FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation:
Computational Engineering

The spring semester, 2022

Open

Author: Einar Salomonsen

Einar Salomonsen
(signature author)

Course coordinator: Aksel Hjort

Supervisor(s): Nestor Cardozo (UiS), Lothar Schulte (Shlumberger)

Thesis title:
ML-based porosity modeling tested on synthetic and subsurface data

Credits (ECTS): 30

Keywords:
F3,
Machine learning,
Sequential Gaussian Simulation,
Synthetic Models,
Porosity,
Impedance

Pages: 90

+ appendix: 66

Stavanger, 14/06/2022
date/year

ML-based porosity modeling tested on synthetic and subsurface data

Einar Salomonsen

2022

Contents

Abstract

This thesis investigates if synthetic porosity models are useful as a basis for comparison between machine learning (ML) approaches to porosity prediction. In addition to the ML methods, the sequential gaussian simulation (SGS) geostatistical method is used as a benchmark. The synthetic models are porosity and impedance cubes constructed from the F3 dataset (offshore Netherlands) well-logs, to mimic specific geological geometries including a sedimentary wedge and a normal fault. Based on the performance of the different methods on the synthetic models, a porosity prediction is performed on the actual F3 dataset as well. The prediction methods discussed are SGS, and ML methods such as KNN-regression, lasso-regression, random forest-regression, and shallow neural network. The geostatistical and geophysical methods are run using Petrel, and the ML methods using Python. ML methods are better at minimizing the error while missing much of the detail of the SGS method. However, for the F3 dataset, random forest appears to capture more details than the other methods. The synthetic models provided a better basis for comparison of the different methods, however the workflow requires improvement.

Acknowledgements

I must sincerely thank my supervising Professor Nestor Cardozo for assisting in the organization and advise in the writing of this thesis. I also thank my external supervisor Doctor Lothar Schulte for his patience and expertise in instructing and reviewing the geostatistics and geophysics portions of this thesis.

1 Introduction

Over the past decade, Machine learning (ML) methods have been applied to many fields of sciences in order to improve data-based workflows. In geosciences, one works with large amounts of data. Thus, it is natural to investigate if ML methods can be an improvement to current subsurface interpretation methods.

This thesis focuses on methods for predicting porosity, which is the ratio between the pore space and the total volume in a rock. The pore space is the volume that can be filled with fluids, and in reservoirs it correlates to some degree with permeability, which is a measure of how easily fluids flow through the reservoir. This makes porosity estimation important for determining migration routes and reservoir volumes for valuable fluids such as oil, as well as the storage of other types of energy (e.g., hydrogen), or the sequestration of CO_2 in the subsurface. The reservoir volume is of great importance for making risk evaluations on whether a prospect is worth exploring.

In the petroleum industry, the available subsurface data are usually well-logs and seismic data. The well-logs are localized and follow a specific path, while seismic data cover a larger, often 3D volume. Additionally, the resolution in well-logs (cm) is about three orders of magnitude finer than seismic. These differences make it challenging to extrapolate well-log data into a 3D seismic volume, as the distances between wells are often large, and the difference in resolution between these two datasets is significant.

While it is possible to make and evaluate ML porosity models using well-logs and seismic data, it is difficult to truly evaluate the performance of these methods without a "known" porosity cube, given the varied geometry and complexity of the subsurface. For this reason, to test the effectiveness of ML for porosity prediction, I use synthetic models which can be derived from well data and typically consists of porosity, seismic and P-impedance cubes. In addition, the synthetic datasets can help us to better understand how the ML models react to different well locations and distributions, seismic signal to noise ratio, and subsurface geometries.

This thesis will compare porosity estimation methods using the classical "sequential Gaussian simulation" approach versus the ML approach. To compare the performance of these two approaches, synthetic, artificially created porosity sections are made as validation sets using established geostatistical methods. The synthetic models are based on a real dataset from offshore Netherlands, the F3 block. The theoretical background of this thesis is thus divided in two parts: the geostatistical part and the machine learning part. Geostatistics is performed using the program Petrel by Schlumberger. Machine learning is performed using Python with the aid of open source modules.

1.1 Thesis Structure

This thesis will first cover the relevant theory for both geostatistical/geophysical methods and machine learning as stated before. The geostatistical/geophysical part includes seismic inversion, the F3 dataset, and geological geometries which are required to build the synthetic models. This part also covers sequential Gaussian simulation as the geostatistical prediction method. The ML theory portion covers all the methods used, which include lasso, KNN, random forest and neural networks. This part also includes the Python tools used and an explanation of cross-validation. The second part is the methodology covering the steps taken to get the results. This includes the classic porosity prediction of the F3 dataset, and the construction and classic prediction of the synthetic models using geostatistical/geophysical theory. The methodology of the ML part covers predictor extraction/editing and hyper-parameter tuning. The results of the predictions are presented and compared in the Results chapter. Finally the results and the overall process of synthetic model construction and porosity prediction are discussed.

2 Background theory

2.1 Geophysics

2.1.1 Seismic inversion and impedance

Seismic has little correlation to porosity directly. A much more useful attribute is acoustic impedance, which typically correlates with porosity. Furthermore, acoustic impedance can be extended from the well-logs to the area of interest covered by the seismic cube, using seismic attributes and seismic inversion [1].

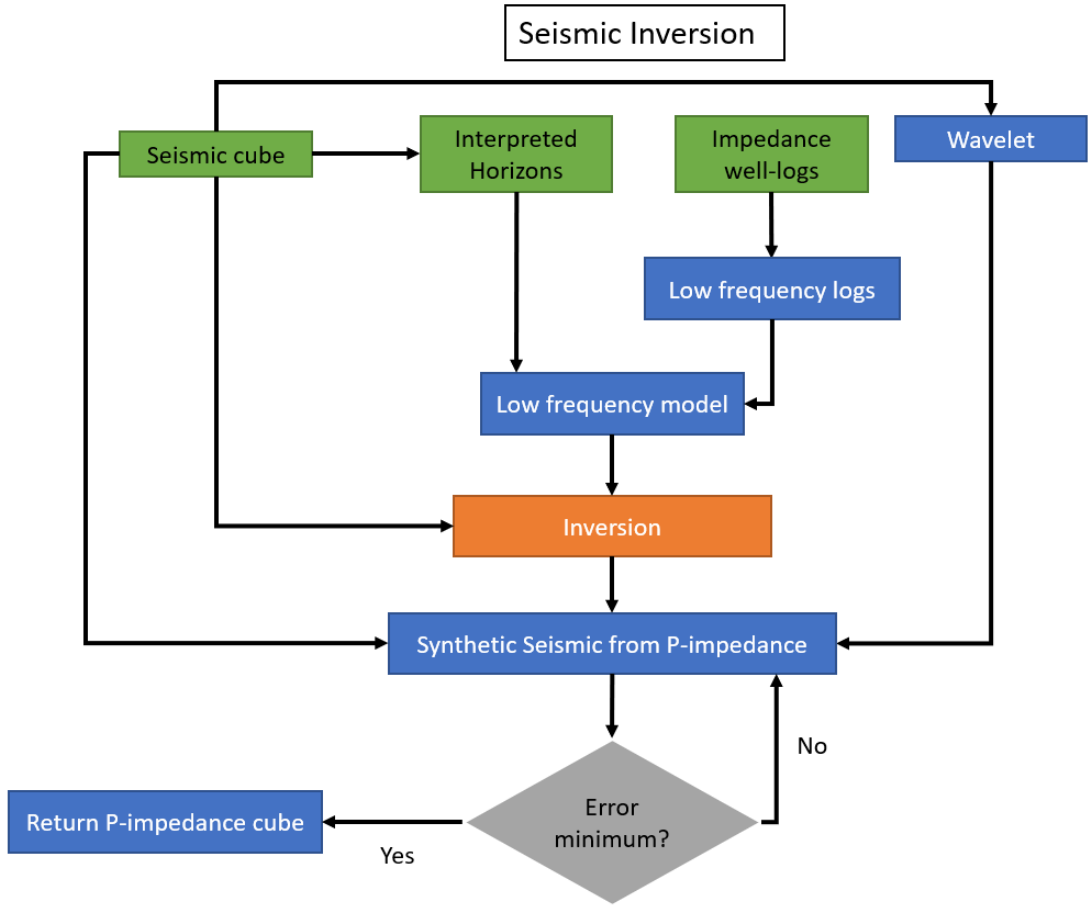

Figure 1: Seismic inversion workflow. The error minimum refers to the local minimum of the cost function that needs to be found.

The general workflow of seismic inversion is shown in Figure 1. Seismic inversion uses a low frequency model (LFM) made by applying a low-pass filter to the acoustic impedance well-logs and guided by the interpreted seismic horizons [1]. The LFM addresses the missing low frequencies of the seismic cube, which are necessary for reliable estimation of the acoustic impedance values.

The process of going from the LFM to the acoustic impedance is applied trace by trace. The LFM traces are modified, then a wavelet is used on the modified LFM trace to calculate the synthetic seismic trace [1]. The synthetic seismic trace is then compared to the real seismic trace using a cost function explained later. The difference between the synthetic and real seismic trace is caused by errors in the estimated impedance trace, and therefore it should be minimized by changing the impedance trace. If the difference is large, the impedance trace is modified and used to calculate a new synthetic seismic. This optimization loop continuous until a local minimum of the cost function is found [1].

The wavelet used for the inversion is an estimation of the source wave used during seismic acquisition. The wavelet is used as a filter to calculate the synthetic seismic from the reflectivity derived from the acoustic impedance. Statistical and deterministic wavelets are considered in this thesis. The primary difference is that the statistical wavelets are constructed based only on the seismic data. Deterministic wavelets are made based on both the seismic data and the well-log data [2].

The construction of the statistical wavelet can be summarized in three steps as displayed in Figure 2. Tapered auto-correlation is used on the traces to address the seismic noise and control the wavelet length. Then the auto-correlated traces are converted into a power spectrum (one spectrum per trace). Finally, the averaged power spectrum is transformed into the time domain giving the wavelet. The deterministic wavelet is based on the seismic trace and the reflectivity derived from the impedance well-logs. The autocorrelation of the reflectivity and the cross correlation between the reflectivity and the seismic trace are calculated. Then these values are tapered and converted into the frequency domain where the amplitude and the phase spectrum of the wavelet is derived. The deterministic wavelet has phase information, as opposed to the statistical wavelet which is zero-phase

and has no phase information [3, 2, 4].

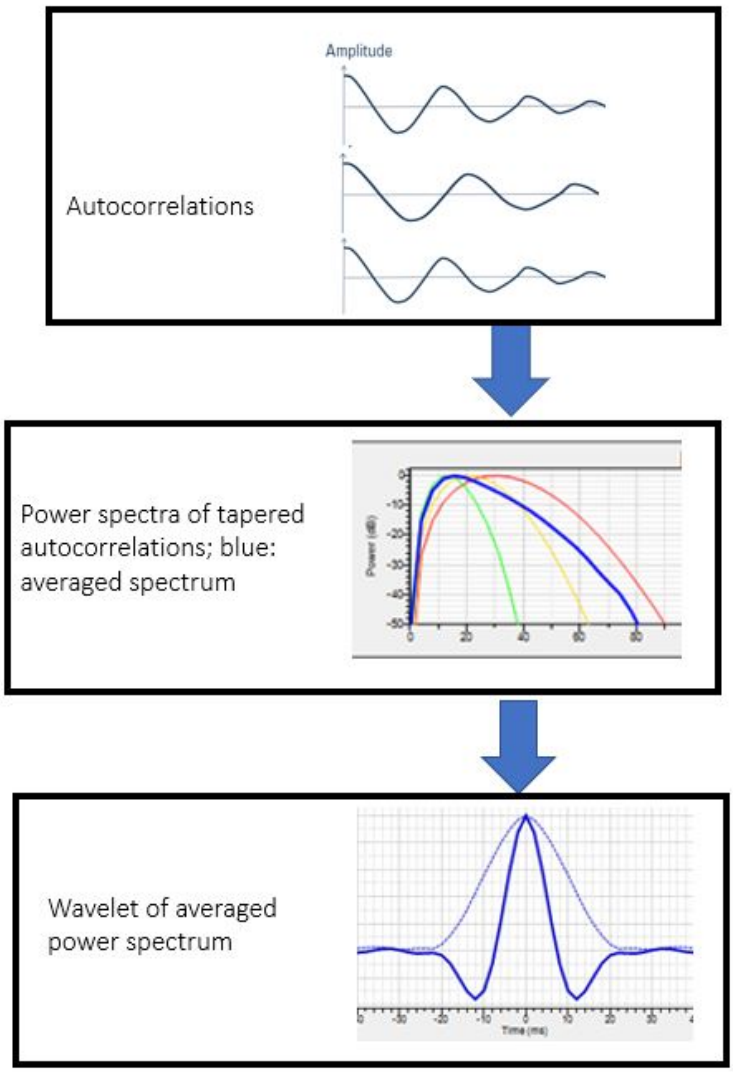

Figure 2: Workflow for constructing a statistical wavelet from seismic data, summarized in three steps. Based on theory from [4] and [3]

The formula for evaluating the quality of the inversion is a cost function with four terms. The first term penalizes for the difference between the synthetic and real seismic. The second term measures the horizontal variation of the impedance. The third term penalizes for the P-impedance deviating from the LFM. Finally, the fourth term determines the number of significant reflections, which are

the reflections that exceed a certain absolute amplitude. This term also penalizes the model for the number of significant reflections that exceed a chosen threshold [1]. The seismic inversion delivers an acoustic impedance cube that covers the area of interest and, it is the main predictor used for determining porosity outside the wells.

2.1.2 Sequential Gaussian simulation, Co-kriging and Variograms

To compare the results of the ML models, a porosity estimation that uses basic geostatistics is used. This estimation involves populating a 3D grid with porosity guided by a P-impedance volume. Taking advantage of the correlation between porosity and acoustic impedance, collocated co-simulation is used based on sequential Gaussian simulation (SGS) to derive the porosity. This method is implemented in modeling software (Petrel, RMS etc) and often applied in the industry for static reservoir modeling [5].

Sequential Gaussian simulation as described by [6] involves the following workflow (Figure 3): Transform the data into a normal Gaussian distribution. Loop over all grid cells randomly. For each cell perform kriging using the weights calculated from the variogram and a coefficient calculated from the correlation between the porosity and the impedance. From kriging, one should get the kriging value and the kriging variance at each grid cell. Then, the Gaussian distribution is derived at each grid cell using the results of kriging. The Gaussian distribution represents the uncertainty of the kriged value. Then, a random value is drawn and is applied to the cumulative distribution function, which is found by integrating the Gaussian distribution. This random value is assigned to the grid cell. When this is done to all the grid points, they are transformed back to the original distribution [6, 5].

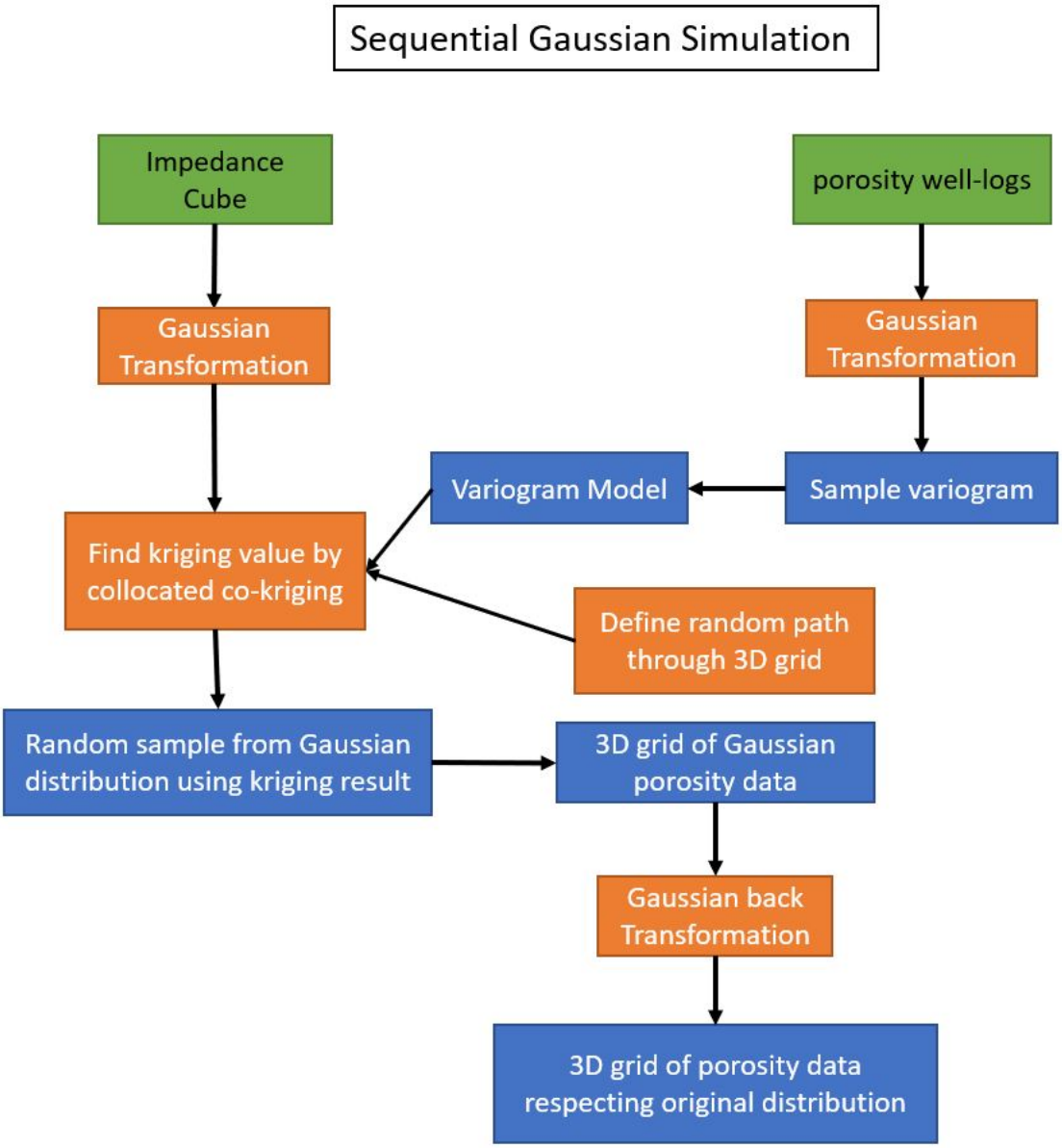

Figure 3: Workflow for the sequential Gaussian simulation.

In Petrel, all modeling is performed in depo-time. This means that all horizons are flattened and the fault displacement is removed. Then, SGS modeling is performed and the result is transformed back to the original domain [5]. This way, the interpreted geological time and it's relationship to

the well-logs is honored.

To explain the concept of the variogram, it is useful to separate it into two main components: the sample variogram and the variogram model. The sample variogram uses known data to calculate the semi-variance on a horizontal or vertical axis, sorted by the spatial distance [7]. The calculation of the semi-variance is:

$$\gamma(h) = \frac{1}{2N(h)} * \sum_{i=1}^{N(h)} (Z(x_i + h) - Z(x_i)^2 \quad (1)$$

Here $N(h)$ is the number of data pairs, h is the spatial distance between x_i and $x_i + h$, and $Z(x)$ is the value of a rock property at location x.

Figure 4 shows an example of the sample variogram based on theory from [7, 6]. The plots on the left show the distribution of the data to be analyzed. The three plots show the same data distribution, each one highlighting data-pairs of different separation distance, as shown by the two-way arrows connecting the data-pairs. On the plot to the right, the distance between the data pairs is plotted along the x axis, and the average semi-variation of the data-pairs along the y-axis (Equation 1).

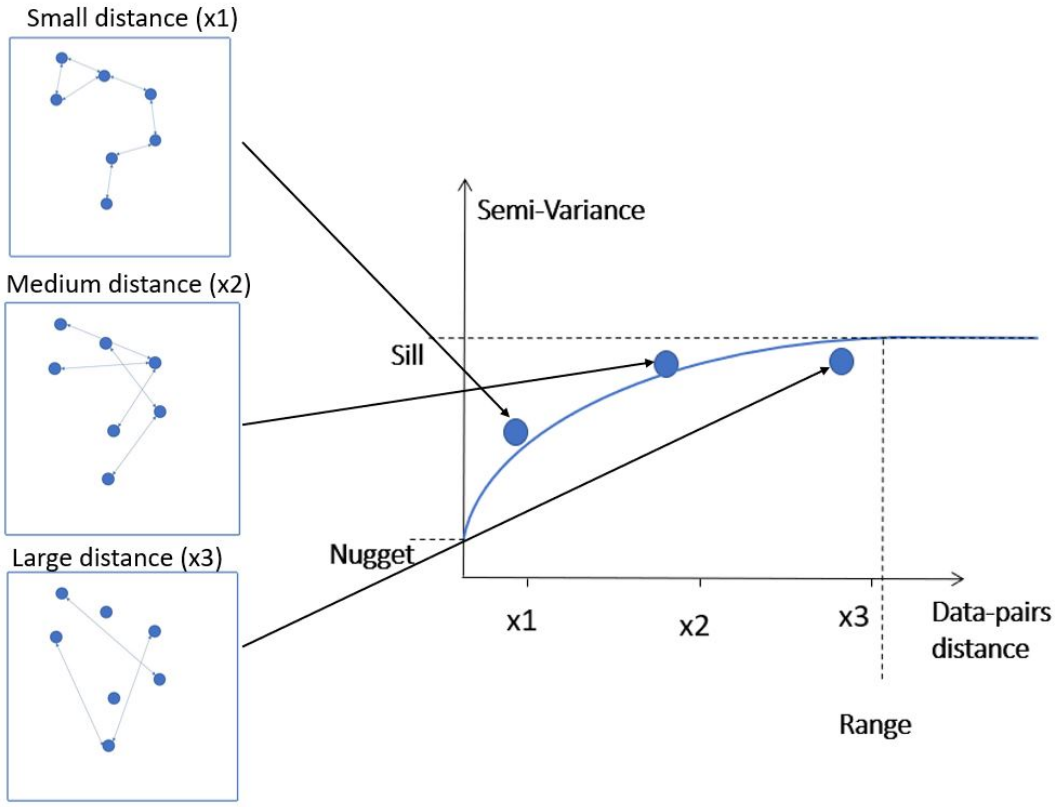

Figure 4: Data distribution with highlighted different data-pairs distances (left), sample variogram (dots on right), and variogram model (blue line on right). The sill, nugget, and range define the variogram model. Based on theory from sources [7, 6]

The blue line fitting the points on the distance versus semi-variance graph (Figure 4) is the variogram model. The model is described by an analytical function which is controlled by the sill, nugget and range parameters. The sill is the plateau of the variogram model, representing the maximum semi-variance. The nugget is the intercept of the variogram model representing the data uncertainty. The range can be interpreted as the separation distance where data pairs no longer correlate. The rock property is regarded as anisotropic when the variogram range is different in two perpendicular horizontal directions. For example, the variogram range of grain size is typically smaller perpendicular to a river than along the river flow axis [7].

By fitting a variogram model (an analytical function) to the sample variogram, the ranges in the horizontal and vertical directions can be estimated. There are several variogram models, the most common ones are exponential, spherical and Gaussian (Figure 5). All variogram models made in this thesis share some properties. They increase from data pairs distance equal zero to distance equal the range. At distances greater than the range, the model value is equal to the sill. Furthermore, the variogram model starts at zero to assume zero variation in rock properties that are at the exact same point (nugget = 0). This implies the assumption that the data are error free [7].

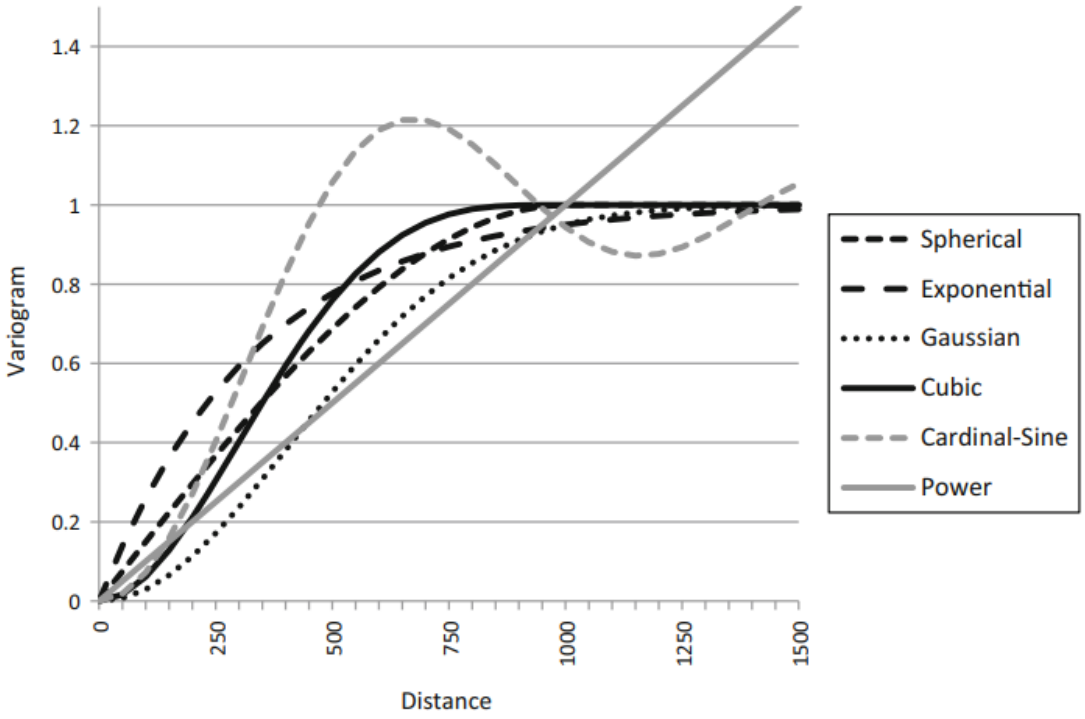

Figure 5: Most common analytical variogram models. In all models but the cardinal-sine and power models, the range is 1000. From [8].

Kriging is a weighted sum algorithm [6, 5]. It follows the equation:

$$z(x_0) = \sum_i^n \lambda_i z(x_i) \quad (2)$$

x_0 is the point of computation, n is the number of datapoints, $z(x_i)$ is the data value at point x_i, and λ_i is the weight assigned to the value at point x_i, determined by the variogram model.

The weights should decrease up to the range, from which the weight value is zero. This is for weights related to hard data, such as the porosity well-logs. If a grid point is outside the data range, the kriging results would be zero. Simple kriging addresses this issue by involving the mean value M provided by all data points [7, 6]. The equation is:

$$z(x_0) = \sum_i^n \lambda_i z(x_i) + [1 - \sum_i^n \lambda_i]M \quad (3)$$

The first term in this equation is the same as in Eq 2. However, there is now a second term which is proportional to the mean M. The influence of this term increases when the sum of weights decreases. This implies that at points away from known data, the simple kriging algorithm defaults to the mean of the data, removing a possible trend [7, 6].

Another kriging algorithm is collocated co-kriging, which is the algorithm used in this thesis. This algorithm includes soft data which guides the kriging. It uses the correlation coefficient between the soft data (P-impedance) and hard data (porosity) as an alternative to the mean (M) used in simple kriging [7, 6].

Co-kriging allows for the population of a 3D grid to use both well-log data and volume data as secondary input or soft data. This means that it respects the distribution of the hard data while correlating well data to the soft data. The kriging results are used for constructing a Gaussian distribution, using the kriging mean and variance. For each point on the 3D grid to be populated, a random sample is drawn from this distribution [7, 6]. The random number generator is controlled by the user-selected seed number. Changing only the seed will change the result. Conversely if the seed and data remain the same, the result will not change when re-running the algorithm. As an example,

Figure 6 shows three runs of the SGS using the exact same parameters and data, but different seeds. This shows that the seed controls the distribution of the high and low porosity values. When one value has been drawn for every point on the 3D grid, the grid is transformed back from the Gaussian distribution to the distribution given by the well-logs [6].

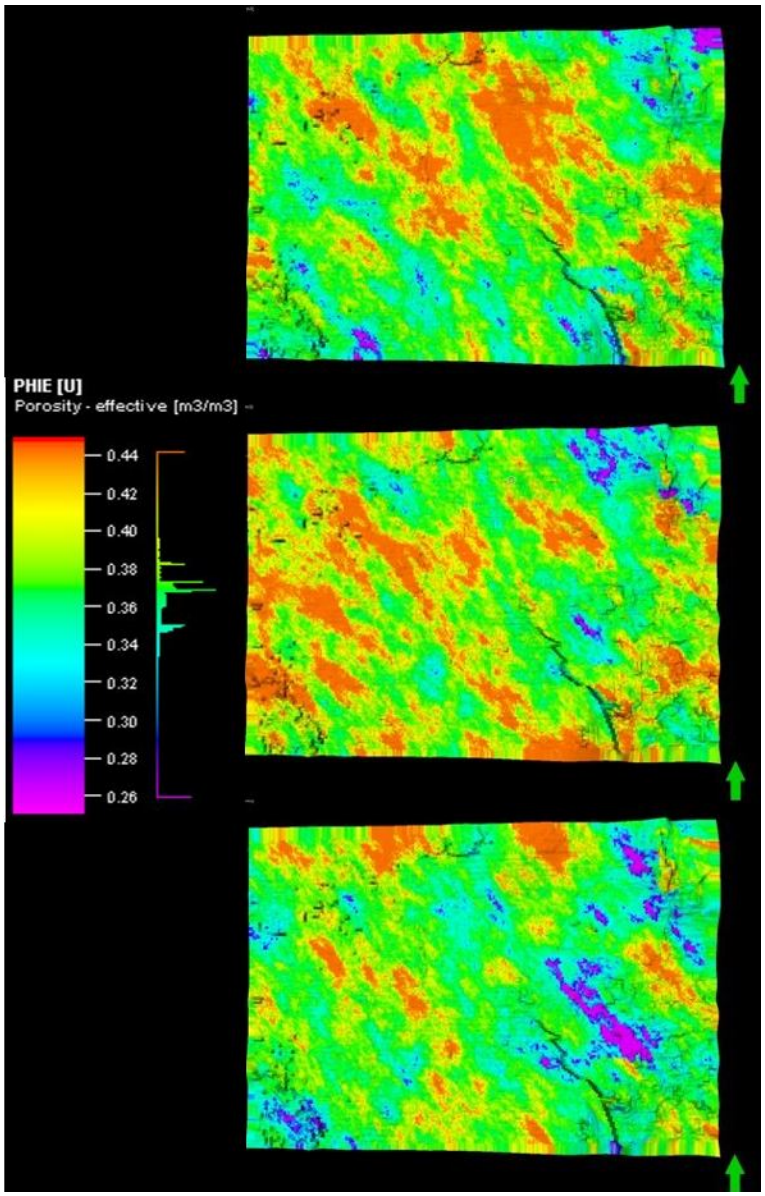

Figure 6: Three runs of sequential Gaussian simulation using the same parameters and data, but different seeds. The simulations were performed in Petrel using the F3 dataset. The porosity values are indicated by the different colors in the porosity scale. The random draws produce significantly different porosity models.

2.1.3 F3 block

The F3 block is a public dataset provided by NAM(Nederlandse Aardolie Maatschappij `https://www.nam.nl/`) and NLOG (Nederlandse Olie- en Gasportaal Hoofdnavigatie `https://www.nlog.nl/`), and further developed by dGB Earth Sciences `dgbes.com`. The dataset includes seismic attribute cubes and several wells with porosity and acoustic impedance (AC impedance) logs. dGB Earth Sciences also provides interpreted seismic horizons [9].

The F3 block is located in the Dutch sector of the North Sea, as shown in Figure 7. The seismic section used in this thesis is displayed in Figure 8 with three interpreted horizons. The location of the section is shown in Figure 9. This figure shows the surface outline of the seismic cube, and the two wells with the AC impedance logs. The prospect for oil and gas is located in Upper Jurassic to Lower Cretaceous sediments. These sediments were deposited in a fluviodeltaic system, and exhibit clinoform wedge-type geometries (Figure 8). The Late Permian Zechstein group is present at the bottom of the section where the AC impedance well-log displays in black (Figure 8). This group is known for evaporites that form salt domes [9].

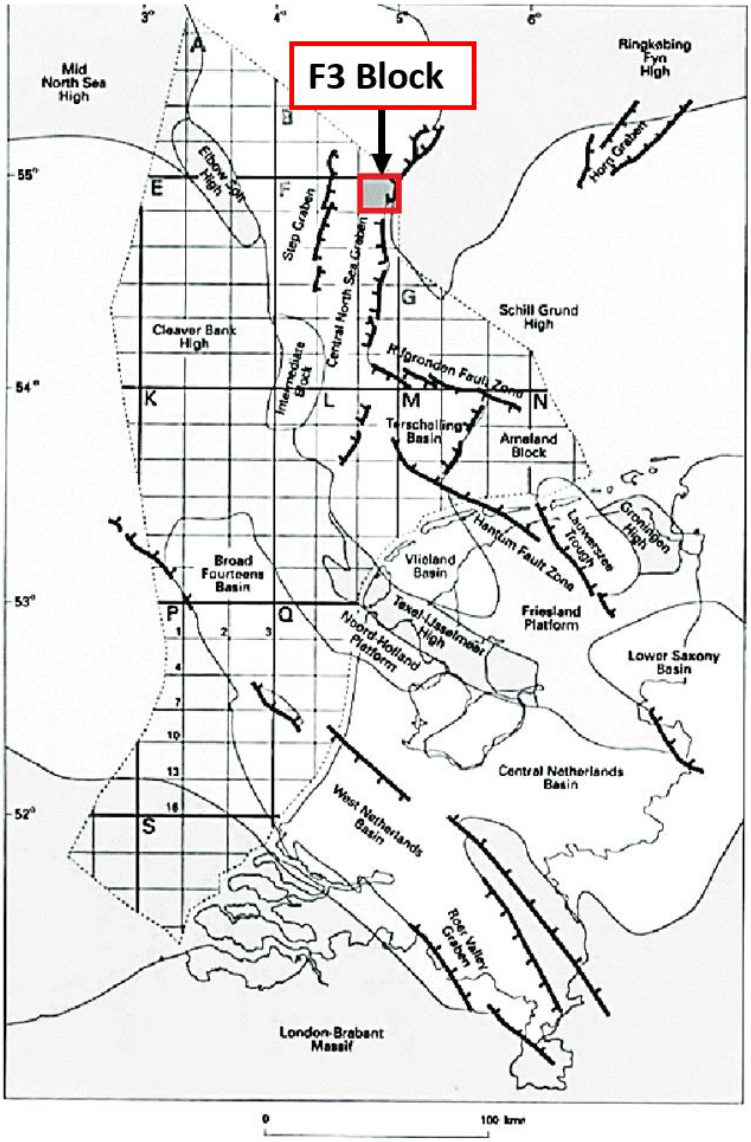

Figure 7: Location of the F3 block (red square) in the Dutch offshore sector. This map was edited from [10] and [11].

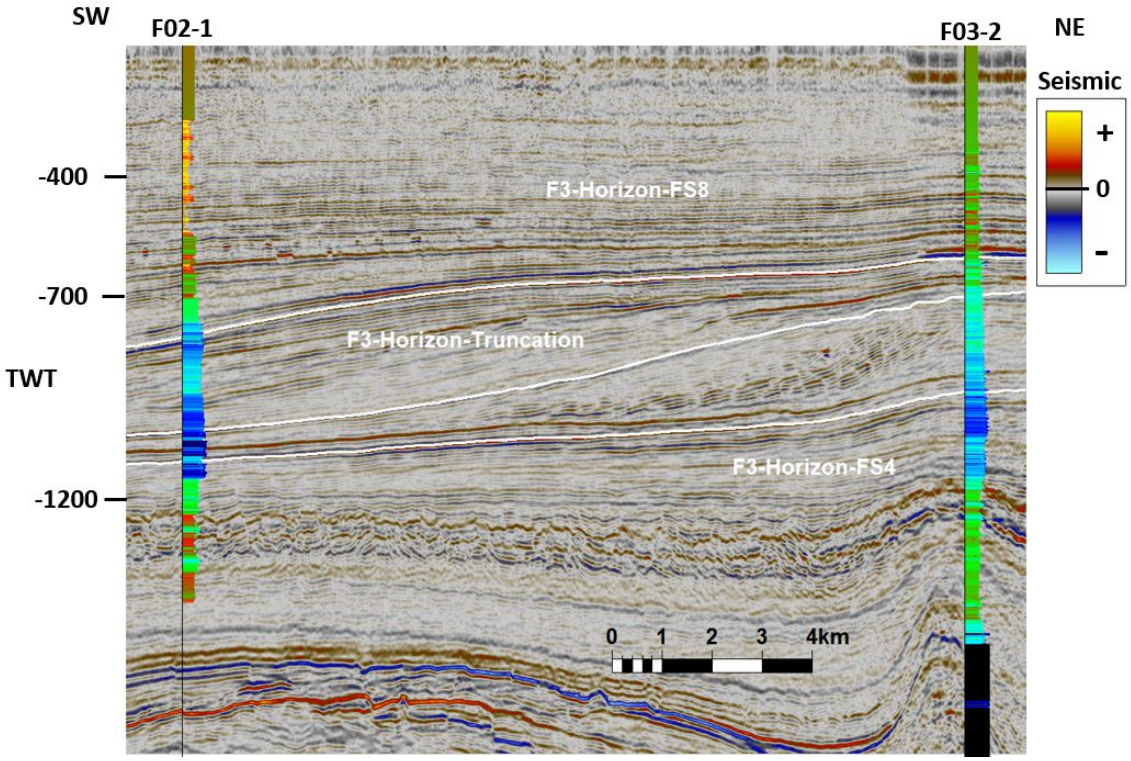

Figure 8: Seismic section containing the wells F02-1 and F03-2. The log displayed in both wells is AC impedance. Well F03-2 reaches the evaporites of the Zechstein group, whose AC impedance displays black.

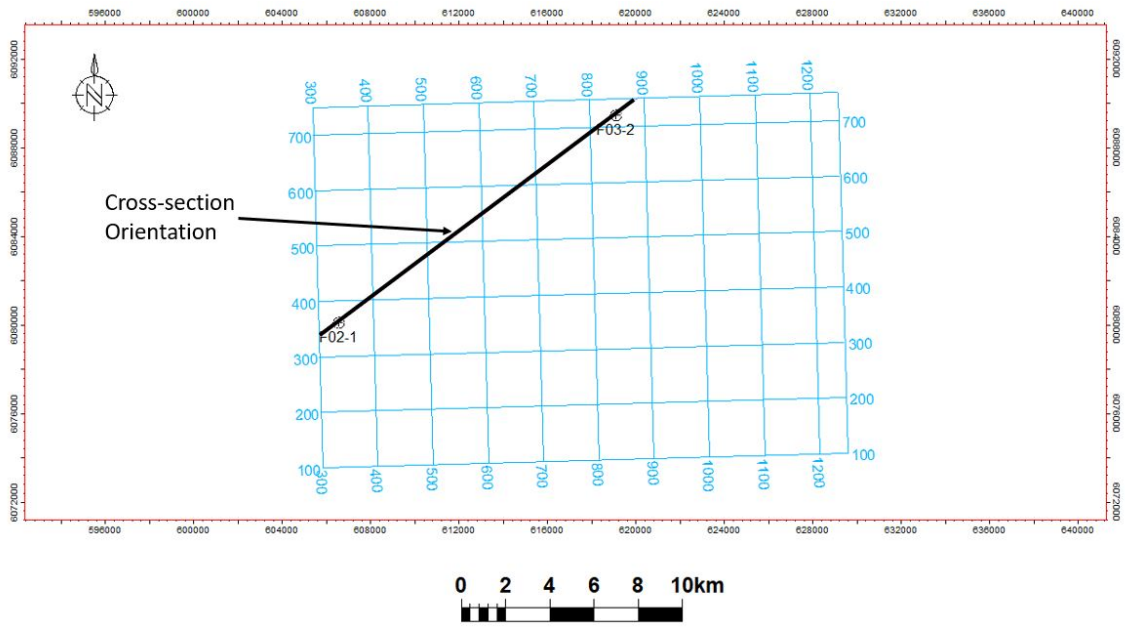

Figure 9: Location of the section in figure 8. The section intersects both wells F02-1 and F03-2.

The simplest synthetic model used in this thesis is the homogeneous wedge model. This model is made using one well, F02-1. An additional well, F03-2 is used to construct the porosity, impedance, and seismic sections of the other two synthetic models, the heterogeneous wedge and fault models. These wells are also used for the porosity estimation of the actual F3 section. The wells and their AC impedance and porosity well-logs are shown in Figure 10. These logs show a negative correlation between acoustic impedance and porosity, i.e. as impedance increases, porosity decreases.

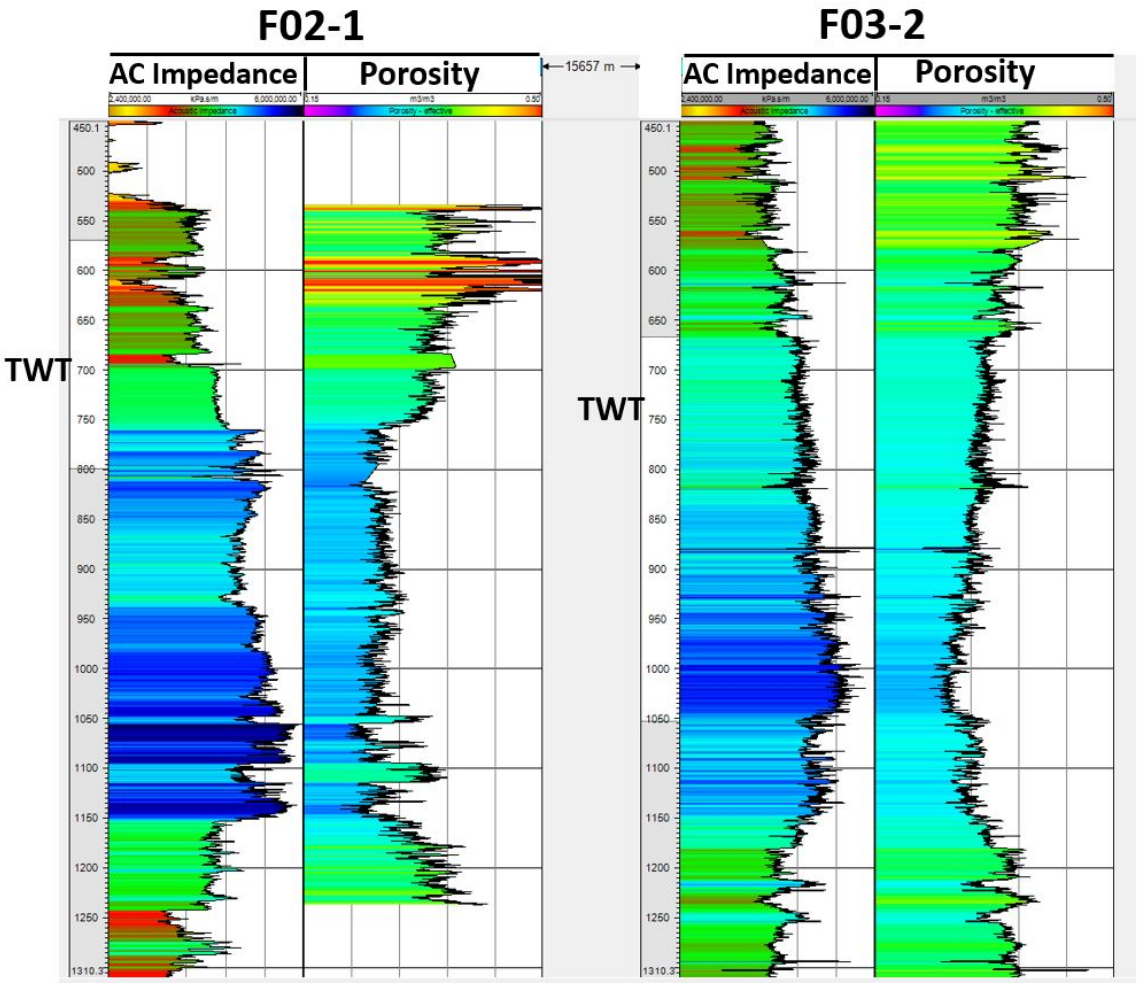

Figure 10: The AC impedance and porosity logs of the wells F02-1 and F03-2.

2.1.4 Geological geometries

Varying subsurface geometries add challenges to the porosity estimation. This is because the models are trained for well locations that may not account for the geological variability in the seismic section. To explore several possibilities, I use different geometries for the synthetic models: homogeneous-wedge, heterogeneous-wedge, and normal fault geometries.

The homogeneous wedge takes a number of layers at a selected location given by the well. From here the layers are vertically stretched in one direction increasing their thickness. In the opposite direction, the layers are squeezed, reducing their thickness until they pinch out. This means that either the top or base surface of the wedge must dip, or both. In the models of this thesis only the base of the wedge dips downwards, meaning that the layers below the base have a constant dip. The heterogeneous-wedge model has the same geometry, but in this case the rock properties change laterally, reflecting different depositional environments.

Wedge geometries can be produced by laterally varying deposition. An example of this exists in the F3 block. A significant portion of the seismic shows indication of having been deposited by a marine delta. An example of this are the sediment onlaps displayed in Figure 8 at approximately 900 ms two-way travel time (TWT) towards the northeast. Deltas often display a clinoform geometry as seen in Figure 11. The middle part of this figure shows a clinoform that has more thickness in the middle than on the edges, just like a wedge. A clinoform surface is a sloping depositional surface commonly associated with sediments prograding into deep water [12].

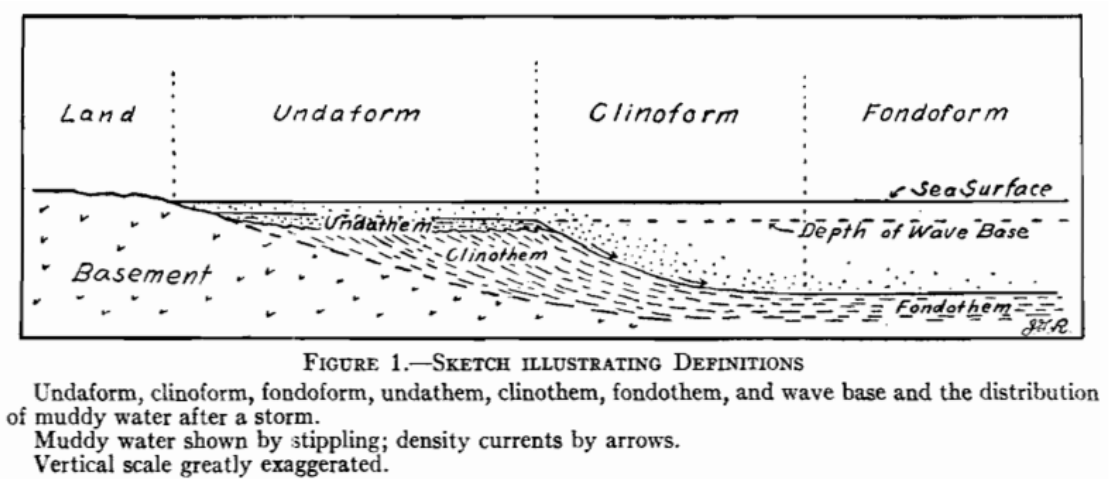

FIGURE 1.—SKETCH ILLUSTRATING DEFINITIONS

Undaform, clinoform, fondoform, undathem, clinothem, fondothem, and wave base and the distribution of muddy water after a storm.
Muddy water shown by stippling; density currents by arrows.
Vertical scale greatly exaggerated.

Figure 11: Change of depositional structures from land to deep water. The clinoform portion shows greater sedimentary thickness landwards, and forms a wedge-like shape. From [13]

The final synthetic model, model 3, involves displacement of rocks and rock properties along a fault-plane. Faults are however not infinite and also have a folding effect in the area near the fault plane as described in [14] and shown in Figure 12. The right plot in Figure 12 is used as a visual aid for constructing seismic horizons consistent with this deformation.

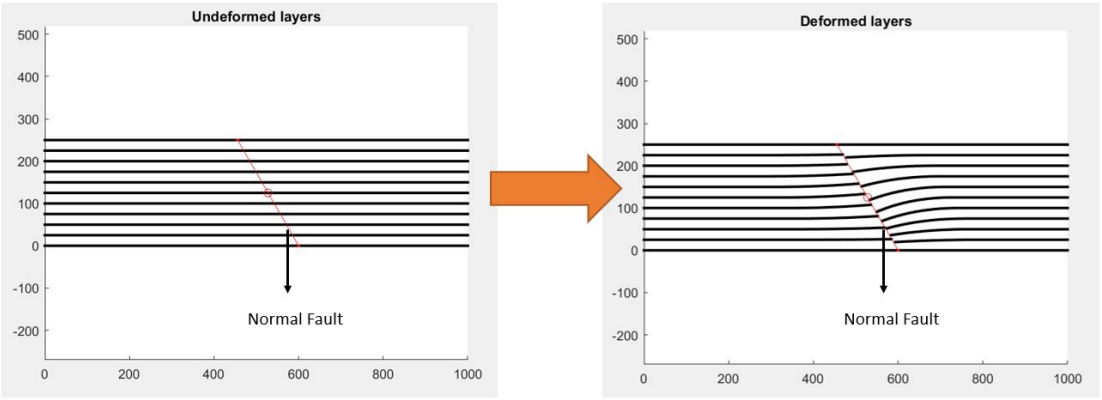

Figure 12: Displacement and deformation of rock layers due to a normal fault. Fault displacement is maximum at the center of the fault (red circle) and diminishes to zero towards the top and base layers. The deformation follows the theory presented by [14]. Matlab code by Nestor Cardozo.

2.2 Machine learning methods

2.2.1 Python tools and Modules

In Python version 3, modules are extensions to the program language, which add new data objects and functionalities. All the work for this thesis is done in the Windows 10 operating system. The coding, running, and debugging of the scripts are done in Sypder [15], while Jupyter Lab [16] is used for the analysis of the results. Both of these programs are open source.

Numerical Python or numpy is used extensively in the thesis. This library contains many useful features, and probably the most important is the implementation of arrays. To those unfamiliar with Python, the language does contain the "list" type made up of several elements, for example [1, 2, 3]. However, if this list is multiplied by 2, the result is [1, 2, 3, 1, 2, 3]. The same operation on a numpy array with the same elements will result in [2, 4, 6], which is what we expect. Thus, numpy arrays are consistent with vectors and matrices (arrays). They allow vectors and matrices operations, while lists don't (at least not directly). Numpy also has a randomization module (numpy.random) which I use in my code [17].

The pyplot module of the Matplotlib library is used for many of the plots in this thesis. The plots are usually made using numpy arrays as the input [18]. The seismic data are stored in segy format, and to work with these files and extract the data needed for Python, the module segyio made by Equinor is used [19].

For all ML methods, except neural networks, scikit-learn is used. This module provides a large library of machine learning algorithms and tools for improving and assisting the machine learning models. An example of this is cross-validation, which is used for all the machine learning models in this thesis. It is important to notice that the random number generation in scikit-learn is handled with numpy.random. This means that changing the seed of numpy.random, also changes the seed for scikit-learn [20].

The Pandas module handles the creation and editing of DataFrames. A DataFrame is an object

that functions similar to a table. Pandas is useful for loading, saving, analyzing and editing the data-sets [21]. Finally, I use the Tensorflow module for neural network modeling. This includes building, using, and saving neural networks [22]. Table 1 summarizes the tools I use in the thesis, and their version for the purpose of recreating the environment of this work.

Tool:	Version:
Windows	21H2 (OS Build 19044.1645)
Python	3.7.11
Spyder	5.1.5
Jupyter Lab	3.2.1
numpy	1.20.3
matplotlib	3.5.0
segyio	1.9.7
scikit-learn	1.0.1
pandas	1.3.4
Tensorflow	2.3.0

Table 1: Tools used in this thesis, and their version.

2.2.2 Introduction to Machine Learning

Machine learning algorithms are data-driven methods. What these methods set out to accomplish is largely dependent on the dataset. For example, if one wants to predict whether a fish is a salmon or a cod, one might have a dataset consisting of the length, weight and number of fins for a large number of fish. These are called predictors related to the target, which is the type of fish. If one does not have data on the type of fish, unsupervised machine learning must be used. In this case, the ML algorithm tries to find groups of data-points that are similar to each other. If the type of fish (the target value) for each data-point is given, then supervised learning is used. In this case, ML uses the known target values with the available predictors to train the model. The model will then predict the target value given a combination of predictor values [23, 24].

There are two types of target values, discrete and continuous, which require classification and regression methods, respectively. Simple examples are shown in Figure 13. Discrete means that there are a limited number of possible answers (classes); these problems are solved by using classification methods. This is shown in Figure 13A, where the classes are (*) and (x). For example a fish can be a salmon or a cod in the earlier example, making it a classification problem. Classifiers make several decision regions equal to the number of classes (possible discrete values). If a data-point is in a decision region, then it is classified as the class corresponding to that region. The different decision regions are separated by a boundary line representing equal probability for these classes, the boundary is shown in Figure 13A as a straight line. When the target has a continuous value range, for example if one is trying to predict the age of a person, a regression model is required. A regression model tries to fit a function to the data with the least amount of error. An example is given in Figure 13B. Here the line is the regression model which tries to fit the data-points, so that for any value of the predictor x, the linear function gives the target value y [23, 24].

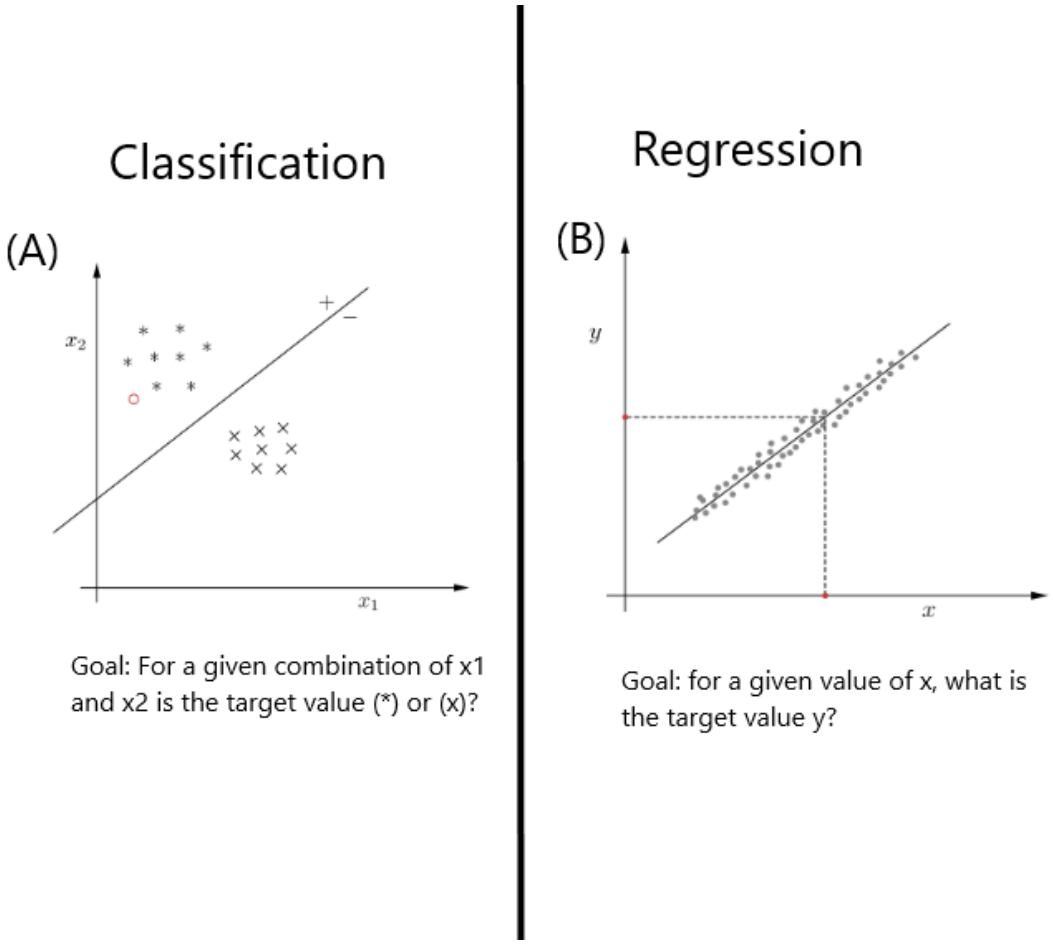

Figure 13: (A) Simple classification problem, where the line represents the boundary between the class regions. (B) Simple regression problem, where the line shows the solution fitted to the data. Plots edited from [24].

The main feature of machine learning is that the resulting models adapt to the data. Supervised ML methods are trained to predict the target value or class using provided data, including predictors and target values for many samples [24]. Therefore, the data quality is important. Since this thesis uses known porosity data as the target values, the focus will be on supervised methods, and since porosity is a continuous value, regression methods are used [23, 24].

Different ML methods adapt to the data in different ways, and all have advantages and disadvantages. A central theme in choosing the best method for the data set is simplicity versus flexibility,

or bias versus variability. This refers to how much the ML method adapts to the data. One can think of this as a sliding scale. Maximum simplicity means that the data has little to no effect on the model, meaning that the model will always predict the same value. Maximum flexibility implies that the model linearly interpolates the data, thus any new data added has a massive effect on the model. Typically, one wants to find a balance between these two [23, 24].

Figure 14 illustrates how two ML methods solve the same classification problem. The two ML models are Bayes classification and KNN-classification. The important thing to notice is that the Bayes classifier produces simpler (more biased) decision boundaries, while the KNN classifier (k = 13) makes more flexible (more variable) decision boundaries. Figure 14 shows that the Bayes classifier has a smoother shape than the KNN classifier, which means that the Bayes classifier is more independent of the data-points than the KNN-classifier [24]. Different ML methods occupy different areas on this sliding scale of bias versus variability. Finding the right method with the right amount of flexibility is the key for consistent predictions that respect the target data [23].

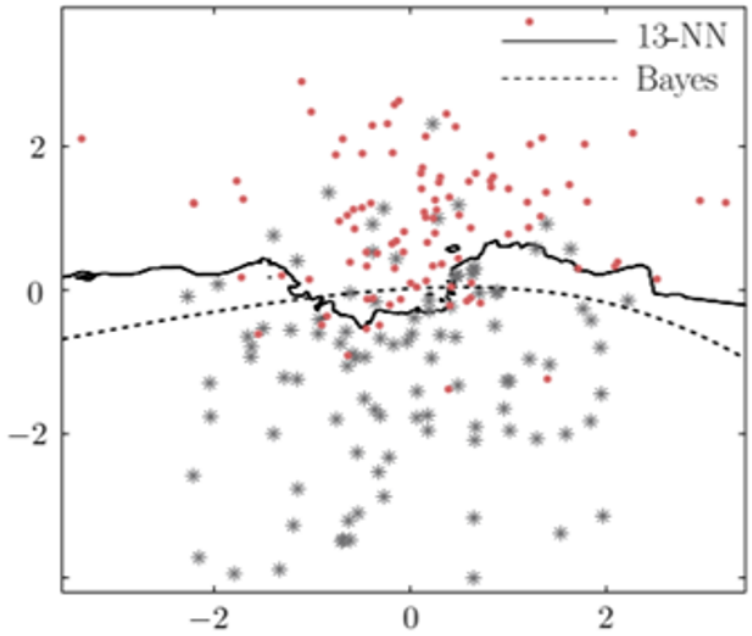

Figure 14: KNN versus Bayes classifiers. The Bayes classifier has a smoother shape, and it adapts less to the data than the KNN classifier. Red dots and grey asterisks represent two classes. From [24].

The ML methods used in this thesis are KNN, random forest, lasso regression and neural networks. SVR (support vector regression) was considered. However, this method scaled poorly to the amount of data in terms of processing time.

The goal is to predict porosity from the seismic attribute acoustic impedance. As porosity is a continuous value, regression models are the natural choice. KNN is chosen as it is a simple method that can interpolate the data-points if this becomes ideal. Lasso regression is an extension of the familiar linear regression, so if the data works well with linear regression, it also works well with lasso regression. However, lasso regression, unlike pure linear regression can also perform automatic predictor selection. Random forest is good at separating datasets into several regions that display different relations. Finally, neural networks are used because of their flexibility. With enough time and tuning, neural networks can be used for any dataset. That said, with a limited amount of time and testing the neural networks may not work as well.

2.2.3 KNN-regression

K-nearest neighbors' regression is a non-parametric regression method. Non-parametric means that the resulting trained ML model does not have a consistent shape. For each point on the predictors grid, the k-nearest data points are considered, the value on the grid equals the mean value of the response of the data points. Because distance is important in this method, it is highly recommended to standardize the predictors. By setting $k = 1$, the method will linearly interpolate the data. If N is the number of data points and $k = N$, then the method will produce the mean data value for every point on the grid. This implies that the method has a wide range of adaptability. Depending on k, KNN can interpolate the data or predict only the mean of the entire data-set. An example of using a maximum, minimum and moderate k value is illustrated in Figure 15. This method is the simplest ML algorithm used in this thesis [23].

2.2.4 Lasso regression

Lasso regression builds on arguably the simplest regression model, linear regression. For every predictor (dimension), a coefficient is found that minimizes a loss function, often residual sum

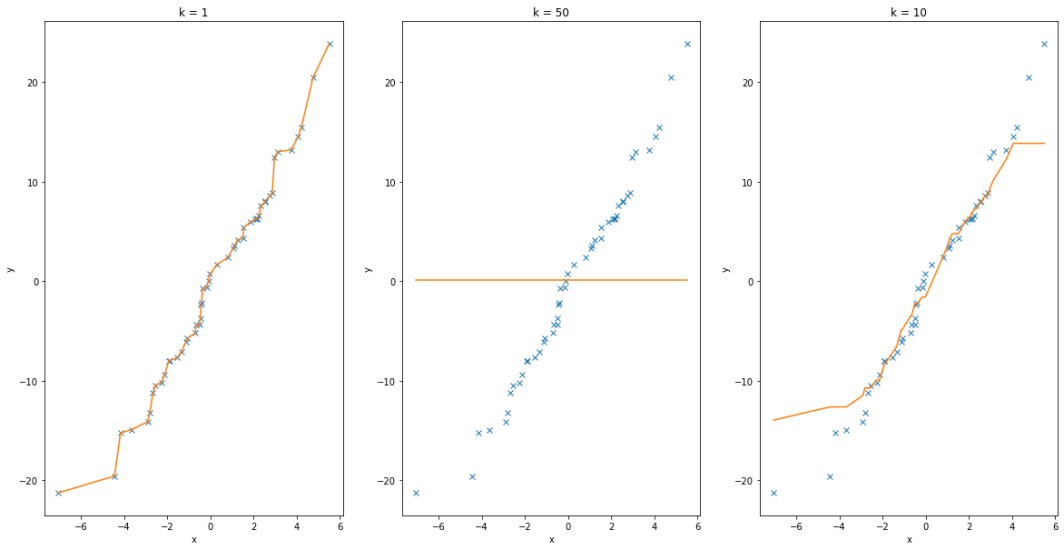

Figure 15: Changing k in KNN-regression changes the predictions (orange lines). The regression uses the data points in blue as training data. Left shows k = 1, meaning maximum variability and interpolation. Middle shows k = number of data points, meaning maximum bias and causing the mean of the data to be chosen no matter of location. Right shows a more reasonable value of k = 10. Made in Python using Numpy, Scikit-learn and Matplotlib.

squares (RSS) [23]. The RSS is calculated as the sum of the squared difference between the target value y and the predictor value x:

$$RSS = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_i j) \quad (4)$$

Here p is the number of predictors, n is the number of data-points, β_0 = is the intercept, β_j is the j^{th} coefficient, and x_j is the j^{th} predictor value.

This is not a flexible method, but it is easy to interpret. The aim is to find coefficients that display the relation between the predictor and the target. However, this relation is often not linear. Fitting a linear model to a non-linear predictor-target relation can be detrimental to the model's accuracy. There are several ways of identifying this problem such as separating the model into the different groups of predictors and use statistical parameters such as R-squared or MSE to identify poor fits. However, there are extensions of linear regression that attempt to automatically limit the influence of poor predictor-target relations. These are known as shrinkage methods. Shrinkage

methods aim to reduce the coefficients β_j by adding a penalty term to the RSS [23]. Figure 16 shows a data-set that displays no linear correlation, x is the predictor and y is the target. To reduce the linear regression and the data's influence on the model a penalty λ can be added. λ is the penalty strength, and a larger λ reduces the poor correlation by reducing the β_j coefficients [23].

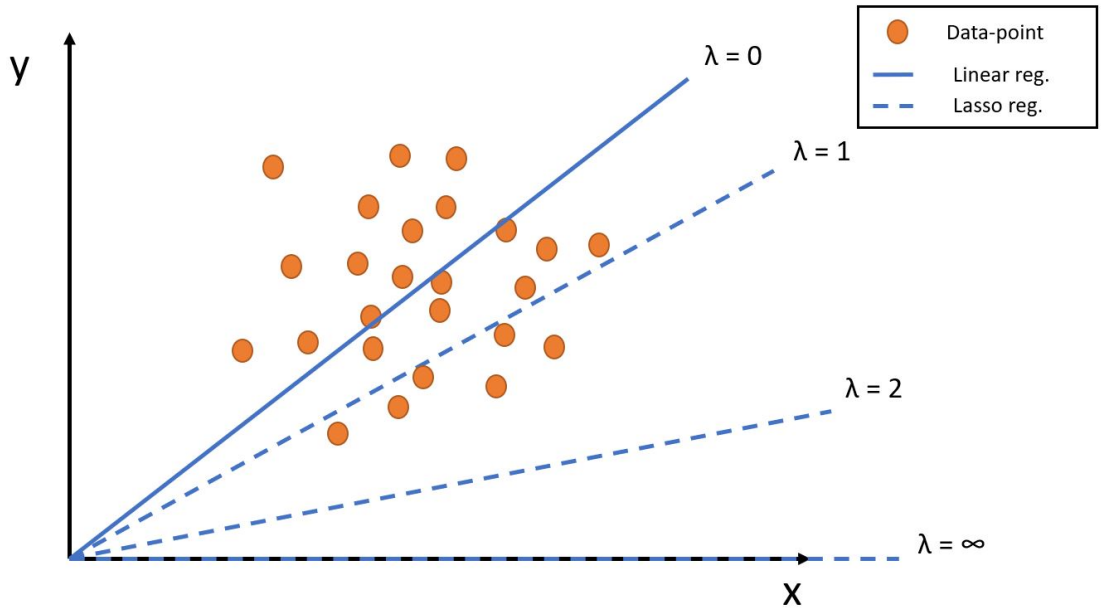

Figure 16: The orange points show a data-set with a nonlinear relationship. The continuous blue line shows the linear regression result or the lasso/ridge regression where $\lambda = 0$. The other dashed blue lines show the lasso/ridge regression using increasing penalty terms λ.

For the general case with multiple predictors, by reducing the j^{th} coefficient one also reduces the impact that the j^{th} predictor has on the model. If the j^{th} predictor has a non-linear relationship to the target data, the full model will likely be improved by reducing this predictor. This is regardless of whether the RSS is increased or decreased. This is why a penalty term is needed to find a balance between the RSS and the total value of all coefficients. One such method is ridge regression [23]. The aim of ridge regression is to minimize the following equation:

$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2 \quad (5)$$

The new term added to the RSS is the shrinkage penalty, where the strength of the penalty is

determined by λ. In order to minimize Equation 5, the model finds the balance between the RSS and the shrinkage penalty, not allowing either one of them to get too big. If $\lambda = \infty$, the minimum of the regression will set all coefficients to zero. If $\lambda = 0$, the standard linear regression will be the result. If λ has a reasonable value (estimated by trial and error), it will decrease the error of the model because it reduces the impact of coefficients that correlate poorly with the response. The ridge regression method has a weakness in that even with a good λ, none of the coefficients are reduced to zero, but just approach zero. This implies that ridge regression can reduce the impact of poor predictors, but has issues eliminating predictors [23].

The lasso regression method seeks to improve this. This method minimizes the following equation:

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j| \quad (6)$$

Here the shrinkage term uses the absolute value of the coefficients instead of the squared value. The following example explains the difference between these two methods. If $\lambda = 1$ and $\beta_j = 0.5$, the penalty for the ridge method is 0.25, while the penalty for the lasso method is 0.5. Another example: Assuming $\lambda = 1$ and $\beta_j = 5$, the penalty for the ridge method is 25, while the penalty for the lasso method is 5. So one can say that ridge regression penalizes large coefficients β_j harsher, while lasso regression penalizes smaller coefficients β_j approaching zero harsher. Therefore, lasso regression is more likely to set the coefficients to zero. I use lasso regression instead of ridge regression, to observe the effects of predictor elimination [23].

2.2.5 Random forest regressor

Decision trees are good at separating different regions in the data into decision regions. A decision tree is composed of several decision levels. Each level represents a condition based on the previous decision. For example is x larger than 1? If this condition is true, then the decision is to move down to the left branch. At the end of the tree, there will be a value representing the estimate based on the decisions taken. This estimate depends on if the tree is a classification tree or a regression tree. A classification tree delivers the most common class in that region as the output. A regression tree usually outputs the mean target value of all the data in the decision region. Building a decision tree

involves making these levels of decisions so that they best split the data [23].

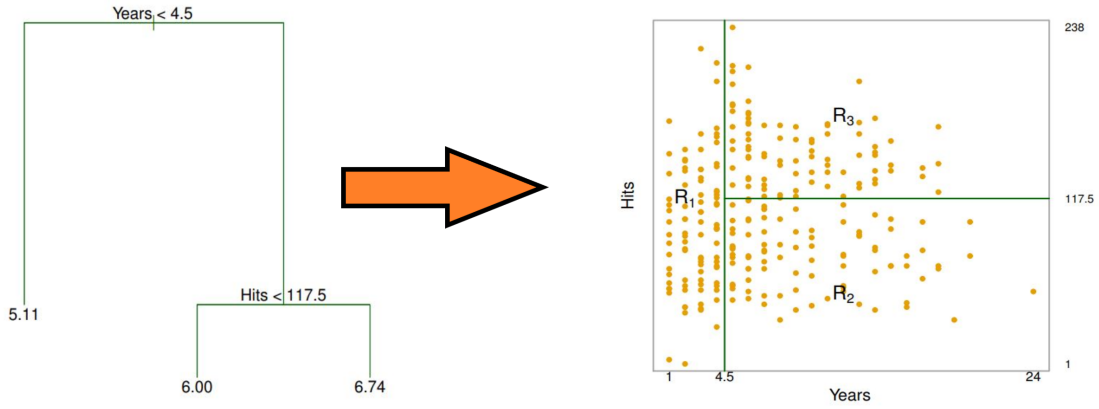

Figure 17: How a decision tree translates to decision regions. Years and Hits are the predictors, the target value is not displayed. The decision tree has two decisions resulting in two splits and three decision regions. The first decision splits the dataset in half based on if Years < 4.5. If this is true, then the decision if to go down to the left, resulting in R_1. If false, another decision is encountered to the right. This second decision splits the decision region into the regions R_2 and R_3. From [23] and their "Hitters" data-set.

A decision tree is made via a top-down recursive binary splitting. Starting from the top decision, the tree splits the data in half based on what area minimizes the loss function, for example RSS or MSE. The split defines two decision areas. Then for each new area a new split that minimizes the loss function is found. This is done until the maximum depth is reached. Another method is to make the entire tree at once many times until the loss function is minimized. This is however, too computationally intensive, even if it is more accurate. Recursive binary splitting makes a large decision tree that often overfits the data [23]. This is the same as saying that the decision trees have high variance and adapt greatly to the given data, which can make the models unreliable.

Random forest is an ML method that makes many decision trees to estimate the response. Random forest makes many smaller decision trees, where the predictors considered at each split are determined randomly. Using random predictors for each tree makes it necessary to reduce the correlation between the trees. The reason for this is mainly to reduce the variance of random forest

in comparison to decision trees. Here variance means that the trained model changes significantly if trained to a subset of the dataset. After all the trees are trained, the trees estimate a value at a sample point based on what decision area the point is in. The mean of these values is returned [23]. In scipy, bootstrapping is used for making random forest models. This method considers two arguments: n_estimators which refers to the number of trees in the forest, and max_depth which refers to the maximum depth of the decision trees.

2.2.6 Neural Networks

Neural Networks are designed to be similar to how mammalian brains function. The brain is composed of neurons which fire electrical signals, the neurons are connected via synapses. The synapses usually convert the neurons electrical signals into chemical signals that are converted back to a new electrical signal that is transported to the next neuron. However, all neurons are not connected to each other. Rather they are connected in a hierarchical layered manner. This allows for the transportation and modification of electrical signals [24]. In a similar manner, a neural network is composed of neurons that collect information and weights connecting the neurons while modifying the information. Sets of weights and their connected neurons form a layered structure such as in Figure 18.

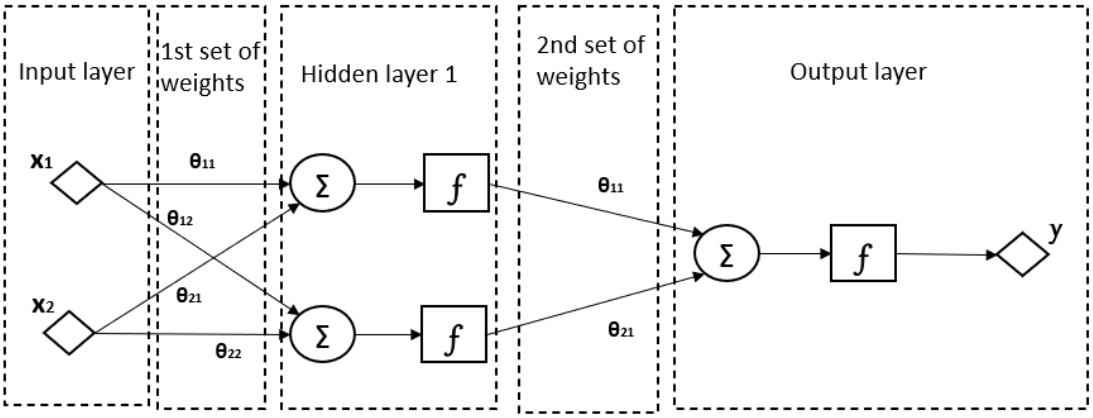

Figure 18: One−layer neural network. The squares with an f represent the activation function. Σ indicates that the inputs to the neuron are summed. θ_{ij} represents a weight (floating point number) which is multiplied with the connected input (j), and the result is sent to the connected neuron (i). Based on theory from [24]

The information here is not electrical, but rather numerical. The neurons sum all incoming numbers together before sending them to the next layer. The weights multiply the numbers by a specific value during transmission. The first set of neurons are the input layer, where the initial data are provided. The last set of neurons is the output layer representing the result of the model. Since the output is a set of neurons, one model can estimate several parameters at once, for example porosity and density [24].

Figure 18 shows a one−layer network. The structure is separated into 5 parts that do not necessarily represent the current nomenclature but is useful for showing how a neural network functions. The input data are added to the input layer. The input is multiplied by the 1st set of weights and passed into the hidden layer. In the hidden layer, the connected data are summed in the neurons represented by the circles. The output of the neurons is passed through an activation function (sometimes called a transfer function depending on the source). The output of the hidden layer is multiplied by the 2nd set of weights and passed to the output layer. The procedure of the output layer is similar to the hidden layer. The result of the output layer and thus the neural network model is y, which can be a vector or a single value depending on the desired output [24].

This ordered system of summation and multiplication allows the neural network to produce virtually any function, making it extremely flexible. There are many extensions to neural networks. The most common extension that is relevant for this thesis is the activation function. The activation function can be used on every neuron. It is simply a function that takes a single argument. For example in this thesis, the sigmoidal function will be used. This is because the sigmoidal function limits the output between 0 and 1; this is useful when trying to predict porosity which is within this range. By using the activation function, the flexibility is limited but it ensures that the model behaves within reasonable limits [24].

After the output layer is calculated, a loss function is used for evaluating how close the output is to the target. The mean−squared error is a common example of a loss function. I have explained so far forward propagation. Backward propagation is how the weights are updated [24]. This is done with an optimizer, for example "Adam", which is used by tensorflow. "Adam" is an algorithm

for first−order gradient−based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments [25]. It calculates how much the set of weights should be changed, which is then applied to the weights. One forward and one backward propagation equals one epoch or one learning cycle of the model [24]. With each epoch, the performance of the model should improve. In this thesis I use 100 to 1000 epochs.

2.2.7 K-fold Cross-validation

Cross validation is a resampling method, which involves taking samples from a training set and fitting the model to these samples before repeating this process to the rest of the dataset. Resampling methods give information on the dataset that cannot be acquired by simply fitting a model. In our case, cross validation gives information on how prone the dataset is to over-training. Over-training means that in the worst-case scenario, the model is simply interpolating the data. This means that the model would likely fail predicting any new data. One can avoid over-training by constraining the models shape (lowering how much it adapts to the data); the worst-case scenario is that the model simply shows a linear relationship while the data shows a completely different relationship, this would be under-training. Resampling methods allow finding the amount of constraints that prevent both under and over-training [23].

Cross validation is useful for avoiding over-training and performing parameter tuning. For example, when using KNN-regression what should be the "k" parameter equal to? In k-fold cross validation, the dataset is split randomly into k folds (parts). Figure 19 shows an illustration of 5-folded cross validation. For each split shown in Figure 19, the model is trained using all other folds, and then the method predicts the current fold to calculate the error (for example MSE). Thus, the method returns a number of error scores equal to the number of folds. The mean of these errors gives an evaluation of the robustness of the model against over-training and its accuracy. Cross validation is ran once for each parameter value that is considered. Then, the model that performs best is picked [23]. TensorFlow does not have cross validation built in, so the sklearn split function is used instead to make the folds, and the cross-validation is manually coded.

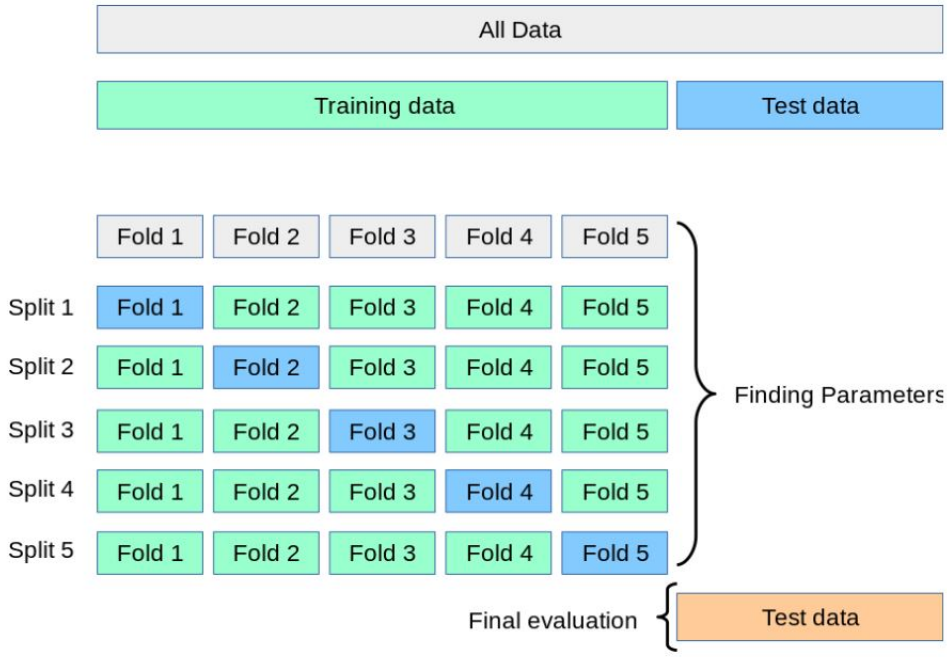

Figure 19: Five folded (parts) cross-validation. One cycle per fold of training to 4 folds and validating using the remaining fold. The validation error (cross-validation score) is calculated for each cycle. The cross-validation mean (CV-score) is the mean of all the validation errors. From [26].

3 Methodology

3.1 Workflow

The overall workflow aims to perform seismic inversion using either synthetic data or data from the F3 block. This provides a P-impedance cube that correlates with porosity. Then the acoustic impedance and porosity well-logs are used to predict the porosity of a selected seismic section using SGS and ML methods. The predictions are later compared to each other in the Results section.

The workflows for the synthetic models and the F3 dataset are different. The workflows are shown in figures 20 and 21 for the F3 dataset and synthetic models, respectively. The primary difference is that the synthetic models have to be constructed, specifically the seismic cube and seismic horizons, as these are the basis for seismic inversion. This is unlike the F3 dataset, which has a seismic cube and interpreted seismic horizons.

For the synthetic models, 3D cubes of porosity and acoustic impedance are made from seismic horizons and well-logs. From the acoustic impedance cube and a wavelet, a 3D seismic cube is calculated (Figure 21). At this point, both the synthetic models and the F3 block have a seismic cube and well-logs. For both situations, seismic inversion is performed to construct a P-impedance cube (figures 20 and 21). Then, SGS is performed to estimate porosity from the P-impedance cube and the porosity well-logs. Finally, the ML methods are used to predict porosity from a section of the P-impedance cube and the porosity well-logs. After that, the ML and SGS predictions are compared to examine where and to what degree they differ from each other and the true porosity model (figures 20 and 21). Notice that the SGS method operates on a 3D basis (seismic cube), while the ML methods operate on a 2D basis (seismic section).

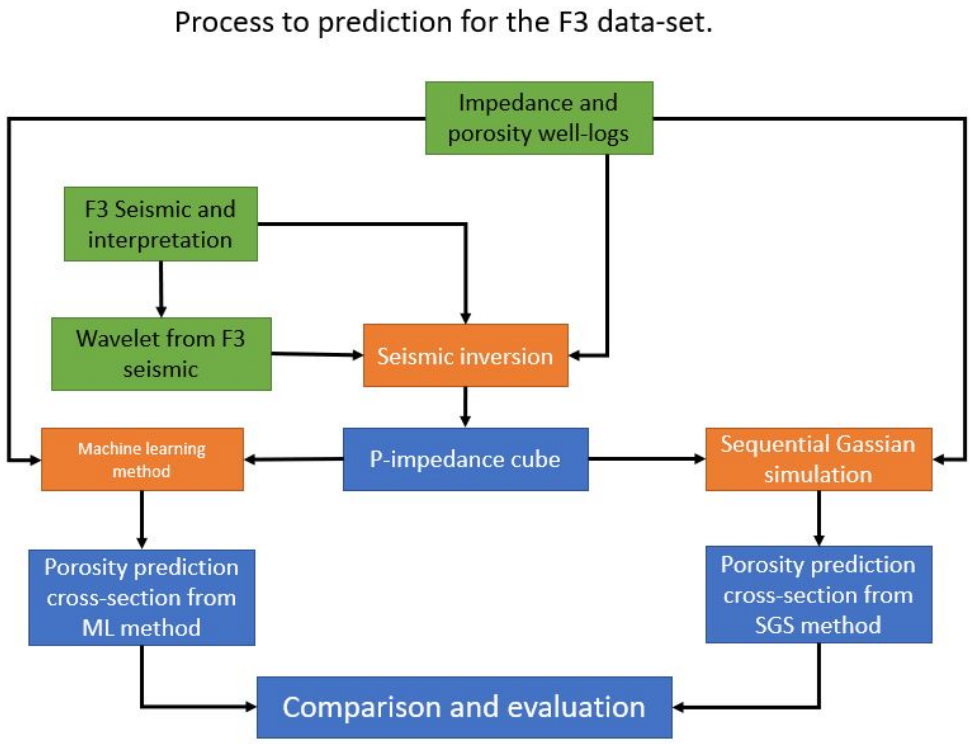

Figure 20: Basic workflow for the F3 dataset. ML = machine learning, SGS = sequential Gaussian simulation.

For the machine learning part there is an important difference if the model is synthetic. The synthetic models have a true porosity cube unlike the F3 data. One advantage of this is that traces in the porosity and acoustic impedance cubes can be used as synthetic well-logs. Thus, by changing the well location(s), the impact of well placement can be investigated. For example, will the ML methods perform well if the well is outside the wedge, despite the lack of information in the wedge?

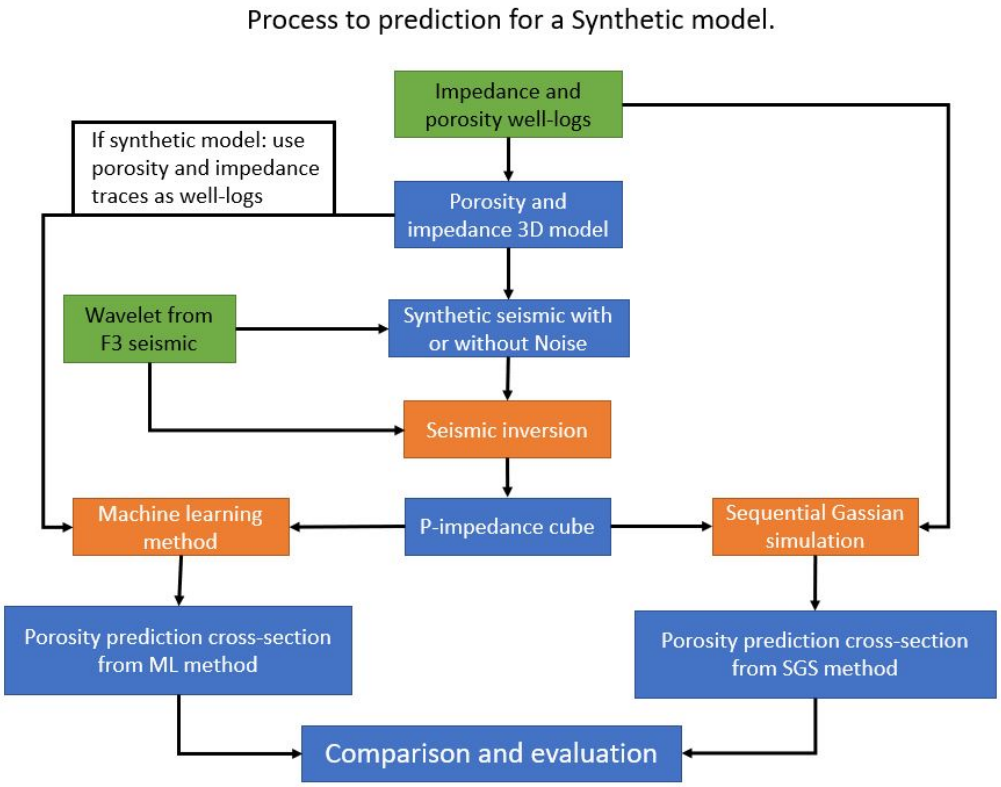

Figure 21: Basic workflow for a synthetic model. ML = machine learning, SGS = sequential Gaussian simulation.

Sections across the synthetic acoustic impedance models are shown in Figure 22. From left to right, the models are the homogeneous wedge, heterogeneous wedge, and normal fault. The models without spectral noise are the three top sections, the models with spectral noise are the bottom sections. The models are constructed to evaluate how the seismic inversion, geostatistics (SGS), and ML methods handle these types of subsurface geometries and noise levels.

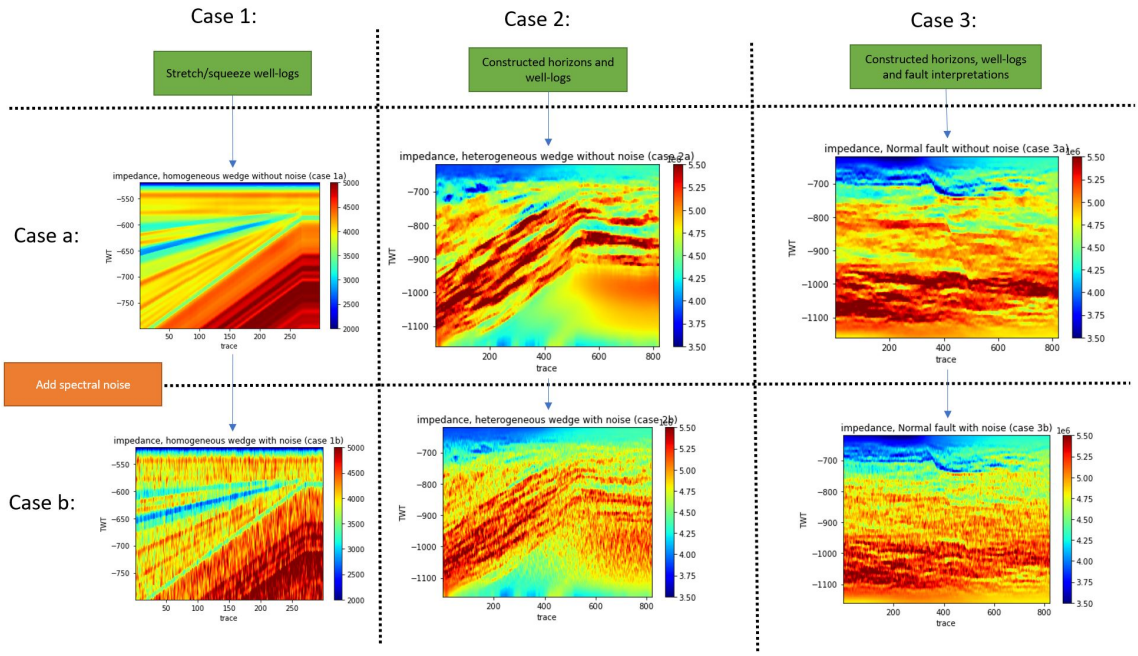

Figure 22: Sections across the synthetic acoustic impedance models. Case 1 is the homogeneous wegde, case 2 is the heterogeneous wedge, and case 3 is the fault model. The first row are impedance sections without noise. The second row are impedance sections with spectral noise. Case 1 has a differant scale because it is using kPa.s/m, while the other cases use Pa.s/m

3.2 Seismic inversion and porosity estimation of F3 dataset

The inversion and porosity prediction of the F3 dataset are covered before the synthetic models, as the construction of the synthetic models are based on the F3 dataset. The statistical wavelet, wells and anisotropy are derived from the F3 dataset and used in the synthetic models.

The deterministic wavelet extracted from the F3 seismic data is displayed in Figure 23. This wavelet shows a reversed polarity which was not obvious from the seismic.

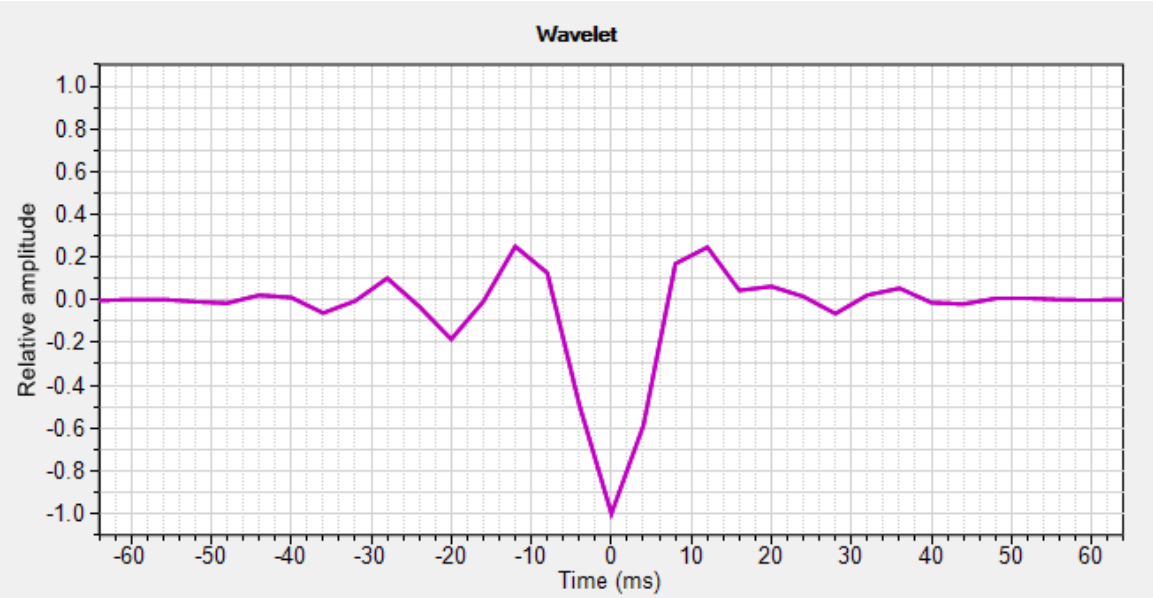

Figure 23: Deterministic wavelet from the F3 dataset. The wavelet is made from the seismic cube and logs from well F02-1.

The lower-frequency model (LFM) is displayed in Figure 24. The extrapolation of the LFM is guided by the interpreted horizons. There are uncertainties in the low frequency impedance between the wells, which is indicated by the LFM not following the horizons far away from the wells. Via seismic inversion, the LFM and wavelet (Figure 23) are used to derive the P-impedance shown in Figure 25. This P-impedance and the porosity well-logs are the input for the porosity estimation.

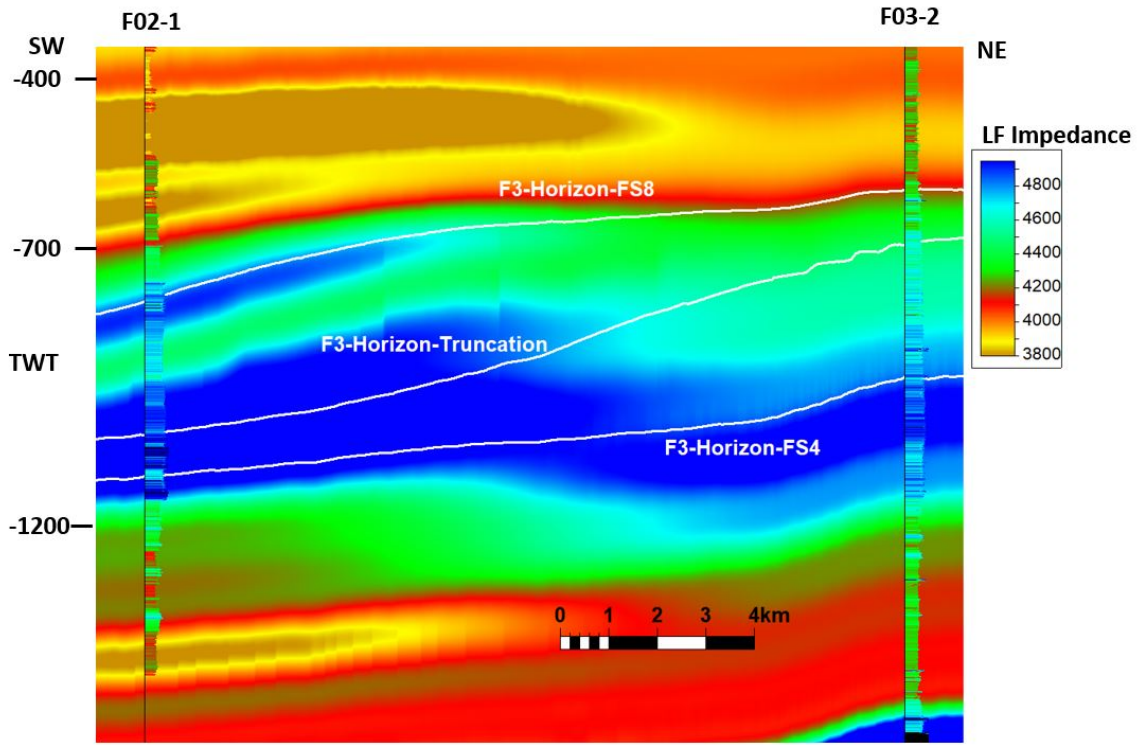

Figure 24: F3 LFM impedance section with wells F02-1 and F03-2 and acoustic impedance logs. Same location as in Figure 8. The impedance is in kPa.s/m.

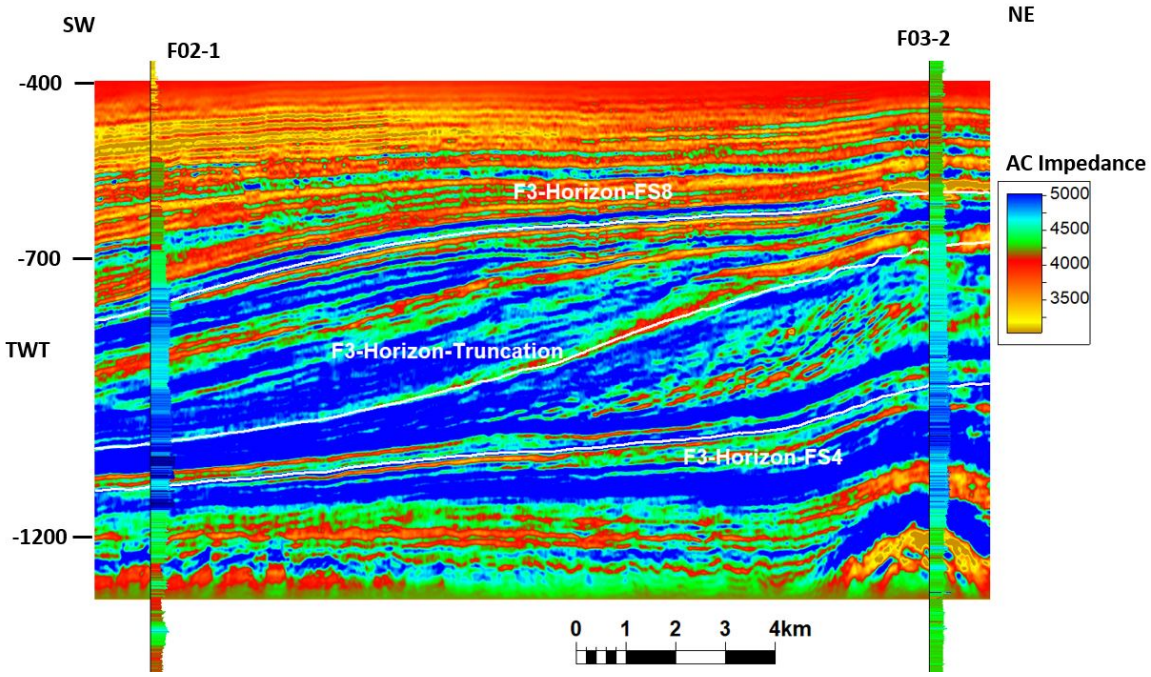

Figure 25: Seismic inversion and resulting P-impedance section of the selected F3 section. Wells F02-1 and F03-2 and acoustic impedance logs are included. The impedance is in kPa.s/m. Same location as in Figure 8

The parameters for the SGS method are determined from the P-impedance cube and porosity well-logs. The number of wells is not enough to derive a horizontal variogram. As the porosity correlates with the acoustic impedance, the impedance cube is used to estimate the variogram horizontal major and minor ranges as well as their azimuth. Figure 26 shows the the major and minor horizontal variogram directions for the F3 data. The major range is estimated to be 8000 m with an azimuth of 320°. The minor range is estimated to be 4000 m, half of the major range, and with azimuth 050°.

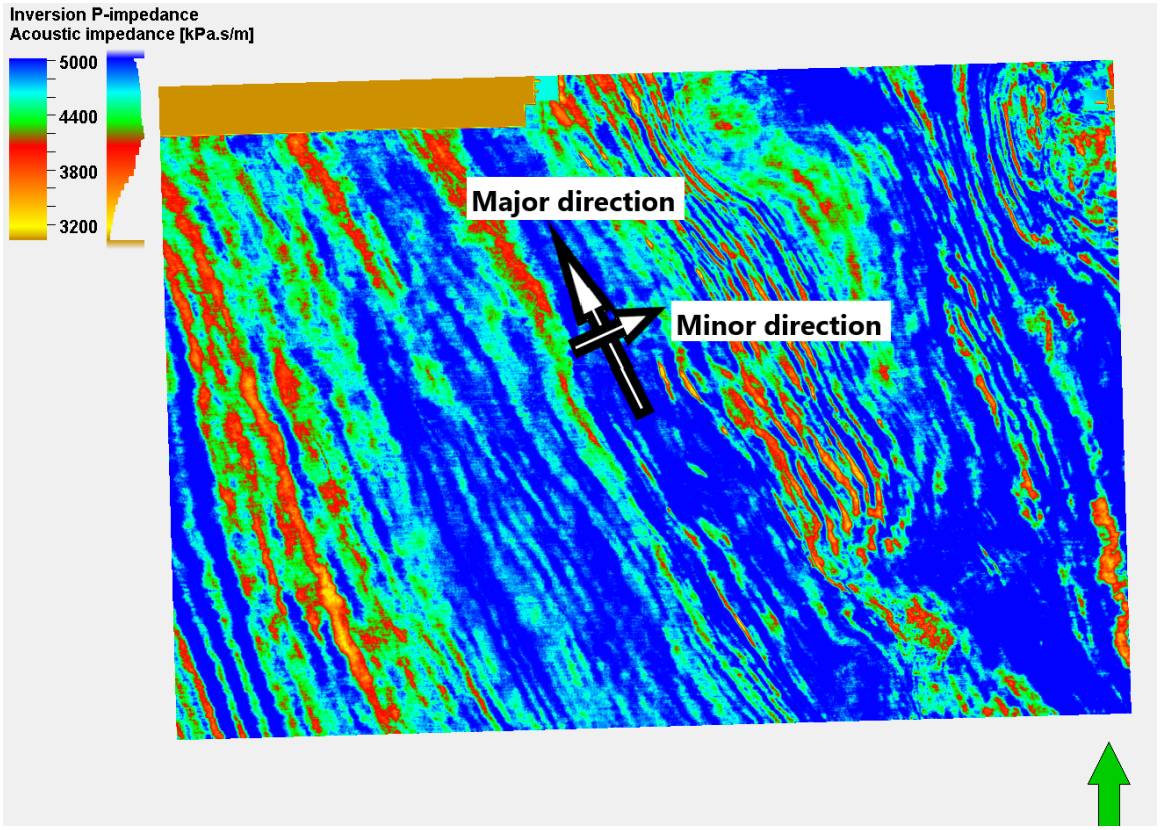

Figure 26: Time-slice of the F3 dataset and inverted P-impedance. Many time-slices are used to estimate major and minor variogram ranges as well as their azimuth. These are shown as the white arrows. The impedance is in kPa.s/m.

The vertical range however can be estimated from the well-logs using the variogram in Figure 27. The nugget is set to zero, assuming that the data have no errors. The vertical range is 11.6 m. Also, the vertical variogram model fits well with an exponential function. The variogram parameters used in the SGS algorithm are shown in Table 2.

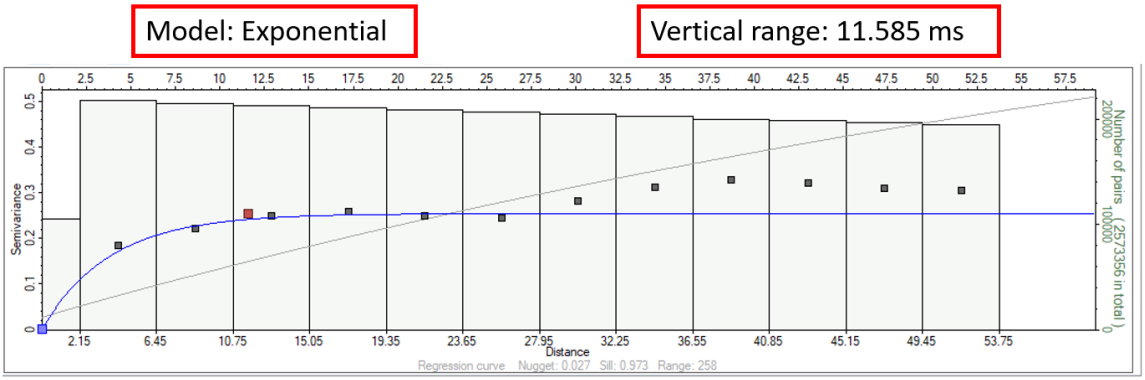

Figure 27: Variogram of the porosity log in well F02-1. The fitted model gives a range of 11.6 m with an exponential function shown by the blue line. The red boxes show the model function and vertical range.

Parameter:	Value:
Vertical range	11.6 m
Major range	8000 m
Minor range	4000 m
Azimuth major axis	320°
Azimuth minor axis	050°
Variogram model	exponential

Table 2: Variogram parameters used in SGS modelling

3.3 Construction of synthetic models

3.3.1 Case 1: Homogeneous wedge

Case 1 is a homogeneous wedge. The model was made using the Petrel tool: "RokDoc - 2D Forward Modeling". This tool makes the two horizons that define the top and base of the wedge. The top horizon is horizontal while the base horizon dips to the left. The layers below the wedge dip the same than the base of the wedge. The horizons in the wedge increase in dip from the top to the base of the wedge (Figure 28). The well F02-1 is used as the basis for constructing the porosity and acoustic impedance models. The porosity and acoustic impedance logs are extended from the well location to the rest of the section following the horizons. Since the vertical distance between the horizons changes laterally, the well-logs are squeezed to the right where the wedge pinches out, and

stretched to the left where the wedge thickens. The synthetic acoustic impedance model is displayed in Figure 28.

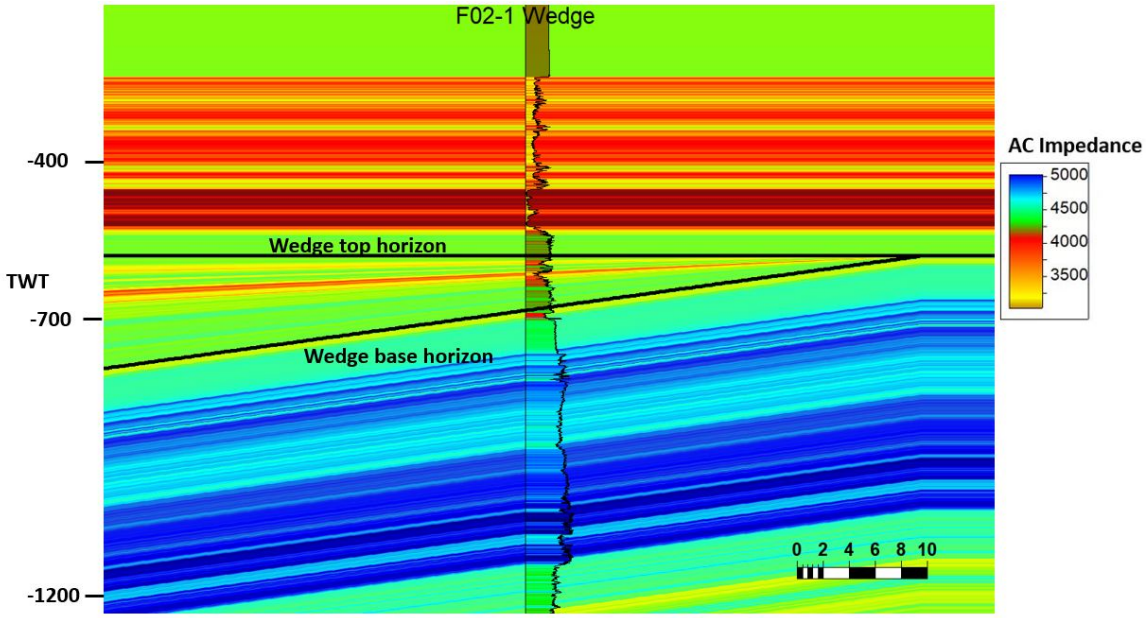

Figure 28: Case 1a: Acoustic impedance model of the homogeneous wedge without noise. The well, impedance log and horizons used for the construction of the sections are shown. The impedance is in kPa.s/m.

The synthetic seismic section for the homogeneous wedge is derived from the impedance model (Figure 28) using a statistical wavelet shown in figure 30. Once the synthetic seismic section is made, noise is added to the section, with a spectrum that is shaped by the analytical wavelet. Seismic inversion is performed on both the seismic section without noise and the section with noise to determine the P-impedance cube. The wedge model is only used for simple evaluation of the ML performance. Therefore, no porosity estimation using SGS was done.

3.3.2 Cases 2 and 3: Heterogeneous wedge and fault models

These models are three-dimensional and they were made covering an area around the wells F02-1 and F03-2. The vertical TWT range should include the well-logs and is therefore limited to -400 and -1300 ms TWT. In order to preserve the lateral variability, the major and minor variogram ranges as well as the anisotropy azimuth are consistent with the F3 block (Table 2). These parameters

are also used in the porosity estimation by sequential Gauss simulation, meaning that they are not subject to uncertainty.

For these two models, four horizons (models 2 and 3) and a fault plane (model 3) are constructed to build the structural model. The wedge model (model 2) requires two horizons that are parallel in one area, before diverging with the lower horizon dipping downward as shown in Figure 29A. In model 3, a fault plane is made shown as the blue surface in Figure 29B. At the top and bottom of the fault plane, the horizons are approximately horizontal and indicate zero fault displacement. Between the top and bottom horizons, Figure 29B shows that the horizons dip sharply in the area close to the fault. These horizons and fault interpretations are used to define the shape and resolution of the 3D grids that are populated with porosity and acoustic impedance.

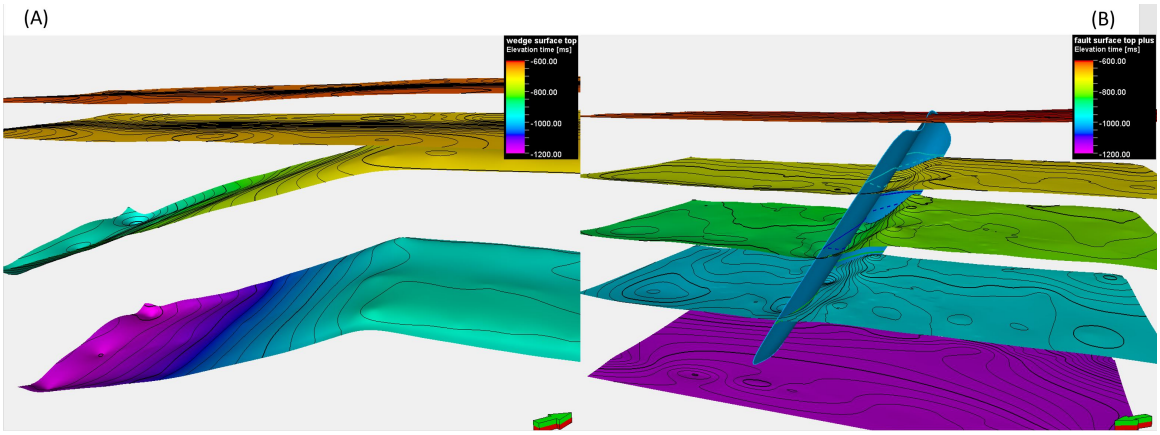

Figure 29: Seismic horizons and fault used to construct the synthetic models of cases 2 and 3

For both models, the 3D grid is populated with acoustic impedance using SGS. The horizontal variogram ranges for the major and minor directions are the same as in Table 2. Similar to the variogram model in Figure 27, the exponential model was chosen for the vertical variogram. The porosity model is made using SGS guided by the acoustic impedance model using the variogram parameters specified earlier. The synthetic seismic is made using the impedance model and the statistical F3 wavelet shown in Figure 30. Noise is added with a frequency spectrum defined by the statistical wavelet. The noise is necessary for challenging the estimation methods. The addition of

the noise separates these two cases into sub-cases with and without noise.

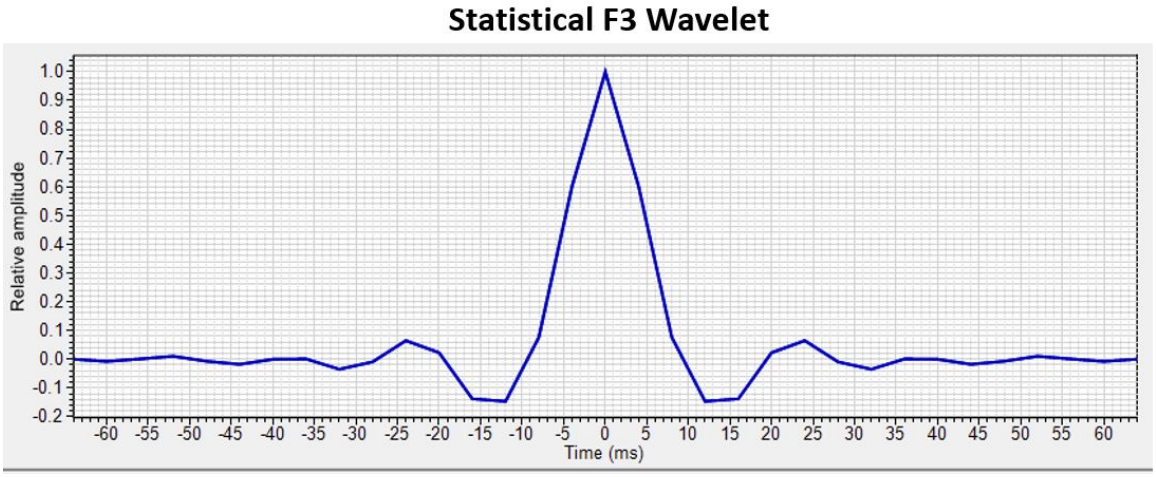

Figure 30: Statistical wavelet made from F3 seismic.

From the synthetic seismic, a deterministic wavelet is calculated using the reflectivity derived from the impedance log of well F03-2 and the seismic traces near that well. Seismic inversion is then performed to derive the P-impedance cube. Using the P-impedance cube with SGS (same parameters as in Table 2), the porosity cube is estimated. The section used for testing the machine learning algorithms contains the wells F02-1 and F03-2. Their location is shown in Figure 8.

3.4 Design of ML methods for porosity prediction

3.4.1 Synthetic well-logs

The synthetic sections contain the true porosity and impedance. This means that traces from these sections can be extracted as synthetic well-logs and used for training the ML models. This is done to test the effects that different well locations have on the predictions. For example, placing the synthetic well inside the wedge versus outside the wedge and examining the effects this has on the error. A negative consequence of using traces as synthetic well-logs is that the sampling frequency is lower than in a real well-log. This means less data than if real well-logs are used, which can reduce the ML model accuracy.

3.4.2 Window functions

In machine learning, predictor and target data are used to train the ML model. After the model is made, it can predict the target value based on the input predictors for one data-point. If an ML model makes several predictions, it does not consider previous or future predictors, only the current input predictors (there might be some exceptions). In sedimentary rocks, there is a certain expectation that vertical sequences in the rock properties have some continuation laterally. Therefore, it would be useful to provide the ML models with information on how the rock properties change vertically.

To provide the ML models with information about vertical properties changes, rolling window functions are used. The rolling median and mean windows calculate the median and/or the mean within a predetermined interval at every point and add this as a predictor. The interval refers to the number of data-points above and below the point of computation. The rolling selection window works in a similar way. It adds all data points within an interval as new predictors. I use only a window size of 10 data-points. This window size is included to examine if it improves the ML models. This thesis will not look at optimizing the window size.

3.4.3 Geological time (depo-time)

Adding geological time as a predictor might improve the estimation for the same reason as the window functions, since it provides the ML methods with information about the geometry and geology. For example, the homogeneous wedge is made using three points: the beginning of the top and base, the end of the base, and the end of the top. This same principle can be used for reconstructing the geological time. From the top of the section to the bottom, time increases continuously where the wedge is present. Where the wedge is not present, there is a gap in time. The depo-time constructed in Python is shown in Figure 31. The upper layers above the wedge are horizontal while the lower layers below the wedge dip to the left. Also, on the right of the plot indicated by the red rectangle, the depo-time jumps from about 300 to about 500 ms, indicating that the wedge is not present here.

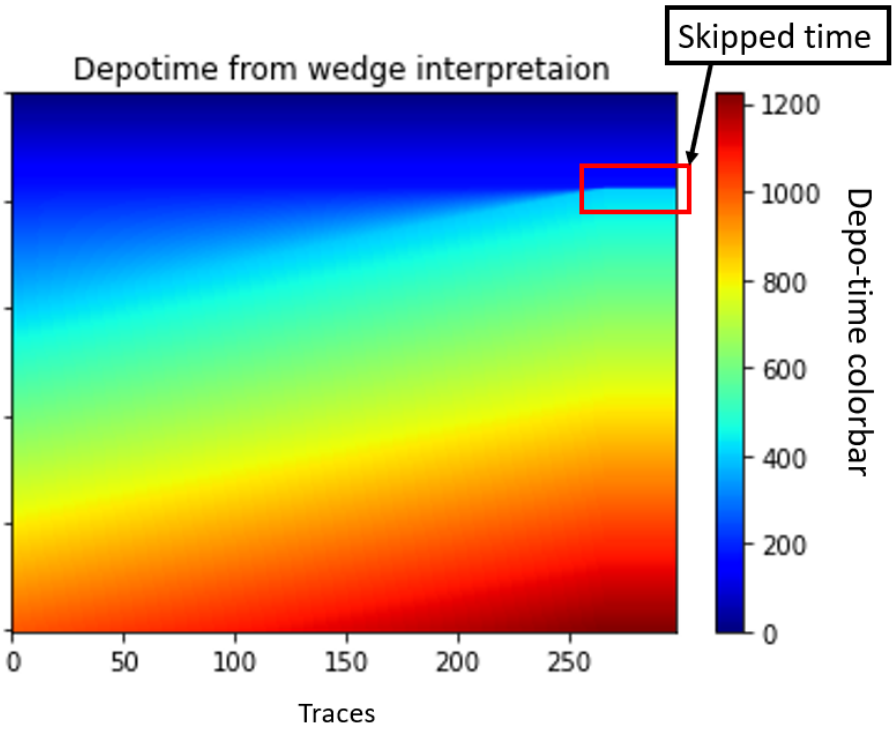

Figure 31: The depo-time for the wedge geometry used in case 1. This diagram is constructed based on the assumption that time increases continuously from the top to the base of the wedge. The red box shows the area of missing time.

In the heterogeneous wedge and fault models, depo-time (geological time) is also used. In the process of making these time sections, seismic horizons are necessary. These horizons are surfaces that represent an area where the deposition occurred at approximately the same time. This can be used to derive the relative geological time. An example of the resulting time sections are shown in Figure 32. This method is also applicable to the F3 dataset as seismic horizons are available. Each horizon is first given a label (depo-time), the upmost horizon equals 1, the second horizon equals 2, etc. Then for each trace in the section, the depo-time is linearly interpolated between the horizons. Since standardization (explained in the next section) is used, the depo-time absolute numbers don't matter so long as the rate of increase/decrease in time is constant.

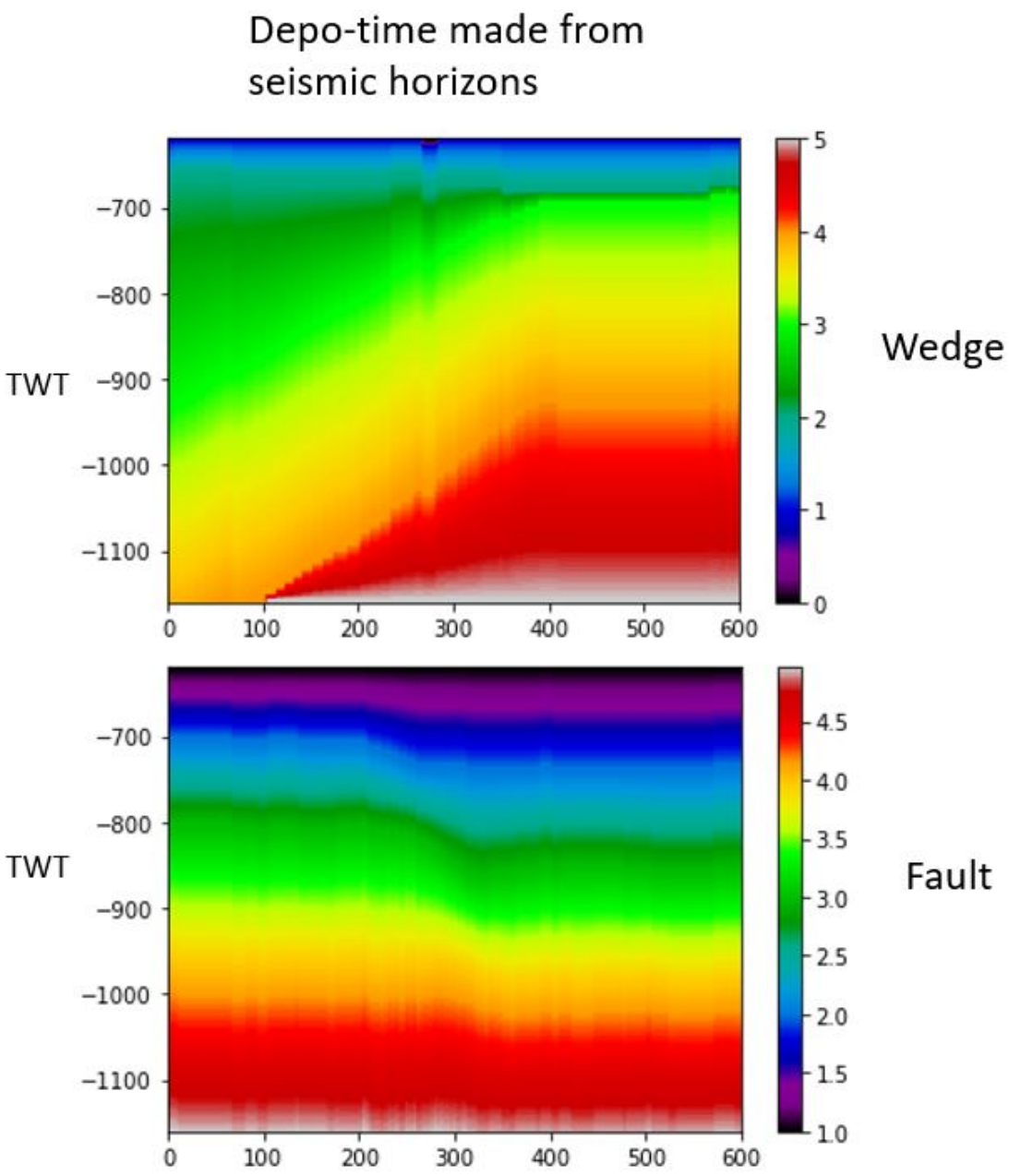

Figure 32: The depo-time constructed from seismic horizons in synthetic models 2 and 3 (heterogeneous wedge and fault). This construction is based on interpolation between the horizons.

3.4.4 Standardization

For ML methods such as neural networks with a sigmoidal activation function and KNN, it is important to standardize the predictors. Standardization ensures that a set of values have a mean equal to zero and a standard deviation equal to 1. Standardization means that variables with different scales, units of measurement, etc. can be compared [23]. For example, acoustic impedance and porosity can be set to the same scale through standardization. This makes equal the scale of predictors so that none have a greater influence than the others. If this is not done, it can result in poor optimization during training. In the case of KNN, the predictors on a larger scale would dominate since distance is important. Standardization is applied using the equation:

$$predictor_{standardized} = \frac{(predictor_v - mean(predictor_v))}{std(predictor_v)} \quad (7)$$

$predictor_v$ stands for predictor vector, i.e. each predictor vector is standardized individually. std() is the standard deviation. Additionally, the training predictors mean and std are saved and latter used in the following function:

$$predictor_{standardized} = \frac{(predictor_{new} - mean(predictor_v))}{std(predictor_v)} \quad (8)$$

This equation is similar to Eq. (7), except that $predictor_{new}$ is the predictor vector in either the validation set or the dataset that lacks known target values. This means that the standardization remains consistent to the standardization in the training dataset.

3.4.5 ML methods Parameter Tuning

The properties of each ML method are largely determined by their parameters. While parameter tuning can be performed using cross-validation, the tuning is only done within a certain specified parameter search range. Figure 33 shows an example of the cross-validation MSE (MSE = mean squared error) plotted against the parameter values. The x-axis shows the parameter range. The orange line in Figure 33 displays where the MSE is closest to zero. The corresponding parameter value on the x-axis will be the right parameter value to use in the model. Determining the range is a matter of starting with a large but coarse range and observing the results. Then, the range is

reduced to the area of best performance (cross validation MSE closest to zero) until a consistent value range of satisfactory performing variables are found. If the MSE closest to zero appears to be outside the search range, the range should be expanded. [23].

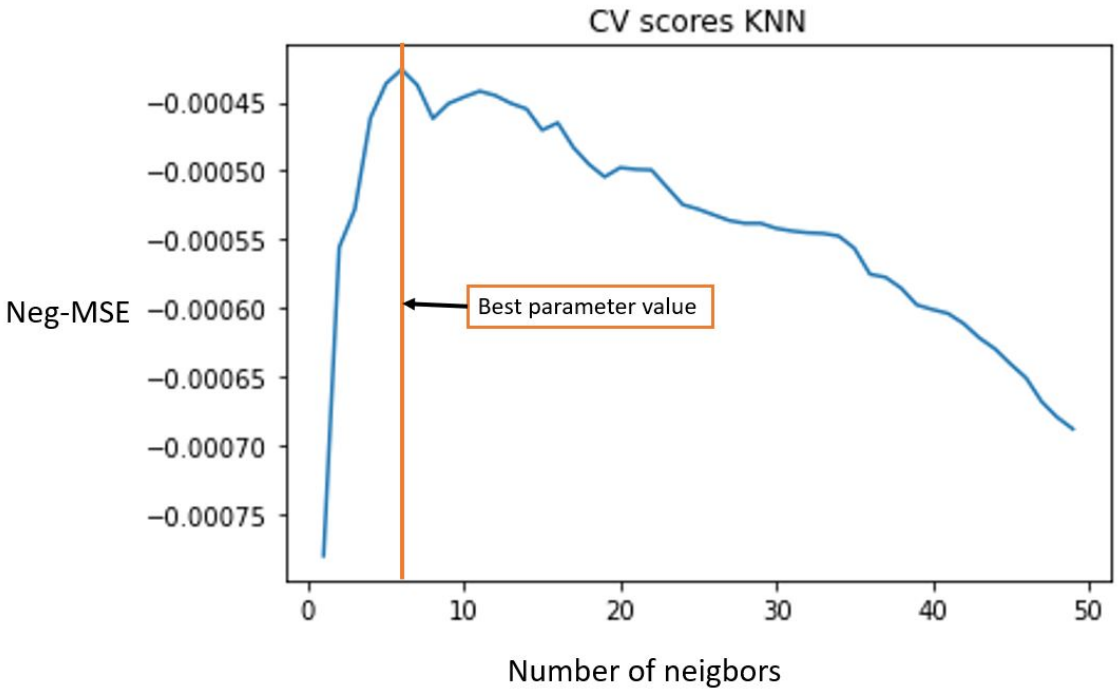

Figure 33: Parameter tuning for the KNN method and specifically the number of neighbors. The optimal number of neighbors is shown as the orange line.

The best parameter search ranges are compiled in Table 3 for the synthetic models, and Table 4 for the F3 dataset.

ML method:	Parameter:	Cross-validation search range:
KNN-regression (case 1)	k	$1 \rightarrow 50$
KNN-regression (cases 2 and 3)	k	$1 \rightarrow 100$
Lasso regression	alpha or λ	$0 \rightarrow 0.005$
Random forest	maximum depth	$4 \rightarrow 9$
Random forest	number of estimators	$25 \rightarrow 110$
Neural network	number of neurons	$1 \rightarrow 40$

Table 3: ML parameter ranges used in cross-validation for the synthetic models.

ML method:	Parameter:	Cross-validation search range:
KNN-regression (case 1)	k	$1 \rightarrow 50$
KNN-regression (cases 2 and 3)	k	$1 \rightarrow 100$
Lasso regression	alpha or λ	$0 \rightarrow 0.005$
Random forest	maximum depth	$1 \rightarrow 20$
Random forest	number of estimators	$25 \rightarrow 110$
Neural network	number of neurons	$1 \rightarrow 40$

Table 4: ML parameter ranges used in cross-validation for the F3 data-set.

3.4.6 F3 well-log issues for Machine learning

To use the window functions, the section and well-log resolution must be equal (4 ms sampling). This can be done by up-scaling the well-logs in Petrel. Upscaling is performed for the porosity and P-impedance logs. This does mean that much of the data are lost in exchange for more predictors. Whether or not this improves the predictions is not known a priori. If the window function is not used, then the well-logs are not up-scaled.

3.5 Evaluating Prediction results

The classical statistical parameters are calculated for the geostatistical and ML approaches. These are: mean absolute error (MAE), mean squared error (MSE), and the r2 score. The r2 score is similar to the classical R-squared metric and is used in the Sklearn module. The best score is 1.0 and the worst score is 0 and negative values [20].

In the synthetic models, to compare the ML models results against the true porosity in the section, the absolute difference at every point on the grid is used. This produces a section of the absolute error, which shows the error in a geometrical/geological context. The cross-validation scores are saved for each ML model.

Boxplots are used to evaluate how the ML models performance changes depending on the cases and predictors. A boxplot is a visualization of the data distribution of one variable. For example, the probability distribution of the MAE. This is illustrated in Figure 34. The box indicates where 50 % of the data are located. The line inside the box is the median of the data. From the box, two vertical lines extend, these are called whiskers. The range of these lines and the box show where the majority of the data are (usually 95-99 % of the data). In this thesis, Pandas is used for making the boxplots. In Pandas, the sum of the whiskers is equal to the range of the box, but they can be slightly shorter. So, the full range should be 100 % aside from outliers [21].

The results comprise every combination of the sub cases, ML methods, and parameters for predictor extraction. This corresponds to a total of 352 models. Because of this large number of results, the overall impact of the different methods will be explored, not the specific impact. For example, the impact of changing the well location for each ML method will not be explored. Instead, for each synthetic model with and without added noise, all the resulting MAE are compiled into a distribution. Then for each parameter (for example ML methods), the compiled MAE are categorised by their values. For example, for case 1 with noise given the parameter ML methods, there will be four boxplots for the MAE distributions, one for each of the ML methods. These distributions contain all combinations of the remaining parameters.

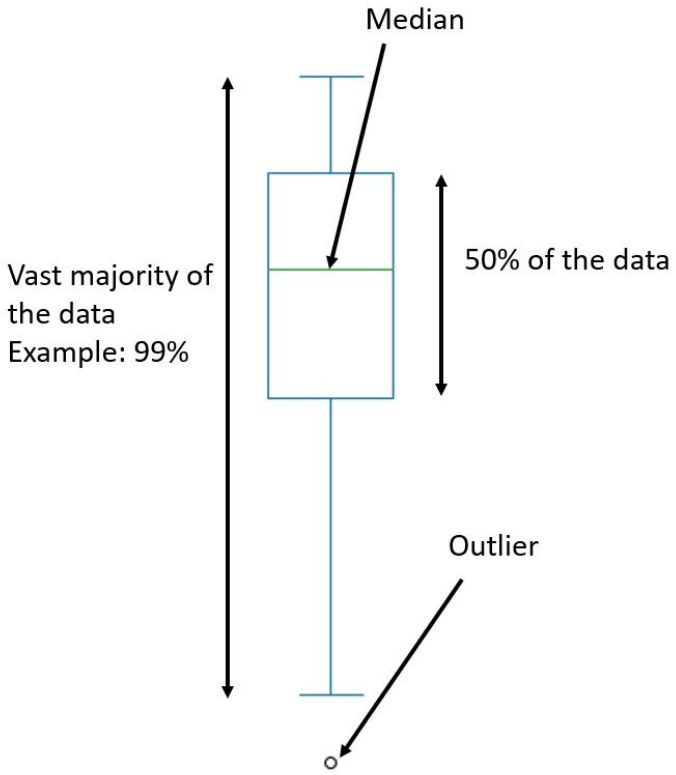

Figure 34: A boxplot made in Pandas using MAE values. The box represents the 50 % range of the data, the line inside the box shows the median of the data. The vertical lines (whiskers) represent where the majority of the data are located. The sum of the whiskers lengths extend no further than the range of the box.

4 Results

As discussed in the methodology, the SGS and ML methods are used to predict the target porosity sections. SGS does this using an impedance cube and the porosity and impedance well-logs. The ML methods always use the impedance section, and synthetic or real well-logs of porosity and impedance. However, using prediction extraction, the ML methods can also work with the rolling window mean, median, data-point selection (see section 3.4.2), and/or the depo-time. Of course, the porosity logs are not used as predictors, but as a target for the training process. When the window functions are used, they are all used at the same time as predictors. Due to the number of results, only the

predictions with added noise will be plotted as sections. This is because noise is expected in real cases, thus these sections are more important to consider.

4.1 Statistical parameters

The statistical parameter used for comparison of the different predictions is the MAE. The other parameters (MSE and r2 score) show the same trends as the MAE. The only exception is the cross-validation (CV) method. The relationship between the CV MSE and MAE is shown in Figure 35. The plot shows that in synthetic cases 1 and 2 (wedge models), for a low CV MSE a low MAE is very likely. However, in synthetic case 3 (fault model), there is no clear relationship. The reason for this has to do with the well locations as it will be discussed latter in section 4.4.2.

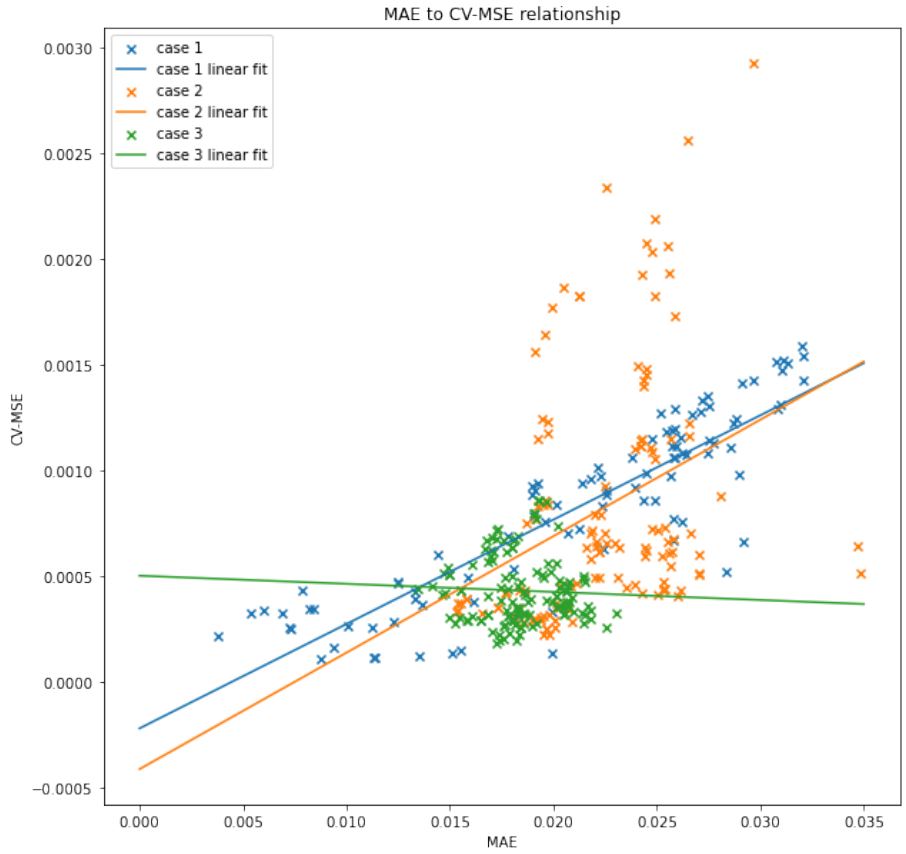

Figure 35: Scatter plot of the CV MSE versus the MAE for synthetic models 1 to 3. Cases 1 and 2 are the wedge models, and case 3 is the normal fault model. Points and linear fits are colored by the case/model.

4.2 Case 1 Homogeneous wedge

Case 1 tests the ML methods performance for predicting the porosity of a homogeneous wedge. Figure 36 is the true porosity section and it is the basis for evaluating the performance of the ML methods. Figure 37 shows the P-impedance sections. The right plot has noise, but it is still possible to discern the wedge geometry.

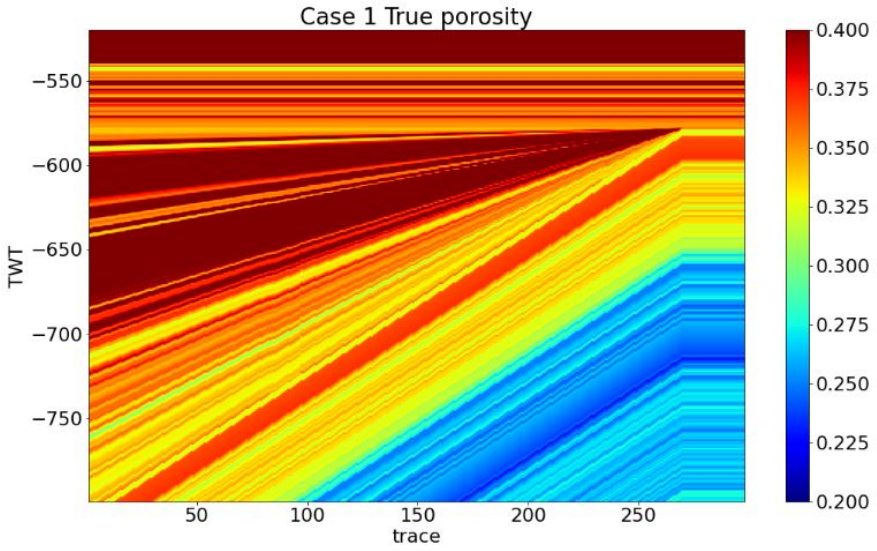

Figure 36: True porosity section of the homogeneous wedge model (case 1).

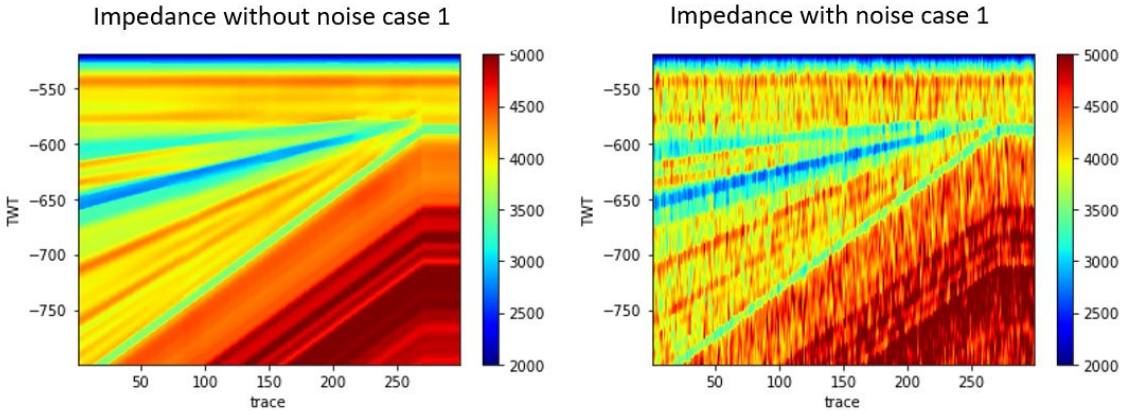

Figure 37: P-impedance sections used for the porosity estimation of case 1 (homogeneous wedge). The left section has no noise, while the right section has noise. The impedance is in kPa.s/m.

4.2.1 ML results

The methods that made the best predictions according to the MAE are shown in Table 5 for the cases without and with noise. The MAE is 0.37% and 1% for the cases without and with noise, respectively. Both predictions used the same well locations and depo-time, but different window size and ML methods (Table 5). These results are useful to keep in mind as we proceed to the boxplots.

Parameter Type:	Parameter value without noise	Parameter value with noise
ML method	KNN	Random Forest
Well location(s)	[100, 200]	[100, 200]
Window size	10	0
Depo-time implemented	True	True
MAE	0.0037	0.01

Table 5: Best prediction for case 1 (homogeneous wedge) with the ML methods and parameters used.

Figure 38 shows the boxplots comparing the impact of different parameters on the mean absolute error (MAE). From the plots a1 and a2, it appears that random forest and KNN regression result in the lowest MAE regardless of noise. Lasso regression is the worst performing ML method, followed by neural network.

Lasso and shallow neural net are less flexible methods (if the N-net has few neurons), so the poor predictions imply that the porosity trend is not simple and requires a more flexible ML method. Figure 38 d1 and d2 show that including the depo-time reduces the error. Figure 38 c1 and c2 show that the window function has little effect on the prediction's overall error, especially for the section with noise. These observations imply that the depo-time provides the ML models with better information on the geometry and geology.

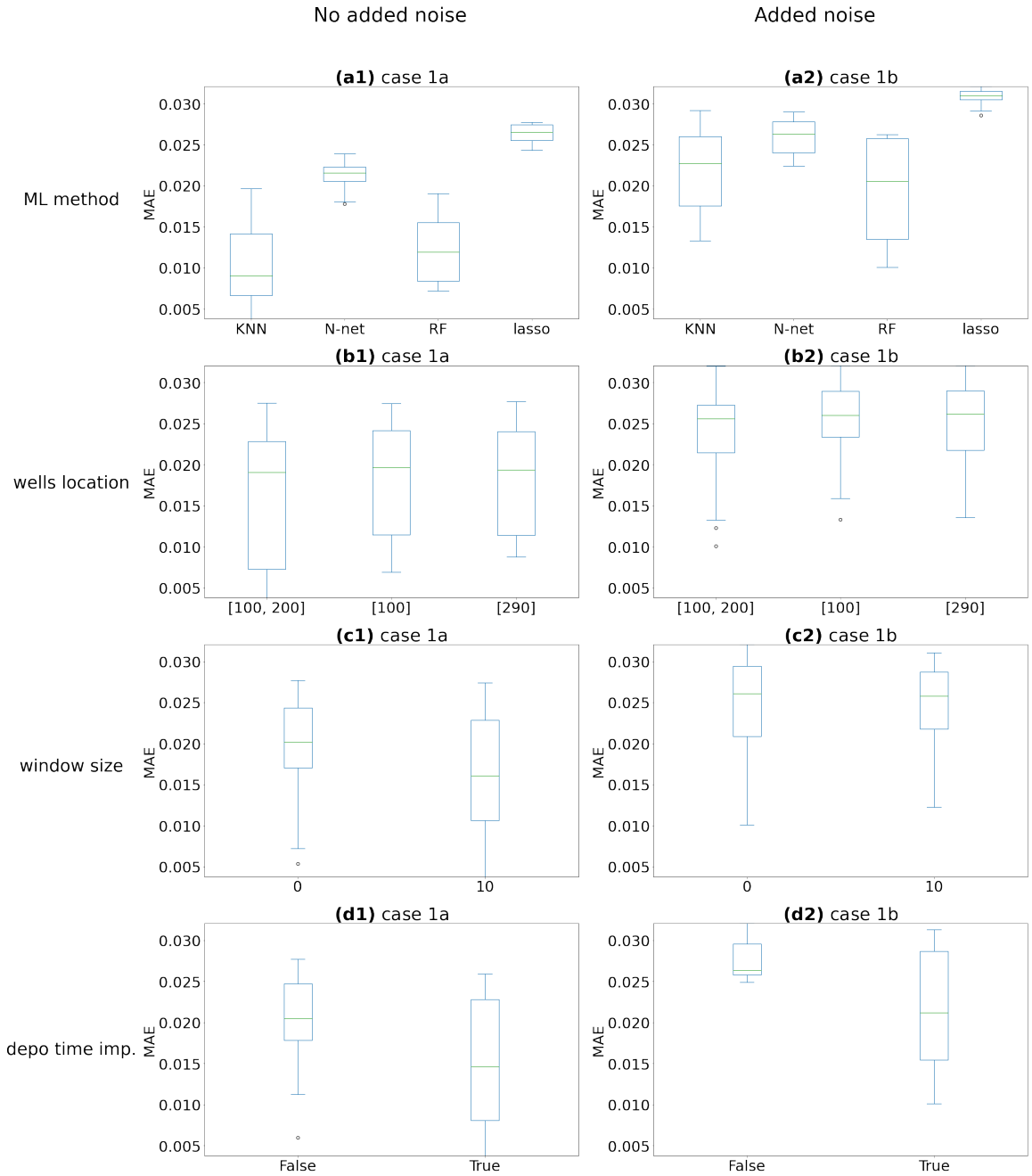

Figure 38: Box plots for homogeneous wedge without noise (a1-d1) and with noise (a2-d2), and exploring the effect of ML method (a1, a2), wells location (b1, b2), window size (c1, c2), and depo-time (d1, d2). For wells location, the numbers withing brackets are the x coordinates where the wells are.

The results in figure 38 b1 and b2 show that the wells locations have little to no effect on the overall accuracy. However, the predicted porosity sections in Figure 39 show that the wells locations matter greatly. Figure 39 a1 and b1 have well location(s) inside the wedge and show good performance in the absolute error plots a2 and b2. Figure 39 c1 shows the prediction using one well at trace 290, which is outside the wedge. The porosity in the wedge in this case is underestimated, and yields a significant error (c2). That the boxplots do not reflect this error means that it is not significant for the overall MAE estimation. However, if the area of interest is inside the wedge, this error would be highly problematic.

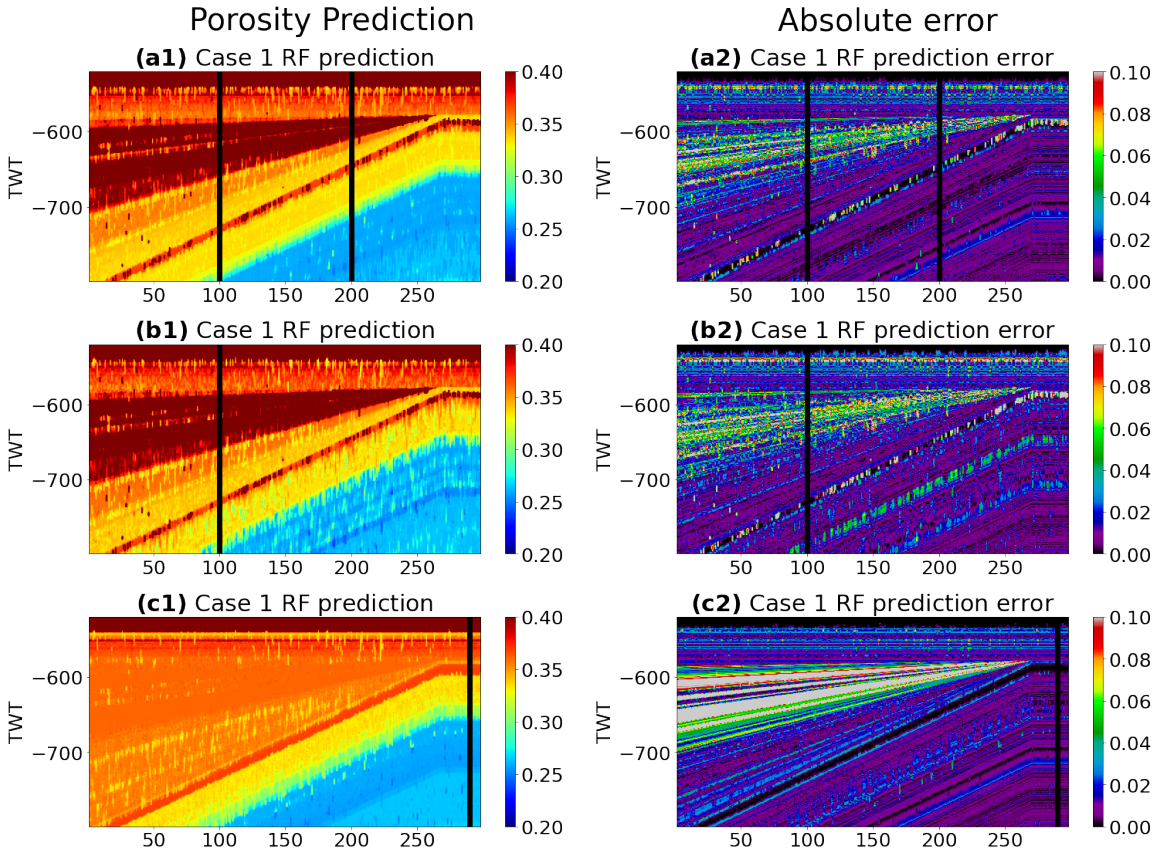

Figure 39: The ML porosity predictions and error sections using the same data in case 1b (with added noise), but with different well location(s). a1, b1 and c1 show the porosity predictions, while a2, b2 and c2 show the absolute error sections. The wells are displayed as black vertical lines.

4.3 Case 2 Heterogeneous wedge

The true porosity section is displayed in Figure 40, while the P-impedance sections are shown in Figure 41, where the left plot has no noise and the right plot has noise. These two sections were used for the ML porosity predictions.

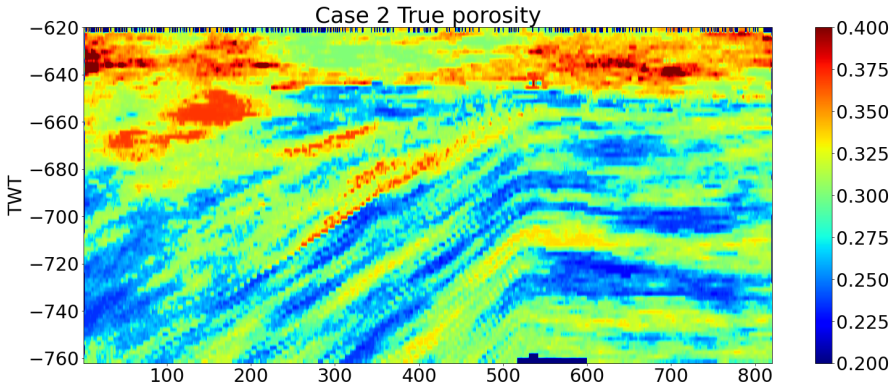

Figure 40: True porosity section in case 2, laterally heterogeneous wedge.

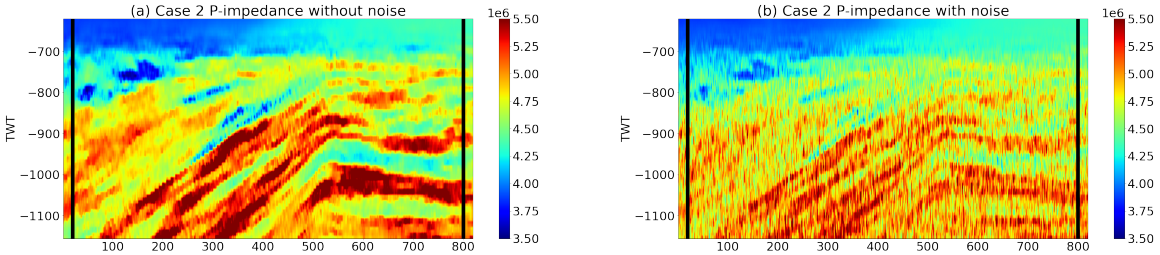

Figure 41: P-impedance sections used for the porosity estimation of case 2 (heterogeneous wedge). (a) has no added noise, while (b) has added noise. The impedance is in Pa.s/m.

4.3.1 Classical approach

Figure 42 shows the statistical error of the SGS prediction. SGS has an MAE of 2.3 % when the P-impedance has no added noise, and 2.5 % when noise was added to the P-impedance. Figure 43 shows the porosity prediction using SGS to the left and the absolute error to the right. The SGS method captures the general structure and heterogeneity of the sedimentary wedge well, aside from the area to the right where it overestimates the porosity.

case	MAE	MSE	r2 score	abs error std
case 2a wedge hetero	0.023256	0.000972	0.477209	0.020761

case	MAE	MSE	r2 score	abs error std
case 2b wedge hetero	0.025217	0.001046	0.437161	0.020258

Figure 42: Statistics of the SGS prediction in case 2, heterogeneous wedge. The upper row is the prediction without noise, while the lower row is the prediction with added noise.

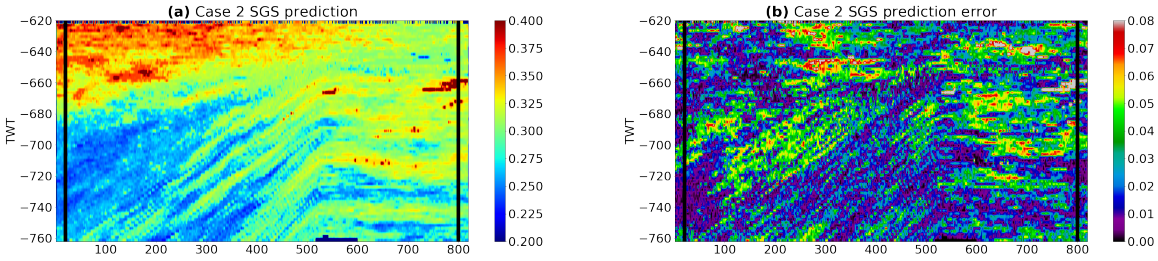

Figure 43: Predicted porosity section for case 2 using SGS. The left plot shows the predicted porosity section. The right plot shows the absolute difference between the predicted and the true porosity.

4.3.2 ML results

The methods that made the best predictions with the lowest MAE are shown in Table 6, for the cases without and with noise. The MAE are 1.5% and 2%, for the cases without noise and with noise, respectively. These are better MAE results than the SGS method just discussed. The predictions use the same window size and depo-time, but different well-locations and ML methods (Table 6). As in case 1, it is useful to keep in mind these results as we proceed to the boxplots.

Parameter Type:	Parameter value without noise	Parameter value with noise
ML method	Neural net	Lasso
Well location(s)	[200, 350, 500, 700]	[700, 750, 650, 775]
Window size	10	10
Depo-time implemented	True	True
MAE	0.015	0.021

Table 6: Best ML predictions for case 2 (heterogeneous wedge) with the parameters used.

Figure 44 shows the boxplots of the MAE using the ML methods. The left column (a1-d1) is the section without noise, and the right column (a2-d2) is the section with noise. Figure 44 a1 and a2 show that the ML methods perform similarly in terms of the median MAE. The KNN and random forest perform decently with a wide distribution of MAE when there is no noise (Figure 44 a1). However, when noise is added, the KNN performs worse and has a narrower MAE range than the other methods (Figure 44 a2). This is regardless the predictors or well-locations. The same plot shows that the random forest (RF) method has a wide error distribution, but it does perform well given the correct well-location(s) and predictors, or it can perform very poorly if this is not the case. Lasso is the best performing method consistently, and has a narrow MAE distribution.

Figure 44 d1 and d2 show that the depo-time has little effect on the accuracy of the prediction. However, the best models in Table 6 use the depo-time, meaning that this implementation has a positive effect. Figure 44 c1 and c2 show that the window implementation has a positive effect on the prediction. These observations imply that, opposite from case 1, in case 2 the window function provides better information on the geometry and geology. Figure 44 b2 shows that with added noise, the well-locations produce similar MAE distributions. The first well-location [200, 350, 500, 700] (wells spread throughout the entire section) has a wider MAE distribution skewed towards lower MAE. While the difference with the other well locations is small, this could imply that the first well-location provides more reliable data. This is supported by Figure 44 b1, where the first well-location is clearly better than the other combinations.

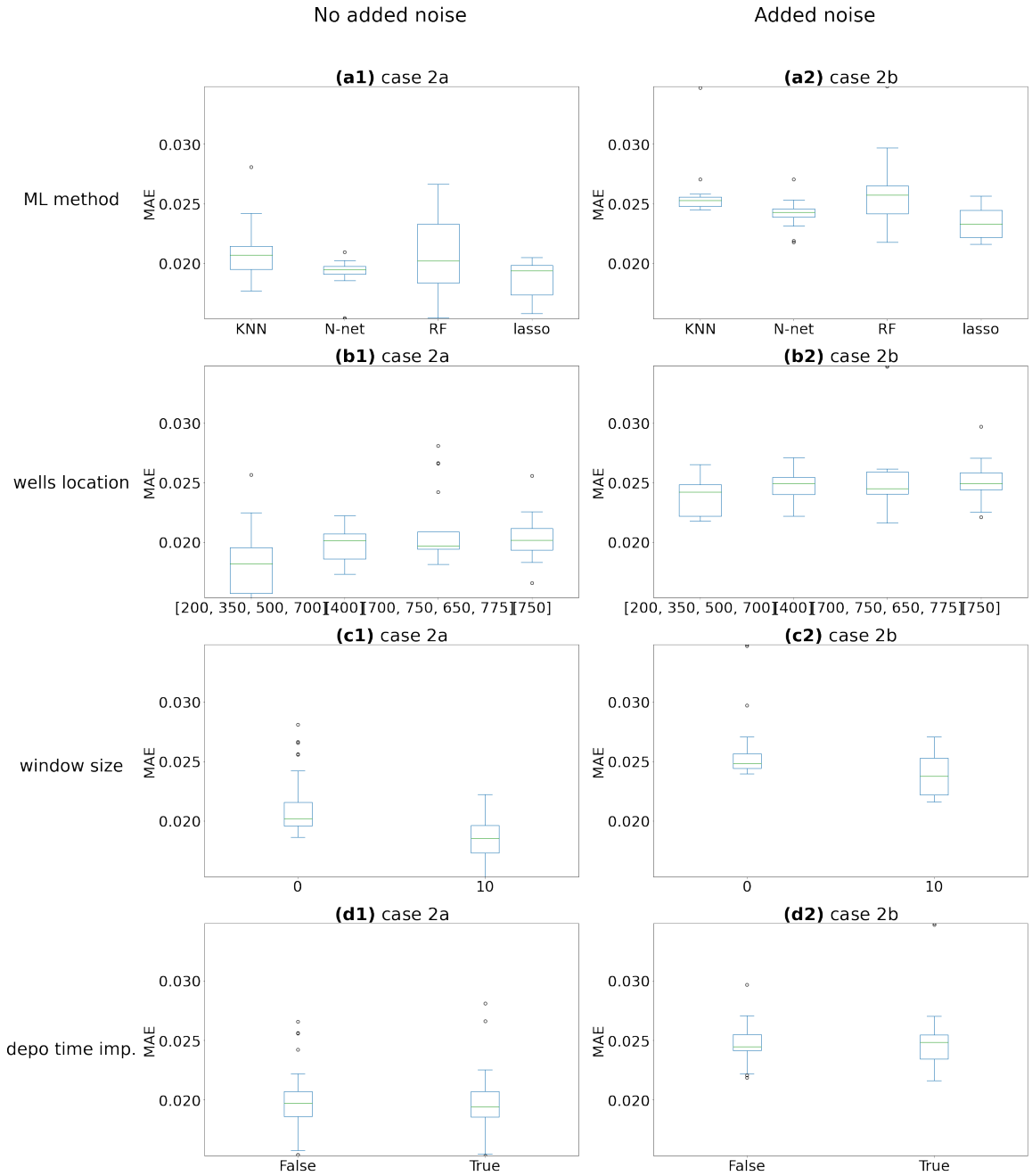

Figure 44: Box plots for heterogeneous wedge without noise (a1-d1) and with noise (a2-d2), and exploring the effect of the ML method (a1, a2), wells location (b1, b2), window size (c1, c2), and depo-time (d1, d2). For wells location, the numbers withing brackets are the x coordinates where the wells are.

In Figure 44 a2 to d2, one can observe two outliers with the worst performance. This occurs for the KNN and RF methods when the wells are concentrated outside the wedge, the window function is not used, and the depo-time is used. This supports the observations made so far and confirms that while the well-locations do not have a large effect, if the wells are placed outside the wedge, the predictions can be very poor.

Figure 45 shows the best predictions based on the well(s) location(s). Case (a) has one well inside of the wedge. Case (b) has four wells concentrated to the right of the section. Case (c) has four wells spread throughout the section. The sections and corresponding error plots (a2, b2, c2) are similar. However, compared to the SGS prediction and true porosity shown in figures 43 and 40 respectively, the ML predictions display less detail. Specifically, the predicted porosity (a1, b1, c1) shows less clear layering than the true section. One could say that it looks like the ML models are interpolating the porosity between the layers. The layers containing higher true porosity (0.325 to 0.4) are predicted as lower porosity values ca. 0.3. This is probably an effect of optimizing the error, meaning that the ML methods predict the most common porosity values to avoid large individual errors.

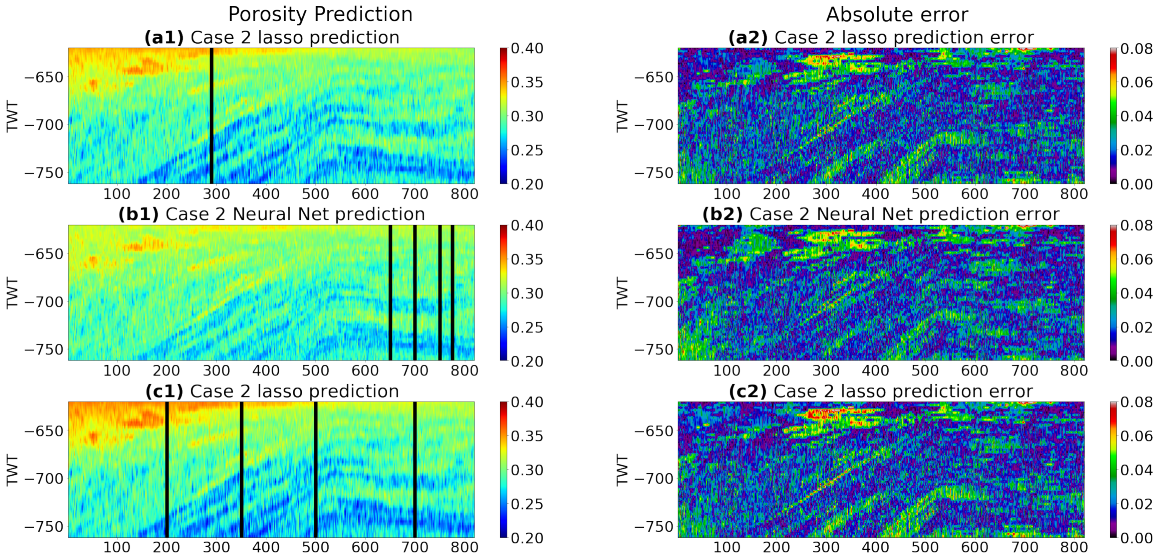

Figure 45: The ML porosity predictions and error sections using the same data than in case 2b (with added noise), but with different well location(s). (a1, b1, c1) show the porosity predictions, (a2, b2, c2) show the absolute error sections. The wells are displayed as black vertical lines.

4.4 Case 3 Normal Fault

The synthetic porosity section in the normal fault case is shown in Figure 46. The P-impedance sections used for the porosity estimation are shown in Figure 47, the left plot is without noise and the right plot has noise. Notably, the fault trace is clear in the P-impedance section without noise, but it is less clear in the P-impedance section with noise.

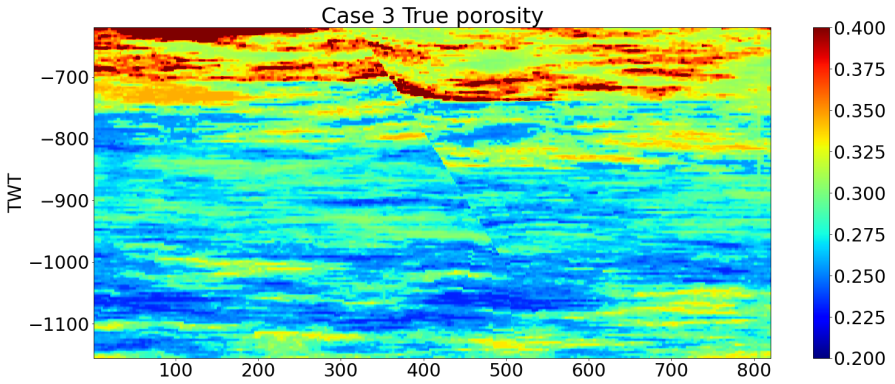

Figure 46: True porosity section in case 3, normal fault.

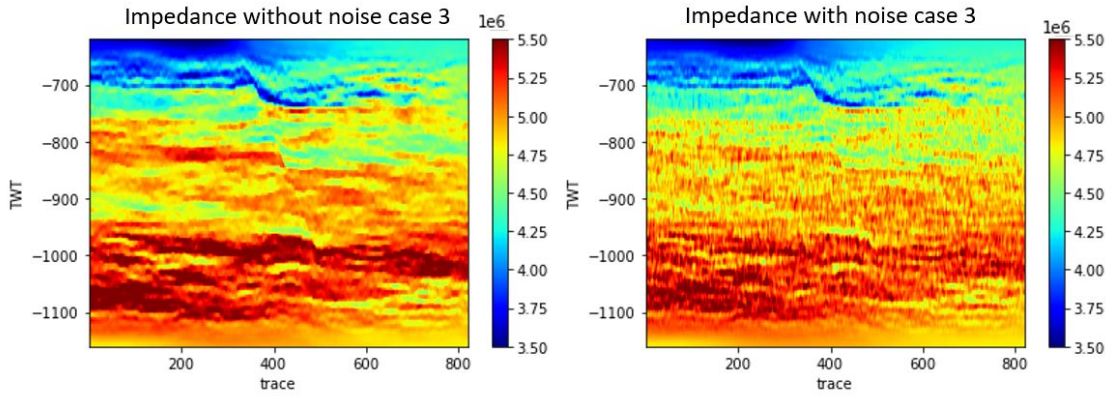

Figure 47: P-impedance sections used for the porosity estimation of case 3 (normal fault). The left section has no added noise, while the right section has added noise. The impedance is in Pa.s/m.

4.4.1 Classical approach

The MAE of the SGS porosity predictions are shown in Figure 48. The MAE for the section without added noise is 2.13%, while in the section with noise the MAE is 2.27%. The porosity prediction for the section with noise is displayed in Figure 49 left, while Figure 49 right shows the absolute error. The absolute error shows that the porosity is largely overestimated in the top left of the section. However, for the rest of the section, the SGS method captures the fault well and displays heterogeneity similar to the true porosity section.

case	MAE	MSE	r2 score	abs error std
case 3a fault	0.021321	0.001354	0.259757	0.029984
case	**MAE**	**MSE**	**r2 score**	**abs error std**
case 3b fault	0.022717	0.001444	0.210382	0.03046

Figure 48: Statistics of the SGS prediction in case 3, normal fault. The first row is the prediction without noise, while the lower row is the prediction with added noise.

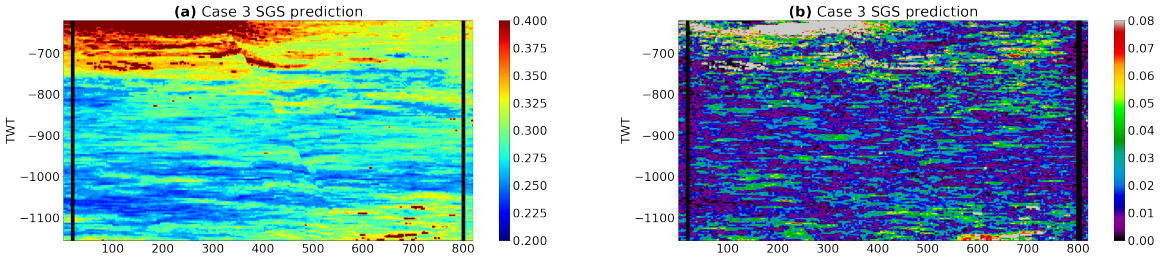

Figure 49: Predicted porosity section in case 3 using SGS. The left plot shows the porosity section. The right plot shows the absolute difference between the predicted and true porosity.

4.4.2 ML results

The best predictions with the lowest MAE are shown in Table 7, for the cases without and with noise. The MAE are 1.3% and 1.6% for the sections without and with noise, respectively. These are better MAE results than the SGS method discussed in the previous section. The best models use the same parameters in both cases, meaning that the noise level has less effect on the parameter optimization than in cases 1 and 2. As in cases 1 and 2, it is useful to keep in mind these results as

we proceed to the boxplots.

Parameter Type:	Parameter value without noise	Parameter value with noise
ML method	Random forest	Random forest
Well location(s)	[200, 350, 500, 700]	[200, 350, 500, 700]
Window size	10	10
Depo-time implemented	True	True
MAE	0.013	0.016

Table 7: Best prediction for case 3 (Normal fault) with the parameters used.

Figure 50 shows the MAE boxplots of the ML predictions. a1-d1 are predictions using the impedance section without noise, while a2-d2 are predictions using the impedance section with noise. Regardless of the noise level, the inclusion of the window functions tend to lower the MAE, as shown by c1 and c2. However, d1 and d2 show that the depo-time implementation has little to no effect on the MAE. As in case 2, these observations imply that the window functions better informs the ML models on the geometry and geology. Additionally, Table 7 shows that the best predictions use the depo-time, implying that this implementation has a positive effect.

Figure 50 a1 and a2 show that the ML methods have similar MAE distributions, although the best predictions use the RF method. Plot (a2) shows more clearly that this is due to the neural network and RF methods producing a wider range of MAE results, both better and worse than the KNN and lasso methods. This means that the neural network and RF methods must be given good parameters, and that the other methods might be more robust.

From Figure 50 b1 and b2, the impact of the well locations shows a clear pattern. Spreading the well locations throughout the section ([200, 350, 500, 700]) causes the lowest MAE. Having the well go straight through the fault ([400]) produces the second best results. The worst predictions occur when the well(s) are located to one side of the section regardless of the amount of wells.

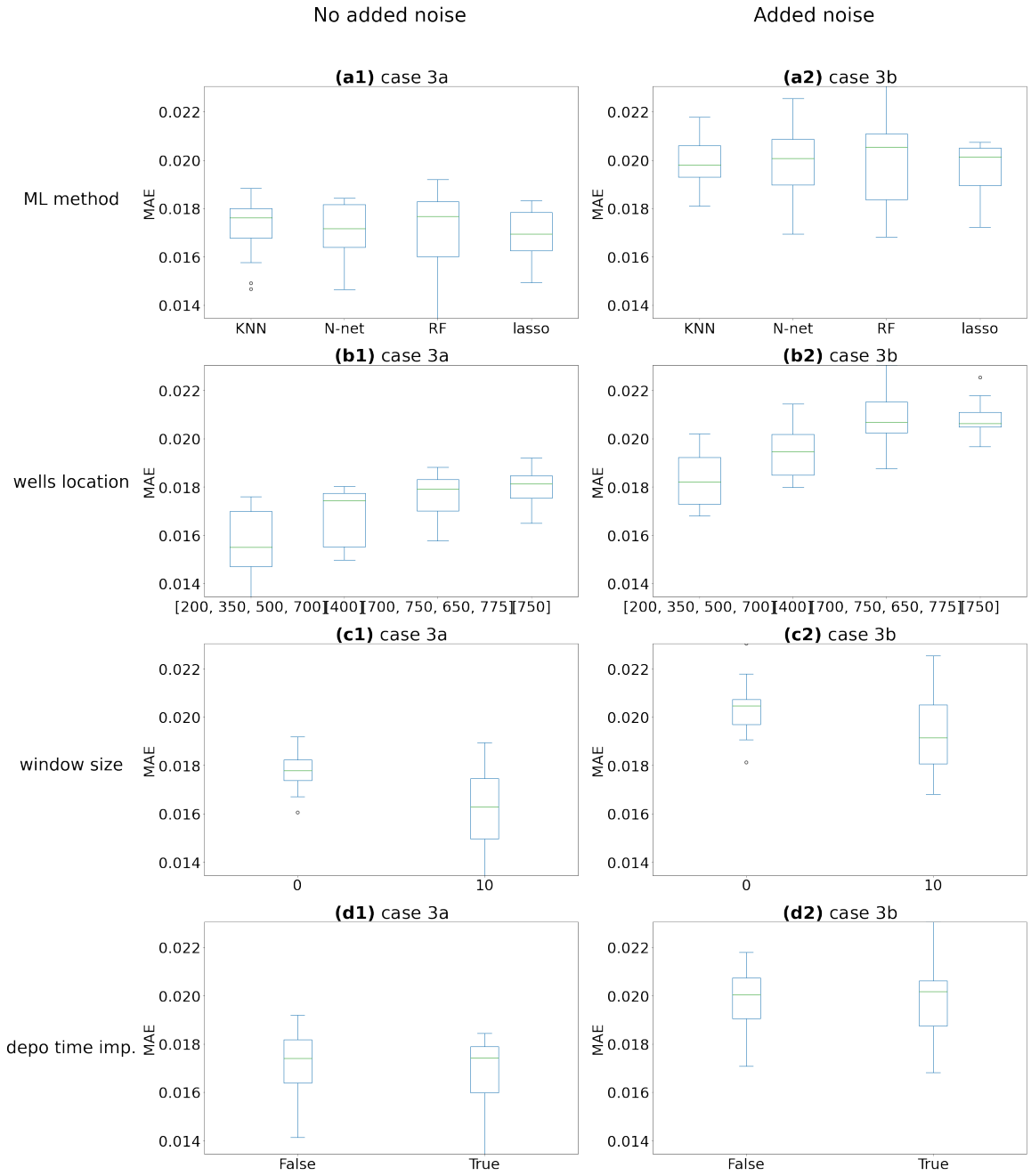

Figure 50: Box plots for case 3, normal fault without noise (a1-d1) and with noise (a2-d2), and exploring the effect of ML method (a1, a2), wells location (b1, b2), window size (c1, c2), and depo-time (d1, d2). For wells location, the numbers withing brackets are the x coordinates where the wells are.

As discussed in section 4.1, case 3 has a unique relationship between the CV MSE and the MAE. Boxplots show that this relationship is usually the same except for the cases displayed in Figure 51. This figure shows the boxplots of the cross-validation MSE by the well locations, which indicate that having a well through the middle of the section (around trace 400) results in a higher error. This is the opposite to the equivalent MAE shown in Figure 50 b1 and b2. The reason is likely poor data quality in the area around trace 750. This is based on this location resulting in high MAE. The cross-validation MSE is low in this area because it was easy to train to the data, but the data was clearly not representative of the full sectionresulting in a higer MAE.

Cross-validation uses only the training data, that is the data at the well locations, while MAE uses the full section. Thus, low cross-validation MSE and high MAE implies that the training data did not capture the important data patterns. If this is the reason, it shows that while cross-validation can limit the effect of over-training, if the training data is of poor quality, the results will most likely have a high degree of error.

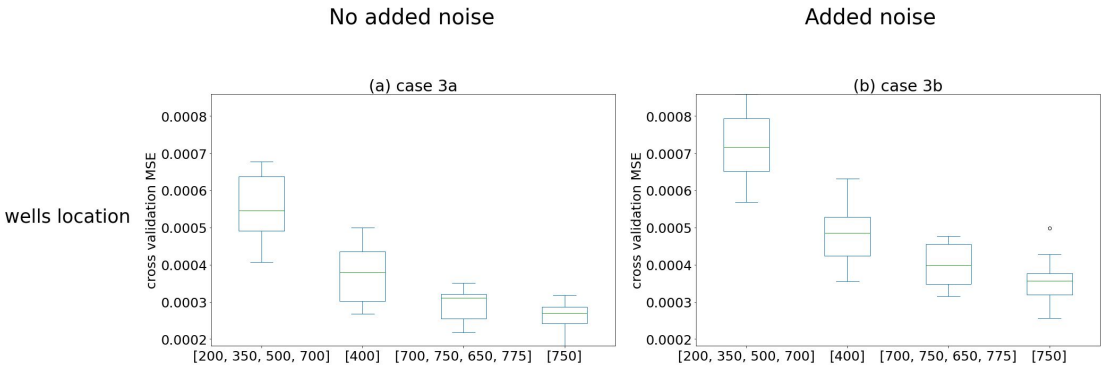

Figure 51: Box plots for normal fault without noise (a) and with noise (b), and exploring the effect of the wells location (a, b) on the cross-validation MSE.

Some of the predicted sections and their error plots are displayed in Figure 52. The best prediction based on the well locations are displayed in each row. Case (a) has the wells spread throughout the section. Case (b) uses a single well through the fault and looks similar to the prediction in case (a). The last case (c) has the worst results of the three, having four wells concentrated towards the

right of the section. This prediction displays some artifacts in the middle of the section, additionally it lacks detail compared to the predictions in (a) and (b). This supports the conclusion from the boxplots in Figure 50 b1 and b2, that the wells should cross the fault to produce a good prediction. Plot (c1) uses four wells, yet performs worse than (b1). This underlines a point that more data do not necessarily improve the predictions, and that the quality of the data is more important. All these predictions used depo-time and the window functions.

Comparing these results to the SGS prediction in Figure 49, we get similar observations to those in case 2. The ML methods predict a smaller range of values near the most common values, producing sections with lower heterogeneity. This means that the porosity in many of the layers with high or low porosity is replaced with a porosity value close to the mean. The effect of this is that detailed information is blurred or removed.

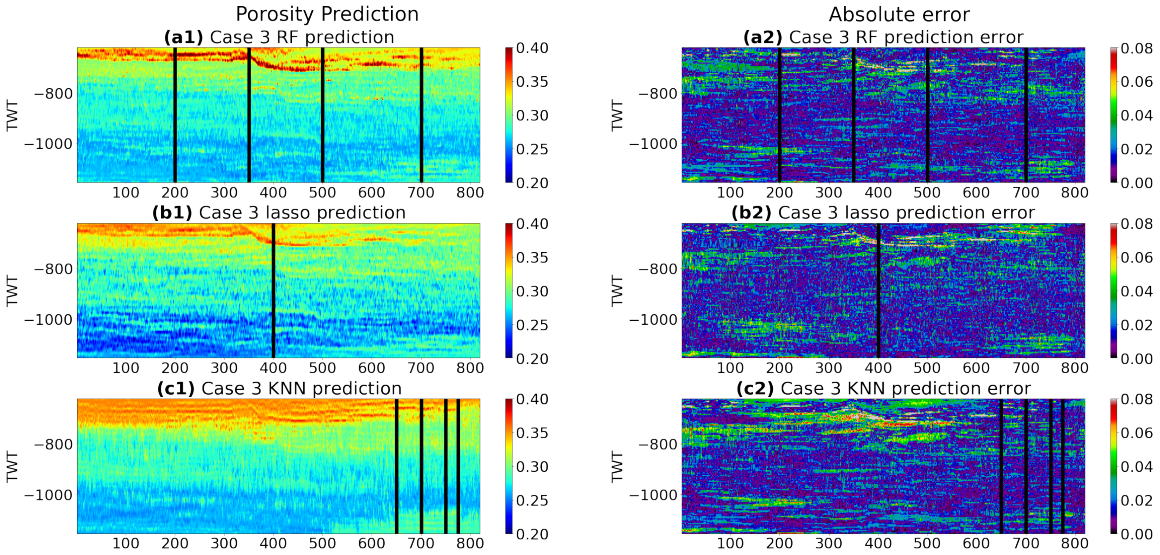

Figure 52: ML porosity predictions and error sections using the same data as in case 3b (with added noise), but with different well location(s). (a1, b1, c1) show the porosity predictions, (a2, b2, c2) show the absolute error sections. The well-placements are displayed as black vertical lines.

4.5 Noise impact and general comparison between cases and approaches

Figure 53 shows each case's box-plot based on the MAE. Figure 53A shows the prediction error using ML methods, while Figure 53B shows the MAE using SGS. A comparison of the median of the ML and SGS methods shows that the ML methods tend to produce lower MAE. This difference, however, is small, circa 0.3 % in MAE.

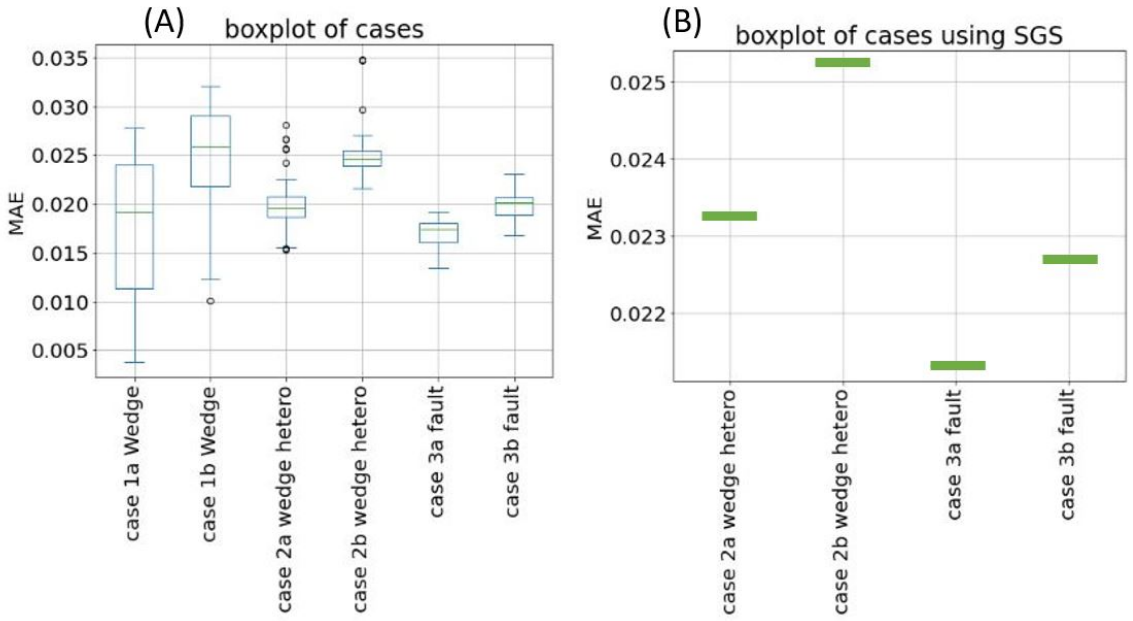

Figure 53: Box-plots comparing the predictions of the different synthetic cases based on the MAE. Cases xa are sections without noise, cases xb are sections with noise. (A) Errors of the ML predictions. (B) Errors of the SGS predictions. These last ones are shown as lines since there is only one SGS prediction for each case.

Figure 53 shows that all of the cases (xa) without spectral noise perform better than the cases (xb) with spectral noise. This is the expected result. Additionally for the ML predictions, case 1 has a wider range of MAE values. This implies that in case 1, the MAE varies more depending on the parameters of the ML model than in the other cases.

4.6 F3 case

4.6.1 Classical approach

The impedance section and porosity prediction using SGS is shown in Figure 54. The porosity section shows that the SGS method displays little vertical heterogeneity, but tends to increase the porosity from left to right. This lateral increase of porosity is likely due to the flow direction of the river delta being from right (coarser grains) to left (finer grains).

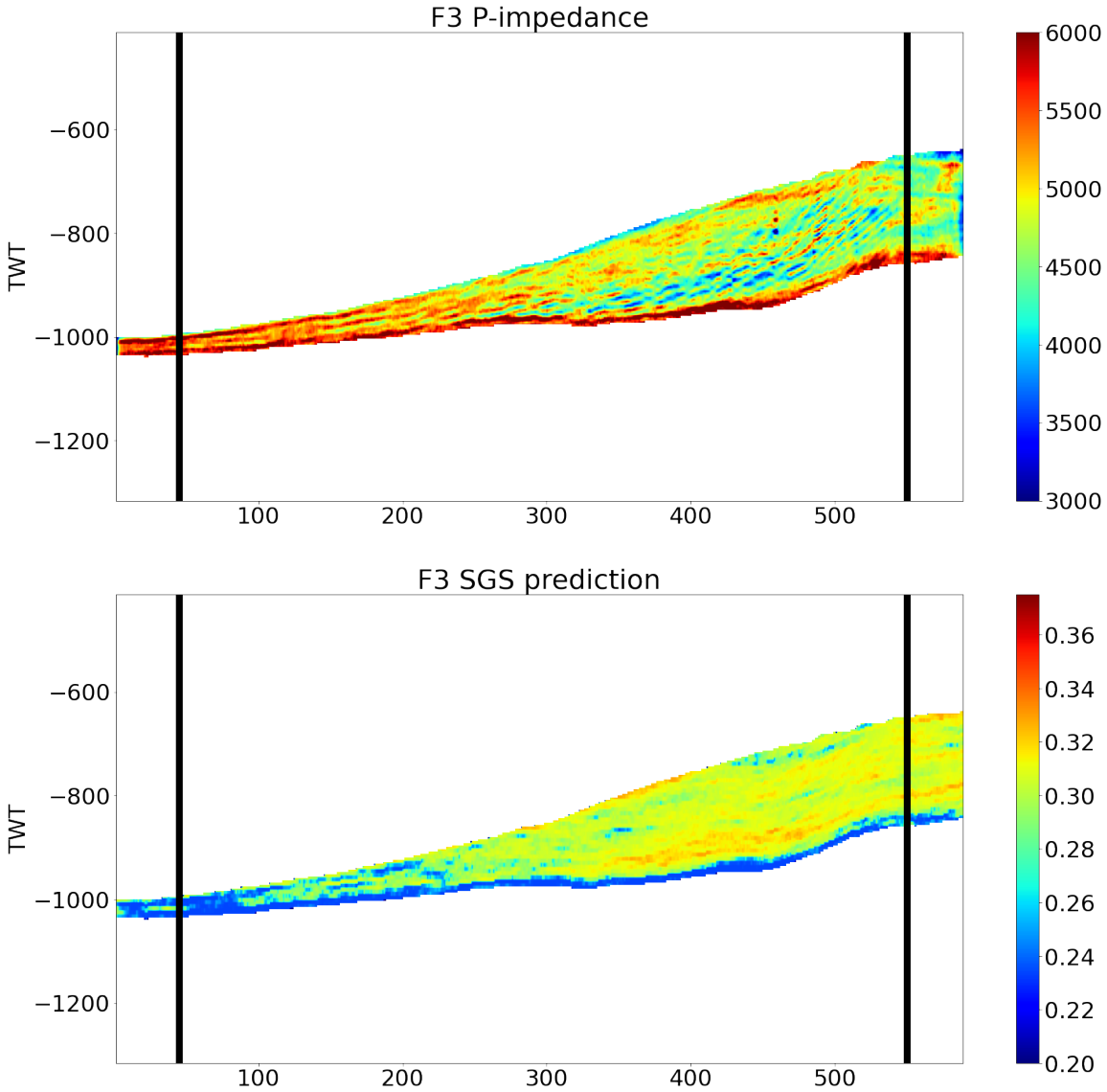

Figure 54: Top: P-impedance section of the F3 dataset from seismic inversion. Bottom: Predicted porosity section for the F3 dataset using SGS. The impedance is in kPa.s/m.

4.6.2 ML results

The results from the synthetic models and MAE boxplots (e.g., Figure 53) were used to select the most likely accurate ML methods for the F3 case. The effect of the window functions and depo-time implementation on the MAE depended on the model, but consistently has a positive effect. The

effectiveness of the ML methods is not always clear from the results, except for the KNN method which consistently was outperformed by the other methods. Figure 55 shows the F3 ML porosity predictions using the random forest (a1 and b1), neural network (c1), and lasso (d1) methods. All these predictions use depo-time, but case a1 does not use the window function, while the other cases b1 to d1 use the window function. In Figure 55, the predictions are ordered from top to bottom by increasing cross-validation MSE. Case a1 has the lowest cross-validation MSE, since this case does not use the window functions and up scaling of the well-logs is not necessary. This means that this model has access to more data.

Figure 55 b2 and c2 show that the prediction using RF and neural network mostly agrees with the SGS method. This is clear by the comparatively large amount of purple in the corresponding difference plots to the right. The lasso prediction (d1) shows the least amount of heterogeneity (detail), populating the section with mostly high porosity values. The neural network prediction (c1) tends to only predict medium (ca. 30%) or high (ca. 40%) porosity. Figure 55 a1 and b1 show that RF produces a larger variety of porosity values. Significantly, RF without a window function (a1) produces thin layers that are laterally consistent. This could mean that using RF with a large weight on the depo-time produces a section with more detail. Based on the results of the synthetic models this is likely to increase the error by a small amount, as the window function almost always reduced the error.

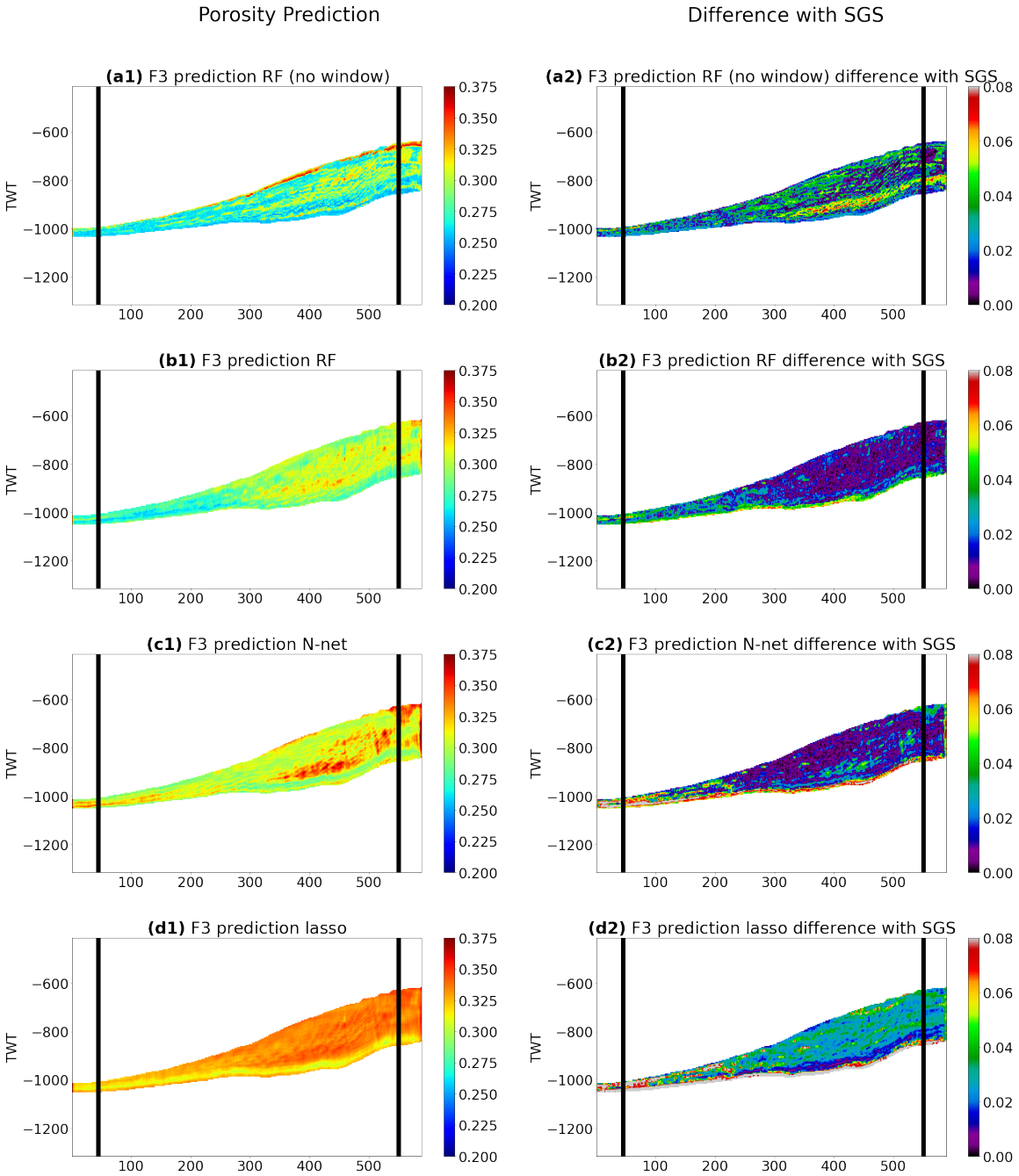

Figure 55: ML predictions of the F3 section. (a1-d1) show the porosity predictions, (a2-d2) show the difference between the ML and the SGS predictions. (a1,a2) use the random forest prediction without a window function, and (b1, b2) use the random forest prediction with a window function. The rest of the predictors use the window function. (c1, c2) use the neural network method, and (d1, d2) use the lasso method. All predictions use the depo-time implementation.

4.7 Impedance-Porosity relationship

Figure 56A shows the scatter plot of the impedance well-logs against the impedance from the seismic inversion, after these data have been assigned to the F3 model 3D grid. The linear correlation coefficient equals 0.7. If the seismic inversion was perfect, this coefficient should equal 1.0. Since this is not the case, this means that the inversion affects the porosity to impedance trend. Figure 56B shows the correlation between the porosity from the well-logs and the impedance from the seismic inversion in the area of interest of the F3 3D model. These data and correlation are used in the SGS method.

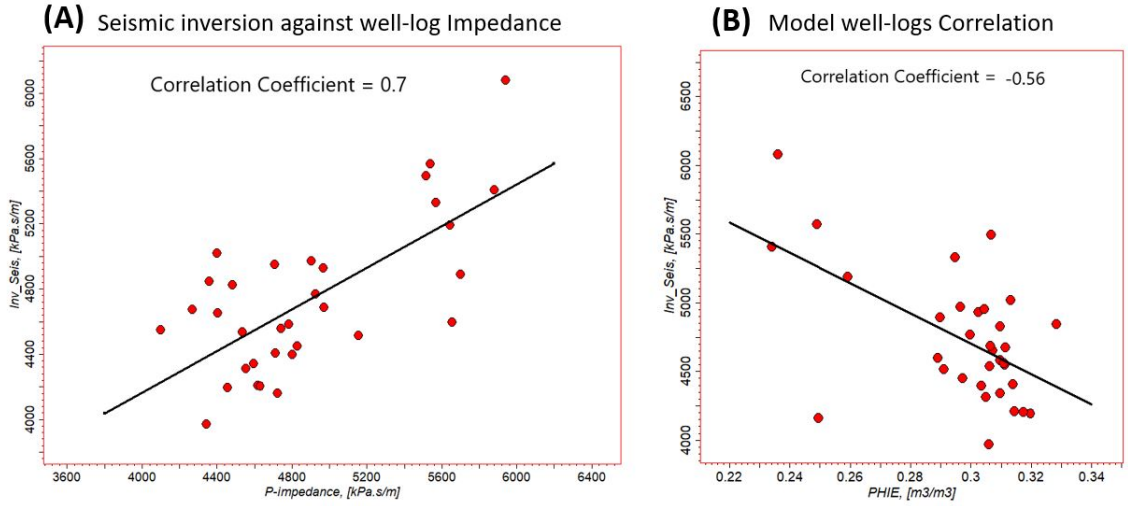

Figure 56: Plot A shows the correlation between the well-logs impedance and the impedance from seismic inversion. Plot B shows the correlation between the porosity from well-logs and the impedance from seismic inversion. This correlation is used in the SGS algorithm.

Figure 57 shows the scatter plots of the porosity against the impedance. The blue points represent the well-log data. The orange points show the distribution of the predicted porosity based on the P-impedance for both the SGS and ML methods. From Figure 56B it is expected that the porosity should increase as the impedance decreases. From plot (a) in Figure 57 this is mostly true, though it is likely that the inversion makes this relationship less clear.

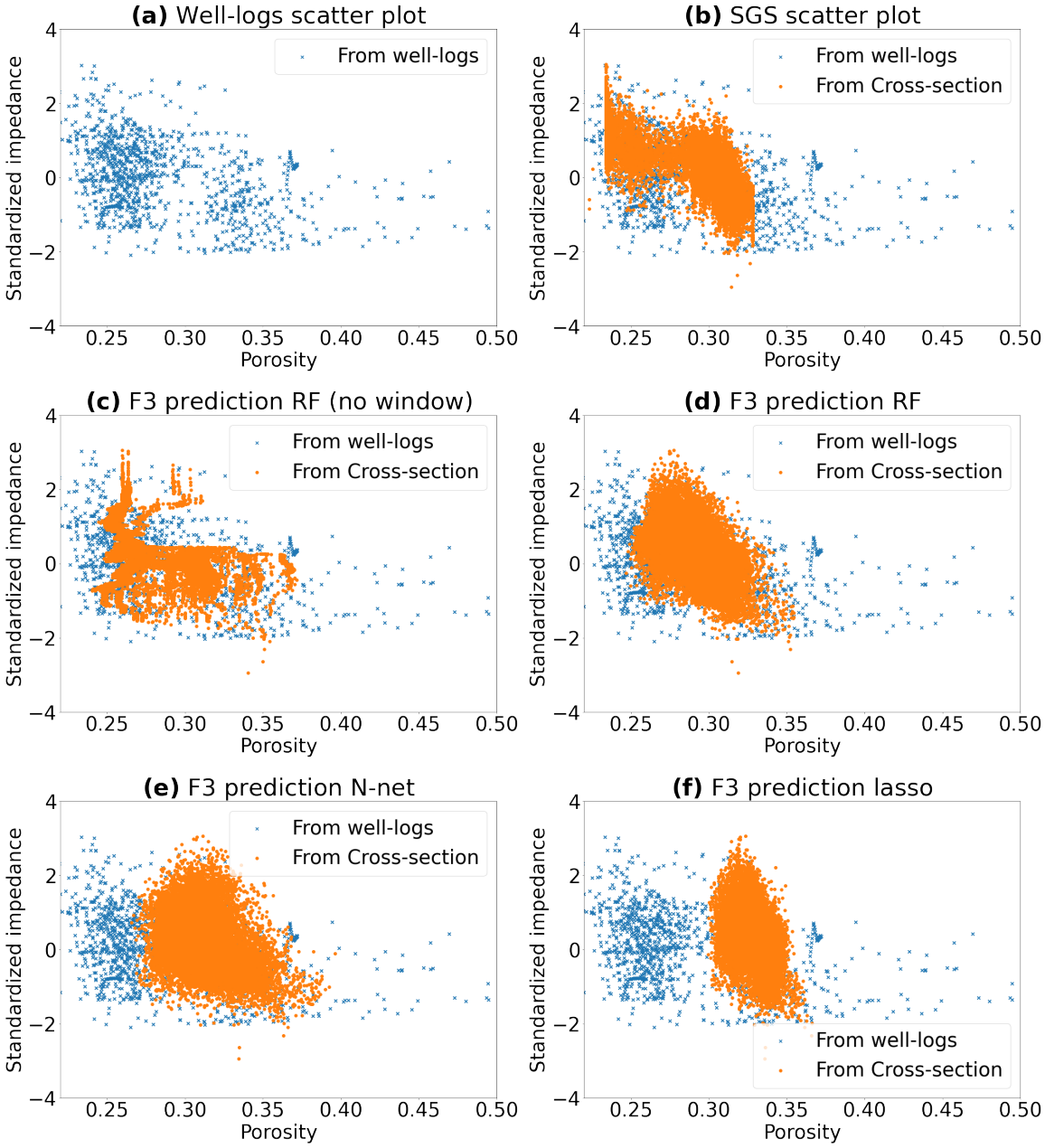

Figure 57: Porosity against impedance for the F3 case. Only the relevant portion of the plots are shown for easier comparison, all axes limits are the same. (a) shows the well-log data, this plot is repeated in the other plots for comparison. (b) shows the SGS results in orange. (c-f) show the results using the ML methods included in Figure 55.

Figure 57a shows a portion of the well-log data relevant to the wedge area considered. The range in the scatter plots includes the expected ranges of impedance and porosity data in this region. Figure 57b shows the SGS results in orange. SGS is expected to follow the original data distribution, yet this is clearly not the case. This is because the scatter plot is only for the 2D section, while SGS predicts a 3D model. Thus, this plot is fine. In fact, it shows a clear trend of decreasing porosity with increasing impedance. However, the data are concentrated at the higher impedance and lower porosity values. Additionally, the porosity values are cut off at approximately 0.23 and 0.33. This is because this is the main porosity range of the well-logs in the area of interest.

Figure 57 d to f show the predictions for the ML RF, N-net, and lasso methods, respectively. All these three methods use the depo-time and window-functions implementations. The ML predictions show little to no porosity-impedance trend, lasso (f) in particular has little correlation to the well-log data compared to the other SGS and ML methods. RF with no window function (Figure 57c) shows a different pattern from the other ML methods. It displays arcs of porosity predictions that deviate from the more common distribution. Since this prediction did not use the window functions, these deviations are most likely an effect of the depo-time. The case in Figure 57c is the prediction shown in Figure 55 a1, which displays the most heterogeneity (detail) and clear layering. That the model uses depo-time instead of the window functions implies that the predicted features occur because the model weights the depo-time highly. This means that if the interpretation of the seismic horizons are accurate, the ML model will predict similar porosity along the horizons.

5 Discussion

Results:

Figure 56 shows that the seismic inversion is not ideal and certainly has an effect on the porosity prediction results. However, testing the methods ability to handle this uncertainty in seismic impedance is valuable, as it gives insight into the robustness of the methods. Since this was likely a consistent issue in cases 2, 3 and the F3 block, it is likely that the methods that performed poorly in these cases would perform better with a better seismic inversion.

The ML methods consistently have lower MAE than the SGS method (Figure (53). This is reasonable since the ML methods are trained to reduce the error, while the SGS algorithm is made to produce a similar porosity distribution in the porosity cube and in the well-logs. A possible consequence of this is that in the synthetic models, the ML methods have a tendency to default to a small range of values to minimize the error as seen in cases 2 and 3, while the SGS method seems to display more heterogeneity and layering.

However for the F3 case, this situation is reversed. The SGS method, neural network and lasso methods (Figures 54 and 55 c and d) show the tendency to reduce the heterogeneity in order to reduce the error. The RF method however, displays more heterogeneity while having a better cross-validation score than the other methods (Figure 55 a and b). This might be because the RF method uses decision trees, which make decision regions that separate data points displaying different patterns. The RF model that does not use the window functions has the clearest thin layering. The arc-like deviations of this model from the normal porosity-impedance distribution (Figure 57 c) indicate that this is probably an effect of the depo-time being weighted highly. Whether this is preferable depends on the validity of the assumption that the depo-time or geological time is significant. Also, this model has the lowest cross-validation MSE, implying that the cross-validation MSE might be a good parameter for selecting the best model.

That stated, in case 3, the cross-validation MSE boxplots categorised by the well location(s)

show the reverse pattern to the equivalent MAE boxplots. This indicates that the cross-validation method cannot counter poor training data. Thus, one must be selective of the data used for training, and make sure that the data displays the relationship patterns and contains minimal noise/outliers.

In machine learning or any data-based method, the model is only as good as the data that it is based on. In the synthetic cases, the amount of wells had little to no effect on the models' performance. However, in case 3, placing the well in a location with good data (across the fault) is far more important than in the other cases. Also, in case 1 the well should be placed inside the wedge to have access to data which show the data trend. This underlines the importance of good well placement. That said, in a real scenario it would be difficult to determine what is a good well-placement, so more wells imply higher chances of including well locations which contain the data distribution representative of the subsurface geometry.

KNN regression is the ML method that performed the worst for synthetic cases 2 and 3. The property that makes KNN different is it's ability to interpolate data points. This is likely the reason why this method performed well in case 1. Because case 1 is a homogeneous wedge, the trend is the same in all layers, and interpolating the data works well in this case.

The lasso-regression method performed very poorly, as did the neural network method in case 1. This might be because the spatial extension of the logs (impedance and porosity) causes the data distribution to be non-linear. This would explain the poor performance of lasso-regression (a linear regression extension). The neural network probably performed badly due to being shallow, meaning that the found relationships are too simplistic, and probably a deeper network is needed to obtain more realistic relationships.

SGS and ML methods:

The SGS approach requires the construction of a variogram model. This allows for professional input into the model, meaning that one has an analytical basis to say why one SGS model should be better than the others. The variogram model used in this thesis for the porosity prediction is

the same than the variogram model used for making the synthetic models. This means that the synthetic cases are close to the best possible results one can expect from an SGS approach. There is some uncertainty due to the wavelet extraction, as one can never derive the true wavelet due to the noise in the seismic.

For the ML methods, this same manual input (example: variogram) from an expert would likely be difficult but very possible to implement. Also, while the SGS method is bound to using only porosity and impedance as data, the ML methods can adapt to a wider variety of data input, including via predictor extraction. For example, adding the mean rolling window, or facies interpretations, etc. In my opinion, the ML methods are more flexible and have more potential for useful predictions.

Conclusion

The SGS method can provide a useful porosity model more quickly, if used by an experienced professional. However, machine learning can potentially provide more useful and accurate predictions due to its flexibility. Additionally, once the predictor extraction methods are implemented by an expert, machine learning models can be used by non-experts. An expert can also produce a large variety of results using different sets of predictors automatically and interpret which predictions make the most geological sense. Such large amount of different predictions was made throughout this thesis. It is clear that this is useful to determine the best ML methods. For example, in the idealised case 1, the KNN method functioned well, but in the other cases it did not. This means that KNN does not function well given a more geometrically complex data-set such as the F3 case.

Out of all the ML methods, random forest proved to be the most reliable method. However, this thesis has not even scratched the surface of neural networks, which can in theory fit any data provided the network is deep enough. The issue is user insight and time constraints rather than algorithm limitations.

To summarize:

The SGS method always requires professional expertise. ML methods have more potential from additional data and flexibility, and once implemented they do not require the same level of expertise for use. The RF method was the preferred method while the neural method has likely more potential. Finally, the workflow of using synthetic data to analyse the impact of methods such as the depo-time and RF, is beneficial for understanding prediction results.

6 Future Work

For future work it is recommended to use only one subsurface model, which can provide a more reliable dataset than for example case 3. There were to many prediction results to analyze given the time constraints. It would be useful to focus on more methods for analyzing such a large set of results on whether they make geological sense. Additionally adding more predictors and observing their effects should prove useful. There are also more ML methods that can be tested, including deep neural networks. While it was not discussed in this thesis, making geological models using forward process modeling could be considered, as these models can provide complex geological scenarios that make physical sense, rather than making a section based on simple geological/geometrical rules, as I did in this thesis.

References

[1] Schlumberger, "Petrel 2014 quantitative interpretation module 4: Simultaneous inversion," 2015.

[2] I. A. S. de Macedo, C. B. da Silva, J. J. S. de Figueiredo, and B. Omoboya, "Comparison between deterministic and statistical wavelet estimation methods through predictive deconvolution; seismic to well tie example from the north sea," *Journal of applied geophysics*, vol. 136, pp. 298–314, 2017.

[3] Schlumberger, "wavelet extraction," accessed: 01/05/2022. [Online]. Available: https://glossary.oilfield.slb.com/en/terms/w/wavelet_extraction

[4] ——, "Petrel geophysical interpretation, 18th edition," 2019.

[5] ——, "Petrel property modeling," 2016.

[6] J. Caers, "Modeling uncertainty in the earth sciences," Chichester, 2011.

[7] Schlumberger, "Petrel property modeling," 2017.

[8] O. Dubrule, "Indicator variogram models; do we have much choice?" *Mathematical geosciences*, vol. 49, no. 4, pp. 441–465, 2017.

[9] TerraNubis, "Project f3 demo 2020," accessed: 01/03/2022. [Online]. Available: https://terranubis.com/datainfo/F3-Demo-2020

[10] M. Ishak, M. A. Islam, M. Shalaby, and N. Hasan, "The application of seismic attributes and wheeler transformations for the geomorphological interpretation of stratigraphic surfaces: A case study of the f3 block, dutch offshore sector, north sea," *Geosciences*, vol. 8, p. 79, 02 2018.

[11] G. Remmelts, "Fault-related salt tectonics in the southern north sea, the netherlands," *Salt tectonics : a global perspective : based on the Hedberg International Research Conference, Bath, U.K., September 1993*, vol. 65, pp. 261–272, 1995.

[12] R. M. Mitchum, "Seismic stratigraphy and global changes of sea level; part 11, glossary of terms used in seismic stratigraphy," *Memoir - American Association of Petroleum Geologists*, no. 26, pp. 205–212, 1977.

[13] J. L. Rich, "Three critical environments of deposition, and criteria for recognition of rocks deposited in each of them," *Bulletin of the Geological Society of America*, vol. 62, no. 1, pp. 1–19, 1951.

[14] J. Walsh and J. Watterson, "Distributions of cumulative displacement and seismic slip on a single normal fault surface," *Journal of Structural Geology*, vol. 9, no. 8, pp. 1039–1046, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0191814187900125

[15] P. Raybaut, "Spyder-documentation," *Available online at: pythonhosted. org*, 2009.

[16] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[18] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[19] Equinor, J. Kvalsvik, and other Github contributors, "segyio," Apr. original-date: 27/09/2016, accessed: 01/03/2022, Copyright (C) 2007 Free Software Foundation, Inc. http://fsf.org/. [Online]. Available: https://github.com/equinor/segyio

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org, Accessed: 01/02/2022. [Online]. Available: https://www.tensorflow.org/

[23] G. James, D. Witten, T. Hastie, and R. Tibshirani, "An introduction to statistical learning : with applications in r," New York, 2017.

[24] S. Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*. San Diego: Elsevier Science Technology, 2020.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," accessed: 01/04/2022. [Online]. Available: https://arxiv.org/abs/1412.6980

[26] scikit-learn developers, "3.1. cross-validation: Evaluating estimator performance," accessed: 01/03/2022. [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html

Appendices

A Manual

A.1 Python environment

Tool:	Version:
Windows	21H2 (OS Build 19044.1645)
Python	3.7.11
Spyder	5.1.5
Jupyter Lab	3.2.1
numpy	1.20.3
matplotlib	3.5.0
segyio	1.9.7
scikit-learn	1.0.1
pandas	1.3.4
Tensorflow	2.3.0

Table 8: The environment used in this thesis, and their version.

A.2 Summary of Python classes and scripts

A brief explanation of the code developed for this thesis is included. In general, the code can be separated into two types based on its functionality. Class scripts which contain the Python classes used in the thesis; these are accessed by other scripts to perform actions. In Python, a class is a broad category that contains attributes and methods. Classes can be instantiated to create objects, and methods can be sent to objects to perform actions. The second type of scripts are those that perform specific tasks.

class: seismic_ext.py

This class uses the segyio module to extract the raw traces and metadata from a .segy file. This returns the seismic section.

class: load_well.py

This class loads well-logs. The functionality depends on if the model is synthetic or the actual F3 dataset. If the synthetic model is used, extraction of the trace(s) as well-log(s) is performed. For the F3 data-set, the well-logs are extracted from csv files.

class: predictor_ext.py

This class uses the extracted well-logs as basis for predictor extraction. It includes the window functions discussed in section 3.4.2. It also contains the depo-time (geological time) construction and extraction based on either the geometry of the wedge or the interpreted horizons. The standardization is also implemented here.

class: Mlearning.py

This class initialises, trains, tunes and applies machine learning algorithms. The tuning is performed by cross-validation. Based on cross-validation, the best set of parameters within a predetermined search range is selected. The prediction of the porosity section is performed for every trace. This is mostly to simplify the implementation of the window functions.

class: plot_results.py

This class has the functions for generating reproducible plots to monitor the resulting predictions during runs and catch bugs early. It also saves these figures and the predictions. The saved predictions are important as one can later visualize and plot them in a desirable way.

class: useful_functions.py

This class contains some functions that don't fit into the categories of the other classes but are necessary. For example, it has a function that maps two input arrays to an output array without any for loops (this is important for efficiency).

class: manage_cases.py

This class functions as "glue" code, stitching the other classes together. It also automatically retrieves the required files based on the case name.

Script: section_horizon_coord.py

This script uses a segy file and horizon point-sets to find and save the intersections between the section and the seismic horizons.

Script: make_new_wells.py

This script takes the well-logs extracted from Petrel and formats them to be more easily usable in the class load_well.

Script: upscale.py

This script uses the existing well-logs and an upscaled well-log as input. The upscaled well log is made in Petrel. The script matches the MD (measured depth) in the upscaled well-log with the closet MD in the real well-logs. This upscales the well-logs to match the vertical sampling frequency of the seismic traces.

Script: well_paths_horizon.py

This script uses the well paths and seismic horizons as input. It locates where the horizons intersect the well paths and saves this information. This information is later used to make the depo-time in the well-logs.

Script: top_cases_script.py

This script uses primarily the manage_cases class to perform the predictions and comparisons automatically for several combinations of parameters. This is used to perform all predictions of the synthetic models in one run without supervision.

Script: F3_script.py

This script has the same functionality as the top_cases script, but for the F3 case instead of the synthetic models. This has to be a separate script because the top_cases script needs a porosity section, which the F3 block doesn't have one.

A.3 Folder composition

The organization of the code, data, and results of this thesis in directories is as follows:

```
Main (should be the directory)
classes
    load_well.py
    Mlearning.py
    plot_results.py
    predictor_ext.py
    seismic_ext.py
    manage_cases_class.py
    usefull_functons.py
scripts
    top_cases_scripts.py
    F3_script.py
    make_new_wells.py
    section_horizon_coord.py
    well_paths_horizon_coord.py
    upscale.py
data
    case 1a Wedge
        Seis_Inv_Wedge_IIIc.segy
        Por_Wedge_IIIc.segy
    case 1b Wedge
        Seis_Noise_Inv_Wedge_IIIc.segy
        Por_Wedge_IIIc.segy
    case 2a wedge hetero
        case 2a wedge imp no noise.segy
        case 2a wedge porosity.segy
        case 2a wedge porosity estimation no noise.segy
    case 2b wedge hetero
        case 2b wedge imp noise.segy
        case 2b wedge porosity.segy
        case 2b wedge porosity estimation noise.segy
    case 3a fault
        case 3a fault imp no noise.segy
        case 3a fault porosity.segy
        case 3a fault porosity no noise.segy
    case 3b fault
        case 3b fault imp noise.segy
        case 3b fault porosity.segy
        case 3b fault porosity estimation noise.segy
    case F3
        Seis Inv depth Random line [2D Converted].segy
        F3 porosity estimation.segy

        AI resampled.xlsx [Upscaled impedance log]
        F02_1 well path.txt
        F03_2 well path.txt
        F02_1.xlsx [well-logs]
        F03_2.xlsx [well-logs]
        F3-Horizon-FS8 (Z).txt
        F3-Horizon-MFS4 (Z).txt
        F3-Horizon-Truncation (Z).txt


    horizons

        (Horizons point sets as .txt files, and locations where horizons intersect with cross-sections)

results

    case 1a Wedge

        figures (plots of the results, and save .npy files of the result arrays)

        info (dataframes of the results saved as csv files)

    case 1b Wedge

        figures
```

info
case 2a wedge hetero
figures
info
case 2b wedge hetero
figures
info
case 3a fault
figures
info
case 3b fault
figures
info
case F3
figures
info
dataframes

B Running the scripts for the results

The data and code is provided on my personal google drive and github. This folder can be accessed atdrive.google.com or github.com/ESalomonsen. The google drive contains a zip file with the main directory. The structure is the same as the general structure described above. Two text files, "requirements.txt" and "ReadMe.txt" provide information on the environment specifications and general information about the code/data, respectively. The environment is also decribed in table 8.

For reproducing the results of this thesis, just go to the "scripts" folder. Then run the "top_cases_script.py" and "F3_script.py" scripts. If only interested in reproducing the results one can ignore the rest if this appendix.

These scripts loop over lists of parameters for example, the combinations of well-locations. So, to run a specific parameter combination simply enter the parameter values into the appropriate list. For example, if one wants to test window size equal to 1 for the synthetic models, change line 16 in "top_cases_script.py" from window_list = [0, 10] to window_list = [1]. The scripts use the "manage_cases_class.py" class as a compilation/glue code of the other classes. An example for producing one prediction result is shown below.

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 13 17:13:05 2022

Example of how to make one predction

@author: Eier
"""
from classes.manage_cases_class import manage_cases_class
import numpy as np

case = 'case 2a wedge hetero'
window = 5
geo_int = True
method = 'Random Forest'
wells_loc = [200, 500]

# start class which sets the basic parameters
case_class = manage_cases_class(case = case, wells_loc = wells_loc, window = window,geo_int = geo_int)
# get the cross-sections
case_class.load_sections()
# load the well-logs and perform predictor extraction
case_class.load_synthetic_wells_and_predictors_ext()
# train the ML model
case_class.ML_with_cross_val(method, max_depth = np.arange(4,9, 1),n_estimators = np.arange(25, 110, 10))
# apply the model
case_class.predict()
# get the prediction:
prediction = case_class.pred_map
```

Many of the figures of the thesis were made using the script "making results.ipynb". This script compiles and plots the results for investigation using pandas.

The data used can be described as following: The sections of porosity, impedance, ect. are extracted from Petrel as 2D .segy files. The seismic horizon data have the form of a point set in 3D, directly extracted from Petrel. The well-paths are also directly extracted from Petrel. The well-logs are copied into an excel file. An example of these data file are shown in the figure below.

Well-logs

	A	B	C	D	E	F
1		MD	Por.Eff.		MD	P-imp.
2		48.000			30.000	
3		48.150			30.500	
4		48.300			31.000	
5		48.450			31.500	
6		48.600			32.000	
7		48.750			32.500	
8		48.900			33.000	
9		49.050			33.500	
10		49.200			34.000	
11		49.350			34.500	
12		49.500			35.000	
13		49.650			35.500	
14		49.800			36.000	
15		49.950			36.500	
16		50.100			37.000	
17		50.250			37.500	
18		50.400			38.000	

Horizon pointset

```
605835.473498 6073556.399259 816.055054
605860.464073 6073557.097688 815.907349
605885.454648 6073557.796116 815.747314
605910.445223 6073558.494544 815.593384
605935.435798 6073559.192973 815.445374
605960.426373 6073559.891401 815.279968
605985.416948 6073560.589829 815.045044
606010.407523 6073561.288257 814.689392
606035.398098 6073561.986686 814.189941
606060.388673 6073562.685114 813.561157
606085.379247 6073563.383542 812.853027
606110.369822 6073564.081971 812.127686
606135.360397 6073564.780399 811.437378
606160.350972 6073565.478827 810.807800
606185.341547 6073566.177255 810.229980
606210.332122 6073566.875684 809.668945
606235.322697 6073567.574112 809.085266
606260.313272 6073568.272540 808.463806
606285.303847 6073568.970969 807.818237
606310.294422 6073569.669397 807.178711
606335.284996 6073570.367825 806.571594
606360.275571 6073571.066254 806.002808
606385.266146 6073571.764682 805.453552
606410.256721 6073572.463110 804.894348
```

Well-path

```
# WELL TRACE FROM PETREL
# WELL NAME:                F03-2
# DEFINITIVE SURVEY:        X Y TVD survey
# WELL HEAD X-COORDINATE: 619101.00000000 (m)
# WELL HEAD Y-COORDINATE: 6089491.00000000 (m)
# WELL DATUM (KB, Kelly bushing, from MSL): 30.00000000 (m)
# WELL TYPE:                UNDEFINED
# MD AND TVD ARE REFERENCED (=0) AT WELL DATUM AND INCREASE DOWNWARDS
# ANGLES ARE GIVEN IN DEGREES
# XYZ TRACE IS GIVEN IN COORDINATE SYSTEM PowerPlan:TM-NL (MENTOR:PowerPlan:TM-NL:TM CM 5E on Dutch ED50) [SIS,501820]
# AZIM_TN: azimuth in True North
# AZIM_GN: azimuth in Grid North
# DX DY ARE GIVEN IN GRID NORTH IN m-UNITS
# DEPTH (Z, tvd_z) GIVEN IN m-UNITS
# ANGLES ARE NOT EXACT (TRACE WAS NOT IMPORTED USING ANGLES)
# MD IS NOT EXACT (TRACE WAS NOT IMPORTED WITH MD-DATA)
#=============================================================================================================================================
 MD X Y Z TVD DX DY AZIM_TN INCL DLS AZIM_GN
#=============================================================================================================================================
 0.0000000000 619101.00000 6089491.0000 30.000000000 0.0000000000 -0.000000000 0.0000000000 1.5217867566 0.0000000000 0.0000000000 0.0000000000
 2140.0000000 619101.00000 6089491.0000 -2110.000000 2140.0000000 -0.000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 358.47821324
```

Figure 58: Logs of one well compiled into an excel file to the top left. The P. imp. and Por.Eff. columns has numbers further down in the file, but it is more important to show the headers. To the top right is an example of the point set for one seismic horizon, the coordinates are X, Y and depth. The lower text file is an example of the well-path of one well.

Information on the scripts and classes can be seen in section A.2. As only some of this information is repeated here. Before the code can be run, the data must be altered using some of the scripts. That said, the drive contains the altered and original data, making this process unnecessary. The section_horizon_coord.py script is ran using the 2D cross-section and horizon point sets and produces a .npy file for each horizon named after the original horizon file +TWT. After this, one should run make_new_wells.py to make a .csv file from the excel well-logs file. This file is named the same as the excel file + new. I used the upscale.py file to upscale the well-logs based on a previously made upscaled log, however it does not matter how the upscaling is done. The last script for editing the

data is the well_paths_horizon.py file. It uses the horizon files and the well path to make a .csv file with the horizon locations in the well-logs, called the same as the well-path name + horizon loc.

If one wants to implement their own cases, this is done in the manage_cases.py file, in the __init__ function. In the function, the necessary data files can be added to an elif block similarly to the other cases shown in the other elif blocks (line 92 to 213).

I made the top cases script and the F3 script to automatically produce results for all parameter values and compile the results. This is not strictly necessary, but should be used as a reference for how to use the manage_cases.py to produce results.

C Code

The code below is provided in case the online code becomes inaccessible:

C.1 top_cases_script.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Fri Mar 11 19:19:29 2022
4
5  @author: Eier
6  """
7  from classes.manage_cases_class import manage_cases_class
8  import numpy as np
9  import pandas as pd
10
11
12 # define parameters
13
14 # 'case 1a Wedge', 'case 1b Wedge', 'case 2a wedge hetero', 'case 2b wedge hetero', 'case 3a fault', 'case
       3b fault'
15 case_list =['case 1a Wedge', 'case 1b Wedge', 'case 2a wedge hetero', 'case 2b wedge hetero', 'case 3a
      fault', 'case 3b fault']
16 window_list = [0, 10]
17 geo_int_list = [False, True]
18 # 'lasso', 'KNN','Random Forest', 'Neural Net'
19 method_list = ['lasso', 'KNN','Random Forest', 'Neural Net']
20
21
22 for case in case_list:
23
24     # prepare to store data
25     MAE_list = []
26     MSE_list = []
27     Rs_list = []
```

```
28      CV_MSE_list = []
29      AE_std_list = []
30
31      case_list_new = []
32      wells_list_new = []
33      window_list_new = []
34      geo_int_list_new = []
35      method_list_new = []
36
37      # define wells depending on the model
38      if case == 'case 1a Wedge' or case == 'case 1b Wedge':
39          wells_loc_list = [[100, 200], [100], [290]]
40          # crop the sections?
41          vcut= [120, 400]
42          hcut = False
43
44      else:
45          # [200, 700], [200], [800], [400]
46          wells_loc_list = [[400], [200, 350, 500, 700], [700, 750, 650, 775], [750]]
47          # crop the sections?
48          vcut= False
49          hcut = False
50
51      # for every combination of the parameters
52      for wells_loc in wells_loc_list:
53          for window in window_list:
54              for geo_int in geo_int_list:
55                  for method in method_list:
56                      # define seed
57                      np.random.seed(seed = 1)
58
59                      # initiallise
60                      case_class = manage_cases_class(case = case, wells_loc = wells_loc, window = window,
                                geo_int = geo_int)
61
62                      # get cross-sections
63                      case_class.load_sections(plot = False)
64
65                      # get wells and perform predictor extraction
66                      case_class.load_synthetic_wells_and_predictors_ext()
67
68                      # train machine learning models with cross validation
69                      if case == 'case 2a wedge hetero' or case == 'case 2b wedge hetero': # less data due
                                to resolusion means KNN needs a k that will not exceed the number of datapoints
70                          cv_mse = case_class.ML_with_cross_val(method, layers_list = [1],N = np.arange(1,
                                    50, 1), cv = 10, plot = True,
71                                                  neurons = np.arange(1, 40, 1),
72                                                  max_depth = np.arange(4,9, 1),n_estimators = np.arange(25,
                                                        110, 10)
73                                                  , al_array = np.arange(0, 0.005, 0.00005))
74                      else:
75                          cv_mse = case_class.ML_with_cross_val(method, layers_list = [1],N = np.arange(1,
                                    100, 1), cv = 10, plot = True,
76                                                  neurons = np.arange(1, 40, 1),
77                                                  max_depth = np.arange(4,9, 1),n_estimators = np.arange(25,
                                                        110, 10)
78                                                  , al_array = np.arange(0, 0.005, 0.00005))
```

```
79
80                          # predict porosity cross-section
81                          case_class.predict()
82
83                          # compare to True porosity
84                          MAE, MSE, Rs, AE_std = case_class.result_eval(vcut= vcut,hcut = hcut)
85
86                          # store data
87                          MAE_list.append(MAE)
88                          MSE_list.append(MSE)
89                          Rs_list.append(Rs)
90                          CV_MSE_list.append(cv_mse)
91                          AE_std_list.append(AE_std)
92
93                          case_list_new.append(case)
94                          wells_list_new.append(wells_loc)
95                          window_list_new.append(window)
96                          geo_int_list_new.append(geo_int)
97                          method_list_new.append(method)
98                          if case == 'case 1a Wedge' or case == 'case 1b Wedge':
99                              pass
100                         # compare SGS solution to true porosity, and compile reults
101                         else:
102
103                             MAE, MSE, r_squared, AE_std = case_class.petrel_solution_eval()
104
105                             df2 = pd.DataFrame()
106                             df2['case'] = [case]
107                             df2['MAE'] = [MAE]
108                             df2['MSE'] = [MSE]
109                             df2['R-squared'] = [r_squared]
110                             df2['abs error std'] = [AE_std]
111
112                             df2.to_csv('results\dataframes\{} classic solution'.format(case)+'.txt',sep='/',
                                    index = False, encoding = 'utf-8')
113
114         # compile results of ML
115         df = pd.DataFrame()
116
117         df['case'] = case_list_new
118         df['ML method'] = method_list_new
119         df['wells location'] = wells_list_new
120         df['window size'] = window_list_new
121         df['depo time implementation'] = geo_int_list_new
122         df['cross validation MSE'] = CV_MSE_list
123         df['MAE'] = MAE_list
124         df['MSE'] = MSE_list
125         df['abs error std'] =AE_std_list
126         df['R-squared'] = Rs_list
127
128         df.to_csv('results\dataframes\case {} auto results'.format(case)+'.txt',sep='/', index = False,
              encoding = 'utf-8')
```

C.2 F3_script.py

```
1  # -*- coding: utf-8 -*-
2  """
```

```
3   Created on Mon Apr  4 13:30:55 2022
4
5   @author: Eier
6   """
7   from classes.manage_cases_class import manage_cases_class
8   import numpy as np
9   import pandas as pd
10  import matplotlib.pyplot as plt
11
12  # np.random.seed(seed = 1)
13
14  case_type = 'real'
15
16  case_list =['case F3']
17
18  well_paths_horizons = ['data\case F3\F02_1 well path horizon loc.txt', 'data\case F3\F03_2 well path
        horizon loc.txt']
19
20  window_list = [0, 10]
21
22  well_files = ['data\case F3\F02_1_new.txt', 'data\case F3\F03_2_new.txt']
23
24  geo_int_list = [False, True]
25  # 'lasso', 'KNN','Random Forest', 'Neural Net'
26  method_list = ['lasso', 'KNN','Random Forest', 'Neural Net']
27
28  for case in case_list:
29      MAE_list = []
30      MSE_list = []
31      Rs_list = []
32      CV_MSE_list = []
33
34      case_list_new = []
35      wells_list_new = []
36      window_list_new = []
37      geo_int_list_new = []
38      method_list_new = []
39
40      for window in window_list:
41          if window == 0:
42              well_files = ['data\case F3\F02_1_new.txt', 'data\case F3\F03_2_new.txt']
43          else:
44              well_files = ['data\case F3\F02_1_new_upsacale.txt', 'data\case F3\F03_2_new_upsacale.txt']
45          for geo_int in geo_int_list:
46              for method in method_list:
47
48                  np.random.seed(seed = 1)
49
50                  case_class = manage_cases_class(case = case, wells_loc = None, window = window,geo_int =
                        geo_int)
51
52                  case_class.load_sections(plot = False, scaling = False)
53
54                  case_class.load_wells_and_predictors_ext(well_files = well_files, well_paths_horizons =
                        well_paths_horizons)
55
56
```

```
57                  cv_mse = case_class.ML_with_cross_val(method, layers_list = [1],N = np.arange(1, 100, 1),
                        cv = 10, plot = True,
58                                      neurons = np.arange(1, 40, 1),
59                                      max_depth = np.arange(1,20, 1),n_estimators = np.arange(1, 41, 10),
60                                      al_array = np.arange(0, 0.005, 0.00005))
61
62                  case_class.predict(case_type = case_type)
63                  case_class.result_eval_F3()
64
65                  # case_class.predict(case_type = case_type)
66                  # plt.imshow(case_class.pred_map.T)
67                  # plt.show()
68
69                  # MAE_list.append(MAE)
70                  # MSE_list.append(MSE)
71                  # Rs_list.append(Rs)
72                  CV_MSE_list.append(cv_mse)
73
74                  case_list_new.append(case)
75                  # wells_list_new.append(wells_loc)
76                  window_list_new.append(window)
77                  geo_int_list_new.append(geo_int)
78                  method_list_new.append(method)
79
80                  # df2 = pd.DataFrame()
81                  # df2['case'] = case
82                  # df2['MAE'] = MAE
83                  # df2['MSE'] = MSE
84                  # df2['R-squared'] = r_squared
85
86                  # df2.to_csv('results\dataframes\{} classic solution'.format(case)+'.txt',sep='/', index =
                        False, encoding = 'utf-8')
87
88      df = pd.DataFrame()
89
90      df['case'] = case_list_new
91      df['ML method'] = method_list_new
92      # df['wells location'] = wells_list_new
93      df['window size'] = window_list_new
94      df['depo time implementation'] = geo_int_list_new
95      df['cross validation MSE'] = CV_MSE_list
96      # df['MAE'] = MAE_list
97      # df['MSE'] = MSE_list
98      # df['R-squared'] = Rs_list
99
100     df.to_csv('results\dataframes\case {} auto results'.format(case)+'.txt',sep='/', index = False,
            encoding = 'utf-8')
```

C.3 manage_cases_class.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Fri Mar 11 17:43:48 2022
4
5   @author: Eier
6   """
7   import numpy as np
```

```
8  import segyio
9  import pandas as pd
10 import matplotlib.pyplot as plt
11 import tensorflow as tf
12 import pickle
13 from sklearn.metrics import r2_score
14
15 from classes.load_well import load_well
16 from classes.predictor_ext import predictor_ext
17 from classes.Mlearning import Mlearning
18 from classes.plot_results import plot_results
19 from classes.usefull_functions import usefull_functions
20 from classes.seismic_ext import seismic_ext
21
22
23 class manage_cases_class:
24     """
25     Class for connecting the other classes and data.
26     This makes it easier to run many different parameter combinations automatically (iteratively).
27     """
28
29     def __init__(self, case, wells_loc, window, geo_int=False,  # first line is all that is used for the
           thesis
30                  file_ai=None, file_seis=None, file_por=None, max_TWT=None, min_TWT=None,
31                  w_start=None, w_end_top=None, w_end_base=None, horizons_list=None, petrel_solution=None):
32         """
33         sets the parameters of the case from a preset or defines them here
34
35         Parameters
36         ----------
37         case : str
38             the case identification as to get all the preset parameters.
39         wells_loc : list
40             the trace locations of the wells.
41         window : TYPE
42             window size to be used in predictor extraction.
43         geo_int : str, optional
44             Should the depositional time be implemented and if so how?. The default is False.
45         file_ai : str, optional
46             path to segy file with impedance. The default is None.
47         file_seis : str, optional
48             path to segy file with seismic. The default is None.
49         file_por : str, optional
50             path to segy file with porosity. The default is None.
51         max_TWT : int, optional
52             maximum value of the TWT. The default is None.
53         min_TWT : int, optional
54             minimum value of the TWT. The default is None.
55          w_start : list, optional
56              the starting position of the wedge: the pinch point. The default is None.
57         w_end_top : list, optional
58            the end of the top surface of the wedge. The default is None.
59         w_end_base : list, optional
60             the end of the base surface of the wedge. The default is None.
61         horizons_list : list or None, optional
62             list of numpy arrays that describe where the horizons intersect with the cross-section. The
                   default is None.
```

```
63          petrel_solution : str, optional
64              file path to the SGS solution of porosity. The default is None.
65
66          Returns
67          -------
68          None.
69
70          """
71
72          self.case = case
73          self.win = window
74          self.wells_loc = wells_loc
75          self.geo_int = geo_int
76
77          ###################################
78          # for new case:
79          self.file_ai = file_ai
80          self.file_seis = file_seis
81          self.file_por = file_por
82          self.max_TWT = max_TWT
83          self.min_TWT = min_TWT
84          self.w_start = w_start
85          self.w_end_top = w_end_top
86          self.w_end_base = w_end_base
87          self.horizons_list = horizons_list
88          self.petrel_solution = petrel_solution
89
90          ###################################
91          # select the relevant files based on the case
92          if case == 'case 1a Wedge':
93              self.file_ai = 'data\case 1a Wedge\Seis_Inv_Wedge_IIIc.segy'
94              #self.file_seis = 'data\case 1a Wedge\Synth_Wedge_IIIc.segy'
95              self.file_por = 'data\case 1a Wedge\Por_Wedge_IIIc.segy'
96              # self.max_TWT = -618 # max TWT of the seismic section
97              # self.min_TWT = -1162 # min TWT of the seismic section
98              # wedge start, the relitive location where the wedge has just thinned out
99              self.w_start = [267, 178]
100             # the relitive location where the top of the wedge ends
101             self.w_end_top = [0, 178]
102             # the relitive location where the base of the wedge ends
103             self.w_end_base = [0, 400]
104             self.horizons_list = False
105
106             if self.geo_int == True:
107                 self.geo_int = 'wedge'
108
109         elif case == 'case 1b Wedge':
110             self.file_ai = 'data\case 1b Wedge\Seis_Noise_Inv_Wedge_IIIc.segy'
111             #self.file_seis = 'data\case 1b Wedge\Synth_Noise_Wedge_IIIc.segy'
112             self.file_por = 'data\case 1b Wedge\Por_Wedge_IIIc.segy'
113             # self.max_TWT = -618 # max TWT of the seismic section
114             # self.min_TWT = -1162 # min TWT of the seismic section
115             # wedge start, the relitive location where the wedge has just thinned out
116             self.w_start = [267, 178]
117             # the relitive location where the top of the wedge ends
118             self.w_end_top = [0, 178]
119             # the relitive location where the base of the wedge ends
```

```
120             self.w_end_base = [0, 400]
121             self.horizons_list = False
122 
123             if self.geo_int == True:
124                 self.geo_int = 'wedge'
125 
126         elif self.case == 'case 2a wedge hetero':
127             self.file_ai = 'data\case 2a wedge hetero\case 2a wedge imp no noise.segy'
128             #self.file_seis = 'data\case 2a wedge hetero\case 2 wedge seis.segy'
129             self.file_por = 'data\case 2a wedge hetero\case 2a wedge porosity.segy'
130             self.petrel_solution = 'data\case 2a wedge hetero\case 2a wedge porosity estimation no noise.
                    segy'
131 
132             hor_file4 = 'data\case 2a wedge hetero\horizons\wedge surface base plus'
133             hor_file3 = 'data\case 2a wedge hetero\horizons\wedge surface base'
134             hor_file2 = 'data\case 2a wedge hetero\horizons\wedge surface top'
135             hor_file1 = 'data\case 2a wedge hetero\horizons\wedge surface top plus'
136 
137             self.horizons_list = [np.load(hor_file1+'TWT'+'.npy'), np.load(
138                 hor_file2+'TWT'+'.npy'), np.load(hor_file3+'TWT'+'.npy'), np.load(hor_file4+'TWT'+'.npy')]
139             # self.max_TWT = -618
140             # self.min_TWT = -1162
141 
142             if self.geo_int == True:
143                 self.geo_int = 'from horizons'
144 
145         elif self.case == 'case 2b wedge hetero':
146             self.file_ai = 'data\case 2b wedge hetero\case 2b wedge imp noise.segy'
147             #self.file_seis = 'data\case 2b wedge hetero\case 2 wedge seis.segy'
148             self.file_por = 'data\case 2b wedge hetero\case 2b wedge porosity.segy'
149             self.petrel_solution = 'data\case 2b wedge hetero\case 2b wedge porosity estimation noise.segy
                    '
150 
151             hor_file4 = 'data\case 2b wedge hetero\horizons\wedge surface base plus'
152             hor_file3 = 'data\case 2b wedge hetero\horizons\wedge surface base'
153             hor_file2 = 'data\case 2b wedge hetero\horizons\wedge surface top'
154             hor_file1 = 'data\case 2b wedge hetero\horizons\wedge surface top plus'
155 
156             self.horizons_list = [np.load(hor_file1+'TWT'+'.npy'), np.load(
157                 hor_file2+'TWT'+'.npy'), np.load(hor_file3+'TWT'+'.npy'), np.load(hor_file4+'TWT'+'.npy')]
158             # self.max_TWT = -618
159             # self.min_TWT = -1162
160 
161             if self.geo_int == True:
162                 self.geo_int = 'from horizons'
163 
164         elif self.case == 'case 3a fault':
165             self.file_ai = 'data\case 3a fault\case 3a fault imp no noise.segy'
166             #self.file_seis = 'data\case 3a fault\case 3 fault seis.segy'
167             self.file_por = 'data\case 3a fault\case 3a fault porosity.segy'
168             self.petrel_solution = 'data\case 3a fault\case 3a fault porosity no noise.segy'
169 
170             hor_file1 = 'data\case 3a fault\horizons\cfault surface 1'
171             hor_file2 = 'data\case 3a fault\horizons\cfault surface 4'
172             hor_file3 = 'data\case 3a fault\horizons\cfault surface 3'
173             hor_file4 = 'data\case 3a fault\horizons\cfault surface 5'
174             hor_file5 = 'data\case 3a fault\horizons\cfault surface 2'
```

```
175             self.horizons_list = [np.load(hor_file1+'TWT'+'.npy'), np.load(hor_file2+'TWT'+'.npy'), np.
                    load(
176                 hor_file3+'TWT'+'.npy'), np.load(hor_file4+'TWT'+'.npy'), np.load(hor_file5+'TWT'+'.npy')]
177             # self.max_TWT = -618
178             # self.min_TWT = -1162
179
180             if self.geo_int == True:
181                 self.geo_int = 'from horizons'
182
183         elif self.case == 'case 3b fault':
184             self.file_ai = 'data\case 3b fault\case 3b fault imp noise.segy'
185             #self.file_seis = 'data\case 3b fault\case 3 fault seis.segy'
186             self.file_por = 'data\case 3b fault\case 3b fault porosity.segy'
187             self.petrel_solution = 'data\case 3b fault\case 3b fault porosity estimation noise.segy'
188             hor_file1 = 'data\case 3b fault\horizons\cfault surface 1'
189             hor_file2 = 'data\case 3b fault\horizons\cfault surface 4'
190             hor_file3 = 'data\case 3b fault\horizons\cfault surface 3'
191             hor_file4 = 'data\case 3b fault\horizons\cfault surface 5'
192             hor_file5 = 'data\case 3b fault\horizons\cfault surface 2'
193             self.horizons_list = [np.load(hor_file1+'TWT'+'.npy'), np.load(hor_file2+'TWT'+'.npy'), np.
                    load(
194                 hor_file3+'TWT'+'.npy'), np.load(hor_file4+'TWT'+'.npy'), np.load(hor_file5+'TWT'+'.npy')]
195             # self.max_TWT = -618
196             # self.min_TWT = -1162
197
198             if self.geo_int == True:
199                 self.geo_int = 'from horizons'
200
201         elif self.case == 'case F3':
202             self.file_ai = 'data\case F3\Seis Inv depth Random line [2D Converted].segy'
203             self.petrel_solution = 'data\case F3\F3 porosity estimation.segy'
204
205             hor_file1 = 'data\case F3\F3-Horizon-FS8 (Z)'
206             hor_file2 = 'data\case F3\F3-Horizon-Truncation (Z)'
207             hor_file3 = 'data\case F3\F3-Horizon-MFS4 (Z)'
208
209             self.horizons_list = [np.load(hor_file1+'TWT'+'.npy'), np.load(
210                 hor_file2+'TWT'+'.npy'), np.load(hor_file3+'TWT'+'.npy')]
211
212             if self.geo_int == True:
213                 self.geo_int = 'from horizons'
214
215         else:
216             pass
217
218     def handle_CV(self, name, CV):
219         """
220         handles cross-validation result
221
222         Parameters
223         ----------
224         name : str
225             file path to save the CV result.
226         CV : dict
227             dictionary of cross-validation results.
228
229         Returns
```

```
230             --------
231             best : str
232                 string of the best parameters.
233             best_ : list
234                 list of the best parameters.
235
236             """
237             uf = usefull_functions()
238             best_, self.neg_mse = uf.save_cv(name + ".pkl", self.ML_method, CV)
239             best = best_.split('/')
240             file = open(name + ".pkl", "rb")
241             CV_KNNoutput = pickle.load(file)
242
243             return best, best_
244
245         def load_sections(self, plot=True, scaling=False):
246             """
247             load the cross-sections
248
249             Parameters
250             ----------
251             plot : str, optional
252                 if true plot the sections. The default is True.
253             scaling : str, optional
254                 should the sections be standardized (not working currently). The default is False.
255
256             Returns
257             --------
258             None.
259
260             """
261
262             # get impedance
263             ext = seismic_ext(self.file_ai)
264             self.grid2, self.extent2 = ext.syn_seismic(plot=plot, scaling=scaling)
265
266             # get porosity
267             if self.file_por == None:
268                 pass
269             else:
270                 ext = seismic_ext(self.file_por)
271                 self.grid3, self.extent3 = ext.syn_seismic(
272                     plot=plot, scaling=False)
273
274             # get petrel solution
275
276             if self.petrel_solution == None:
277                 pass
278             else:
279                 ext = seismic_ext(self.petrel_solution)
280                 self.grid_pet, self.extent_pet = ext.syn_seismic(
281                     plot=plot, scaling=False)
282
283                 if self.case == 'case F3':
284
285                     self.grid_pet = np.flip(self.grid_pet, 0)
286
```

```
287             # get max and min depth
288             self.max_TWT = self.extent2[3]
289             self.min_TWT = self.extent2[2]
290 
291             # slice the sections based on the case
292             if self.case == 'case 3a fault' or self.case == 'case 3b fault':
293                 self.grid2 = self.grid2[:, 0:135]
294             elif self.case == 'case 2a wedge hetero' or self.case == 'case 2b wedge hetero':
295                 # self.grid1 = self.grid1[:, 0:75]
296                 self.grid2 = self.grid2[:, 0:75]
297                 self.grid3 = self.grid3[:, 0:75]
298 
299         def load_wells_and_predictors_ext(self, well_files, win_names=['imp'], well_paths_horizons=None):
300             """
301             load the well-logs as predictors and target, then perform predictor extraction
302             on the real data-set
303 
304             Parameters
305             ----------
306             well_files : TYPE
307                 DESCRIPTION.
308             win_names : TYPE, optional
309                 DESCRIPTION. The default is ['imp'].
310             well_paths_horizons : TYPE, optional
311                 DESCRIPTION. The default is None.
312 
313             Returns
314             -------
315             None.
316 
317             """
318 
319             if well_paths_horizons == None:
320                 well_paths_horizons = np.linspace(0, 1, len(well_files))
321 
322             grid_list = [self.grid2]
323             self.grid_list = grid_list
324             grid_names = ['imp']
325             self.grid_names = grid_names
326 
327             well = well_files[0]
328             log1 = load_well(file_name=well)
329 
330             data, MD = log1.from_csv()
331 
332             Por = data['por'].to_numpy()
333             Por_df = pd.DataFrame(Por, columns=['Por'])
334 
335             data.drop('por', axis=1, inplace=True)
336 
337             new_pred = predictor_ext(data)
338 
339             for name in win_names:
340                 new_pred.roll_and_win_sel_well(data_name=name, win=self.win,
341                                                geo_int=self.geo_int, grid=self.grid2, horizons_list=self.
                                                    horizons_list, MD=MD, well_path=well_paths_horizons[0])
342             new_pred2 = predictor_ext(Por_df)
```

```
343             new_pred2.remove_outside_window(win=self.win)
344
345             data1 = new_pred2.data
346             data2 = new_pred.data
347             response = data1.to_numpy()
348
349             pred = data2.to_numpy()
350
351             for n, well in enumerate(well_files):
352                 if n == 0:
353                     pass
354                 else:
355                     log1 = load_well(file_name=well)
356
357                     data, MD = log1.from_csv()
358
359                     Por = data['por'].to_numpy()
360                     Por_df = pd.DataFrame(Por, columns=['Por'])
361
362                     data.drop('por', axis=1, inplace=True)
363
364                     new_pred = predictor_ext(data)
365
366                     for name in win_names:
367                         new_pred.roll_and_win_sel_well(data_name=name, win=self.win,
368                                                        geo_int=self.geo_int, grid=self.grid2, horizons_list=
                                                             self.horizons_list, MD=MD, well_path=
                                                             well_paths_horizons[n])
369                     new_pred2 = predictor_ext(Por_df)
370                     new_pred2.remove_outside_window(win=self.win)
371
372                     data1 = new_pred2.data
373                     data2 = new_pred.data
374
375                     response2 = data1.to_numpy()
376
377                     pred2 = data2.to_numpy()
378
379                     response = np.vstack((response, response2))
380                     pred = np.vstack((pred, pred2))
381
382             # standardize
383             pred, self.mean_arr, self.std_arr = new_pred.stand_pred(pred)
384
385             # randomize index to scramble the data
386             idx = np.random.rand(*response.shape).argsort(axis=0)
387             print(pred)
388             response = np.take_along_axis(response, idx, axis=0)
389             pred = np.take_along_axis(pred, idx, axis=0)
390             print(pred)
391
392             self.response = response
393             self.pred = pred
394
395         def load_synthetic_wells_and_predictors_ext(self):
396             """
397             load the wells from the synthetic sections as predictors and
```

```
398
399         Returns
400         -------
401         None.
402
403         """
404         #
                ##########################################################################################################################
405
406         log1 = load_well(file_name='data_2d_wedge_F3\F03_2_por_eff.xlsx')
407
408         grid_list = [self.grid2, self.grid3]
409         self.grid_list = grid_list
410         grid_names = ['imp', 'Por']
411         self.grid_names = grid_names
412
413         # get the impedance and porosity at the trace == col
414         df = log1.from_synthetic(grid_list, grid_names, col=self.wells_loc[0])
415
416         # move the porosity to a different dataframe as it is the target
417         Por = df['Por'].to_numpy()
418         Por_df = pd.DataFrame(Por, columns=['Por'])
419
420         df.drop('Por', axis=1, inplace=True)
421         #
                ##########################################################################################################################
422         name = 'imp'
423
424         new_pred = predictor_ext(df)
425
426         # select values in  window, get window mean and median and add all these as predictors
427         new_pred.roll_and_win_sel(data_name=name, win=self.win,
428                                   geo_int=self.geo_int, grid=self.grid2, col=self.wells_loc[0],
429                                   w_start=self.w_start, w_end_top=self.w_end_top, w_end_base=self.
                                        w_end_base,
430                                   horizons_list=self.horizons_list, max_TWT=self.max_TWT, min_TWT=self.
                                        min_TWT)
431
432         new_pred2 = predictor_ext(Por_df)
433         # remove target values outside the window
434         new_pred2.remove_outside_window(win=self.win)
435         data1 = new_pred2.data
436         data2 = new_pred.data
437
438         response = data1.to_numpy()
439
440         pred = data2.to_numpy()
441         #
                ##########################################################################################################################
442         if len(self.wells_loc) > 1:
443             # for each well location perform the same  predictor extraction as above
444             for i in range(len(self.wells_loc)):
445                 if i == 0:
446                     pass
```

```
447                 else:
448                     #
                            ##########################################################################################################
449                     log2 = load_well(
450                         file_name='data_2d_wedge_F3\F03_2_por_eff.xlsx')
451
452                     df2 = log2.from_synthetic(
453                         grid_list, grid_names, col=self.wells_loc[i])
454
455                     Por2 = df2['Por'].to_numpy()
456                     Por_df2 = pd.DataFrame(Por2, columns=['Por'])
457
458                     df2.drop('Por', axis=1, inplace=True)
459                     #
                            ##########################################################################################################
460                     new_pred = predictor_ext(df2)
461
462                     new_pred.roll_and_win_sel(data_name=name, win=self.win,
463                                               geo_int=self.geo_int, grid=self.grid2, col=self.wells_loc[i
                                                    ],
464                                               w_start=self.w_start, w_end_top=self.w_end_top, w_end_base=
                                                    self.w_end_base,
465                                               horizons_list=self.horizons_list, max_TWT=self.max_TWT,
                                                    min_TWT=self.min_TWT)
466
467                     new_pred2 = predictor_ext(Por_df2)
468                     new_pred2.remove_outside_window(win=self.win)
469
470                     data1 = new_pred2.data
471                     data2 = new_pred.data
472
473                     response2 = data1.to_numpy()
474
475                     pred2 = data2.to_numpy()
476
477                     response = np.vstack((response, response2))
478                     pred = np.vstack((pred, pred2))
479                     #
                            ##########################################################################################################
480         else:
481             pass
482
483         # standardize
484         pred, self.mean_arr, self.std_arr = new_pred.stand_pred(pred)
485
486         # randomize index to scramble the data
487         idx = np.random.rand(*response.shape).argsort(axis=0)
488         print(pred)
489         response = np.take_along_axis(response, idx, axis=0)
490         pred = np.take_along_axis(pred, idx, axis=0)
491         print(pred)
492
493         self.response = response
494         self.pred = pred
```

```
495
496     def ML_with_cross_val(self, ML_method, n_estimators=np.arange(4, 200, 10), max_depth=np.arange(1, 12,
            1),
497                           neurons=np.arange(10, 300, 50), layers_list=[1, 3, 5], act_list=['sigmoid'],
                                  epochs=100,
498                           N=np.arange(1, 100, 1), al_array=np.arange(0, 0.03, 0.0001), cv=10, plot=True):
499         """
500         Performes machine learning cross-validation and training.
501
502         Parameters
503         ----------
504         ML_method : str
505             the machine learning method to be used.
506         n_estimators : numpy array, optional
507             number of decision trees. The default is np.arange(4, 200, 10).
508         max_depth : numpy array, optional
509             max depth for each decision tree. The default is np.arange(1, 12, 1).
510         neurons : numpy array, optional
511             range of number of neurans in every layer considered. The default is np.arange(10, 300, 50).
512         layers_list : list, optional
513             range of number of hidden layers considered. The default is [1,3, 5].
514         act_list : list, optional
515             activation functions considered. The default is ['sigmoid'].
516         epochs : int, optional
517             numnber of epochs or cycles of training. The default is 100.
518         N : numpy array, optional
519             number of neigbors considered. The default is np.arange(1, 100, 1).
520         al_array : numpy array, optional
521             penalty coefficients considered. The default is np.arange(0, 0.03, 0.0001).
522         cv : int, optional
523             number of folds. The default is 10.
524         plot : bool, optional
525             should the cross validation scores the plotted by tuning parameter. The default is True.
526
527         Returns
528         -------
529         float
530             cross-validation MSE (CV score).
531
532         """
533
534         self.ML_method = ML_method
535         self.ML = Mlearning(self.pred, self.response, self.grid2)
536         uf = usefull_functions()
537         self.LOSS = dict()
538
539         # perform cross-validation, all if sections follow a similar logic
540         if self.ML_method == 'Random Forest':
541             CV_RF = self.ML.RF_cross_val(
542                 n_estimators=n_estimators, max_depth=max_depth, cv=cv, plot=plot)  # perform cross
                        validation
543             # retreve best cross-validation result
544             best, best_ = self.handle_CV('CV_RF', CV_RF)
545
546             # fit model according to the best parameters
547             self.ML.RF_init(n_estimators=int(best[1]), max_depth=int(best[3]))
548
```

```
549             elif self.ML_method == 'Neural Net':
550                 CV_net = self.ML.n_net_cross_val(
551                     N=neurons, layers_list=layers_list, act_list=act_list, epochs=epochs, cv=cv, plot=plot)  #
                            perform cross validation
552                 # retreve best cross-validation result
553                 best, best_ = self.handle_CV('CV_net', CV_net)
554 
555                 self.ML.n_net_init(self.pred, self.response, int(best[5]), int(
556                     best[1]), activation=best[3], epochs=epochs)  # fit model according to the best parameters
557 
558             elif self.ML_method == 'lasso':
559                 CV_lasso = self.ML.lasso_cross_val(
560                     al_array=al_array, cv=cv, plot=plot)  # perform cross validation
561 
562                 # retreve best cross-validation result
563                 best, best_ = self.handle_CV('CV_lasso', CV_lasso)
564                 # fit model according to the best parameters
565                 self.ML.lasso_init(float(best[1]))
566 
567             else:
568                 CV_KNN = self.ML.KNN_cross_val(
569                     N=N, cv=cv, plot=plot)  # perform cross validation
570                 # retreve best cross-validation result
571                 best, best_ = self.handle_CV('CV_KNN', CV_KNN)
572                 # fit model according to the best parameters
573                 self.ML.KNN_init(int(best[1]))
574 
575             self.best_ = best_  # save the best parameters as class object
576 
577             return self.neg_mse
578 
579         def predict(self, case_type='synthetic'):
580             """
581             apply trained ML method to case
582 
583             Parameters
584             ----------
585             case_type : str, optional
586                 if the model is synthetic uses ML.predict_syn_grid, uses ML.predict_syn_grid if not. The
                        default is 'synthetic'.
587 
588             Returns
589             -------
590             None.
591 
592             """
593 
594             if case_type == 'synthetic':
595                 if self.ML_method == 'Random Forest':
596 
597                     pred_map = self.ML.predict_syn_grid(self.grid_list, self.grid_names, self.win, method='RF'
                                , geo_int=self.geo_int,
598                                                         w_start=self.w_start, w_end_top=self.w_end_top,
                                                                w_end_base=self.w_end_base,
599                                                         horizons_list=self.horizons_list, max_TWT=self.max_TWT
                                                                , min_TWT=self.min_TWT,
600                                                         mean_arr=self.mean_arr, std_arr=self.std_arr)
```

```
601             else:
602
603                 pred_map = self.ML.predict_syn_grid(self.grid_list, self.grid_names, self.win, method=self
                        .ML_method, geo_int=self.geo_int,
604                                                     w_start=self.w_start, w_end_top=self.w_end_top,
                                                            w_end_base=self.w_end_base,
605                                                     horizons_list=self.horizons_list, max_TWT=self.max_TWT
                                                            , min_TWT=self.min_TWT,
606                                                     mean_arr=self.mean_arr, std_arr=self.std_arr)
607
608         else:
609             if self.ML_method == 'Random Forest':
610
611                 pred_map = self.ML.predict_grid(self.grid_list, self.grid_names, self.win, method='RF',
                        geo_int=self.geo_int,
612                                                 horizons_list=self.horizons_list, max_TWT=self.max_TWT,
                                                        min_TWT=self.min_TWT,
613                                                 mean_arr=self.mean_arr, std_arr=self.std_arr)
614             else:
615
616                 pred_map = self.ML.predict_grid(self.grid_list, self.grid_names, self.win, method=self.
                        ML_method, geo_int=self.geo_int,
617                                                 horizons_list=self.horizons_list, max_TWT=self.max_TWT,
                                                        min_TWT=self.min_TWT,
618                                                 mean_arr=self.mean_arr, std_arr=self.std_arr)
619
620         self.pred_map = pred_map
621
622     def result_eval(self, vcut=False, hcut=False, ylab='TWT', vminp=0.2, vmaxp=0.4):
623         """
624         compare the ML prediction against the true porosity, assuming a synthetic model.
625         Returns statistical values of the comparison.
626         Plots the true porosity, predicted porosity and the absolute difference between the two.
627
628         Parameters
629         ----------
630         vcut : list, optional
631             vertical slice of the array (TWT). The default is False.
632         hcut : list, optional
633             horizontal slice of the array (traces). The default is False.
634         ylab : str, optional
635             y label. The default is 'TWT'.
636         vminp : float, optional
637             minimum porosity value in colormap. The default is 0.2.
638         vmaxp : float, optional
639             maximum porosity value in colormap. The default is 0.4.
640
641         Returns
642         -------
643         float
644             mean absolute error.
645         float
646             mean squared error.
647         float
648             r2 score.
649         list
650             standard deviation of the absolute error.
```

```
651
652         """
653
654         vmin = np.min(self.grid3)
655         vmax = np.max(self.grid3)
656
657         # in order to take into account the window size if used
658         por_grid = self.grid3[:, self.win:len(self.grid3[0])-self.win]
659
660         pl = plot_results()
661         # titles = ['using {}, Wells at {}, window = {}'.format(self.ML_method, str(self.wells_loc),str(
                self.win)),
662         #           'Estimate', 'absolute error with {}, Wells at {}, window = {}'.format(self.ML_method,
                str(self.wells_loc),str(self.win))]
663         titles = ['Prediction on bottom using {}, Wells at {}'.format(self.ML_method, str(self.wells_loc))
                ,
664                   'Estimate', 'absolute error with {}, Wells at {}'.format(self.ML_method, str(self.
                        wells_loc))]
665
666         # file name
667         best_ = self.best_.replace("/", "")
668         file_name = 'results/{}/figures/{}, using {}, wells at {}, window = {}, geo int = {}'.format(
669             self.case, self.ML_method, best_, self.wells_loc, str(self.win), self.geo_int)
670         file_name = file_name.replace("[", "")
671         file_name = file_name.replace("]", "")
672         file_name = file_name.replace(":", "")
673
674         pl.grids_comp(por_grid, self.pred_map, titles=titles, vcut=vcut, hcut=hcut, file_name=file_name,
675                       extent=self.extent2, ylab=ylab, vmin=vminp, vmax=vmaxp)
676
677         r_squared = r2_score(por_grid.flatten(), self.pred_map.flatten())
678
679         self.LOSS[self.ML_method +
680                   ' AE_std'] = np.std(abs(por_grid.flatten()-self.pred_map.flatten()))
681         self.LOSS[self.ML_method +
682                   ' MAE'] = np.mean(abs(por_grid.flatten()-self.pred_map.flatten()))
683         self.LOSS[self.ML_method+' MSE'] = np.mean((por_grid-self.pred_map)**2)
684         self.LOSS[self.ML_method+' r2 score'] = r_squared
685
686         res = {'case: ': [self.case],
687                'wells location ': [str(self.wells_loc)],
688                'method: ': [self.ML_method],
689                'window for prediction extraction :': [self.win],
690                'Geological interpretation': [self.geo_int],
691                'hyperparameter tuning result: ': [best_],
692                'Cross validation negative MSE ': [self.neg_mse],
693                'validation error MAE: ': [self.LOSS[self.ML_method+' MAE']],
694                'validation error MSE: ': [self.LOSS[self.ML_method+' MSE']],
695                'abs error std': [self.LOSS[self.ML_method+' AE_std']],
696                'r2 score': [self.LOSS[self.ML_method+' r2 score']]
697                }
698         df = pd.DataFrame(res)
699         file_name = file_name.replace("figures", "info")
700         df.to_csv(file_name+'.txt', sep='/', index=False, encoding='utf-8')
701
702         return self.LOSS[self.ML_method+' MAE'], self.LOSS[self.ML_method+' MSE'], self.LOSS[self.
                ML_method+' r2 score'], [self.LOSS[self.ML_method+' AE_std']]
```

```
703
704     def result_eval_F3(self, vcut=False, hcut=False, ylab='TWT', vminp=0.2, vmaxp=0.4):
705         """
706         compare the ML prediction against the true porosity in the F3 case.
707         Returns statistical values of the comparison.
708         Plots the true porosity, predicted porosity and the absolute difference between the two.
709
710         vcut : list, optional
711             vertical slice of the array (TWT). The default is False.
712         hcut : list, optional
713             horizontal slice of the array (traces). The default is False.
714         ylab : str, optional
715             y label. The default is 'TWT'.
716         vminp : float, optional
717             minimum porosity value in colormap. The default is 0.2.
718         vmaxp : float, optional
719             maximum porosity value in colormap. The default is 0.4.
720
721         Returns
722         -------
723         res : dict
724             dictionary of the parameters used, the cross-validation result and the cross-validation MSE.
725
726         """
727
728         vmin = np.min(self.pred_map)
729         vmax = np.max(self.pred_map)
730
731         # in order to take into account the window size if used
732         pet_grid = self.grid_pet[:, self.win:len(self.grid_pet[0])-self.win]
733
734         pl = plot_results()
735
736         titles = ['Prediction on bottom using {}'.format(self.ML_method),
737                   'Estimate', 'absolute difference between solutions']
738
739         # file name
740         best_ = self.best_.replace("/", "")
741         file_name = 'results/{}/figures/{}, using {}, window = {}, geo int = {}'.format(
742             self.case, self.ML_method, best_, str(self.win), self.geo_int)
743         file_name = file_name.replace("[", "")
744         file_name = file_name.replace("]", "")
745         file_name = file_name.replace(":", "")
746         pl.grids_comp(pet_grid, self.pred_map, titles=titles, vcut=vcut, hcut=hcut, file_name=file_name,
747                       extent=self.extent2, ylab=ylab, vmin=vminp, vmax=vmaxp)
748
749         res = {'case: ': [self.case],
750                'method: ': [self.ML_method],
751                'window for prediction extraction :': [self.win],
752                'Geological interpretation': [self.geo_int],
753                'hyperparameter tuning result: ': [best_],
754                'Cross validation negative MSE ': [self.neg_mse]
755                }
756         df = pd.DataFrame(res)
757         file_name = file_name.replace("figures", "info")
758         df.to_csv(file_name+'.txt', sep='/', index=False, encoding='utf-8')
759
```

```
760             return res
761
762         def petrel_solution_eval(self):
763             """
764             Compare the Sequential Gaussian simulation against the true porosity.
765             Returns statistical values of the comparison.
766             Plots the true porosity, predicted porosity and the absolute difference between the two.
767
768             Returns
769             ---------
770             MAE : float
771                 mean absolute error.
772             MSE : float
773                 mean squared error.
774             r_2 : float
775                 r2 score.
776             AE_std : float
777                 standard deviation of the absolute error.
778
779             """
780             ext = seismic_ext(self.petrel_solution)
781             gridsol = ext.syn_seismic(plot=False, scaling=False)[0]
782
783             if self.case == 'case 2a wedge hetero' or self.case == 'case 2b wedge hetero':
784                 gridsol = gridsol[:, 0:75]
785
786             pl = plot_results()
787             titles = ['True porosity: top, classic solution: below',
788                       'Classical solution', 'Difference petrel solution']
789
790             pl.grids_comp(self.grid3, gridsol, titles=titles, file_name='results/{}/figures/{} Petrel_solution
                    '.format(self.case, self.case),
791                           extent=self.extent2)
792
793             self.gridsol = gridsol
794
795             AE_std = np.std(abs(self.grid3.flatten() - gridsol.flatten()))
796             r_2 = r2_score(self.grid3.flatten(), gridsol.flatten())
797             MAE = np.mean(abs(self.grid3.flatten() - gridsol.flatten()))
798             MSE = np.mean(((self.grid3.flatten() - gridsol.flatten())**2))
799
800             return MAE, MSE, r_2, AE_std
```

C.4 seismic_ext.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Sun Dec 19 13:54:10 2021
4
5   @author: Eier
6   """
7
8
9   import segyio
10  import matplotlib.pyplot as plt
11  import numpy as np
12  from scipy import ndimage as ndi
```

```
13
14
15
16  class seismic_ext:
17      """
18      Exctracts sections using segyio
19      """
20
21      def __init__(self, file_name):
22          """
23          initialise class
24
25          Parameters
26          ----------
27          file_name : float
28              file path of the segy file.
29
30          Returns
31          -------
32          None.
33
34          """
35
36          self.file = file_name
37
38      def syn_seismic(self, plot = False, scaling = False):
39          """
40          uses segyio to get the section as a numpy grid.
41          also gets the depth ready for the class plot results.
42
43          Parameters
44          ----------
45          plot : bool, optional
46              Should the sections be plotted for evaluation? The default is False.
47          scaling : bool, optional
48              should the section numpy grid be standardized? The default is False.
49
50          Returns
51          -------
52          grid : 2D numpy array
53              section as numpy array, traces as rows, depth as columns.
54          extent : list
55              axis extent as traces and TWT, for later plotting.
56
57          """
58
59          # open file
60          f = segyio.open(self.file, ignore_geometry=True)
61
62          # get the metadata, incl. TWT and trace ranges
63          sec = segyio.tools.metadata(f)
64          TWT = -sec.samples # get depth data
65          TWT_max = TWT.max()
66          TWT_min = TWT.min()
67          trace_min = f.header[0][segyio.TraceField.TraceNumber]
68          trace_max= f.header[-1][segyio.TraceField.TraceNumber]
69
```

```
70          extent = [trace_min, trace_max, TWT_min, TWT_max] # get depth and trace numbers for the plots
71
72          # get cross-section
73          grid = f.trace.raw[:]
74          f.close()
75
76          if plot == True:
77              vm = np.percentile(grid, 99.5)
78              plt.imshow(grid.T, cmap='jet', aspect='auto', vmin=-vm, vmax=vm)
79              plt.title(self.file)
80              plt.show()
81
82          else:
83              pass
84
85          if scaling == True:
86              grid = (grid - np.mean(grid)) / np.std(grid) # standardize data
87
88          self.grid = grid
89
90          return grid, extent
```

C.5 load_well.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Sun Dec 19 14:26:19 2021
4
5   @author: Eier
6   """
7   import pandas as pd
8   import matplotlib.pyplot as plt
9   import numpy as np
10
11  class load_well:
12      """
13      Handles the extraction and editing of the well-data.
14      This can be real well data or synthetic well data from a 2D array of synthetic data.
15      """
16      def __init__(self, file_name):
17          """
18          Handles the extraction and editing of the well-data.
19          This can be real well data or synthetic well data from a 2D array of synthetic data.
20
21          Parameters
22          ----------
23          file_name : str
24              The file location of the well data.
25
26          Returns
27          -------
28          None.
29
30          """
31
32          self.file = file_name
33
```

```
34      def from_excel(self):
35          """
36          Imports well-log data from an excel file.
37
38          Returns
39          -------
40          data : pandas.DataFrame
41              dataframe of the well-logs.
42              one column per well-log.
43
44          """
45
46          data= pd.read_excel(self.file, header = 2) #file_name = 'data_2d_wedge_F3\F03-2_por_eff.xlsx'
47
48          data.drop('Unnamed: 0', inplace = True, axis = 1)
49          data = data.dropna()
50          #data = data.set_index('MD')
51          data.drop('MD', inplace = True, axis = 1)
52
53          return data
54      def from_csv(self, drop_MD = True):
55          """
56          Imports well-log data from an csv file.
57
58          Parameters
59          ----------
60          drop_MD : bool, optional
61              if True; will drop the 'MD' column. The default is True.
62
63          Returns
64          -------
65          data : pandas.DataFrame
66              dataframe of the well-logs.
67              one column per well-log.
68          MD : numpy array
69              the measured depth.
70
71          """
72
73          data = pd.read_csv(self.file)
74          MD = data['MD'].to_numpy()
75          if drop_MD==True:
76
77              data.drop('MD', inplace = True, axis = 1)
78
79          return data, MD
80
81
82      def from_synthetic(self, grid_list, name_list, col = 150):
83          """
84          Imports a trace from one or more synthetic sections as if it was a well-log.
85
86          Parameters
87          ----------
88          grid_list : list
89              list of 2d arrays representing the cross-sections.
90          name_list : list
```

```
91              list of grid names, for example ['imp', 'por'].
92          col : int, optional
93              trace number. The default is 150.
94
95          Returns
96          -------
97          df : pandas.DataFrame
98              dataframe of the well-logs.
99              one column per well-log.
100
101         """
102         n = 0
103         l = len(grid_list[0].T)
104         data_ar = np.zeros((l, len(grid_list)))
105
106         # get trace = col in every cross-section
107         for grid in grid_list:
108             data_ar[:,n] = grid[col] # row in grid should be verticle ---> .T provides correct plot
109             n = n+1
110
111         # compile traces into dataframe
112         df = pd.DataFrame(data_ar, columns = name_list)
113         return df
114
115
116 # def win_select(data, data_name = 'Por. Eff.',win =4):
117
118 #     data_col= data[data_name].to_numpy()
119
120 #     win_2 = int(win/2)
121 #     l = len(data_col)
122
123 #     for w in range(win_2):
124 #         w = w+1
125
126
127 #         # start with 1 and -1
128 #         #
129 #         up = np.zeros(l)
130 #         up[:] = np.nan
131 #         up[w:l] = data_col[0:l-w]
132
133 #         down = np.zeros(l)
134 #         down[:] = np.nan
135 #         down[0:l-w] = data_col[0+w:l]
136
137 #         data['+{} wind'.format(w)] = up
138 #         data['-{} wind'.format(w)] = down
139 #     return data
140
141
142 # df = pd.read_excel('data\case F3\F02_1.xlsx')
143 # df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
144
145 # df_por = pd.DataFrame()
146 # df_por['MD'] = df['MD']
147 # df_por['Por. Eff.'] = df['Por. Eff.']
```

```
148 # df_por = df_por.dropna()
149
150 # df_imp = pd.DataFrame()
151 # df_imp['MD'] = df['MD.1']
152 # df_imp['P-imp.'] = df['P-imp.']
153 # df_imp = df_imp.dropna()
154
155 # por_MD_lim = [df_por['MD'].min(), df_por['MD'].max()]
156
157 # df_imp = df_imp[df_imp['MD']>por_MD_lim[0]]
158 # df_imp = df_imp[df_imp['MD']<por_MD_lim[1]]
159
160
161 # df_imp = df_imp.reset_index()
162 # df_por = df_por.reset_index()
163
164 # df_well = pd.DataFrame()
165
166 # def nearest(number, arr):
167 #     """
168 #     Gets index of number nearest the "number"
169
170 #     Parameters
171 #     ----------
172 #     number : TYPE
173 #         DESCRIPTION.
174 #     arr : TYPE
175 #         DESCRIPTION.
176
177 #     Returns
178 #     -------
179 #     TYPE
180 #         DESCRIPTION.
181
182 #     """
183 #     search = abs(arr-number)
184 #     m = search.min()
185 #     return np.where(search == m)[0]
186
187 # md_list = []
188 # imp_list = []
189 # por_list = []
190
191
192 # for n, i in enumerate(df_imp['MD']):
193
194 #     arr = df_por['MD'].to_numpy()
195
196 #     ind = nearest(i, arr)[0]
197
198 #     md = df_imp.iloc[n]['MD']
199 #     md_list.append(md)
200 #     imp = df_imp.iloc[n]['P-imp.']
201 #     imp_list.append(imp)
202 #     por = df_por.iloc[ind]['Por.Eff.']
203 #     por_list.append(por)
204
```

```
205 # df_well['MD'] = md_list
206 # df_well['imp'] = imp_list
207 # df_well['por'] = por_list
208
209 # plt.scatter(df_well['imp'], df_well['por'])
```

C.6 predictor_ext.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Dec 23 16:06:26 2021
4
5  @author: Eier
6  """
7  import numpy as np
8  import matplotlib.pyplot as plt
9  from classes.usefull_functions import usefull_functions
10 import pandas as pd
11
12 class predictor_ext:
13
14     def __init__(self, data):
15         """
16
17
18         Parameters
19         ----------
20         data : pandas DataFrame
21             predictor data.
22
23         Returns
24         -------
25         None.
26
27         """
28
29         self.data = data
30
31     def add_well_loc(self, well_loc):
32         """
33         adds the well location as a predictor.
34         Only works for synthetic models
35
36         Parameters
37         ----------
38         well_loc : int
39             trace (well) location.
40
41         Returns
42         -------
43         None.
44
45         """
46         self.data['well location'] = np.repeat(well_loc, len(self.data))
47
48
49     def median_and_mean(self, data_name = 'Por.Eff.',win =4):
```

```
        """
        adds the rolling mean, median window results as predictors

        Parameters
        ----------
        data_name : str, optional
            name of the column that the rolling windows should apply to. The default is 'Por.Eff.'.
        win : int, optional
            window size. The default is 4.

        Returns
        -------
        None.

        """
        self.roll_mean(data_name = data_name,win = win)
        self.roll_median(data_name = data_name,win = win)
        self.remove_outside_window(win = win)

    def roll_and_win_sel(self, data_name = 'Por.Eff.',win =4,
                                geo_int = 'none', horizons_list = None,grid = 1, col = 1,
                                w_start = [267, 178], w_end_top = [0, 178], w_end_base = [0, 273],
                                max_TWT = -618, min_TWT = -1162):
        """
        adds the rolling mean, median and selections window results as predictors.
        also adds the depositional time if apporopriate.

        Parameters
        ----------
        data_name : str, optional
            name of the column that the rolling windows should apply to. The default is 'Por.Eff.'.
        win : int, optional
            window size. The default is 4.
        geo_int : str, optional
            Should the depositional time be implemented and if so how?. The default is 'none'.
        horizons_list : TYPE, optional
            DESCRIPTION. The default is None.
        grid : numpy array, optional
            2D array of a cross-section. The default is 1.
        col : int, optional
            trace (well) location. The default is 1.
        w_start : list, optional
            the starting position of the wedge: the pinch point. The default is [267, 178].
        w_end_top : list, optional
            the end of the top surface of the wedge. The default is [0, 178].
        w_end_base : list, optional
            the end of the base surface of the wedge. The default is [0, 273].
        max_TWT : int, optional
            maximum value of the TWT. The default is -618.
        min_TWT : int, optional
            minimum value of the TWT. The default is -1162.

        Returns
        -------
        None.

        """
```

```
107
108             # self.add_well_loc(well_loc=col)
109
110             # add window functions
111             self.roll_mean(data_name = data_name,win = win)
112             self.roll_median(data_name = data_name,win = win)
113             self.win_select(data_name = data_name,win = win)
114
115             # add depo-time
116             if geo_int == 'wedge':
117                 self.construct_timelines_wedge_df(grid = grid, w_start = w_start, w_end_top=w_end_top,
                        w_end_base=w_end_base,
118                                                   col = col, win = win)
119             elif geo_int == 'from horizons':
120                 self.construct_timelines_from_horizons(grid, horizons_list, max_TWT = max_TWT, min_TWT =
                        min_TWT)
121                 self.depotime_well(well_loc = col)
122
123             # remove data outside of the windows
124             self.remove_outside_window(win = win)
125
126         def roll_and_win_sel_well(self, data_name = 'Por.Eff.',win =4,
127                                   geo_int = 'none', horizons_list = None,grid = 1, MD = None, well_path = None
                                        ):
128             self.roll_mean(data_name = data_name,win = win)
129             self.roll_median(data_name = data_name,win = win)
130             self.win_select(data_name = data_name,win = win)
131
132             # add depo-time
133             if geo_int == 'from horizons':
134                 self.depotime_well_path(MD = MD, well_path = well_path)
135
136
137             self.remove_outside_window(win = win)
138     ##########################################################################################
139         def roll_mean(self, data_name = 'Por.Eff.',win =4):
140             """
141             applies the rolling mean to a column in the dataframe
142
143             Parameters
144             ----------
145             data_name : str, optional
146                 name of the column that the rolling windows should apply to. The default is 'Por.Eff.'.
147             win : int, optional
148                 window size. The default is 4.
149
150             Returns
151             -------
152             data : pandas DataFrame
153                 full dataframe of the predictors.
154
155             """
156             data = self.data
157             # failsafe for if no window is wanted
158             if win == 0:
159                 pass
160             else:
```

```
161             data['roll mean'] = data[data_name].rolling(window=win).mean()
162             self.data = data
163         return data
164 
165     def roll_median(self, data_name = 'Por.Eff.',win =4):
166         """
167         applies the rolling median to a column in the dataframe
168 
169         Parameters
170         ----------
171         data_name : str, optional
172             name of the column that the rolling windows should apply to. The default is 'Por.Eff.'.
173         win : int, optional
174             window size. The default is 4.
175 
176         Returns
177         -------
178         data : pandas DataFrame
179             full dataframe of the predictors.
180 
181         """
182         data = self.data
183         # failsafe for if no window is wanted
184         if win == 0:
185             pass
186         else:
187             data['roll median'] = data[data_name].rolling(window=win).median()
188             self.data = data
189         return data
190 
191     def win_select(self, data_name = 'Por.Eff.',win =4):
192         """
193         applies window selection to a predictor.
194         This means that in a window of [-win/2, +win/2] every data point from the point of consideration
                is added as a predictor
195 
196         Parameters
197         ----------
198         data_name : str, optional
199             name of the column that the rolling windows should apply to. The default is 'Por.Eff.'.
200         win : int, optional
201             window size. The default is 4.
202 
203         Returns
204         -------
205         data : pandas DataFrame
206             full dataframe of the predictors.
207 
208         """
209 
210         data = self.data
211         # failsafe for if no window is wanted
212         if win == 0:
213             pass
214         else:
215             data_col= data[data_name].to_numpy()
216 
```

```
217
218             win_2 = int(win/2)
219             l = len(data_col)
220
221             # for every point above and below the point of computation
222             for w in range(win_2):
223                 w = w+1
224
225                 # prepare to get higher value
226                 up = np.zeros(l)
227                 up[:] = np.nan
228                 # get value
229                 up[w:l] = data_col[0:l-w]
230
231                 # prepare to get lower value
232                 down = np.zeros(l)
233                 down[:] = np.nan
234                 # get value
235                 down[0:l-w] = data_col[0+w:l]
236
237                 # save values
238                 data['+{} wind'.format(w)] = up
239                 data['-{} wind'.format(w)] = down
240
241             self.data = data
242         return data
243
244     def remove_outside_window(self, win):
245         """
246         Remove the top and bottom rows equal to the window size.
247
248         Parameters
249         ----------
250         win : int
251             window size.
252
253         Returns
254         -------
255         None.
256
257         """
258
259         self.data = self.data.iloc[win:]
260         self.data = self.data.iloc[0:len(self.data)-win]
261
262
263     def construct_timelines_wedge(self, grid, w_start = [267, 178], w_end_top = [0, 178], w_end_base = [0,
            273]):
264         """
265         A non-ideal function that constructs an array of the depositional time for a wedge that is
                thinning to the right.
266
267         Parameters
268         ----------
269         grid : numpy array
270             2D array of a cross-section.
271         w_start : list, optional
```

```
272             the starting position of the wedge: the pinch point. The default is [267, 178].
273         w_end_top : list, optional
274             the end of the top surface of the wedge. The default is [0, 178].
275         w_end_base : list, optional
276             the end of the base surface of the wedge. The default is [0, 273].
277
278         Returns
279         -------
280         res : numpy array
281             2D array with the relative depositional time.
282
283         """
284
285         inter = 1
286         if w_start > w_end_top:
287             direction = 'thinning right'
288
289         rows = np.arange(min([w_start[0], w_end_top[0]]), max([w_start[0], w_end_top[0]]), 1) #  what rows
                are relavant
290
291         if direction == 'thinning right':
292             # start_top0 = np.linspace(),len(rows)).round()
293             start_top1 = np.linspace(min(w_start[1], w_end_top[1]),max(w_start[1], w_end_top[1]),len(rows)
                    ).round()
294
295             # start_base0 = np.linspace(min(w_start[0], w_end_base[0]),max(w_start[0], w_end_base[0]),len(
                    rows)).round()
296             start_base1 = np.linspace(max(w_start[1], w_end_base[1]),min(w_start[1], w_end_base[1]),len(
                    rows)).round()
297
298         else:
299             start_top0 = np.linspace(min(w_start[0], w_end_top[0]),max(w_start[0], w_end_top[0]),len(rows)
                    ).round()
300             start_top1 = np.linspace(min(w_start[1], w_end_top[1]),max(w_start[1], w_end_top[1]),len(rows)
                    ).round()
301
302             start_base0 = np.linspace(min(w_start[0], w_end_base[0]),max(w_start[0], w_end_base[0]),len(
                    rows)).round()
303             start_base1 = np.linspace(min(w_start[1], w_end_base[1]),max(w_start[1], w_end_base[1]),len(
                    rows)).round()
304
305         top_ts = np.arange(0, int(start_top1[1]), inter) # timelines above wedge
306
307         missing = np.arange(int(start_top1[1]), int(max(start_base1)), inter) # timelines inside wedge
308
309         base_ts = np.arange(int(max(start_base1)),len(grid[0]), inter) # timelines below wedge
310
311         L = abs(w_start[0]-w_end_top[0])
312
313         change_in_inter = np.linspace(inter, 0, L)
314
315         res = np.zeros(np.shape(grid))
316
317         for n, row in enumerate(grid): # vertical, left to right in imshow
318             print(n)
319             l = len(row)
320             new_row = np.zeros(l)
```

```
321
322             if n in rows:
323                 if direction == 'thinning right':
324
325                     change = change_in_inter[n]
326
327                     new_row[0:int(start_top1[n])] = np.linspace(top_ts[0],top_ts[-1], len(rows[0:int(
                            start_top1[n])]))
328                     ll = len(row[int(start_top1[n]):int(start_base1[n])])
329                     new_row[int(start_top1[n]):int(start_base1[n])] = np.linspace(missing[0], missing[-1],
                             len(new_row[int(start_top1[n]):int(start_base1[n])]))
330                     new_row[int(start_base1[n]):] = np.linspace(base_ts[0], base_ts[0]+len(new_row[int(
                            start_base1[n]):]), len(new_row[int(start_base1[n]):]))
331
332             else:
333                 new_row[0:int(start_top1[0])] = np.linspace(top_ts[0], top_ts[-1], len(new_row[0:int(
                        start_top1[0])]))
334                 new_row[int(start_top1[0]):] = np.linspace(base_ts[0], base_ts[0]+len(new_row[int(
                        start_top1[0]):]), len(new_row[int(start_top1[0]):] ))
335
336             res[n] = new_row
337
338         res = res.round()
339         # res = (res - np.mean(res)) / np.std(res) # standardizing data
340
341         return res
342
343     def construct_timelines_wedge_df(self, grid, w_start = [267, 178], w_end_top = [0, 178], w_end_base
            = [0, 273],
344                                      col = 100):
345         """
346         Make the relative depositional time for a synthetic wedge,
347         then exctract one of the columns as a predictor.
348
349         Parameters
350         ----------
351         grid : numpy array
352             2D array of a cross-section.
353         w_start : list, optional
354             the starting position of the wedge: the pinch point. The default is [267, 178].
355         w_end_top : list, optional
356             the end of the top surface of the wedge. The default is [0, 178].
357         w_end_base : list, optional
358             the end of the base surface of the wedge. The default is [0, 273].
359         col : int, optional
360             trace (well) location. The default is 100.
361
362         Returns
363         -------
364         self.data : pandas DataFrame
365             full dataframe of the predictors.
366
367         """
368
369         # get the timelines
370         self.timelines = self.construct_timelines_wedge(grid, w_start, w_end_top, w_end_base)
371
```

```
372             data = self.timelines[col]
373
374             # save the timelines for one trace
375             self.data['time lines {}'.format(col)] = data
376
377             # plt.imshow(self.timelines.T, aspect = 'auto',cmap='jet')
378             # plt.title('Depotime from wedge interpretaion')
379             # plt.colorbar()
380             # plt.show()
381
382
383             # self.data = (self.data - np.mean(self.data)) / np.std(self.data)
384             return self.data
385         def construct_timelines_from_horizons(self, grid, horizons_list, max_TWT = -618, min_TWT = -1162,
                standardizing = True):
386             """
387             make the deopsitional time array from where the horizons interect with the cross-sections
388
389             Parameters
390             ----------
391             grid : numpy array
392                 2D array of a cross-section.
393             horizons_list : list or None, optional
394                 list of numpy arrays that describe where the horizons intersect with the cross-section. The
                        default is None.
395             max_TWT : int, optional
396                 maximum value of the TWT. The default is -618.
397             min_TWT : int, optional
398                 minimum value of the TWT. The default is -1162.
399             standardizing : bool, optional
400                 should the resulting array be standardized?. The default is True.
401
402             Returns
403             -------
404             res : numpy array
405                 2D array of the depositional time.
406
407             """
408             # ordered from max to min
409             n_traces, n_samples= np.shape(grid)
410             interval = -abs((max_TWT-min_TWT)/n_samples)
411             print(n_samples)
412
413             res = np.zeros(np.shape(grid))
414
415             # for every trace
416             for s in range(n_traces):
417                 stack = np.array([])
418                 # for every horizon
419                 for i in range(len(horizons_list)):
420                     # find the upper and lower horizons
421                     upper_h = horizons_list[i-1]
422                     lower_h = horizons_list[i]
423
424                     # firs (top) horizon
425                     if i == 0:
426                         # if the upper horizon extends beyond the cross-section then skip
```

```
427                     if upper_h[s] > max_TWT:
428                         pass
429                     # if the upper horizon does no extend beyond the cross-section then fill in the
                            missing values above the horizon
430                     else:
431
432                         upper_h = horizons_list[i]
433                         times = np.arange(max_TWT, upper_h[s], interval)
434                         times = times[times<=max_TWT]
435
436                         times = np.linspace(i, i+1, len(times))
437
438                         stack = np.hstack((stack, times))
439                 else:
440
441                     times = np.arange(upper_h[s], lower_h[s], interval)
442
443                     times = times[times<=max_TWT]
444
445                     times = np.linspace(i, i+1, len(times))
446
447                     stack = np.hstack((stack, times))
448
449             if len(stack) < len(res[s]): # happens if the bottom horizon is not beyond the min_TWT,
                    meaning the remaining time needs to be filled in
450                 diff = abs(len(stack) - len(res[s]))
451                 stack = np.hstack((stack, np.linspace(i+1, i+2, diff)))
452
453
454             res[s] = stack[0:len(res[s])]
455
456
457             self.times = res
458
459         # plt.imshow(res.T, aspect = 'auto',cmap='jet')
460         # plt.title('Depotime from horizons')
461         # plt.colorbar()
462         # plt.show()
463
464         if standardizing == True:
465             pass
466             # res = (res - np.mean(res)) / np.std(res) # standardizing data
467         # plt.imshow(res.T, cmap='nipy_spectral', aspect='auto', extent=[0,600,min_TWT,max_TWT])
468         # plt.colorbar()
469         # plt.show()
470         return res
471
472     def depotime_well_path(self, MD = None, well_path = None):
473         """
474         Add the depositional time as a predictor.
475         This is for a real well, not synthetic data.
476
477         Parameters
478         ----------
479         MD : numpy array, optional
480             measured depth of the well-logs. The default is None.
481         well_path : str, optional
```

```
482             path to a file containing the horizon locations intersecting the cross-section. The default is
                    None.
483
484         Returns
485         -------
486         None.
487
488         """
489
490         MD = -MD
491
492         # get the horizon locations in the well
493         well_p = pd.read_csv(well_path)
494         well_p_np = -well_p.to_numpy()
495
496         well_p_np.sort()
497         well_p_np = -well_p_np[0]
498
499         uf = usefull_functions()
500
501         indx_list = []
502
503         # for each horizon
504         for hor_loc in well_p_np:
505
506             # find the horizon location in the MD-log
507             indx = uf.nearest(hor_loc, MD)[0]
508             indx_list.append(indx)
509
510         indx_list.sort()
511
512         # interpolating the time
513         time = np.zeros(len(MD))
514         i_old = 0
515         n_old = 0
516         for n, i in enumerate(indx_list):
517
518             time[i_old:i] = np.linspace(n_old, n+1,len(time[i_old:i]) )
519
520             i_old = i
521             n_old = n+1
522
523         time[i_old:] = np.linspace(n_old, n_old+1,len(time[i_old:]) )
524
525         self.time = time
526
527         self.data['depotime'] = self.time
528
529     def depotime_well(self, well_loc):
530         self.data['depo time at {}'.format(well_loc)] = self.times[well_loc]
531
532     def stand_pred(self, pred, mean_list = None, std_list = None):
533         """
534         standardize all the predictors and save the standard deviation and mean used for the
                standardization.
535
536         Parameters
```

```
        ----------
        pred : numpy array
            numpy array of predictors.
        mean_list : numpy array, optional
            array of the mean values to be used in the standization. The default is None.
        std_list : numpy array, optional
            array of the standard deviation values to be used in the standization. The default is None.

        Returns
        -------
        pred : numpy array
            numpy array of predictors.
        mean_arr : numpy array
            array of the mean values used in the standization.
        std_arr : numpy array
            array of the standard deviation values used in the standization.

        """

        # reserving memory
        mean_arr = np.zeros(len(pred[0]))
        std_arr = np.zeros(len(pred[0]))

        # if previous mean and std are to be used
        if type(mean_list)==np.ndarray and type(std_list) ==np.ndarray:
            for n in range(len(pred[0])):
                pred[:,n] = ( pred[:,n] - mean_list[n]) / std_list[n]

        else:

            for n in range(len(pred[0])):
                mean_arr[n] = np.mean( pred[:,n])
                std_arr[n] = np.std( pred[:,n])



                pred[:,n] = ( pred[:,n] - np.mean( pred[:,n])) / np.std( pred[:,n])



        return pred, mean_arr, std_arr
```

C.7 Mlearning.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 27 13:00:06 2021

@author: Eier
"""
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
import sklearn
from sklearn.model_selection import KFold
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
14  from sklearn.model_selection import cross_val_score
15  import tensorflow as tf
16  from sklearn.ensemble import RandomForestRegressor
17  from classes.load_well import load_well
18  from sklearn import linear_model
19
20
21  class Mlearning:
22      """
23      Handels the cross validation, training and application (predictiion)
24      of the machine learning methods.
25      """
26      def __init__(self, pred, response, grid):
27          """
28          Handels the cross validation, training and application (predictiion)
29          of the machine learning methods.
30
31          Parameters
32          ----------
33          pred : numpy array
34              predictors
35          response : numpy array
36              response or target values
37          grid : numpy array
38              grid to map the predictions on, for the thesis: the impedance section
39
40          Returns
41          -------
42          None.
43
44          """
45          # set the seeds, means that the result will always be the same given the same data and parameters
46          np.random.seed(seed = 1)
47          tf.random.set_seed(1)
48
49          self.pred =pred
50          self.response=response
51          self.grid=grid
52
53      def KNN_init(self, k):
54          """
55          Initiallise and fit KNN
56
57          Parameters
58          ----------
59          k : int
60              number of neigbors considered
61
62          Returns
63          -------
64          None.
65
66          """
67          self.model = KNeighborsRegressor(n_neighbors=k)
68          self.model.fit(self.pred, self.response)
69
70      def RF_init(self, n_estimators = 100,max_depth = 3):
```

```
71          """
72          initiallise and fit random forest
73
74          Parameters
75          ----------
76          n_estimators : int, optional
77              number of trees. The default is 100.
78          max_depth : int, optional
79              max depth of all decision trees. The default is 3.
80
81          Returns
82          -------
83          None.
84
85          """
86          self.model = RandomForestRegressor(n_estimators = n_estimators,max_depth=max_depth)
87          response = np.ravel(self.response)
88          self.model.fit(self.pred, response)
89
90
91      def lasso_init(self, alpha = 0.1):
92          """
93          initialise and fit lasso
94
95          Parameters
96          ----------
97          alpha : float, optional
98              coefficiant for the penalty term. The default is 0.1.
99
100         Returns
101         -------
102         None.
103
104         """
105         self.model = linear_model.Lasso(alpha = alpha)
106         self.model.fit(self.pred, self.response)
107
108     def n_net_init(self, pred, response, n, layers = 1,epochs=10, activation='sigmoid', optimizer = 'adam'
            , verbose=0):
109         """
110         initialise and fit neural network
111
112         Parameters
113         ----------
114         pred : numpy array
115             predictors
116         response : numpy array
117             response or target values
118         n : int
119             number of neurans in each layer.
120         layers : int, optional
121             number of hidden layers. The default is 1.
122         epochs : int, optional
123             number of forward and backward prop. The default is 10.
124         activation : str, optional
125             transfer function. The default is 'sigmoid'.
126         optimizer : str, optional
```

```
127             optimizer function. The default is 'adam'.
128         verbose : int, optional
129             if 0: stops the network printouts, if 1 the printouts are enabaled. The default is 0.
130 
131         Returns
132         -------
133         None.
134 
135         """
136         # self.n_model = tf.keras.models.Sequential([
137         #     tf.keras.Input(np.shape(self.pred[0])),
138         #     tf.keras.layers.Dense(n, activation=activation),
139         #     tf.keras.layers.Dense(1, activation=activation)
140         # ])
141 
142         self.model = tf.keras.models.Sequential()
143         # add input layer
144         self.model.add(tf.keras.Input(np.shape(self.pred[0])))
145 
146         # add hidden layers
147         for i in range(layers):
148             self.model.add(tf.keras.layers.Dense(n, activation=activation))
149 
150         # add output layer
151         self.model.add(tf.keras.layers.Dense(1, activation=activation))
152 
153         self.model.compile(optimizer=optimizer,
154                            loss='MeanSquaredError',#'mean_squared_logarithmic_error',#'
                                   categorical_crossentropy',#'MeanSquaredError',
155                            #loss='MeanAbsoluteError',
156                            metrics=['mse'])
157 
158         self.model.fit(self.pred, self.response, epochs=epochs, verbose=0)
159 
160 
161     def RF_cross_val(self, n_estimators = np.arange(100, 250, 1),max_depth = np.arange(2, 5, 1), cv = 10,
          plot = True):
162         """
163         Random forest cross validation
164 
165         Parameters
166         ----------
167         n_estimators : numpy array, optional
168             number of decision trees. The default is np.arange(100, 250, 1).
169         max_depth : numpy array, optional
170             max depth for each decision tree. The default is np.arange(2, 5, 1).
171         cv : int, optional
172             number of folds. The default is 10.
173         plot : bool, optional
174             should the cross validation scores the plotted by tuning parameter. The default is True.
175 
176         Returns
177         -------
178         res : dictionary
179             resutls of the cross validation by the tuning parameters.
180 
181         """
```

```
182         # prepare to store data
183         res = dict()
184         i = 0
185         scores = np.zeros(len(n_estimators)*len(max_depth))
186         estimators_scores = np.zeros(len(n_estimators))
187 
188         # for all parameters in range
189         for k, n in enumerate(n_estimators):
190             depth_scores = np.zeros(len(max_depth))
191             for l, d in enumerate(max_depth):
192                 print(i)
193                 n = int(n)
194 
195                 RF = RandomForestRegressor(n_estimators=n,max_depth=d)
196 
197                 # perform cross validation
198                 score = cross_val_score(RF, self.pred, self.response, cv = cv, scoring='
                        neg_mean_squared_error').mean()
199 
200                 # store data
201                 scores[i] = score
202                 depth_scores[l] = score
203                 i = i+1
204                 res['n_estimators: /{}/, max_depth: /{}/'.format(n,d)] = score
205             estimators_scores[k] = depth_scores.mean()
206 
207         # plot cross validation result
208         if plot == True:
209             plt.plot(max_depth, depth_scores)
210             plt.xlabel('max depth')
211             plt.ylabel('scores')
212             plt.title('CV scores RF')
213             plt.show()
214 
215             plt.plot(n_estimators, estimators_scores)
216             plt.xlabel('n_estimators')
217             plt.ylabel('scores')
218             plt.title('CV scores RF')
219             plt.show()
220         return res
221 
222 
223     def KNN_cross_val(self, N = np.arange(100, 250, 1), cv = 10, plot = True):
224         """
225         KNN cross validation
226 
227         Parameters
228         ----------
229         N : numpy array, optional
230             number of neigbors considered. The default is np.arange(100, 250, 1).
231         cv : int, optional
232             number of folds. The default is 10.
233         plot : bool, optional
234             should the cross validation scores the plotted by tuning parameter. The default is True.
235 
236         Returns
237         -------
```

```
238             res : dictionary
239                 resutls of the cross validation by the tuning parameters.
240
241             """
242             # prepare to store data
243             res = dict()
244             i = 0
245             scores = np.zeros(len(N))
246
247             # for all parameters in range
248             for n in N:
249                 print(i)
250                 n = int(n)
251                 KNN = KNeighborsRegressor(n_neighbors=n)
252
253                 # perform cross validation
254                 score = cross_val_score(KNN, self.pred, self.response, cv = cv, scoring='
                        neg_mean_squared_error').mean()
255
256                 # store data
257                 scores[i] = score
258                 i = i+1
259                 res['k: /{}/'.format(n)] = score
260
261             # plot cross validation result
262             if plot == True:
263                 plt.plot(N, scores)
264                 plt.xlabel('k')
265                 plt.ylabel('scores')
266                 plt.title('CV scores KNN')
267                 plt.show()
268             return res
269
270         def lasso_cross_val(self, al_array = np.arange(0.01, 10, 0.01), cv = 10, plot = True):
271             """
272             Lasso-regression cross validation
273
274             Parameters
275             ----------
276             al_array : numpy array, optional
277                 penalty coefficients considered. The default is np.arange(0.01, 10, 0.01).
278             cv : int, optional
279                 number of folds. The default is 10.
280             plot : bool, optional
281                 should the cross validation scores the plotted by tuning parameter. The default is True.
282
283             Returns
284             -------
285             res : dictionary
286                 resutls of the cross validation by the tuning parameters.
287
288             """
289             # prepare to store data
290             res = dict()
291             i = 0
292             scores = np.zeros(len(al_array))
293
```

```
294             # for all parameters in range
295             for alpha in al_array:
296                 print(i)
297                 lasso = linear_model.Lasso(alpha = alpha)
298 
299                 # perform cross validation
300                 score = cross_val_score(lasso, self.pred, self.response, cv = cv, scoring='
                        neg_mean_squared_error').mean()
301 
302                 # store data
303                 scores[i] = score
304                 i = i+1
305                 res['alpha: /{}/'.format(alpha)] = score
306 
307 
308             # plot cross validation result
309             if plot == True:
310                 plt.plot(al_array, scores)
311                 plt.xlabel('alpha')
312                 plt.ylabel('scores')
313                 plt.title('CV scores lasso')
314                 plt.show()
315             return res
316 
317         def n_net_cross_val(self, N = np.arange(10, 500, 100),layers_list = [1,2, 3], act_list = ['sigmoid'],
                cv = 10,epochs=10, plot = True):
318             """
319             Neural Network cross validation
320 
321             Parameters
322             ----------
323             N : numpy array, optional
324                 range of number of neurans in every layer considered. The default is np.arange(10, 500, 100).
325             layers_list : list, optional
326                 range of number of hidden layers considered. The default is [1,2, 3].
327             act_list : list, optional
328                 activation functions considered. The default is ['sigmoid'].
329             cv : int, optional
330                 number of folds. The default is 10.
331             epochs : int, optional
332                 numnber of epochs or cycles of training. The default is 10.
333             plot : bool, optional
334                 should the cross validation scores the plotted by tuning parameter.
335 
336             Returns
337             -------
338             res : dictionary
339                 resutls of the cross validation by the tuning parameters.
340 
341             """
342 
343             # prepare to store data
344             res = dict()
345 
346 
347             kf = KFold(n_splits=cv)
348 
```

```
349         # for all parameters in range
350         for act in act_list:
351             layers_scores = np.zeros(len(layers_list))
352 
353             for l, layers in enumerate(layers_list):
354                 N_scores = np.zeros(len(N))
355 
356                 for m, n in enumerate(N):
357                     i = 0
358                     scores = np.zeros(cv)
359 
360                     # perform cross validation
361                     # split the data and loop for each split
362                     for train, test in kf.split(self.pred, self.response):
363 
364                         self.n_net_init(self.pred[train], self.response[train],n = n, layers = layers,
                                activation = act,epochs=epochs)
365 
366                         # store data
367                         scores[i] = self.model.evaluate(self.pred[test], self.response[test], verbose=0)
                                [0]
368                         i+=1
369 
370                     # store data
371                     score = scores.mean()
372                     N_scores[m] = score
373                     res['layers: /{}/, activation: /{}/, neurons: /{}/'.format(layers,act, n)] = scores.
                            mean()
374                     tf.keras.backend.clear_session()
375                 layers_scores[l] = N_scores.mean()
376 
377             # plot cross validation result
378             if plot == True:
379                 plt.plot(N, N_scores)
380                 plt.xlabel('neurons')
381                 plt.ylabel('scores')
382                 plt.title('CV scores N-net')
383                 plt.show()
384 
385                 plt.plot(np.array(layers_list), layers_scores)
386                 plt.xlabel('layers')
387                 plt.ylabel('scores')
388                 plt.title('CV scores N-net')
389                 plt.show()
390 
391         return res
392 
393 
394     def predict_syn_grid(self,grid_list,grid_names, win, method = 'KNN', grid_ext = 'case1', geo_int = '
            none',
395                          w_start = [267, 178], w_end_top = [0, 178], w_end_base = [0, 273],
396                          max_TWT = -618, min_TWT = -1162,horizons_list = None,
397                          mean_arr = None, std_arr = None):
398         """
399         Use Trained ML model to predict the target values of the synthetic cross-section (grid)
400 
401         Parameters
```

```
402             ----------
403             grid_list : list
404                 list of 2d arrays representing the cross-sections.
405             grid_names : list
406                 list of grid names, for example ['imp', 'por'].
407             win : int
408                 window size for mean and median rolling window and window selection.
409             method : str, optional
410                 machine learning method. The default is 'KNN'.
411             grid_ext : str, optional
412                 predictor extraction preset. The default is 'case1'.
413             geo_int : str, optional
414                 Should the depositional time be implemented and if so how?. The default is 'none'.
415             w_start : list, optional
416                 the starting position of the wedge: the pinch point. The default is [267, 178].
417             w_end_top : list, optional
418                 the end of the top surface of the wedge. The default is [0, 178].
419             w_end_base : list, optional
420                 the end of the base surface of the wedge. The default is [0, 273].
421             max_TWT : int, optional
422                 maximum value of the TWT. The default is -618.
423             min_TWT : int, optional
424                 minimum value of the TWT. The default is -1162.
425             horizons_list : list or None, optional
426                 list of numpy arrays that describe where the horizons intersect with the cross-section. The
                        default is None.
427             mean_arr : float, optional
428                 the mean value used in the standardization. The default is None.
429             std_arr : float, optional
430                 the standard deviation used in the standardization. The default is None.
431
432             Returns
433             -------
434             pred_map : numpy array
435                 result of the prediction.
436
437             """
438
439             v, h = np.shape(self.grid)
440             # reserve memory for result
441             pred_map = np.zeros((v, h-win*2))
442             # plt.imshow(self.grid.T)
443             # plt.show()
444
445             # for every trace
446             for col in range(len(self.grid)):
447                 print('col = ', col)
448                 # c = self.grid[col]
449                 # data = pd.DataFrame(c, columns = ['AI'])
450
451                 ################### Load well ###################################
452                 log1 = load_well(file_name = 'data_2d_wedge_F3\F03_2_por_eff.xlsx') # arbitrary file name
453                 data = log1.from_synthetic(grid_list, grid_names, col = col)
454                 data.drop('Por', axis=1, inplace=True)
455                 ###################################
456
457                 # predictor extraction
```

```
458             from classes.predictor_ext import predictor_ext
459             new_pred = predictor_ext(data)
460             name = 'imp'
461             # new_pred.add_well_loc(well_loc=col)
462             ######################################################
463             if grid_ext == 'case1':
464                 new_pred.roll_mean(data_name = name,win = win)
465                 new_pred.roll_median(data_name = name,win = win)
466                 new_pred.win_select(data_name = name,win = win)
467 
468             elif grid_ext == 'case2':
469                 new_pred.roll_mean(data_name = name,win = win)
470                 new_pred.roll_median(data_name = name,win = win)
471             else:
472                 pass
473             ######################################################
474 
475             if geo_int == 'wedge':
476                 new_pred.construct_timelines_wedge_df(self.grid,  w_start = w_start, w_end_top=w_end_top,
                        w_end_base=w_end_base,
477                                                       col = col, win = win)
478             elif geo_int == 'from horizons':
479                 new_pred.construct_timelines_from_horizons(self.grid, horizons_list, max_TWT = max_TWT,
                        min_TWT = min_TWT)
480                 new_pred.depotime_well(well_loc = col)
481 
482             ######################################################
483 
484 
485             new_pred.remove_outside_window(win = win)
486             self.new_pred = new_pred
487 
488             pred = new_pred.data.to_numpy()
489 
490             # standardize predictor based on previous standardization
491             if type(mean_arr)==np.ndarray and type(std_arr) ==np.ndarray:
492                 pred, dummy1, dummy2 = new_pred.stand_pred(pred,mean_list = mean_arr, std_list = std_arr)
493             else:
494                 pass
495 
496             # predict
497             prediction = self.model.predict(pred)
498             ######################################################
499             if method == 'RF' or method == 'lasso':
500                 pred_map[col] = prediction
501             else:
502                 pred_map[col] = prediction[:,0]
503             ######################################################
504         return pred_map
505 
506     def predict_grid(self, grid_list,grid_names, win, method = 'KNN', grid_ext = 'case1', geo_int = 'none'
        ,
507                      horizons_list = None, max_TWT = -618, min_TWT = -1162,
508                      mean_arr = None, std_arr = None):
509         """
510         Use Trained ML model to predict the target values of the cross-section (grid)
511 
```

```
512             Parameters
513             ----------
514             grid_list : list
515                 list of 2d arrays representing the cross-sections.
516             grid_names : list
517                 list of grid names, for example ['imp', 'por'].
518             win : int
519                 window size for mean and median rolling window and window selection.
520             method : str, optional
521                 machine learning method. The default is 'KNN'.
522             grid_ext : str, optional
523                 predictor extraction preset. The default is 'case1'.
524             geo_int : str, optional
525                 Should the depositional time be implemented and if so how?. The default is 'none'.
526             horizons_list : list or None, optional
527                 list of numpy arrays that describe where the horizons intersect with the cross-section. The
                        default is None.
528             max_TWT : int, optional
529                 maximum value of the TWT. The default is -618.
530             min_TWT : int, optional
531                 minimum value of the TWT. The default is -1162.
532             mean_arr : float, optional
533                 the mean value used in the standardization. The default is None.
534             std_arr : float, optional
535                 the standard deviation used in the standardization. The default is None.
536
537             Returns
538             -------
539             pred_map : numpy array
540                 result of the prediction.
541
542             """
543
544
545             v, h = np.shape(self.grid)
546             # reserve memory for result
547             pred_map = np.zeros((v, h-win*2))
548             # plt.imshow(self.grid.T)
549             # plt.show()
550
551             # for every trace
552             for col in range(len(self.grid)):
553                 print('col = ', col)
554                 # c = self.grid[col]
555                 # data = pd.DataFrame(c, columns = ['AI'])
556
557                 ################### Load well ###################################
558                 log1 = load_well(file_name = 'data_2d_wedge_F3\F03_2_por_eff.xlsx')
559                 data = log1.from_synthetic(grid_list, grid_names, col = col)
560                 # data.drop('Por', axis=1, inplace=True)
561                 ###################################
562
563                 # predictor extraction
564                 from classes.predictor_ext import predictor_ext
565                 new_pred = predictor_ext(data)
566                 name = 'imp'
567                 # new_pred.add_well_loc(well_loc=col)
```

```
568             ######################################################
569             if grid_ext == 'case1':
570                 new_pred.roll_mean(data_name = name,win = win)
571                 new_pred.roll_median(data_name = name,win = win)
572                 new_pred.win_select(data_name = name,win = win)
573 
574             elif grid_ext == 'case2':
575                 new_pred.roll_mean(data_name = name,win = win)
576                 new_pred.roll_median(data_name = name,win = win)
577             else:
578                 pass
579             ######################################################
580             if geo_int == 'from horizons':
581                 new_pred.construct_timelines_from_horizons(self.grid, horizons_list, standardizing = False
                        , max_TWT = max_TWT, min_TWT = min_TWT)
582                 new_pred.depotime_well(well_loc = col)
583 
584             ######################################################
585 
586 
587             new_pred.remove_outside_window(win = win)
588             self.new_pred = new_pred
589 
590             pred = new_pred.data.to_numpy()
591 
592             # standardize predictor based on previous standardization
593             if type(mean_arr)==np.ndarray and type(std_arr) ==np.ndarray:
594                 pred, dummy1, dummy2 = new_pred.stand_pred(pred,mean_list = mean_arr, std_list = std_arr)
595             else:
596                 pass
597 
598             # predict
599             prediction = self.model.predict(pred)
600             ######################################################
601             if method == 'RF' or method == 'lasso':
602                 pred_map[col] = prediction
603             else:
604                 pred_map[col] = prediction[:,0]
605             ######################################################
606         return pred_map
```

C.8 plot_results.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Jan 10 11:22:11 2022
4 
5  @author: Eier
6  """
7  import matplotlib.pyplot as plt
8  import numpy as np
9  from classes.usefull_functions import usefull_functions
10 class plot_results:
11 
12     def __init__(self):
13         self.sel = 'placeholder'
14 
```

```
15      def plot_generic(grid, vmin, vmax, title, xlab, ylab, extent,cmap = 'jet',
16                       vcut=False, hcut = False):
17          """
18          Plot one cross-section
19
20          Parameters
21          ----------
22          grid : numpy array
23              2D array of the cross-section.
24          vmin : float
25              minimum value of the plotted array.
26          vmax : float
27              maximum value of the plotted array.
28          title : str
29              title of the plot.
30          xlab : str
31              x label.
32          ylab : str
33              y label.
34          extent : list
35              axis extent as traces and TWT.
36          cmap : str, optional
37              colormap. The default is 'jet'.
38          vcut : list, optional
39              vertical slice of the array (TWT). The default is False.
40          hcut : list, optional
41              horizontal slice of the array (traces). The default is False.
42
43          Returns
44          -------
45          None.
46
47          """
48
49          if vcut == False:
50              pass
51          else:
52              TWT = np.linspace(extent[3], extent[2], len(grid[1, :]))
53              extent[3] = TWT[vcut[0]:vcut[1]][0]
54              extent[2] = TWT[vcut[0]:vcut[1]][-1]
55
56              grid = grid[:, vcut[0]:vcut[1]]
57
58          if hcut == False:
59              pass
60          else:
61              grid = grid[hcut[0]:hcut[1],:]
62
63
64          plt.imshow(grid.T, cmap=cmap, aspect='auto',vmin=vmin, vmax=vmax, extent=extent)
65          plt.xlabel(xlab)
66          plt.ylabel(ylab)
67          plt.title(title)
68          plt.colorbar()
69          plt.show()
70
71      def grids_comp(self, grid1, grid2, vcut=False, hcut = False, cmap = 'jet',
```

```
72                  titles = ['True','Estimate', 'MSE'], file_name = False,
73                  extent=[-1,1,-1,1], ylab = 'TWT', vmin =0.2, vmax = 0.4):
74          """
75          plot two 2D arrays and the absolute difference between them.
76          The numpy arrays used in the plotting are saved.
77
78          Parameters
79          ----------
80          grid1 : numpy array
81              array to be plotted and compared against grid2.
82          grid2 : numpy array
83              array to be plotted and compared against grid1.
84          vcut : list, optional
85              vertical slice of the array (TWT). The default is False.
86          hcut : list, optional
87              horizontal slice of the array (traces). The default is False.
88          cmap : str, optional
89              colormap. The default is 'jet'.
90          titles : list, optional
91              list of titles for the different plots. The default is ['True','Estimate', 'MSE'].
92          file_name : str, optional
93              file path for the results to be saved to. The default is False.
94          extent : list, optional
95              axis extent as traces and TWT. The default is [-1,1,-1,1].
96          ylab : str, optional
97              y label. The default is 'TWT'.
98          vmin : float, optional
99              minimum value of the plotted array. The default is 0.2.
100         vmax : float, optional
101             maximum value of the plotted array. The default is 0.4.
102
103         Returns
104         -------
105         None.
106
107         """
108
109         if vcut == False:
110             pass
111         else:
112             TWT = np.linspace(extent[3], extent[2], len(grid1[1, :]))
113             extent[3] = TWT[vcut[0]:vcut[1]][0]
114             extent[2] = TWT[vcut[0]:vcut[1]][-1]
115
116             grid1 = grid1[:, vcut[0]:vcut[1]]
117             grid2 = grid2[:, vcut[0]:vcut[1]]
118
119         if hcut == False:
120             pass
121         else:
122             grid1 = grid1[hcut[0]:hcut[1],:]
123             grid2 = grid2[hcut[0]:hcut[1],:]
124         plt.subplot(211)
125
126         plt.imshow(grid1.T, cmap=cmap, aspect='auto',vmin=vmin, vmax=vmax, extent=extent)
127         plt.xlabel('Trace')
128         plt.ylabel(ylab)
```

```
129         plt.title(titles[0])
130
131         plt.subplot(212)
132         plt.imshow(grid2.T, cmap=cmap, aspect='auto',vmin=vmin, vmax=vmax, extent=extent)
133         plt.xlabel('Trace')
134         plt.ylabel(ylab)
135         cax = plt.axes([0.95, 0.1, 0.075, 0.8]) # https://matplotlib.org/stable/gallery/
                subplots_axes_and_figures/subplots_adjust.html#sphx-glr-gallery-subplots-axes-and-figures-
                subplots-adjust-py
136         plt.colorbar(cax=cax)
137         if file_name != False:
138             np.save(file_name+' grid1', grid1)
139             np.save(file_name+' grid2', grid2)
140             plt.savefig(file_name+'.png', bbox_inches='tight')
141         plt.show()
142         plt.subplot(111)
143         uf = usefull_functions()
144         sim = uf.my_map2(grid1, grid2, uf.diff)
145         plt.imshow(sim.T, cmap='nipy_spectral', aspect='auto', vmin = 0, vmax = 0.15, extent=extent)
146         plt.xlabel('Trace')
147         plt.ylabel(ylab)
148         plt.title(titles[2])
149         cax = plt.axes([0.95, 0.1, 0.075, 0.8])
150         plt.colorbar(cax=cax)
151         if file_name != False:
152             np.save(file_name+' difference', sim)
153             plt.savefig(file_name+' difference plot.png', bbox_inches='tight')
154         plt.show()
```

C.9 usefull_functions.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Wed Jan 19 15:53:33 2022
4
5   @author: Eier
6   """
7   import pickle
8   import numpy as np
9   import matplotlib.pyplot as plt
10  import scipy.signal
11
12  class usefull_functions:
13
14      def __init__(self):
15          pass
16
17      def save_cv(self, path = 'CV.pkl', method = 'KNN', dictionary = ''):
18
19          # save the cv results
20          a_file = open(path, "wb")
21          pickle.dump(dictionary, a_file)
22          a_file.close()
23
24          # https://www.tutorialsteacher.com/articles/sort-dict-by-value-in-python
25          marklist=sorted((value, key) for (key,value) in dictionary.items())
26          sortdict=dict([(k,v) for v,k in marklist])
```

```
27            self.cv_res = sortdict
28
29            # get the best results
30            if method == 'Neural Net':
31                best = list(sortdict)[0]
32            else:
33                best = list(sortdict)[-1]
34
35            return best, sortdict[best]
36
37
38        def nearest(self, number, arr):
39            """
40            Gets index of number nearest the "number"
41
42            Parameters
43            ----------
44            number : TYPE
45                DESCRIPTION.
46            arr : TYPE
47                DESCRIPTION.
48
49            Returns
50            -------
51            TYPE
52                DESCRIPTION.
53
54            """
55            search = abs(arr-number)
56            m = search.min()
57            return np.where(search == m)[0]
58
59        def diff(self, v1, v2):
60            return abs(v1-v2)
61
62        def my_map(self, grid, func):
63            """
64            I find the existing map function to be limited, this is an alternitive
65            that applies a function to all elements in a grid/matrix.
66            """
67            n,k = np.shape(grid)
68            l = list(grid.flatten())
69            m = list(map(func, l))
70            a = np.array(m)
71            return a.reshape(n, k)
72
73        def my_map2(self, grid1, grid2, func, cross = True):
74            """
75            I find the existing map function to be limited, this is an alternitive
76            that applies a function to all elements in a grid/matrix.
77            """
78            # if cross == True:
79            #     mean1 = np.mean(grid1)
80            #     std1 = np.std(grid1)
81            #     mean2 = np.mean(grid2)
82            #     std2 = np.std(grid2)
83
```

```
84          n,k = np.shape(grid1)
85          l1 = list(grid1.flatten())
86          l2 = list(grid2.flatten())
87          m = list(map(func, l1, l2))
88          a = np.array(m)
89          return a.reshape(n, k)
```

C.10 section_horizon_coord.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Sat Feb 26 20:45:55 2022
4
5   @author: Eier
6   """
7
8   # finds the locations where horizons intersect the cross-section
9
10
11  import segyio
12  import pandas as pd
13  import numpy as np
14  #fault seis.segy
15  #wedge seis.segy'
16  # file_seis = 'data\case Oseberg\case oseberg LFM imp.segy'
17  # hor_file = 'data\horizons\Drake 100ms below'
18  # hor_file = 'data\horizons\Drake LFM'
19  # hor_file = 'data\horizons\Brent LFM'
20  # hor_file = 'data\horizons\Shetland LFM'
21  # hor_file = 'data\horizons\Shetland 50 ms above'
22
23  # cross section
24  file_imp = 'data\case F3\Seis Inv depth Random line [2D Converted].segy'
25
26  f = segyio.open(file_imp, ignore_geometry=True)
27  #'data\case F3\F3-Horizon-FS8 (Z)',
28
29  # horizons
30  hor = ['data\case F3\F3-Horizon-MFS4 (Z)', 'data\case F3\F3-Horizon-Truncation (Z)']
31
32  # for each horizons
33  for hor_file in hor:
34
35      # get TWT of section
36      sec = segyio.tools.metadata(f)
37      TWT = -sec.samples
38      top_twt = TWT.max()
39      base_twt = TWT.min()
40      l = len(f.header)
41
42      s = len(f.trace[1])
43
44      # inter = (top_twt-base_twt)/161
45
46      inter = (top_twt-base_twt)/s
47
48      resx = np.zeros((l, s))
```

```
49      resy = np.zeros((l, s))
50
51
52      for trace in range(l):
53          scale = f.header[trace][segyio.TraceField.SourceGroupScalar]
54          # get the coordinates of the cross-section
55          if scale <0:
56
57              x = f.header[trace][segyio.TraceField.SourceX]/abs(scale)
58              y = f.header[trace][segyio.TraceField.SourceY]/abs(scale)
59
60          else:
61              x = f.header[trace][segyio.TraceField.SourceX]*abs(scale)
62              y = f.header[trace][segyio.TraceField.SourceY]*abs(scale)
63
64          resx[trace] = x
65          resy[trace] = y
66
67          samples = f.trace[trace]
68
69
70
71      ##############################
72
73      TWT_array = np.zeros(len(resx))
74
75      # load horizon
76      hor = pd.read_csv(hor_file, header = None, sep = ' ')
77      z = np.zeros(len(hor))
78
79      for coo in range(len(resx)):
80          print(coo)
81          for i in range(len(hor)):
82              # horizon point
83              point_a = np.array([hor[0][i], hor[1][i]])
84
85              # cross-section point
86              point_b = np.array([resx[coo, 0], resy[coo, 0]])
87
88              # distance between points
89              dist = np.sqrt((point_a[0]-point_b[0])*(point_a[0]-point_b[0])+
90                             (point_a[1]-point_b[1])*(point_a[1]-point_b[1]))
91
92              z[i] = dist
93
94          # get smallest distance
95          in_min = np.argmin(z)
96
97          TWT = hor.iloc[in_min][2]
98          TWT_array[coo] = -TWT
99
100     np.save(hor_file+'TWT',TWT_array)
101 f.close()
102
103 # np.load(hor_file+'TWT'+'.npz')
```

C.11 make_new_wells.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr  3 20:42:06 2022

@author: Eier
"""

# the logs of a well may not match
# this script matches the logs

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

well_files = ['data\case F3\F02_1.xlsx', 'data\case F3\F03_2.xlsx']

new_well_files = ['data\case F3\F02_1_new.txt', 'data\case F3\F03_2_new.txt']

for k,file in enumerate(well_files):
    df = pd.read_excel(file)
    df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

    df_por = pd.DataFrame()
    df_por['MD'] = df['MD']
    df_por['Por.Eff.'] = df['Por.Eff.']
    df_por = df_por.dropna()

    df_imp = pd.DataFrame()
    df_imp['MD'] = df['MD.1']
    df_imp['P-imp.'] = df['P-imp.']
    df_imp = df_imp.dropna()

    por_MD_lim = [df_por['MD'].min(), df_por['MD'].max()]

    df_imp = df_imp[df_imp['MD']>por_MD_lim[0]]
    df_imp = df_imp[df_imp['MD']<por_MD_lim[1]]


    df_imp = df_imp.reset_index()
    df_por = df_por.reset_index()

    df_well = pd.DataFrame()

    def nearest(number, arr):
        """
        Gets index of number nearest the "number"

        Parameters
        ----------
        number : TYPE
            DESCRIPTION.
        arr : TYPE
            DESCRIPTION.

        Returns
```

```
56              --------
57              TYPE
58                  DESCRIPTION.
59
60              """
61              search = abs(arr-number)
62              m = search.min()
63              return np.where(search == m)[0]
64
65          md_list = []
66          imp_list = []
67          por_list = []
68
69
70          for n, i in enumerate(df_imp['MD']):
71
72              arr = df_por['MD'].to_numpy()
73
74              ind = nearest(i, arr)[0]
75
76              md = df_imp.iloc[n]['MD']
77              md_list.append(md)
78              imp = df_imp.iloc[n]['P-imp.']
79              imp_list.append(imp)
80              por = df_por.iloc[ind]['Por.Eff.']
81              por_list.append(por)
82
83          df_well['MD'] = md_list
84          df_well['imp'] = imp_list
85          df_well['por'] = por_list
86
87          plt.scatter(df_well['imp'], df_well['por'])
88
89          df_well.to_csv(new_well_files[k], index = False)
```

C.12 upscale.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Thu Apr 14 13:18:12 2022
4
5   @author: Eier
6   """
7
8   # upscales the well logs based on an existing already upscaled well log
9
10  import pandas as pd
11  import numpy as np
12  import matplotlib.pyplot as plt
13
14  def nearest(number, arr):
15      """
16      Gets index of number nearest the "number"
17
18      Parameters
19      ----------
20      number : float
```

```
21              the value to find in arr.
22          arr : numpy array
23              array of values.
24
25          Returns
26          -------
27          int
28              the index where arr is closest to number.
29
30          """
31          search = abs(arr-number)
32          m = search.min()
33          return np.where(search == m)[0]
34
35      # get upscaled well log
36      upscale = 'data\case F3\AI resampled.xlsx'
37
38      up_df = pd.read_excel(upscale)
39      up_df = up_df.loc[:, ~up_df.columns.str.contains('^Unnamed')]
40
41      # well logs to be upscaled
42      well_files = ['data\case F3\F02_1_new', 'data\case F3\F03_2_new']
43
44      # for each well
45      for k,file in enumerate(well_files):
46          df = pd.read_csv(file+'.txt')
47          df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
48
49          df_por = pd.DataFrame()
50          df_por['MD'] = df['MD']
51          df_por['por'] = df['por']
52          df_por = df_por.dropna()
53
54          df_imp = pd.DataFrame()
55          df_imp['MD'] = df['MD']
56          df_imp['imp'] = df['imp']
57          df_imp = df_imp.dropna()
58
59          plt.scatter(df['imp'], df['por'])
60          plt.show()
61
62          por_MD_lim = [df_por['MD'].min(), df_por['MD'].max()]
63
64          df_imp = df_imp[df_imp['MD']>por_MD_lim[0]]
65          df_imp = df_imp[df_imp['MD']<por_MD_lim[1]]
66
67
68          df_imp = df_imp.reset_index()
69          df_por = df_por.reset_index()
70
71          df_well = pd.DataFrame()
72
73          md_list = []
74          imp_list = []
75          por_list = []
76
77          # for every point in the upscaled well-log MD
```

```
78      for n, i in enumerate(up_df['MD']):
79
80          arr_por = df_por['MD'].to_numpy()
81          arr_imp = df_imp['MD'].to_numpy()
82
83          # find the closest point in the real well-logs and save
84          ind_por = nearest(i, arr_por)[0]
85          ind_imp = nearest(i, arr_imp)[0]
86
87          md = up_df.iloc[n]['MD']
88          md_list.append(md)
89
90          imp = df_imp.iloc[ind_imp]['imp']
91          imp_list.append(imp)
92
93          por = df_por.iloc[ind_por]['por']
94          por_list.append(por)
95
96      # store the new upscaled well logs in a dataframe
97      df_well['MD'] = md_list
98      df_well['imp'] = imp_list
99      df_well['por'] = por_list
100
101     plt.scatter(df_well['imp'], df_well['por'])
102     plt.show()
103     # df_well.to_csv(file+'_upsacale.txt', index = False)
```

C.13 well_paths_horizon.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Fri Mar 11 16:44:24 2022
4
5   @author: Eier
6   """
7   # implements the horizon locations to the well.
8   # the well path is given with coordinates (interpolation is requiered)
9   # the horizon point set is given with coordinates
10
11  import pandas as pd
12  import numpy as np
13  import matplotlib.pyplot as plt
14
15  def TWT_from_MD(MD):
16      z = np.load('TDR\_fit TDR result.npy')
17      p = np.poly1d(z)
18
19      return p(MD)
20
21
22  file = 'data\case F3\F03_2 well path'
23  # file = 'data\case Oseberg\w30_9_J_13_well_path'
24  hor_file1 = 'data\case F3\F3-Horizon-FS8 (Z)'
25  hor_file2 = 'data\case F3\F3-Horizon-MFS4 (Z)'
26  hor_file3 = 'data\case F3\F3-Horizon-Truncation (Z)'
27
28  hor_file_list = [hor_file1, hor_file2, hor_file3]
```

```
29
30  ##################################################################################################
31  df = pd.read_csv(file, skiprows=17, sep = ' ')
32  df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
33  df = df.dropna()
34  ############## from https://stackoverflow.com/questions/66466080/python-pandas-insert-empty-rows-after-
        each-row ####################
35  n = 10
36  new_index = pd.RangeIndex(len(df)*(n+1))
37  new_df = pd.DataFrame(np.nan, index=new_index, columns=df.columns)
38  ids = np.arange(len(df))*(n+1)
39  new_df.loc[ids] = df.values
40  df = new_df
41  ############################################## interpolate
        ##############################################################################################################

42  df = df.interpolate()
43  df = df.reset_index()
44  ################################### MD to Z in well path
        ###########################################################
45  z = np.polyfit(df['MD'], df['Z'], 12)
46  np.save(file[:18]+'MD to Z '+file[18:], z)
47  ##################################################################################################
48
49  hor_location_5S = pd.DataFrame()
50
51  # for each horizon
52  for hor_file in hor_file_list:
53      print(hor_file)
54
55      #
            ##########################################################################################################################

56      hor = pd.read_csv(hor_file, header = None, sep = ' ')
57      hor[2] = -hor[2]
58
59      ################################################## reduce the search area for well location
            ##########################################################################################################

60      upperX = df['X'].max()+10
61      upperY = df['Y'].max()+10
62
63      lowerX = df['X'].min()-10
64      lowerY = df['Y'].min()-10
65
66      hor = hor.drop(hor[(hor[0] < lowerX)].index)
67      hor = hor.drop(hor[(hor[0] > upperX)].index)
68
69      hor = hor.drop(hor[(hor[1] < lowerY)].index)
70      hor = hor.drop(hor[(hor[1] > upperY)].index)
71      hor = hor.reset_index()
72      #
            ##########################################################################################################################

73      # df['TWT'] = TWT_from_MD(df['MD'])
74
75
```

```
76        # reserve memory
77        z = np.zeros(len(hor)*len(df))
78        twt = np.zeros(len(hor)*len(df))
79        n = 0
80
81        # follow the well path interpolated points
82        for coo in range(len(df)):
83            print(coo)
84            for i in range(len(hor)):
85                # horizon point
86                point_a = np.array([hor[0][i], hor[1][i], hor[2][i]])
87
88                # well point
89                point_b = np.array([df['X'][coo], df['Y'][coo], df['Z'][coo]])
90
91                # distance between points
92                dist = np.sqrt((point_a[0]-point_b[0])*(point_a[0]-point_b[0])+
93                               (point_a[1]-point_b[1])*(point_a[1]-point_b[1])+
94                               (point_a[2]-point_b[2])*(point_a[2]-point_b[2]))
95                z[n] = dist
96                twt[n] = hor[2][i]
97                n+=1
98
99        # find the two closest points
100       in_min = np.argmin(z)
101
102       # record the horizon location in the well path
103       TWT_loc = twt[in_min]
104       hor_location_5S[hor_file[14:]] = [TWT_loc]
105
106   # compile results
107   hor_location_5S.to_csv(file+' horizon loc'+'.txt', index = False, encoding = 'utf-8')
108
109   # plt.plot(df['X'], df['Y'])
110   # plt.show()
111   # plt.plot(df['X'], df['Z'])
112   # plt.show()
113   # plt.plot(df['Y'], df['Z'])
114   # plt.show()
```