2). pp. 233-239.

11. Awad, M., Khanna, R. (2015). Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers. Springer Nature. https://doi.org/10.1007/978-1-4302-5990-9.

12. El Morr, C., Jammal, M., Ali-Hassan, H., El-Hallak, W. (2022). Logistic Regression. In: Machine Learning for Practical Decision Making. International Series in Operations Research & Management Science, vol 334. Springer, Cham. https://doi.org/10.1007/978-3-031-16990-8_7

13. Brereton, R.G., & Lloyd, G.R. (2010). Support vector machines for classification and regression. The Analyst, vol. 135 2, 230-67.

14. Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica 31 (2007). Pp. 249 – 268. Retrieved from IJS website: http://wen.ijs.si/ojs2.4.3/index.php/informatica/article/download/148/140

15. Mucherino A., Papajorgji P. J., Pardalos P. M., (2009). Data Mining in Agriculture, Springer Optimization and Its Applications, Springer, number 978-0-387-88615-2.

16. Cutler, A., Cutler, D.R. Stevens, J.R. (2012) Random Forests. In: Zhang, C. and Ma, Y.Q., Eds., Ensemble Machine Learning, Springer, New York, pp. 157-175. http://dx.doi.org/10.1007/978-1-4419-9326-7_5

17. Basavaiah J., Audre A. A., (2020). Tomato Leaf Disease Classification using Multiple Feature Extraction Techniques. Wireless Personal Communications. 115, pp. 633-651. 10.1007/s11277-020-07590-x.

18. Introduction to Remote Sensing' Humboldt-Universität zu Berlin. Department of Geography, 2021. Image classification - Random Forest. https://pages.cms.hu-berlin.de/EOL/geo_rs/S09_Image_classification2.html

19. Breiman, L., (2001) Random Forests. Machine Learning vol. 45, pp. 5–32. https://doi.org/10.1023/A:1010933404324

20. The Carpathians, Tan P.N., Steinbach M., Anuj V. K., (2019). Introduction to Data Mining, Global Edition, Pearson, 2nd Edition.

21. Agrawal, T. (2021). Hyperparameter Optimization in Machine Learning. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6579-6_1

22. Bergstra, J., Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. J. Mach. Learn. Res., vol. 13, pp. 281-305.

23. Xue Y., (2019). An Overview of Overfitting and its Solutions. Journal of Physics: Conference Series. 1168. 022022. 10.1088/1742-6596/1168/2/022022.

24. Boumédiène D., Nisrine.M., (2022). An overview of the infrastructure seismic resilience assessment using Artificial Intelligence and Machine-learning algorithms. 3rd International Conference on Natural Hazards & Infrastructure 5-7 July 2022, Athens, Greece.

25. S. Fortmann-Roe. (n.d.). Understanding the Bias-Variance Tradeoff. Retrieved April 26, 2022, from https://scott.fortmann-roe.com/docs/BiasVariance.html

26. Tharwat, A. (2021), "Classification assessment methods", Applied Computing and Informatics, Vol. 17 No. 1, pp. 168-192. https://doi.org/10.1016/j.aci.2018.08.003

27. Deepti C., Roopal K.,. (2023). Introduction to Machine Learning with Python. Bentham Science, 1st Edition, 94 p. 10.2174/9789815124422123010004.

28. G. James, D. Witten, T. Hastie, and R. Tibshirani, (2017). An introduction to statistical learning: with applications in r, New York.

29. Salomonsen E., (2022). Ml-based porosity modeling tested on synthetic and subsurface data. [Master's Thesis, University of Stavanger]. https://hdl.handle.net/11250/3023681

30. In He, H., & In Ma, Y. (2013). Imbalanced learning: Foundations, algorithms, and applications. Hoboken, New Jersey: John Wiley & Sons, Inc. http://site.ebrary.com/id/10716627

31. Williamson DF, Parker RA, Kendrick JS., (1989). The box plot: a simple visual method to interpret data. Annals of Internal Medicine. vol. 110(11), pp. 916-921. DOI: 10.7326/0003-4819-110-11-916. PMID: 2719423

32. Metwalli, F.I., Shendi, EA.H., Fagelnour, M.S. (2019). Seismic facies analysis of thin sandstone reservoirs, North Western Desert, Egypt. J Petrol Explor Prod Technol 9, 793–808. https://doi.org/10.1007/s13202-018-0541-5

Appendixes

Python appendices included in this section is published in GitHub and can be found using the following link:

https://github.com/aigulakberova/Machine-learning-based-seismic-classification-for-facies-prediction

Appendix 1 Reading SEG-y files

```
1
2  import segyio
3  from mpl_toolkits.axes_grid1 import make_axes_locatable
4  from segysak.segy import segy_header_scan
5  from IPython.display import display
6  import numpy as np
7  import pandas as pd
8  import matplotlib.pyplot as plt
9  from matplotlib import colors
10 import matplotlib.pyplot as plt
11 import matplotlib
12
13
14 def segy_file(file_name='Synth_seismic_1.segy', color_map='seismic'):
15     """The function reads SEGY file of 2D cross-section, makes a plot for Seismic data, its attributes
16     In addition, defines Extent of the cross-section
17
18     Parameters:
19         file_name (str): Specify the name of the SEGY-file. Defaults to 'Synth_seismic_1.segy'.
20         color_map (str): Specify the color_map. Defaults to 'seismic'.
21
22     Returns:
23         data_file (2D array): 2D numpy array of cross-section
24         extent (list): List with depth and trace numbers for the plots
25     """
26
27
28     with segyio.open(file_name, ignore_geometry=True) as file:
29         # Get basic attributes
30         n_traces = file.tracecount
31         sample_rate = segyio.tools.dt(file) / 1000
32         n_samples = file.samples.size
33         twt = file.samples
34         data_file = file.trace.raw[:]  # Get all data into memory (could cause on big files)
35         # Load headers
36         bin_headers = file.bin
37     f'N Traces: {n_traces}, N Samples: {n_samples}, Sample rate: {sample_rate}ms'
38     d = data_file.flatten()
39
40     fig = plt.figure(figsize=(10, 4))
41     ax = fig.add_subplot(1, 1, 1)
42     extent = [1, n_traces, -twt[-1], -twt[0]]  # define extent
43
```

```
43
44      # Choose color_map
45      if color_map =='seismic':
46          # Customise color map for sesimic
47          custom_norm=colors.TwoSlopeNorm(vmin=min(d), vcenter=0, vmax=max(d))
48          im = ax.imshow(data_file.T, origin='upper', cmap="seismic", extent=extent, aspect='auto', norm=custom_norm)
49          fig.colorbar(im)
50          #ax.set_title('Seismic section')
51
52
53      elif color_map == 'facies':
54          # Customise color map for facies
55          facies_name = ['','Coarse Sand', 'Sand', 'Fine Sand', 'Shale', 'Carbonate']
56          facies_color = ['#FFFFFF' ,'#E69076', '#FFFF00', '#FFCC00', '#A6A6A6', '#8080FF']
57          cmap = matplotlib.colors.ListedColormap(facies_color)
58          bounds = [-1.5, -0.5, 0.5, 1.5, 2.5, 3.5, 4.5]
59          norm = matplotlib.colors.BoundaryNorm(bounds, cmap.N)
60          im = ax.imshow(data_file.T, cmap=cmap, aspect='auto', extent=extent, norm=norm, vmin=0-1.5, vmax=4+0.5)
61          fig.colorbar(im, ticks=np.arange(0, 5))
62          #ax.set_title('Facies section')
63
64
65      else:
66          im = ax.imshow(data_file.T, cmap="jet", vmin=min(d), vmax=max(d), aspect='auto', extent=extent)
67          fig.colorbar(im)
68          ax.set_title('')
69
70      colormap1 = fig.axes[1]
71      colormap1.tick_params(labelsize=12)
72      ax.set_xlabel('trace number', fontsize=14)
73      plt.xticks(fontsize=12)
74      plt.yticks(fontsize=12)
75      ax.set_ylabel('TWT [ms]', fontsize=14)
76      plt.show()
77
78      return data_file, extent
```

Appendix 2 - Data Processing

```python

import numpy as np
import pandas as pd



def replace_nonexisting_data_with_NaN(df, nonexisting_data):
    """This function removes data that are out of reservoir (for example, for depths it is data that =250)
    and replaces with NaN.

    Args:
        df (DataFrame): DataFrame
        nonexisting_data (float): nonexisting data that have to be replaced by NaN

    Returns:
        df (DataFrame): DataFrame without nonexisting data
    """
    df = df.replace(nonexisting_data, np.NaN)
    return df




def take_data_that_exist(df, df2):
    """THis function removes data that are out of reservoir by filtering by another data

    Args:
        df (DataFrame): Data that need to be filtered
        df2 (DataFrame): Filter

    Returns:
        map_df (DataFrame): Filtered DataFrame that contains data within the reservoir
    """
    r, c = np.shape(df)
    map_df = np.zeros((r, c))
    map_df[:] = np.NaN

    # for every trace (column)
    for every_col in range(len(df.columns)):
        non_empty_rows = df2.iloc[:,every_col].dropna(axis=0)

        # Filter
        map_df[non_empty_rows.index, every_col] = df.iloc[non_empty_rows.index, every_col]

    return pd.DataFrame(map_df)   # returns df


```

```python


def standartization(df, value_to_drop=True):

    '''
    Implement standartization to a dataset.
    When calculating MEAN and STD excludes the values that belong to empty cells (in case, they were not removed earlier)

    Parameters:
        df (DataFrame)
        value_to_drop (float): specify the value that is outside reservoir


    Returns:
        df_standard (DataFrame): data after implementing Standartization
    '''
    df_array = df.values.flatten()
    # assign 'values_to_drop' a value
    if value_to_drop == True:
        to_drop = df.iloc[0,0]
        # calculate mean of data after excluding values_to_drop from the data
        df_mean = np.mean(df_array[(df_array != to_drop)])
        # calculate STD
        df_STD = np.std(df_array[(df_array != to_drop)])
        # standartization
        df_standard = pd.DataFrame((df.values - df_mean) / df_STD)
    else:
        df_standard = pd.DataFrame((df - np.nanmean(df.values)) / np.nanstd(df.values))

    # return df_standard.add_prefix(str(feature_name))
    return df_standard


```

```python
##########################################################################################################
def standartization_3D(feature_3D_array):

    '''
    Implement standartization to a dataset.
    When calculating MEAN and STD excludes the values that belong to empty cells.

    Parameters:

    Returns:
    '''
    # Flatten 3D array (to 1D)
    feature_array_1d = feature_3D_array.reshape(-1)

    # Calculate STD
    std_all = np.nanstd(feature_array_1d)

    # Calculate Mean
    mean_all = np.nanmean(feature_array_1d)

    # Apply Standartization
    feature_std_3D = (feature_3D_array - mean_all) / std_all

    return feature_std_3D



def df_wells_from_section(df, wells_list):

    '''
    returns DataFrame with particular columns

    Parameters:
        df (DataFrame)
        wells_list (list): list of traces (wells)

    Returns:
        DataFrame with particular columns
    '''
    df_wells = df.iloc[:,wells_list]
    return df_wells
```

```python
##########################################################################################################
# Extract properties for wells
# We have wells coordinates
def extract_wells_with_data_3D(data_3D, x_coord_wells, y_coord_wells):
    wells_list = []

    for well_x in x_coord_wells:
        for well_y in y_coord_wells:

            each_well = data_3D[x_coord_wells, y_coord_wells, :]

    wells_list.append(each_well)

    # Convert to list and Remove nan from well_all
    well_3d_array = np.asarray(wells_list)
    well_2d_array = np.reshape(well_3d_array, (len(x_coord_wells), data_3D.shape[2]))
    # wells_1d_array = well_2d_array.flatten()

    return well_2d_array
```

Appendix 3 - Concatenate facies and features

```python

import pandas as pd

def concat_features_RelAI_Seis_SeisInv(df_facies_wells, feature_list_RelAI_Seis_SeisInv):
    """The function concatenate features (RelAI, Seismic, Seismic Inversion) with Facies

    Args:
        df_facies_wells (DataFrame): Facies
        feature_list_RelAI_Seis_SeisInv (lisr): list of features

    Returns:
        facies_and_features (DataFrame): concatenated dataframe with facies and 3 features (Relative AI, Seismic, Seis_inversion)
    """
    def create_empty_lists(a):
        for i in range(a):
            yield []

    XX0, XX1, XX2 = create_empty_lists(len(feature_list_RelAI_Seis_SeisInv))
    YY1 = []

    for every_well in range(len(df_facies_wells.columns)):

        X0 = feature_list_RelAI_Seis_SeisInv[0].iloc[:,every_well]
        X1 = feature_list_RelAI_Seis_SeisInv[1].iloc[:,every_well]
        X2 = feature_list_RelAI_Seis_SeisInv[2].iloc[:,every_well]

        Y = df_facies_wells.iloc[:,every_well]

        XX0.append(X0)
        XX1.append(X1)
        XX2.append(X2)

        YY1.append(Y)

    XX0 = pd.concat(XX0, ignore_index=False, axis=0)
    XX1 = pd.concat(XX1, ignore_index=False, axis=0)
    XX2 = pd.concat(XX2, ignore_index=False, axis=0)

    facies = pd.concat(YY1, ignore_index=False)

    features = pd.concat([XX0, XX1, XX2], axis=1)

    features = features.rename(columns = {0:'relai',
                                          1:'seis',
                                          2: 'seis_inv'
                                          })
    features

    facies = pd.DataFrame(facies).rename(columns={0:'facies'})
    facies

    facies_and_features = pd.concat([facies, features], axis=1)
    facies_and_features = facies_and_features.dropna(axis=0)

    return facies_and_features

```

```python


########################################################################################################################
def concat_features_Seis_SeisInv(df_facies_wells, feature_list_Seis_SeisInv):
    """The function concatenate features (RelAI, Seismic, Seismic Inversion) with Facies

    Args:
        df_facies_wells (DataFrame): Facies
        feature_list_RelAI_Seis_SeisInv (lisr): list of features

    Returns:
        facies_and_features (DataFrame): concatenated dataframe with facies and 2 features (Seismic and Seismic Inversion)
    """
    def create_empty_lists(a):
        for i in range(a):
            yield []

    XX0, XX1 = create_empty_lists(len(feature_list_Seis_SeisInv))
    YY1 = []

    for every_well in range(len(df_facies_wells.columns)):

        X0 = feature_list_Seis_SeisInv[0].iloc[:,every_well]
        X1 = feature_list_Seis_SeisInv[1].iloc[:,every_well]

        Y = df_facies_wells.iloc[:,every_well]

        XX0.append(X0)
        XX1.append(X1)

        YY1.append(Y)

    XX0 = pd.concat(XX0, ignore_index=False, axis=0)
    XX1 = pd.concat(XX1, ignore_index=False, axis=0)

    facies = pd.concat(YY1, ignore_index=False)

    features = pd.concat([XX0, XX1], axis=1)

    features = features.rename(columns = {0:'seis',
                                          1:'seis_inv'
                                          })
    features

    facies = pd.DataFrame(facies).rename(columns={0:'facies'})
    facies

    facies_and_features = pd.concat([facies, features], axis=1)
    facies_and_features = facies_and_features.dropna(axis=0)

    return facies_and_features



```

```python

def concat_features_RelAI_Seis_Envel_InstFreq(df_facies_wells, feature_list_RelAI_Seis_Envel_InstFreq):
    """The function concatenate features (RelAI, Seismic, Envelope, InstFreq) with Facies

    Args:
        df_facies_wells (DataFrame): Facies
        feature_list_RelAI_Seis_Envel_InstFreq (lisr): list of features

    Returns:
        facies_and_features (DataFrame): concatenated dataframe with facies and 4 features (Relative AI,
        Seismic, Envelope, Instantaneous Frequency)
    """
    def create_empty_lists(a):
        for i in range(a):
            yield []

    XX0, XX1, XX2, XX3 = create_empty_lists(len(feature_list_RelAI_Seis_Envel_InstFreq))
    YY1 = []

    for every_well in range(len(df_facies_wells.columns)):

        X0 = feature_list_RelAI_Seis_Envel_InstFreq[0].iloc[:,every_well]
        X1 = feature_list_RelAI_Seis_Envel_InstFreq[1].iloc[:,every_well]
        X2 = feature_list_RelAI_Seis_Envel_InstFreq[2].iloc[:,every_well]
        X3 = feature_list_RelAI_Seis_Envel_InstFreq[3].iloc[:,every_well]

        Y = df_facies_wells.iloc[:,every_well]

        XX0.append(X0)
        XX1.append(X1)
        XX2.append(X2)
        XX3.append(X3)

        YY1.append(Y)

    XX0 = pd.concat(XX0, ignore_index=False, axis=0)
    XX1 = pd.concat(XX1, ignore_index=False, axis=0)
    XX2 = pd.concat(XX2, ignore_index=False, axis=0)
    XX3 = pd.concat(XX3, ignore_index=False, axis=0)

    facies = pd.concat(YY1, ignore_index=False)

    features = pd.concat([XX0, XX1, XX2, XX3], axis=1)

    features = features.rename(columns = {0:'relai',
                                          1:'seis',
                                          2: 'envel',
                                          3: 'inst_freq'
                                          })
    features

    facies = pd.DataFrame(facies).rename(columns={0:'facies'})
    facies

    facies_and_features = pd.concat([facies, features], axis=1)
    facies_and_features = facies_and_features.dropna(axis=0)

    return facies_and_features
```

```
174
175
176
177
178
179  ##############################################################################################
180
181  def concat_features_RelAI_Seis_SeisInv_Depth(df_facies_wells, feature_list_RelAI_Seis_SeisInv_Depth):
182      """The function concatenate features (RelAI, Seismic, Seismic Inversion) with Facies
183
184      Args:
185          df_facies_wells (DataFrame): Facies
186          feature_list_RelAI_Seis_SeisInv (lisr): list of features
187
188      Returns:
189          facies_and_features (DataFrame): concatenated dataframe with facies and 4 features (Relative AI,
190          Seismic, Seismic Inversion, Geological Time)
191      """
192      def create_empty_lists(a):
193          for i in range(a):
194              yield []
195
196      XX0, XX1, XX2, XX3 = create_empty_lists(len(feature_list_RelAI_Seis_SeisInv_Depth))
197      YY1 = []
198
199      for every_well in range(len(df_facies_wells.columns)):
200
201          X0 = feature_list_RelAI_Seis_SeisInv_Depth[0].iloc[:,every_well]
202          X1 = feature_list_RelAI_Seis_SeisInv_Depth[1].iloc[:,every_well]
203          X2 = feature_list_RelAI_Seis_SeisInv_Depth[2].iloc[:,every_well]
204          X3 = feature_list_RelAI_Seis_SeisInv_Depth[3].iloc[:,every_well]
205
206          Y = df_facies_wells.iloc[:,every_well]
207
208          XX0.append(X0)
209          XX1.append(X1)
210          XX2.append(X2)
211          XX3.append(X3)
212
213          YY1.append(Y)
214
215      XX0 = pd.concat(XX0, ignore_index=False, axis=0)
216      XX1 = pd.concat(XX1, ignore_index=False, axis=0)
217      XX2 = pd.concat(XX2, ignore_index=False, axis=0)
218      XX3 = pd.concat(XX3, ignore_index=False, axis=0)
219
220      facies = pd.concat(YY1, ignore_index=False)
221
222      features = pd.concat([XX0, XX1, XX2, XX3], axis=1)
223
224      features = features.rename(columns = {0:'relai',
225                                            1:'seis',
226                                            2: 'seis_inv',
227                                            3: 'depth'
228                                            })
229      features


229      features
230
231      facies = pd.DataFrame(facies).rename(columns={0:'facies'})
232      facies
233
234      facies_and_features = pd.concat([facies, features], axis=1)
235      facies_and_features = facies_and_features.dropna(axis=0)
236
237      return facies_and_features
238
239
240
```

```
240
241 ##########################################################################################
242
243 def concat_features_RelAI_Seis_Envel_InstFreq_Depth(df_facies_wells, feature_list_RelAI_Seis_Envel_InstFreq_Depth):
244     """The function concatenate features (RelAI, Seismic, Envelope, InstFreq) with Facies
245
246     Args:
247         df_facies_wells (DataFrame): Facies
248         feature_list_RelAI_Seis_Envel_InstFreq (lisr): list of features
249
250     Returns:
251         facies_and_features (DataFrame): concatenated dataframe with facies and 5 features (Relative AI,
252         Seismic, Envelope, Instant Frequency, Geological Time)
253     """
254     def create_empty_lists(a):
255         for i in range(a):
256             yield []
257
258     XX0, XX1, XX2, XX3, XX4 = create_empty_lists(len(feature_list_RelAI_Seis_Envel_InstFreq_Depth))
259     YY1 = []
260
261     for every_well in range(len(df_facies_wells.columns)):
262
263         X0 = feature_list_RelAI_Seis_Envel_InstFreq_Depth[0].iloc[:,every_well]
264         X1 = feature_list_RelAI_Seis_Envel_InstFreq_Depth[1].iloc[:,every_well]
265         X2 = feature_list_RelAI_Seis_Envel_InstFreq_Depth[2].iloc[:,every_well]
266         X3 = feature_list_RelAI_Seis_Envel_InstFreq_Depth[3].iloc[:,every_well]
267         X4 = feature_list_RelAI_Seis_Envel_InstFreq_Depth[4].iloc[:,every_well]
268
269         Y = df_facies_wells.iloc[:,every_well]
270
271         XX0.append(X0)
272         XX1.append(X1)
273         XX2.append(X2)
274         XX3.append(X3)
275         XX4.append(X4)
276
277         YY1.append(Y)
278
279     XX0 = pd.concat(XX0, ignore_index=False, axis=0)
280     XX1 = pd.concat(XX1, ignore_index=False, axis=0)
281     XX2 = pd.concat(XX2, ignore_index=False, axis=0)
282     XX3 = pd.concat(XX3, ignore_index=False, axis=0)
283     XX4 = pd.concat(XX4, ignore_index=False, axis=0)
284
285     facies = pd.concat(YY1, ignore_index=False)
286
287     features = pd.concat([XX0, XX1, XX2, XX3, XX4], axis=1)
288
289     features = features.rename(columns = {0:'relai',
290                                           1:'seis',
291                                           2: 'envel',
292                                           3: 'inst_freq',
293                                           4: 'depth'
294                                           })
295     features
```

```
    facies = pd.DataFrame(facies).rename(columns={0:'facies'})
    facies

    facies_and_features = pd.concat([facies, features], axis=1)
    facies_and_features = facies_and_features.dropna(axis=0)

    return facies_and_features
```

```
##############################################################################################

def concat_features_RelAI_Seis_SeisInv_SpecDec(df_facies_wells, feature_list_RelAI_Seis_SeisInv_SpecDec):
    """The function concatenate features (RelAI, Seismic, Envelope, InstFreq) with Facies

    Args:
        df_facies_wells (DataFrame): Facies
        feature_list_RelAI_Seis_Envel_InstFreq (lisr): list of features

    Returns:
        facies_and_features (DataFrame): concatenated dataframe with facies and 5 features (Relative AI,
        Seismic, Envelope, Instant Frequency, Geological Time)
    """
    def create_empty_lists(a):
        for i in range(a):
            yield []

    XX0, XX1, XX2, XX3 = create_empty_lists(len(feature_list_RelAI_Seis_SeisInv_SpecDec))
    YY1 = []

    for every_well in range(len(df_facies_wells.columns)):

        X0 = feature_list_RelAI_Seis_SeisInv_SpecDec[0].iloc[:,every_well]
        X1 = feature_list_RelAI_Seis_SeisInv_SpecDec[1].iloc[:,every_well]
        X2 = feature_list_RelAI_Seis_SeisInv_SpecDec[2].iloc[:,every_well]
        X3 = feature_list_RelAI_Seis_SeisInv_SpecDec[3].iloc[:,every_well]

        Y = df_facies_wells.iloc[:,every_well]

        XX0.append(X0)
        XX1.append(X1)
        XX2.append(X2)
        XX3.append(X3)

        YY1.append(Y)

    XX0 = pd.concat(XX0, ignore_index=False, axis=0)
    XX1 = pd.concat(XX1, ignore_index=False, axis=0)
    XX2 = pd.concat(XX2, ignore_index=False, axis=0)
    XX3 = pd.concat(XX3, ignore_index=False, axis=0)

    facies = pd.concat(YY1, ignore_index=False)

    features = pd.concat([XX0, XX1, XX2, XX3], axis=1)

    features = features.rename(columns = {0:'relai',
                                          1:'seis',
                                          2: 'seis_inv',
                                          3: 'spec'
                                          })
    features

    facies = pd.DataFrame(facies).rename(columns={0:'facies'})
    facies

    facies_and_features = pd.concat([facies, features], axis=1)
    facies_and_features = facies_and_features.dropna(axis=0)
```

```
368
369         return facies_and_features
370
371
372
373
374
375
376 ##########################################################################################
377
378 def concat_features_RelAI_Seis_Envel_InstFreq_SeisInv(df_facies_wells, feature_list_RelAI_Seis_Envel_InstFreq_SeisInv):
379     """The function concatenate features (RelAI, Seismic, Envelope, InstFreq) with Facies
380
381     Args:
382         df_facies_wells (DataFrame): Facies
383         feature_list_RelAI_Seis_Envel_InstFreq (lisr): list of features
384
385     Returns:
386         facies_and_features (DataFrame): concatenated dataframe with facies and 5 features (Relative AI,
387         Seismic, Envelope, Instant Frequency, Geological Time)
388     """
389     def create_empty_lists(a):
390         for i in range(a):
391             yield []
392
393     XX0, XX1, XX2, XX3, XX4 = create_empty_lists(len(feature_list_RelAI_Seis_Envel_InstFreq_SeisInv))
394     YY1 = []
395
396     for every_well in range(len(df_facies_wells.columns)):
397
398         X0 = feature_list_RelAI_Seis_Envel_InstFreq_SeisInv[0].iloc[:,every_well]
399         X1 = feature_list_RelAI_Seis_Envel_InstFreq_SeisInv[1].iloc[:,every_well]
400         X2 = feature_list_RelAI_Seis_Envel_InstFreq_SeisInv[2].iloc[:,every_well]
401         X3 = feature_list_RelAI_Seis_Envel_InstFreq_SeisInv[3].iloc[:,every_well]
402         X4 = feature_list_RelAI_Seis_Envel_InstFreq_SeisInv[4].iloc[:,every_well]
403
404         Y = df_facies_wells.iloc[:,every_well]
405
406         XX0.append(X0)
407         XX1.append(X1)
408         XX2.append(X2)
409         XX3.append(X3)
410         XX4.append(X4)
411
412         YY1.append(Y)
413
414     XX0 = pd.concat(XX0, ignore_index=False, axis=0)
415     XX1 = pd.concat(XX1, ignore_index=False, axis=0)
416     XX2 = pd.concat(XX2, ignore_index=False, axis=0)
417     XX3 = pd.concat(XX3, ignore_index=False, axis=0)
418     XX4 = pd.concat(XX4, ignore_index=False, axis=0)
419
420     facies = pd.concat(YY1, ignore_index=False)
421
422     features = pd.concat([XX0, XX1, XX2, XX3, XX4], axis=1)
423
424     features = features.rename(columns = {0:'relai',
425                                           1:'seis',
426                                           2: 'envel',
427                                           3: 'inst_freq',
428                                           4: 'seis_inv'
429                                           })
430     features
431
432     facies = pd.DataFrame(facies).rename(columns={0:'facies'})
433     facies
```

```
434
435     facies_and_features = pd.concat([facies, features], axis=1)
436     facies_and_features = facies_and_features.dropna(axis=0)
437
438     return facies_and_features
```

Appendix 4 - Machine learning part

```
1
2
3   from sklearn.model_selection import train_test_split
4   from sklearn.linear_model import LogisticRegression
5   import pandas as pd
6   import numpy as np
7   from sklearn.model_selection import cross_val_score
8   import matplotlib.pyplot as plt
9   from sklearn.inspection import permutation_importance
10
11
12  from sklearn.metrics import f1_score
13  from sklearn.metrics import precision_recall_fscore_support
14  from sklearn import metrics
15  from sklearn.metrics import classification_report
16  from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
17  from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
18
```

```
##############################################################################################
def predict_2d_RelAI_Seis_SeisInv(df_facies, relai, seis, seis_inv, model):
    """This function uses ML model and attributes to predict 2D facies cross-section
    It takes the shape of the ground-truth facies and predicts facies using Rel AI, Seismic, Seis_Inv

    Args:
        df_facies (DataFrame): facies cross-section
        relai (DataFrame): 2D cross-section of Rel AI
        seis (DataFrame): 2D cross-section of Seismic
        seis_inv (DataFrame): 2D cross-section of Seis Inversion
        model: ML model

    Returns:
        map_facies_1 (array) : the predicted facies
        df_f_comparison (array): the filtered actual facies for comparison
    """

    # create empty Numpy array with the same shape as facies
    r, c = np.shape(df_facies)
    map_facies = np.zeros((r, c))
    map_facies[:] = np.NaN
    map_facies_1 = map_facies.copy()

    # create array for facies filter
    df_f_comparison = map_facies.copy()

    # for every trace (column)
    for every_col in range(len(df_facies.columns)):

        # concat every column
        new_df = pd.concat([relai.iloc[:,every_col],
                            seis.iloc[:,every_col],
                            seis_inv.iloc[:,every_col]]
                            , axis=1)

        # remove NaN values from trace
        new_df_1 = new_df.dropna(axis=0)

        # remove NaN values from Facies trace
        non_empty_facies = df_facies.iloc[:,every_col].dropna(axis=0)

        # predict
        map_facies[new_df_1.index, every_col] = model.predict(new_df_1)

        # Filter Facies to compare (since Facies and Features have different number of NaN and noNaN values)
        map_facies_1[non_empty_facies.index, every_col] = map_facies[non_empty_facies.index, every_col]
        df_f_comparison[new_df_1.index, every_col] = df_facies.iloc[new_df_1.index, every_col]

    return map_facies_1, df_f_comparison

```

```python


#############################################################################################
def predict_2d_Seis_SeisInv(df_facies, seis, seis_inv, model):

    # create empty Numpy array with the same shape as facies
    r, c = np.shape(df_facies)
    map_facies = np.zeros((r, c))
    map_facies[:] = np.NaN
    map_facies_1 = map_facies.copy()

    # create array for facies filter
    df_f_comparison = map_facies.copy()

    # for every trace (column)
    for every_col in range(len(df_facies.columns)):

        # concat every column
        new_df = pd.concat([seis.iloc[:,every_col],
                            seis_inv.iloc[:,every_col]]
                            , axis=1)

        # remove NaN values from trace
        new_df_1 = new_df.dropna(axis=0)

        # remove NaN values from Facies trace
        non_empty_facies = df_facies.iloc[:,every_col].dropna(axis=0)

        # predict
        map_facies[new_df_1.index, every_col] = model.predict(new_df_1)

        # Filter Facies to compare (since Facies and Features have different number of NaN and noNaN values)
        map_facies_1[non_empty_facies.index, every_col] = map_facies[non_empty_facies.index, every_col]
        df_f_comparison[new_df_1.index, every_col] = df_facies.iloc[new_df_1.index, every_col]

    return map_facies_1, df_f_comparison



```

```
###########################################################################################################
def predict_2d_RelAI_Seis_SeisInv_Depth(df_facies, relai, seis, seis_inv, depth,  model):

    # create empty Numpy array with the same shape as facies
    r, c = np.shape(df_facies)
    map_facies = np.zeros((r, c))
    map_facies[:] = np.NaN
    map_facies_1 = map_facies.copy()

    # create array for facies filter
    df_f_comparison = map_facies.copy()

    # for every trace (column)
    for every_col in range(len(df_facies.columns)):

        # concat every column
        new_df = pd.concat([relai.iloc[:,every_col],
                            seis.iloc[:,every_col],
                            seis_inv.iloc[:,every_col],
                            depth.iloc[:,every_col]]
                            , axis=1)

        # remove NaN values from trace
        new_df_1 = new_df.dropna(axis=0)

        # remove NaN values from Facies trace
        non_empty_facies = df_facies.iloc[:,every_col].dropna(axis=0)

        # predict
        map_facies[new_df_1.index, every_col] = model.predict(new_df_1)

        # Filter Facies to compare (since Facies and Features have different number of NaN and noNaN values)
        map_facies_1[non_empty_facies.index, every_col] = map_facies[non_empty_facies.index, every_col]
        df_f_comparison[new_df_1.index, every_col] = df_facies.iloc[new_df_1.index, every_col]

    return map_facies_1, df_f_comparison
```

```
184 ##########################################################################################################
185 def predict_2d_RelAI_Seis_Envel_InstFreq_Depth(df_facies, relai, seis, envel, inst_freq, depth,  model):
186 
187     # create empty Numpy array with the same shape as facies
188     r, c = np.shape(df_facies)
189     map_facies = np.zeros((r, c))
190     map_facies[:] = np.NaN
191     map_facies_1 = map_facies.copy()
192 
193     # create array for facies filter
194     df_f_comparison = map_facies.copy()
195 
196     # for every trace (column)
197     for every_col in range(len(df_facies.columns)):
198 
199         # concat every column
200         new_df = pd.concat([relai.iloc[:,every_col],
201                             seis.iloc[:,every_col],
202                             envel.iloc[:,every_col],
203                             inst_freq.iloc[:,every_col],
204                             depth.iloc[:,every_col]]
205                             , axis=1)
206 
207         # remove NaN values from trace
208         new_df_1 = new_df.dropna(axis=0)
209 
210         # remove NaN values from Facies trace
211         non_empty_facies = df_facies.iloc[:,every_col].dropna(axis=0)
212 
213         # predict
214         map_facies[new_df_1.index, every_col] = model.predict(new_df_1)
215 
216         # Filter Facies to compare (since Facies and Features have different number of NaN and noNaN values)
217         map_facies_1[non_empty_facies.index, every_col] = map_facies[non_empty_facies.index, every_col]
218         df_f_comparison[new_df_1.index, every_col] = df_facies.iloc[new_df_1.index, every_col]
219 
220     return map_facies_1, df_f_comparison
221 
222 
223 
224 
225 
226 
```

```python
###########################################################################################################
def predict_2d_RelAI_Seis_Envel_InstFreq(df_facies, relai, seis, envel, inst_freq, model):

    # create empty Numpy array with the same shape as facies
    r, c = np.shape(df_facies)
    map_facies = np.zeros((r, c))
    map_facies[:] = np.NaN
    map_facies_1 = map_facies.copy()

    # create array for facies filter
    df_f_comparison = map_facies.copy()

    # for every trace (column)
    for every_col in range(len(df_facies.columns)):

        # concat every column
        new_df = pd.concat([relai.iloc[:,every_col],
                            seis.iloc[:,every_col],
                            envel.iloc[:,every_col],
                            inst_freq.iloc[:,every_col]]
                            , axis=1)

        # remove NaN values from trace
        new_df_1 = new_df.dropna(axis=0)

        # remove NaN values from Facies trace
        non_empty_facies = df_facies.iloc[:,every_col].dropna(axis=0)

        # predict
        map_facies[new_df_1.index, every_col] = model.predict(new_df_1)

        # Filter Facies to compare (since Facies and Features have different number of NaN and noNaN values)
        map_facies_1[non_empty_facies.index, every_col] = map_facies[non_empty_facies.index, every_col]
        df_f_comparison[new_df_1.index, every_col] = df_facies.iloc[new_df_1.index, every_col]

    return map_facies_1, df_f_comparison
```

```
########################################################################################################################
def predict_2d_RelAI_Seis_SeisInv_Spec(df_facies, relai, seis, seis_inv, spec, model):

    # create empty Numpy array with the same shape as facies
    r, c = np.shape(df_facies)
    map_facies = np.zeros((r, c))
    map_facies[:] = np.NaN
    map_facies_1 = map_facies.copy()

    # create array for facies filter
    df_f_comparison = map_facies.copy()

    # for every trace (column)
    for every_col in range(len(df_facies.columns)):

        # concat every column
        new_df = pd.concat([relai.iloc[:,every_col],
                            seis.iloc[:,every_col],
                            seis_inv.iloc[:,every_col],
                            spec.iloc[:,every_col]]
                            , axis=1)

        # remove NaN values from trace
        new_df_1 = new_df.dropna(axis=0)

        # remove NaN values from Facies trace
        non_empty_facies = df_facies.iloc[:,every_col].dropna(axis=0)

        # predict
        map_facies[new_df_1.index, every_col] = model.predict(new_df_1)

        # Filter Facies to compare (since Facies and Features have different number of NaN and noNaN values)
        map_facies_1[non_empty_facies.index, every_col] = map_facies[non_empty_facies.index, every_col]
        df_f_comparison[new_df_1.index, every_col] = df_facies.iloc[new_df_1.index, every_col]

    return map_facies_1, df_f_comparison




def accuracy_score_cv(estimator, X, y, cv=10):

    from sklearn.model_selection import cross_val_score
    #Applying 10-fold cross validation
    accuracy_score_cv = cross_val_score(estimator=estimator, X=X, y=y, cv=cv)
    print("accuracy: ", np.mean(accuracy_score_cv))

    return np.mean(accuracy_score_cv)
```

```
# Feature importance
def feature_importance_plot(model, x_train, y_train, random_state):

    from sklearn.inspection import permutation_importance

    res = permutation_importance(model, x_train, y_train, scoring='accuracy', random_state=random_state)
    importance = res.importances_mean
    importance
    importance_res = pd.Series(importance, index=x_train.columns).sort_values(ascending=True)
    importance_res
    # Plot the results
    fig, ax = plt.subplots(figsize=(8,4))
    ax = importance_res.plot.barh()
    ax.set_title('Permutation importance', fontsize=14)
    ax.set_ylabel('Importance score', fontsize=14)
    ax.set_xlabel('Percentage, %', fontsize=12)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.grid()
    plt.show()
```

```python
# Remove data that were used for training
def confusion_matrix_prediction(df_facies_comparison, facies_pred, col_number, facies_class):

    from sklearn.metrics import f1_score
    from sklearn.metrics import precision_recall_fscore_support
    from sklearn import metrics
    from sklearn.metrics import classification_report
    from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
    from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay


    df_f_comparison_pd = pd.DataFrame(df_facies_comparison)
    df_f_comparison_col = df_f_comparison_pd.drop(df_f_comparison_pd.columns[col_number],axis = 1)

    map_facies_pd = pd.DataFrame(facies_pred)
    map_facies_col = map_facies_pd.drop(map_facies_pd.columns[col_number],axis = 1)

    actual_f = df_f_comparison_col.values[~(np.isnan(df_f_comparison_col))]
    predicted_f = map_facies_col.values[~(np.isnan(map_facies_col))]

    conf_matrix = metrics.confusion_matrix(actual_f, predicted_f)
    conf_matrix
    # print(pd.crosstab(actual_f, predicted_f))
    report_print = print(classification_report(actual_f, predicted_f))

    f1_score_per_class = f1_score(actual_f, predicted_f, average=None)
    accuracy_estimation = accuracy_score(actual_f, predicted_f)

    # Extract number of values of each class
    count_facies = np.unique(actual_f, return_counts=True)[1]


    # Plot confusion matrix
    # display_conf_matrix = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=labels_list.classes_)

    display_conf_matrix = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.array(facies_class))
    # plt.plot(figsize=(10, 4))
    display_conf_matrix.plot()
    plt.show()

    return report_print, f1_score_per_class, count_facies, accuracy_estimation
```

```
401 ##########################################################################################################
402 def confusion_matrix_3D(facies_pred, df_facies_comparison, model):
403 
404     from sklearn.metrics import f1_score
405     from sklearn.metrics import precision_recall_fscore_support
406     from sklearn import metrics
407     from sklearn.metrics import classification_report
408     from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
409     from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
410 
411     actual_f = df_facies_comparison[~(np.isnan(df_facies_comparison))]
412     predicted_f = facies_pred[~(np.isnan(facies_pred))]
413 
414     conf_matrix = metrics.confusion_matrix(actual_f, predicted_f)
415     conf_matrix
416 
417     f1_score_per_class = f1_score(actual_f, predicted_f, average=None)
418     accuracy_estimation = accuracy_score(actual_f, predicted_f)
419 
420     report_print = print(classification_report(actual_f, predicted_f))
421 
422     # Extract number of values of each class
423     count_facies = np.unique(actual_f, return_counts=True)[1]
424 
425     # Plot confusion matrix
426     display_conf_matrix = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=model.classes_)
427     display_conf_matrix.plot()
428     plt.show()
429 
430     return report_print, f1_score_per_class, count_facies, accuracy_estimation
431 
432 
433 
434 
435 
436 
```

```
437 ##################################################################################################################
438 def predict_2d_RelAI_Seis_Envel_InstFreq_SeisInv(df_facies, relai, seis, envel, inst_freq, seis_inv,  model):
439
440     # create empty Numpy array with the same shape as facies
441     r, c = np.shape(df_facies)
442     map_facies = np.zeros((r, c))
443     map_facies[:] = np.NaN
444     map_facies_1 = map_facies.copy()
445
446     # create array for facies filter
447     df_f_comparison = map_facies.copy()
448
449     # for every trace (column)
450     for every_col in range(len(df_facies.columns)):
451
452         # concat every column
453         new_df = pd.concat([relai.iloc[:,every_col],
454                             seis.iloc[:,every_col],
455                             envel.iloc[:,every_col],
456                             inst_freq.iloc[:,every_col],
457                             seis_inv.iloc[:,every_col]]
458                             , axis=1)
459
460         # remove NaN values from trace
461         new_df_1 = new_df.dropna(axis=0)
462
463         # remove NaN values from Facies trace
464         non_empty_facies = df_facies.iloc[:,every_col].dropna(axis=0)
465
466         # predict
467         map_facies[new_df_1.index, every_col] = model.predict(new_df_1)
468
469         # Filter Facies to compare (since Facies and Features have different number of NaN and noNaN values)
470         map_facies_1[non_empty_facies.index, every_col] = map_facies[non_empty_facies.index, every_col]
471         df_f_comparison[new_df_1.index, every_col] = df_facies.iloc[new_df_1.index, every_col]
472
473     return map_facies_1, df_f_comparison
474
475
476
477
478
```

```python

#########################################################################################################################
def predict_facies_3D(facies, relai_std, seis_std, envel_std, inst_freq_std, seis_inv_std, model):
    r, c, b = np.shape(facies)

    map_facies = np.zeros((r, c, b))
    map_facies[:] = np.NaN
    map_facies_1 = map_facies.copy()
    df_f_comparison = map_facies.copy()

    # for every trace (column)
    for every_x in range(facies.shape[0]):
        for every_y in range(facies.shape[1]):

            new_df = pd.concat(
                        [pd.DataFrame(relai_std[every_x, every_y]),
                        pd.DataFrame(seis_std[every_x, every_y]),
                        pd.DataFrame(envel_std[every_x, every_y]),
                        pd.DataFrame(inst_freq_std[every_x, every_y]),
                        pd.DataFrame(seis_inv_std[every_x, every_y])]
                        , axis=1)

    ########################################################################

            new_features = new_df.dropna(axis=0)

            #new_features = new_df
            non_empty_facies = pd.DataFrame(facies[every_x, every_y, :]).dropna(axis=0)

            map_facies[every_x, every_y, new_features.index] = model.predict(new_features)


            # Filter
            map_facies_1[every_x, every_y, non_empty_facies.index] = map_facies[every_x, every_y, non_empty_facies.index]
            df_f_comparison[every_x, every_y, new_features.index] = facies[every_x, every_y, new_features.index]

    # df_facies_wells = df_wells_from_section(df_f_copy, col_30)

    map_facies
    map_facies_1
    return map_facies_1, df_f_comparison

```

Appendix 5 - Plot 2D sections

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import colors
import matplotlib.pyplot as plt
import matplotlib




def plot_2D_section(data_file, extent_plot,  color_map='seismic', number_of_facies=5, list_of_wells=None):
    """This function plot 2D cross-section after non-existing  changed by NaN

    Args:
        data_file (DataFrame): cleaned 2D section
        extent_plot (list): List with depth and trace numbers for the plots
        color_map (str): Specify the color_map. Defaults to 'seismic'. Defaults to 'seismic'.
        number_of_facies (int): Specify the number of Facies. Defaults to 5.

    Returns:
        None
    """
    fig = plt.figure(figsize=(10, 4))
    ax = fig.add_subplot(1, 1, 1)
    # d = np.array(data_file)

    data = data_file
    extent = extent_plot
    d = np.array(data_file).flatten()

    if color_map =='seismic':
        custom_norm=colors.TwoSlopeNorm(vmin=min(d), vcenter=0, vmax=max(d))
        im = ax.imshow(data.T, origin='upper', cmap="seismic", extent=extent, aspect='auto', norm=custom_norm)
        fig.colorbar(im)


    elif color_map == 'facies':
        facies_name = ['','Coarse Sand', 'Sand', 'Fine Sand', 'Shale', 'Carbonate']
        facies_color = ['#FFFFFF' ,'#E69076', '#FFFF00', '#FFCC00', '#A6A6A6', '#8080FF']
        # cmap = matplotlib.colors.ListedColormap(['black', 'fuchsia', 'yellow', 'cyan', 'orange', 'red'])
        cmap = matplotlib.colors.ListedColormap(facies_color)
        bounds = [-1.5, -0.5, 0.5, 1.5, 2.5, 3.5, 4.5]
        norm = matplotlib.colors.BoundaryNorm(bounds, cmap.N)
        im = ax.imshow(data_file.T, cmap=cmap, aspect='auto', extent=extent, norm=norm, vmin=0-1.5, vmax=4+0.5)
        # plt.colorbar(im, cmap=cmap, norm=norm, boundaries=bounds)
        fig.colorbar(im, ticks=np.arange(0, number_of_facies))
        # ax.set_title('Facies', fontsize=18)

    elif color_map == 'seis_inv':
        im = ax.imshow(data, cmap="gist_rainbow", aspect='auto', extent=extent)
        fig.colorbar(im)



    else:
        im = ax.imshow(data, cmap="jet", aspect='auto', extent=extent)
        fig.colorbar(im)

```

```
60
61      if list_of_wells==None:
62          pass
63      else:
64          for well in list_of_wells:
65              ax.axvline(x = well, linewidth = 1, color ='black')
66
67
68      colormap1 = fig.axes[1]
69      colormap1.tick_params(labelsize=12)
70
71      ax.set_xlabel('trace number', fontsize=14)
72      ax.set_ylabel('TWT [ms]', fontsize=14)
73      plt.xticks(fontsize=12)
74      plt.yticks(fontsize=12)
75
76      plt.show()
77
78
79
80
81
82
```

```python
# Difference map

def difference_map(df_facies_comparison, facies_predicted, extent, list_of_wells=None):
    """ Function to plot difference map between the ground-truth and predicted facies.

    Args:
        df_facies_comparison (2D array): ground truth facies
        facies_predicted (2D array): predicted facies from ML models
        extent (list): List with depth and trace numbers for the plots
        list_of_wells (list): the list with wells location. Defaults to None.

    Returns:
        nothing

    """


    import matplotlib.pyplot as plt
    import matplotlib.colors

    fig = plt.figure(figsize=(10, 4))
    ax = fig.add_subplot(1, 1, 1)

    extent = extent # define extent
    # The difference
    np_facies_fact = df_facies_comparison
    np_facies_predicted = facies_predicted

    facies_difference = np.subtract(np_facies_fact, np_facies_predicted)

    df1 = pd.DataFrame(facies_difference)
    df1 = df1.apply(np.sign).replace({-4:1, -3:1, -2:1, -1:1, 0:0,
                                      1:1, 2:1, 3:1, 4:1
                                      })
    facies_difference_result = np.array(df1)



    cmap = matplotlib.colors.ListedColormap(['green', 'red'])
    im = ax.imshow(facies_difference_result, cmap=cmap, vmin=0, vmax=1, aspect='auto', extent=extent)

    if list_of_wells==None:
        pass
    else:
        for well in list_of_wells:
            ax.axvline(x = well, linewidth = 1, color ='black')

    ax.set_xlabel('trace number', fontsize=14)
    ax.set_ylabel('TWT [ms]', fontsize=14)
    ax.set_title('Difference map', fontsize=16)
    colormap1 = fig.axes[0]
    colormap1.tick_params(labelsize=12)
    plt.colorbar(im, ticks=[True, False])
    plt.legend()
    plt.show()

```

Appendix 6 - Plot 3D sections

```
from matplotlib import colors
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import pandas as pd




def plot_3D_cube(data, color_map='facies', number_of_facies=5, number_of_wells=7):
    x = np.indices(data.shape)[0]
    y = np.indices(data.shape)[1]
    z = np.indices(data.shape)[2]
    col = data.flatten()

    # 3D Plot
    fig = plt.figure(figsize=(10, 6))
    ax3D = fig.add_subplot(projection='3d')
    # ax3D = plt.axes(projection='3d')



    # We will randomly choose 7 wells for training
    # It should not cross faults
    start_x = [10, 60, 30, 25, 65, 50, 21]
    start_y = [15, 15, 20, 58, 60, 50, 35]
    start_z = [700, 700, 700, 700, 700, 700, 700]

    end_x = [10, 60, 30, 25, 65, 50, 21]
    end_y = [15, 15, 20, 58, 60, 50, 35]
    end_z  =[0, 0, 0, 0, 0, 0, 0]

    ax3D.set_xlabel('x')
    ax3D.set_ylabel('y')
    ax3D.set_zlabel('z')

    if number_of_wells == 7:
        for well in range(number_of_wells):
            ax3D.plot([start_x[well], end_x[well]], [start_y[well], end_y[well]],  zs=[start_z[well], end_z[well]], color='black', linewidth = 1)

    elif number_of_wells == None:
        pass


    if color_map == 'facies':

        if number_of_facies == 5:
            facies_name = ['','Coarse Sand', 'Sand', 'Fine Sand', 'Shale']
            facies_color = ['#FFFFFF' ,'#E69076', '#FFFF00', '#FFCC00', '#A6A6A6', '#8080FF']
            cmap = matplotlib.colors.ListedColormap(facies_color)
            bounds = [-1.5, -0.5, 0.5, 1.5, 2.5, 3.5, 4.5]
            norm = matplotlib.colors.BoundaryNorm(bounds, cmap.N)
            # # 3D Plot
            # fig = plt.figure()
            # ax3D = fig.add_subplot(projection='3d')
            # p3d = ax3D.scatter(x, y, z)
            # p3d = ax3D.scatter(x, y, z, c =data, cmap=cmap)
            p3d = ax3D.scatter(x, y, z, c=data, cmap=cmap)
```

```
            p3d = ax3D.scatter(x, y, z, c=data, cmap=cmap)
            # fig.colorbar(p3d, ticks=np.arange(0, 5))
            fig.colorbar(p3d)

        # im = ax.imshow(data.T, cmap=cmap, aspect='auto', extent=extent, norm=norm, vmin=0-1.5, vmax=4+0.5)
        # # plt.colorbar(im, cmap=cmap, norm=norm, boundaries=bounds)
        # fig.colorbar(im, ticks=np.arange(0, 5))
        # ax.set_title('Facies section')

        if number_of_facies == 4:
            facies_name = ['Coarse Sand', 'Sand', 'Fine Sand', 'Shale']
            facies_color = ['#E69076', '#FFFF00', '#FFCC00', '#A6A6A6', '#8080FF']
            cmap = matplotlib.colors.ListedColormap(facies_color)
            bounds = [-0.5, 0.5, 1.5, 2.5, 3.5, 4.5]
            norm = matplotlib.colors.BoundaryNorm(bounds, cmap.N)
            # # 3D Plot
            # fig = plt.figure()
            # ax3D = fig.add_subplot(projection='3d')
            # p3d = ax3D.scatter(x, y, z)
            # p3d = ax3D.scatter(x, y, z, c =data, cmap=cmap)
            p3d = ax3D.scatter(x, y, z, c=data, cmap=cmap)
            fig.colorbar(p3d, ticks=np.arange(0, 5))

    elif color_map == 'relai':
        # # 3D Plot
        # fig = plt.figure()
        # ax3D = fig.add_subplot(projection='3d')
        # p3d = ax3D.scatter(x, y, z)
        # p3d = ax3D.scatter(x, y, z, c =data, cmap=cmap)
        p3d = ax3D.scatter(x, y, z, c=data, cmap='jet')
        # fig.colorbar(p3d, ticks=np.arange(0, 5))
        fig.colorbar(p3d)

    else:
        # fig = plt.figure()
        # ax3D = fig.add_subplot(projection='3d')
        # p3d = ax3D.scatter(x, y, z)
        # p3d = ax3D.scatter(x, y, z, c =data, cmap=cmap)
        p3d = ax3D.scatter(x, y, z, c=data, cmap='jet')
        # fig.colorbar(p3d, ticks=np.arange(0, 5))
        fig.colorbar(p3d)

    plt.show()
```

```python

##############################################################################################
def difference_map_3D(df_facies_comparison, facies_predicted, number_of_wells=7):
    d, e, f = np.shape(df_facies_comparison)
    facies_difference_map = np.zeros((d, e, f))
    facies_difference_map[:] = np.NaN


    # The difference map
    np_facies_fact = df_facies_comparison
    np_facies_predicted = facies_predicted


    facies_difference = np.subtract(np_facies_fact, np_facies_predicted)
    # np.unique(facies_difference)
    facies_difference_map = np.where(((facies_difference >= 1) | (facies_difference <= -1)),  1, facies_difference)


    x = np.indices(df_facies_comparison.shape)[0]
    y = np.indices(df_facies_comparison.shape)[1]
    z = np.indices(df_facies_comparison.shape)[2]
    col = df_facies_comparison.flatten()

    # 3D Plot
    fig = plt.figure(figsize=(8, 8))
    #fig1=plt.figure(figsize=(8,5))
    ax3D = fig.add_subplot(projection='3d')



    start_x = [10, 60, 30, 25, 65, 50, 21]
    start_y = [15, 15, 20, 58, 60, 50, 35]
    start_z = [700, 700, 700, 700, 700, 700, 700]

    end_x = [10, 60, 30, 25, 65, 50, 21]
    end_y = [15, 15, 20, 58, 60, 50, 35]
    end_z  =[0, 0, 0, 0, 0, 0, 0]

    ax3D.set_xlabel('x')
    ax3D.set_ylabel('y')
    ax3D.set_zlabel('z')

    for well in range(number_of_wells):
        ax3D.plot([start_x[well], end_x[well]], [start_y[well], end_y[well]],  zs=[start_z[well], end_z[well]], color='black', linewidth = 1)


    cmap = matplotlib.colors.ListedColormap(['green', 'red'])
    bounds = [-0.5, 0.5, 1.5]
    norm = matplotlib.colors.BoundaryNorm(bounds, cmap.N)
    p3d = ax3D.scatter(x, y, z, c=facies_difference_map, cmap=cmap)
    # fig.colorbar(p3d, ticks=np.arange(0, 5))
    fig.colorbar(p3d, ticks=np.arange(0, 2))
    plt.show()
```