FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation:
Petroleum Engineering/ Reservoir Engineering

The spring semester, 2022

Open / ~~Confidential~~

Author: Ove A. Kvandal

.....
(signature author)

Course coordinator: Øystein Arild

Supervisor(s): Carita Augustsson, Pål Østebø Andersen

Thesis title: Automatic evaluation of wavelength spectra from quartz by means of machine learning

Credits (ECTS): 30

Keywords: Quartz, Cathodoluminescence, Machine Learning, Classification, Neural Network, kNN algorithm, Random Forest algorithm

Pages: 69

+ appendix: 71

Sandane, 15.06.2022
date/year

Automatic evaluation of wavelength spectra from quartz by means of machine learning

Ove Andre Kvandal
University of Stavanger

Petroleum Engineering - Reservoir Engineering
A thesis presented for the degree of
Master of Science

Stavanger 2022

I dedicate this thesis to my family.

To be human, is not a fact, but a task.

— Søren Kierkegaard

Acknowledgements

I would like to express my sincere gratitude to my supervisors Carita Augustsson and Pål Østebø Andersen for their valuable guidance and support throughout this project.

Abstract

I have for this thesis worked on automating the steps in evaluating wavelength spectra from quartz. To do this I have applied different machine learning algorithms suitable to the automation requirement at each step, and they were implemented using Python.

A neural network is used to fit a curve to cathodoluminescence spectra from quartz. From this curve are feature values extracted, which are then to be used by a machine learning classification algorithm to predict which of three defined groups a quartz sample belong to.

Two machine learning classification algorithms have been evaluated: The kNN algorithm and the Random Forest algorithm. These algorithms were trained on multiple feature subsets derived from two different datasets. The difference between the two datasets is that one, the reduced dataset, has had cathodoluminescence spectra influenced, primarily by feldspar, removed.

The classification algorithm whose model achieved the highest accuracy was the kNN algorithm with 84%. It achieved this when trained on a feature subset derived from the reduced dataset where only features representing intensity were included. The best performing model by the Random Forest algorithm achieved an accuracy of 81%.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Adam ..... Adaptive Moment Estimation

ANN ..... Artificial Neural Network

BG ..... Background Noise

CL ..... Cathodoluminescence

CART ..... Classification And Regression Trees

ELU ..... Exponential Linear Unit

FN ..... False Negative

FP ..... False Positive

IG ..... Information Gain

kNN ..... K-Nearest Neighbours

ML ..... Machine Learning

MPa ..... Mega Pascal

MSE ..... Mean Squared Error

nm ..... nanometer

NN ..... Neural Network

PC ..... Principal Component

PCA ..... Principal Component Analysis

POC ..... Proof of Concept

ReLU ..... Rectified Linear Unit

RMSE ..... Root Mean Squared Propagation

SNN ..... Simulated Neural Network

TN ..... True Negative

TP ..... True Positive

1 | Introduction

Cathodoluminescence (CL) of quartz can be used as an indicator for parent rock of sand. This is due to lattice defects in the quartz crystal structure caused either pre or post crystallisation, often resulting in unique CL spectra signatures corresponding to specific defects.

Analyzing CL spectra can be a time consuming process, especially when the number of spectra needed to be analyzed number in the hundreds to thousands (for example Götze *et al*. (2001); Boggs *et al.* (2002); Augustsson & Reker (2012)).

1.1 Aim and Objective

The aim of this thesis is to examine if machine learning (ML) can be used to automate the process of classify quartz on the basis of raw CL spectra data. The workflow of the steps making up that process are depicted in figure 1.1, and these steps can further be grouped into two sub–processes. In the first sub–process, steps are taken to be able to extracted feature values from the CL spectra data. This include first filtering the CL spectra data and then fitting a curve to it using a neural network.
In the second sub–process, steps are taken to be able to classify quartz samples on the feature values that have extracted in the first sub–process. This include feature selection an testing multiple models and model configurations to find the model with the best performance in terms of accuracy.

Common for the entire process is that I have used Python, and for the ML parts I have relied on the Python libraries from Scikit–learn (classification) and Keras (neural network). These are free and accessible for everyone.

The initial plan for this thesis at the start of the project was to develop an app that could automatically extract feature values based on raw CL spectra data. The scope of this thesis has since then evolve a bit away from the app as an end result, toward a *proof of concept* (POC) for an entire multi–step process for automatically classifying quartz based on raw CL spectra data. Machine learning was always a fundamental part of the thesis objective.

The python scrips developed for this thesis are organized in Jupyter notebooks, and

may be used as is. To go from raw CL spectra data to final quartz group prediction require multiple steps, and there are some step that may require user interference (in the Jupyter notebooks). This is mentioned throughout this thesis where it is relevant.
It is most likely the case that it does not make sense to have the entire process from raw CL spectra data to proposed classification fully automated. Rather, there is a need for user observation and interference, especially in the step described in section 5.3, to ensure quality of the results produced.

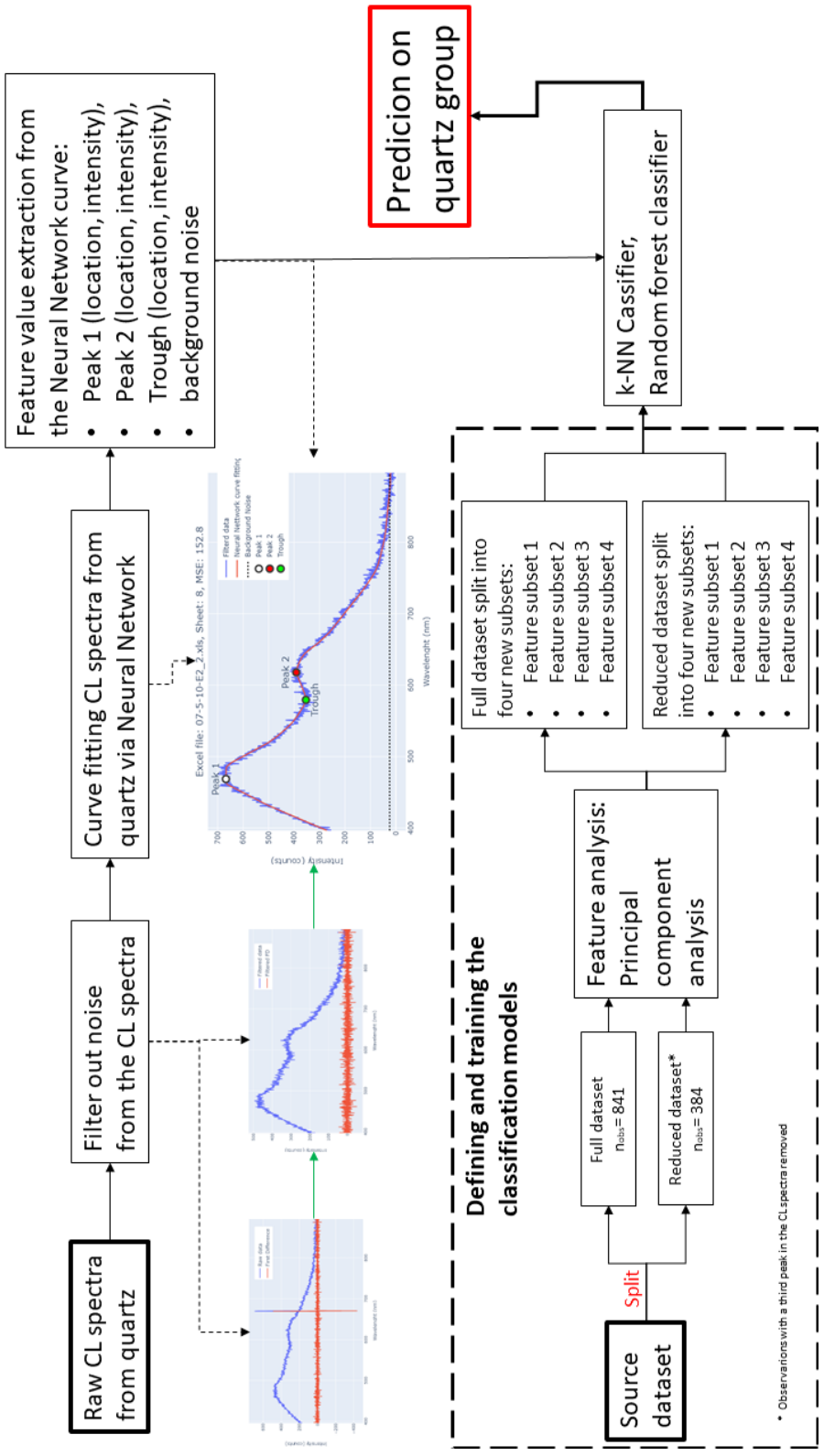


Figure 1.1: Workflow diagram for this thesis. It depict the two sub–processes of this thesis: (1) Being able to automatically extract parameters from CL spectra of quartz, (2) Training a ML model to classify the quartz based on the extracted CL spectra parameters.

2 | Theoretical background

2.1 Rocks and Minerals

2.1.1 Quartz

Quartz is the second most abundant mineral and it makes up 12.6 wt.% of the Earth's crust as crystalline and amorphous silica (Götze, 2009). It is a framework silicate which mean it has a three–dimensional framework comprising SiO_4 tetrahedras that are corner linked together in a network as illustrated in figure 2.1. Because each oxygen atom are shared between two tethrahedras, quartz has the general chemical formula SiO_2, but may have impurities in the form of metalic elements incorporated into its crystal structure.

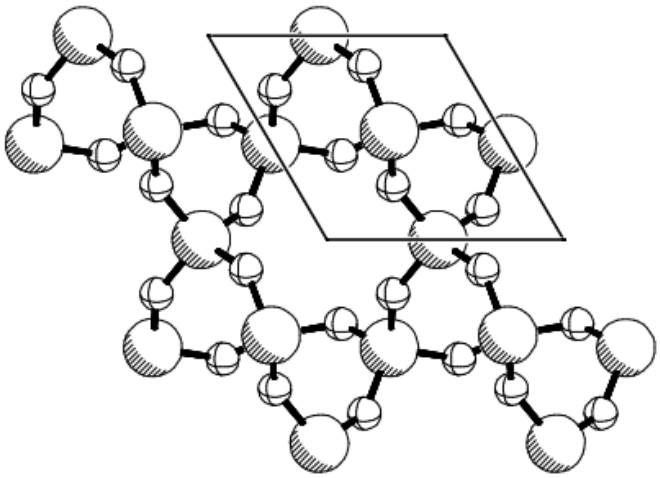


Figure 2.1: Representation of α–quartz structure. The large spheres represent Silicon atoms, and the small spheres represent oxygen atoms (Demuth et al., 1999)

On a side note, the structure as a whole in the quartz lattice has no center symmetry. It is this lack of center symmetry that gives rise to the piezoelectric effect[1] utilized for time keeping in digital watches.

Unlike most minerals, quartz has the ability to crystallize at a range of temperatures in

[1] In certain materials, such as quartz, they can accumulate an electric charge in response to applied mechanical stress.

different geological settings and thereby taking on characteristics related to the environment in which it was formed. This feature of quartz make it useful when determining the conditions at which a reservoir rock was formed, by studying its quartz inclusion.

2.1.2 Metamorphic Rocks

Metamorphism is a process that changes preexisting rocks into new forms because of increases in temperature, pressure, and chemically active fluids (Cheremisinoff, 1997). The pre–metamorphic parent rock is called the protolith.
The metamorphic rocks resulting from metamorphism are often characterized by unique mineral composition, particular texture, and structure to differentiate them from igneous and sedimentary rocks (Zou, 2013).

A metamorphic process takes place at temperatures above $200°C$ and pressure greater than 300 MPa. The upper limit of metamorphism occurs at the pressure and temperature of wet partial melting of the rock in question. Once melting begins, the process changes to an igneous process rather than a metamorphic process (Nelson, 2017).

Further, as pressure and temperature increases the rock undergo prograde metamorphism. That is, the grade of metamorphism increases, as illustrated in figure 2.2

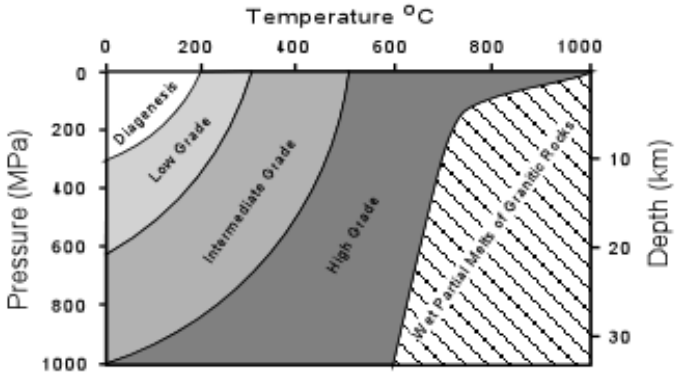


Figure 2.2: Factors affecting degree of retrograde metamorphism (Nelson, 2018)

Quartz is the most common SiO_2 polymorph[2] in metamorphic environments.

[2]In materials science, polymorphism describes the existence of a solid material in more than one form or crystal structure.

2.1.3 Igneous Rocks

Igneous rocks are formed when magma solidifies. This can happen beneath and above the surface, resulting in igneous rocks of two origins:

- plutonic
- volcanic

The texture of an igneous rock can tell whether it is an intrusive or extrusive rock. When magma and lava cool, mineral crystals start to form in the molten mass. Intrusive rock form underground where slow cooling takes place, and therefore allow for larger crystal growth. A special type of intrusive rocks are pegmatitic rocks which have very large crystals due to slow cooling of magma which is extra rich in dissolved water. Extrusive rocks will be more fine–grained due to more rapid cooling near or at Earth's surface.

The most widely used and simplest classification of igneous rocks is according to the silica (SiO_2) content, known as the silica-content classification scheme:

- Rocks with >66 wt.% silica are categorized as felsic (relative abundance in rock of feldspar and silica-quartz minerals),
- Rocks with 52-66 wt.% silica are categorized as intermediate,
- Rocks with 45-52 wt.% silica are categorized as mafic (relative abundance in rock of magnesium and ferrum or iron minerals),
- Rocks with <45 wt.% silica are categorized as ultramafic.

2.2 Cathodoluminescence (CL)

Cathodoluminescence is the light given off by a material when it is struck by an electron beam (Goldberg, 1966). Minerals in sedimentary rocks, such as feldspar, quartz and carbonate minerals, emit characteristic visible luminescence (figure 2.3) when bombarded by a stream of high–energy electrons.

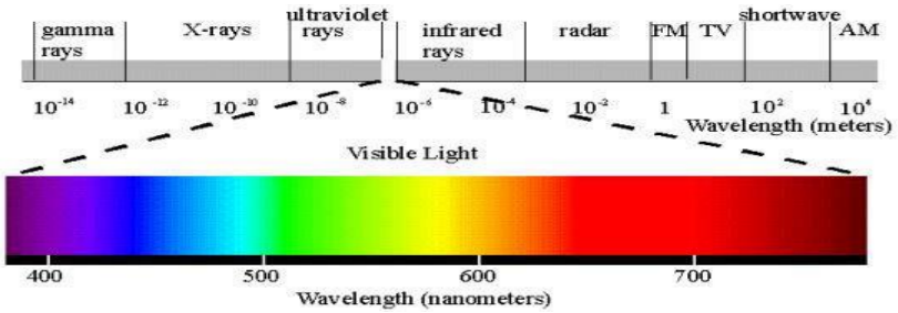


Figure 2.3: The electromagnetic spectrum with the visible light region blown up (Nayak et al., 2020).

In crystals, the energy state of electrons depend upon whether electrons are bound in particular atoms (inner–shell electrons) or are delocalized. Delocalized electrons are electron that are not associated with individual atoms or identifiable chemical bonds (S. Boggs & Krinsley, 2006).

Individual atoms have discrete energy states that are associated with the orbits of shells in of electrons in the atom. Because of the very large number of atoms that interact in a solid material (relative to material state such as gas), the energy levels are so closely spaced that they form bands (figure 2.4). The lower energy band is commonly designated as the **valence band** , and the other bands as the **conductor band**. Electron in the conductor band are easily removed by application of electric field. In insulators, such as quartz, a forbidden zone lie between the valence band and the conduction band –the band gap. Electrons do not reside permanently in the forbidden zone, but travel back and forth through it.

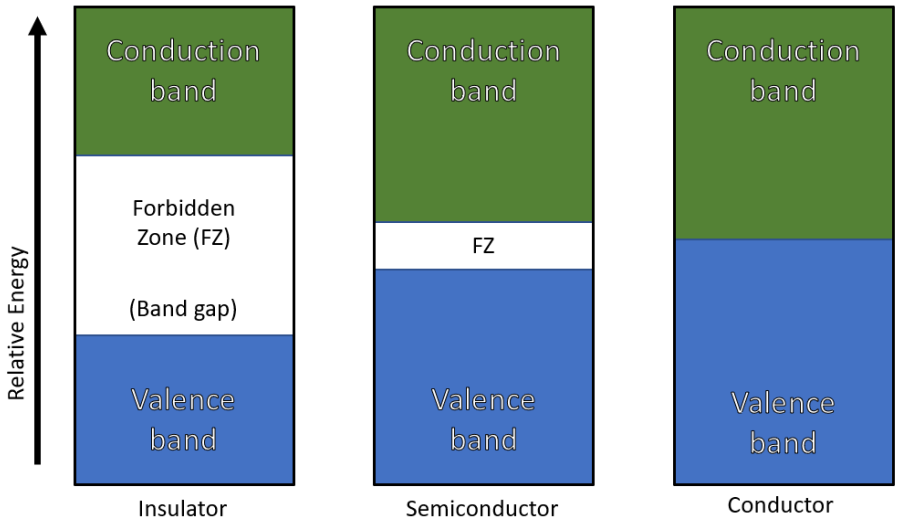


Figure 2.4: Illustration of energy bands in minerals. Insulator minerals, such as quartz, have quite wide band gap, whereas in conductor the band gap is essentially non–existing. Redrawn after (S. Boggs & Krinsley, 2006)

In insulator materials the conduction band hold none electrons as they reside in the lower–energy valence band. In CL, electrons are excited from the valance band into the conductor band as a result of a directed high energy electron beam onto a semiconductor. When this happens, it leaves behind a so–called "hole". Luminescence results when an electron in the conduction band recombines with a hole in the valence band.

2.2.1 Cathodoluminescence Applied to Quartz

The crystalline form of SiO_2 at temperatures below $573°C$ is called α–quartz, or just simply quartz. Above $573°C$ the form of SiO_2 is called β–quartz. The α–quartz structure is more compact than the β–quartz structure and hence is less open to the incorporation of trace elements. These trace elements may be incorporated during crystal growth and re–crystallisation in the cation site of quartz.

When performing CL on quartz, and electrons have been exited from the valance band to the conduction band, they will after a short time begin to de–excite and return to a lower–energy state in the valence band. They do this by moving randomly through the crystal lattice until they encounter a trap caused by defects caused by trace elements in the crystal lattice. The electron remain trapped only momentarily and as the electron exit the trap, a photon is emitted.

If no traps are present, electrons fall directly back to the valence band and emit photons with wavelength in the near ultraviolet. Therefore, the presence of such traps at discrete energy levels within the band gap, is a precondition for emission of photons (cathodoluminescence) in the visible light range (S. Boggs & Krinsley, 2006).

Quartz CL spectrum has several emission bands which are directly related to lattice defects. The type and frequency of lattice defects are influenced by the thermodynamic conditions during mineralization. Post mineralization effects, such as metamorphism, tempering or deformation can change the structural properties of quartz (Götze et al., 2001). For this reason, quartz derived from plutonic, volcanic, metamorphic, and hydrothermal systems emit often unique CL signatures that can aid in identifying different generations of quartz formed in a specific geologic environment (Frelinger, Ledvina, Kyle, & Zhao, 2015).

The visible CL of natural quartz mainly consists of two broad emission bands centered at ∼450 nm (blue emission) and 620–650 nm (red emission), as shown in figure 2.5. The blue band is usually very broad and consists of up to four overlapping component bands centered at 390, 420, 450, and 500 nm (Gorton, Walker, & Burley, 1997). Each of these emission bands, and more, are due to unique lattice defects in the crystal structure of quartz, as summarized in Götze et al. (2001).

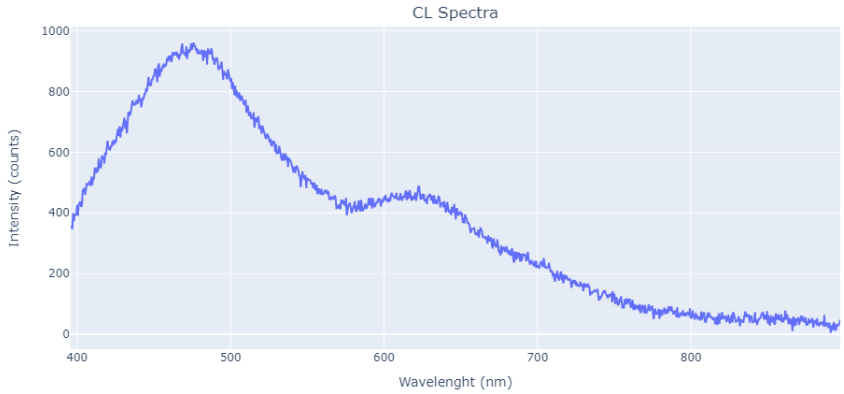


Figure 2.5: CL spectra from a quartz sample. The blue emission band is given by the first peak at around 480 nm, and the red emission band is given by the second peak at around 640 nm.

Zinkernagel (1978) was the first to systematically investigate quartz from different types of crystalline rocks for their CL characteristics (Augustsson & Reker, 2012). Further, he established one of the first classification schemes of quartz CL colors in sandstone.

In Augustsson & Reker (2012) they investigated the CL spectra of more than 1000 quartz crystals from 58 samples of different plutonic, volcanic, metamorphic, and pegmatitic rocks. They came up with a discrimination scheme based on the relative intensity of the two main emission centers in visible light at 470–490 nm and 600–640 nm. In their proposed discrimination scheme they divided rocks into one of three groups. This grouping is further described in chapter 3, as this grouping is what the ML classification models are trained to predict when trained on Augustsson & Reker's dataset consisting of data from more than than 1000 quartz CL spectra.

2.3 Principal Component Analysis

The foundation for principal component analysis was first laid by Karl Pearson in (1901), but was independently developed into the techniqe we know today by Harold Hotelling initially in (1933) and then further in (1936).
Principal Component Analysis (PCA) is a dimensionality-reduction method where the features of the dataset are transformed into linearly independent principal components (PC), where each $PC_i \in [1, d]$ collectively and in decreasing magnitude explain the variance of the dataset. Here, d is the number of features in the dataset. This way, most of the variance of the dataset can be explained by a number of PC's less than the original number of features of the dataset while the loss of information is minimal. For machine learning purposes this can be particularly useful as a method to aid in feature selection. Having more features to train ML models on does not necessarily translate into higher accuracy. It may actually decrease accuracy as some of the feature only create noise.

Having the features of interest represented in a two–dimensional matrix, $\mathbf{X}$, with features as the columns and n rows of observations (equation 2.1), the first step of PCA analysis is to standardizes (or scaled) the features. This can be done by subtracting the mean and dividing by the standard deviation for each value of each variable, as show by equation 2.2. The purpose of this standardization is to avoid that with high variance

features dominate low variance features, causing bias.

$$\mathbf{X} = (x_{i,j}) \in \mathbb{R}^{n \times d} \tag{2.1}$$

$$\vec{Z}_d = \frac{x_{d,j} - \mu_d}{\sigma_d}, \tag{2.2}$$

where μ is the mean, σ is the standard deviation and $j \in [1, .., n]$.

After normalizing the features of matrix $\boldsymbol{X}$, it is now referred to as the matrix $\boldsymbol{Z}$, where

$$\mathbf{Z} = \mathbf{X}_{norm} = (z_{i,j}) \in \mathbb{R}^{n \times d} \tag{2.3}$$

The second step is to calculate the covariance of $\mathbf{Z} = \{\vec{Z}_1, .., \vec{Z}_d\}$, as given by equation 2.4, where $\vec{Z} = [z_1, .., z_n]$:

$$\mathbf{\Sigma_Z} = \begin{pmatrix} Var(\vec{Z}_1) & \cdots & Cov(\vec{Z}_1, \vec{Z}_d) \\ \vdots & \ddots & \vdots \\ Cov(\vec{Z}_d, \vec{Z}_1) & \cdots & Var(\vec{Z}_d) \end{pmatrix} \tag{2.4}$$

In the third step one compute the eigenvectors (equation 2.6) and eigenvalues (equation 2.5) of the covariance matrix to identify the principal components of the dataset. The eigenvectors of the covariance matrix give the direction of the axes where there is the most variance, and the eigenvalues are coefficients attached to the eigenvectors giving the amount of variance carried in each principal component.

$$det(\vec{\lambda}\mathbf{I} - \mathbf{\Sigma}) = 0 \tag{2.5}$$

where $\lambda_i \in [1, d]$ are the eigenvalues and $\mathbf{I}$ is the identity matrix.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you will the principal components in order of significance. The eigenvectors associated with each eigenvalue can then be computed as

$$(\lambda_i \mathbf{I} - \mathbf{\Sigma}) V_i = 0 \tag{2.6}$$

where V_i is the eigenvector.

By dividing each eigenvalue, λ_i, by the total sum of all eigenvalues, $\sum_{i=1}^{d} \lambda_i$, the percentage of variance accounted for by each principal component is found.

Principal components are linear combinations of the the standardized data vectors $Z_1, ..Z_d$, using the eigenvectors as the weights. Take the eigenvalues $\lambda_1, ..\lambda_d$ and sort them from largest to smallest, $q \in [1, .., d]$. In doing so, sorting the corresponding eigenvectors accordingly one find the principal components as:

$$PC_q = V_q^T \mathbf{Z} = \sum_{i=1}^{d} \vec{Z}_i \cdot w_{q,i} \tag{2.7}$$

where $w_{q,i} \in [-1, 1]$ is known as the loading and may be thought of as the correlation between a feature $\vec{Z}_i$ and PC_q.

2.4 Machine Learning (ML)

The term *machine learning* was coined by Arthur Samuel (1959), a pioneer in the field of computer gaming and artificial intelligence.

One can split machine learning into two broad approaches: *Supervised learning* which uses labeled data to help predict outcomes, and *unsupervised learning* which does not use labeled data.

With the supervised learning approach the datasets are conditioned to train or "supervise" algorithms into classifying data or predict outcomes accurately. An example of such data set is seen in table 2.1. In the left-most column we have the target variable which consist of groups (described in chapter 3). The other columns have the predictor variables, or features, which are used to train the algorithm into predicting a rock type.

Table 2.1: Example of data structure used for supervised machine learning. This is a sample of the actual source dataset used for this thesis.

Group	Peak 1, nm	Peak 1, counts	Peak2, nm	Peak 2, counts	Trough, nm	Trough, counts	Bg, counts
3	485,8	325	615,3	304	547,3	234	10
3	490	199	630,5	247	525,5	187	0
1	479	63	642,7	99	621,9	74	5
3	487,4	468	608,1	404	557,7	380	10
1	474,3	453	626,5	356	568,5	241	0
2	490	344	607,1	352	515,6	302	5
3	486,3	181	634,1	456	511,4	179	0
3	483,7	173	635,3	246	531,8	150	0
1	476,3	54	646,7	70	613,7	62	0

Supervised learning can be separated into to groups:

- **Classification**: Target variable consist of categories.
- **Regression**: Target variable is continuous.

Given real–word observations represented by f, the goals of supervised learning is to first find a model $\hat{f}$ that best approximates

$$f : \hat{f} \sim f \tag{2.8}$$

where $\hat{f}$ can be of any ML algorithm applied. Further, the model should discard noise as much as possible, which translates into the end goal - $\hat{f}$ should achieve a low predictive error on unseen datasets. The difficulties in approximating f is seen in

- **Overfitting**: $\hat{f}$ fits the training set noise.
- **Underfitting**: $\hat{f}$ is not flexible enough to approximate f.

The objective of this thesis only call for the use of supervised learning methods. Although, unsupervised learning methods could be used in another thesis, on a related topic to this thesis, for example to search for hidden patterns and correlations in the dataset used.

2.4.1 Neural Network (NN)

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are machine learning techniques that simulate the mechanism of learning in biological organisms. These networks contain computation units that

are referred to as neurons. An example of such a neuron is illustrated in figure 2.6. The computational units are connected to one another through weights, which serve the same role as the strengths of synaptic connections in biological organisms. Each input to a neuron is scaled with a weight, which affects the function computed at that unit. An artificial neural network computes a function of the inputs by propagating the computed values from the input neurons to the output neuron(s) and using the weights as intermediate parameters. Learning occurs by changing the weights connecting the neurons (Aggarwal, 2018).

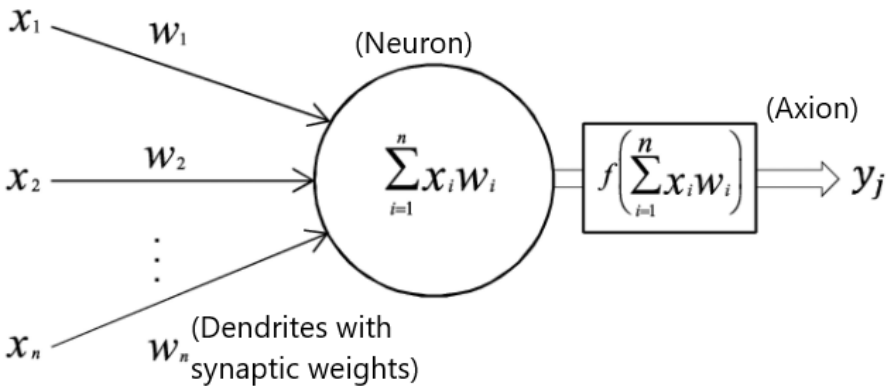


Figure 2.6: The perceptron is the oldest neural network, created by Frank Rosenblatt in 1958. It has a single neuron and is the simplest form of a neural network with a single input layer and an output node. Based on (Vieira et al., 2017).

The perceptron takes several binary inputs, $x_1, .., x_d$ and produces a single binary output. The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum x_i w_i$ is less than or greater than some threshold value (equation 2.9). That is, the neuron is either "ON" or "OFF".

$$output = \begin{cases} 0 & \text{if } \sum x_i w_i \leq \text{threshold} \\ 1 & \text{if } \sum x_i w_i > \text{threshold} \end{cases} \tag{2.9}$$

Whereas the perceptron is a linear classifier, activation functions are applied to neural networks in order to introduce non-linearity into the output of a neuron. In those cases, the outputs may also take on a continuum of values instead of just 0 and 1.

The sum of all weighted inputs, $\sum x_i w_i + bias$ (equation 2.10), is passed through a

nonlinear activation function f, to transform the preactivation level of the neuron to an output y_j. The output y_j then serves as input to a node in the next layer. Several activation functions are available, which differ with respect to how they map a pre-activation level to an output value.

Traditionally, the *sigmoid* activation function (equation 2.13) was the default activation function in the 1990s. Then, the *tanh* function (equation 2.12) became the default activation function for hidden layers. Both the sigmoid and tanh activation functions can make the model more susceptible to issues and problems, especially the so-called vanishing gradients problem[3].

In today's neural networks, the default recommendation is to use the rectified linear unit or *ReLU* as activation function (equation 2.11) (Jarrett, Kavukcuoglu, Ranzato, & LeCun, 2009); (Nair & Hinton, 2010); (Glorot, Bordes, & Bengio, 2011) because it overcomes the vanishing gradient problem, allowing models to learn faster and perform better. However ReLUs are non-negative and, therefore, have a mean activation larger than zero. Units that have a non-zero mean activation act as bias for the next layer. If such units do not cancel each other out, learning causes a bias shift for units in next layer. The more the units are correlated, the higher their bias shift (Clevert, Unterthiner, & Hochreiter, 2016).

Clevert et al., (2016) introduced the *exponential linear unit* (ELU) (equation 2.14). In contrast to ReLUs, ELUs have negative values which allows them to push mean unit actuation's closer to zero like batch normalization but with lower computational complexity. Mean shifts toward zero speed up learning by bringing the normal gradient closer to the unit natural gradient because of a reduced bias shift effect.

$$y_j = \bar{\mathbf{x}} \cdot \bar{\mathbf{w}} + bias = \sum_{i=1}^{d} x_i w_i + bias \tag{2.10}$$

Where in equation 2.10 $bias \equiv -threshold$ in equation 2.9, $\bar{\mathbf{x}} = [x_1, .., x_d]$ and $\bar{\mathbf{w}} = [w_1, .., w_d]$.

[3]In machine learning, the vanishing gradient problem is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, during each iteration of training each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training (Basodi, Ji, Zhang, & Pan, 2020)

$$f(y_j) = \begin{cases} y_j & \text{if } y_j > 0 \\ 0 & \text{if } y_j \leq 0 \end{cases} \tag{2.11}$$

$$f(y_j) = tanh(y_j) \tag{2.12}$$

$$f(y_j) = \frac{1}{1 + e^{-y_j}} \tag{2.13}$$

$$f(y_j) = \begin{cases} y_j & \text{if } y_j > 0 \\ \alpha(e^{y_j} - 1) & \text{if } y_j \leq 0 \end{cases} \tag{2.14}$$

where α in equation 2.14 is a constant with default value of 1.

By connecting more neurons together in layers, as shown in figure 2.7, one can make more complex decisions.

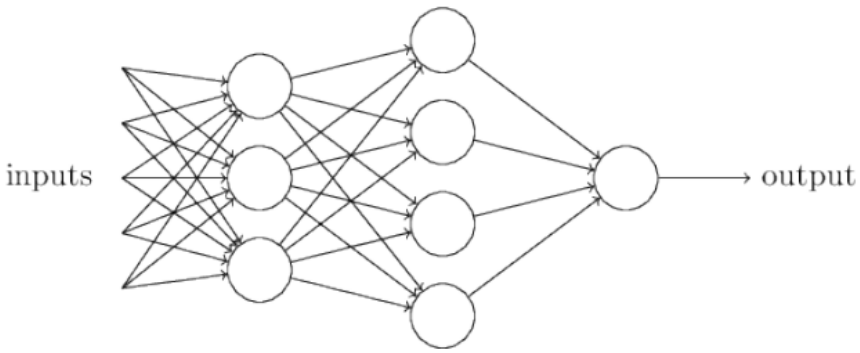


Figure 2.7: Example of a neural network. This specific neural network has 3 neurons in the first hidden layer, and 4 neurons in the second hidden layer. Figure is from (Nielsen, 2013).

An optimizer is an algorithm or method used to change the attributes, such as weights and learning rate, in order to reduce loss. Loss is a metric on how a model is performing at a given instance, and the goal is (in general) to minimize loss through a process of optimization. The loss function C must be suited to the kind of problem a neural network is applied toward. In this thesis for example, mean squared error (MSE),

equation 2.15, is used as loss function ($C = MSE$) because the goal is to fit a curve to data points.

$$C(\mathbf{Y}, \bar{\mathbf{x}}, \bar{\mathbf{w}}, bias) = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{y}_i)^2, \tag{2.15}$$

where $\mathbf{Y} = [Y_1, .., Y_N]$ are the true data points being predicted by $\hat{\boldsymbol{y}}$ via the neural network and N is the number of data points.

When it comes to choice of optimizer, there are numerous to choose from (Ruder, 2016). Common for most of them is that they are based on the method of gradient decent, an algorithm where parameters θ, which include the weights (**w**), are adjusted until a (local) minimum of the loss function ($C(\theta)$) is presumably found. This is illustrated in figure 2.8.

The gradient decent algorithm (equation 2.16) has two requirements toward a function: (**1**) The function is differentiable (since this algorithm is based on gradients) and (**2**) the function is convex (given that the goal is to minimize the function).

$$\theta_{t+1} = \theta_t - \alpha\nabla_\theta C(\theta) \tag{2.16}$$

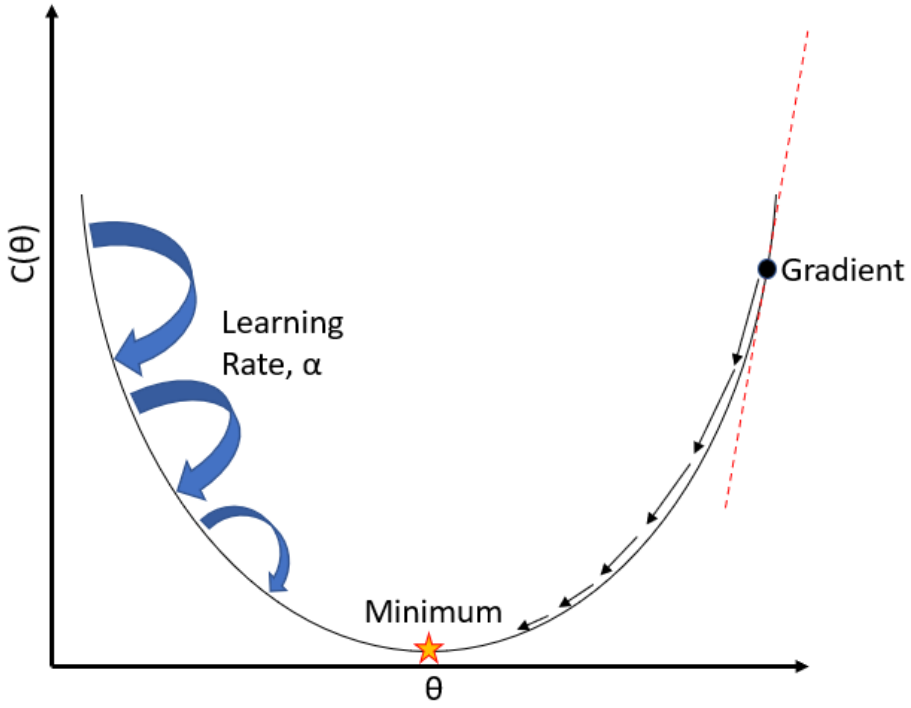


Figure 2.8: The basic working of gradient decent-based optimizers. The collection of parameters θ are adjusted in the search for a minimum. The (local) minimum is also known as *point of convergence*.

The *Adam* (Adaptive moment estimation) optimizer (equation 2.17) is a popular optimizer because it work well in most scenarios due to it in general being fast and converging rapidly (Kingma & Ba, 2014).
Adam build on previous optimizers before it, and aim to accommodate their combined strengths. To be specific, Adam is a combination of two gradient decent methods; *momentum* and *Root Mean Square Propagation* (RMSP).

Momentum is used to accelerate the gradient descent algorithm by taking into consideration the exponentially weighted average of the gradients (equations 2.20 and 2.21), making the algorithm converge toward a minimum at an increased pace. In other words, instead of using only the gradient of the current step, momentum also accumulates the gradient of the past steps to speed up the decent. Further, this also has the benefit of reducing the likelihood of the optimizer getting stuck at a local minima.

In RMSP, the learning rate adapts based on the exponential moving average of the magnitudes of the recent gradients. By adapt, what is meant is that the learning rate

start in larger steps and then reduces, or decays (β), as the minima approaches. This allows for a faster convergence due to taking larger steps initially.

Adam, like the other optimizers, is an iterative process and the update rule is given by:

$$\theta_{t+1} = \theta_t - \alpha \cdot \hat{m}_t \left(\frac{1}{\sqrt{\hat{v}_t} + \epsilon} \right) \tag{2.17}$$

where

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.18}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.19}$$

and where

$$m_t = (1 - \beta_1) g_t + \beta_1 m_{t-1} \tag{2.20}$$

$$v_t = (1 - \beta_2) g_t^2 + \beta_2 v_{t-1} \tag{2.21}$$

ϵ in equation 2.17 is a small $+ve$ constant to avoid "division by 0" error when $v_t \longrightarrow 0$. m_t and v_t in equations 2.20 and 2.21, are estimates of the first moment and the second moment of the gradients, respectively.
Further, $g_t = \nabla_\theta C(\theta_{t-1})$, and β_1, β_2 are decay rates. The decay rates are usually set at $\beta_1 = 0.9$ and $\beta_2 = 0.999$
Finally, because m_t and v_t has a tendency to be biased toward 0 as both β_1 and $\beta_2 \sim 1$. This is fixed in equations 2.18 and 2.19 by "bias–correcting" m_t and v_t.

2.4.2 k-Nearest Neighbours (kNN) Algorithm

The k-nearest neighbor algorithm (kNN) is a widely used and easy to implement classification algorithm. It consider the k closest labeled data points to an unknown data point *x*, and classify it by taking a "majority vote" (equation 2.23). Deciding which

points are nearest is done according to some pre-specified distance. The number of data points (k) can be user-defined constant (k-nearest neighbour learning), or vary based on the local density of points (radius-based neighbor learning).

Figure 2.9 illustrate how classification of an unknown data point is done in the kNN algorithm. In the case illustrated, the algorithm is defined to make a prediction on unknown data points by taking the class majority of the three closest labeled data points.

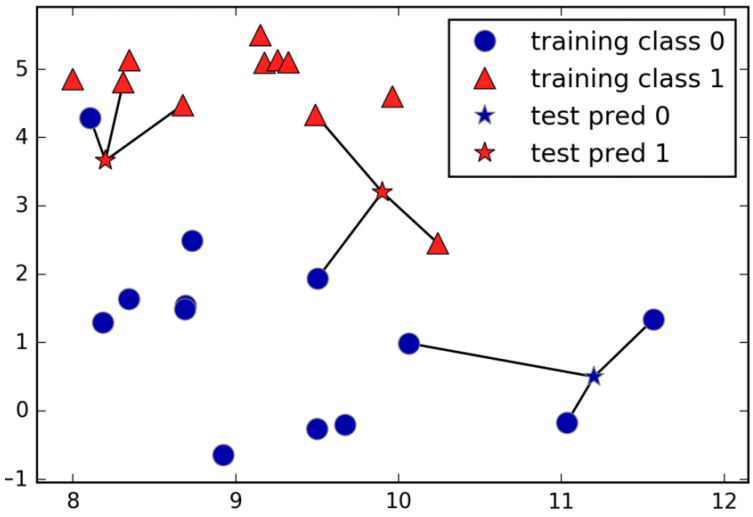


Figure 2.9: Prediction made by a three-nearest-neighbors algorithm. A majority vote decide the classification of the unknown data point (Müller & Guido, 2016).

Fundamentally, there are two important parameters to the kNN algorithm: The number of neighbors, k, and how you measure distance between data points. The default method for measuring distance between data points is by Euclidean distance as defined by equation 2.22 for a N-dimensional domain

$$d(\mathbf{x}', \mathbf{x}_i) = \sqrt{\sum_{j=1}^{N} (x_{i,j} - x'_j)^2}. \tag{2.22}$$

The kNN algorithm is summarized as follows (Yigit, 2013):
Consider $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \in D$ as the training set, where $\mathbf{x}_i$ is the N-dimensional training vector and y_i is the corresponding class label. Given a query $\mathbf{x}'$ from the test set $(\mathbf{x}', y')$, its unknown class label y' is determined as follows:

1. Compute distance d between $\mathbf{x}'$ and each $(\mathbf{x}_i, y_i)$
2. Arrange all distances in an ascending order
3. Assign $w_i = 1$ for equally-weighted kNN rule,
4. The class label of $\mathbf{x}'$ is assigned based on majority vote of its nearest neighbors,

$$y' = \arg\max_{\chi} \sum_{(\mathbf{x}_i, y_i) \in D} w_i \delta(\chi = y_i') \tag{2.23}$$

where χ is a class label, y_i' is the class label for the ith nearest neighbors among its k nearest neighbors, $\delta(\cdot)$ is the Dirac-Delta function that takes the value 1 if its argument is true and 0 otherwise.

The value of k in the kNN algorithm is related to the error rate of the model. A small value of k could lead to over-fitting, just as a big value of k can lead to under-fitting. Over-fitting imply that the model will do well with regard accuracy on the training data, but has poor performance when generalized to unseen data. An example of this is seen in figure 2.10.

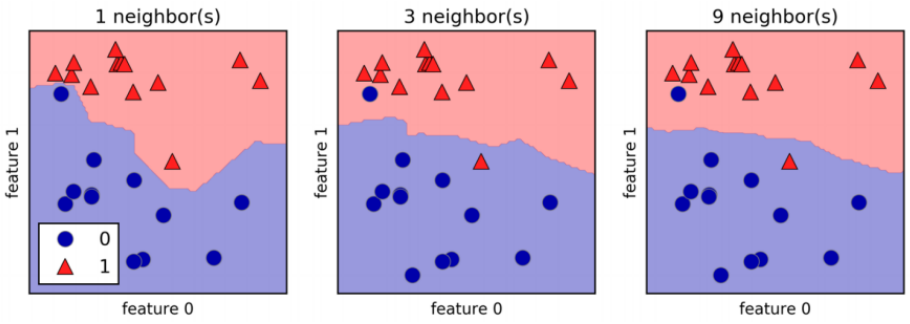


Figure 2.10: Example of decision boundaries created by a kNN model for different values of k. A smaller k give a more complex model but can lead to over-fitting, whereas a larger k give a smoother decision boundary but can lead to a less complex model. (Müller & Guido, 2016).

One common rule of thumb followed in the data science community regarding the value of k, is to set it equal to the square root of the number of observations in the training dataset.

A more precise way of deciding k is to split the data set into a training set and a test set (also known as train/test–split and described in detail in section 2.4.4), and

then plotting different values of k against the training and testing accuracy (see section 2.4.4 for definition on accuracy) as illustrated in figure 2.11. Optimal number of k neighbours is generally taken to be where testing accuracy is maximized, but this point should also be located close to the *Elbow point* of the training accuracy curve. The *Elbow point* can be recognised as the point of the *training accuracy* curve where the rate of change suddenly become marginal.

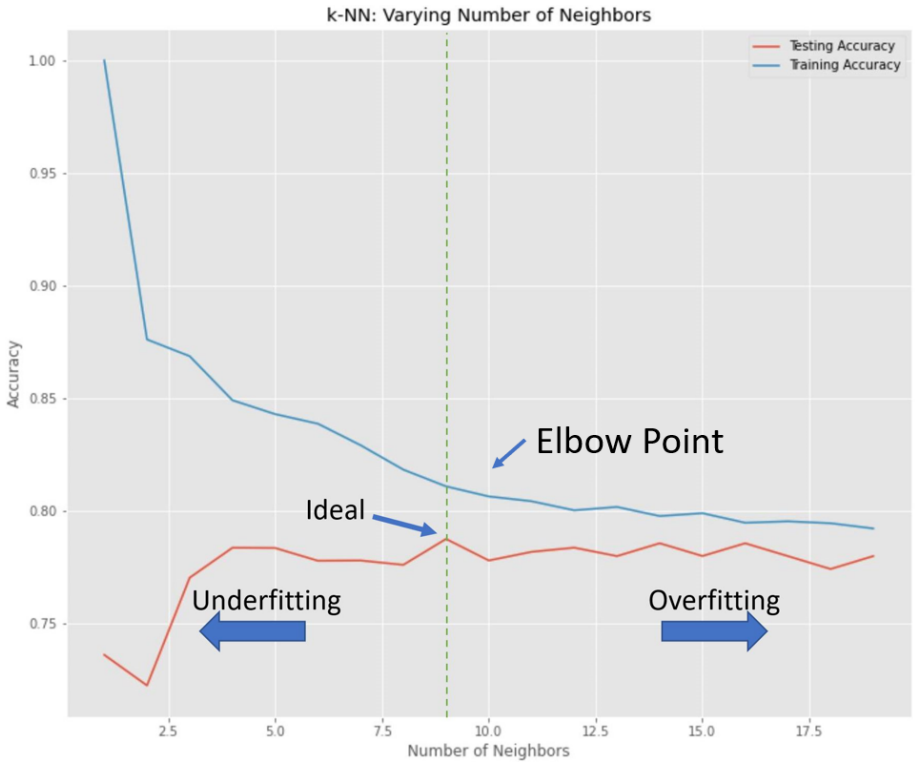


Figure 2.11: Example of how the optimal k can be selected for a kNN model by plotting different values of k against the training and testing accuracy (see section 2.4.4 for definition of "accuracy"). For this case the optimal would be $k = 9$.

By taking a more analytical approach to deciding k, the chance of over or under fitting by the model is reduced.

2.4.3 Tree-Based Algorithms

The *classification and regression tree* algorithm, (CART) for short, was first introduced by Breiman et al. (1984). In the CART algorithm, the objective is to minimize a cost function, for example the Gini Index (equation 2.24) or entropy (equation 2.25), at each

node of a classification tree. A classification tree, as illustrated in figure 2.12, can be described as a series of if-else questions about individual features, and by doing this achieve the objective which is to predict class labels.
The main strengths of this algorithm is that it is able to capture non-linear relationships between features and classes, and it does not require feature scaling (e.g., standardization).

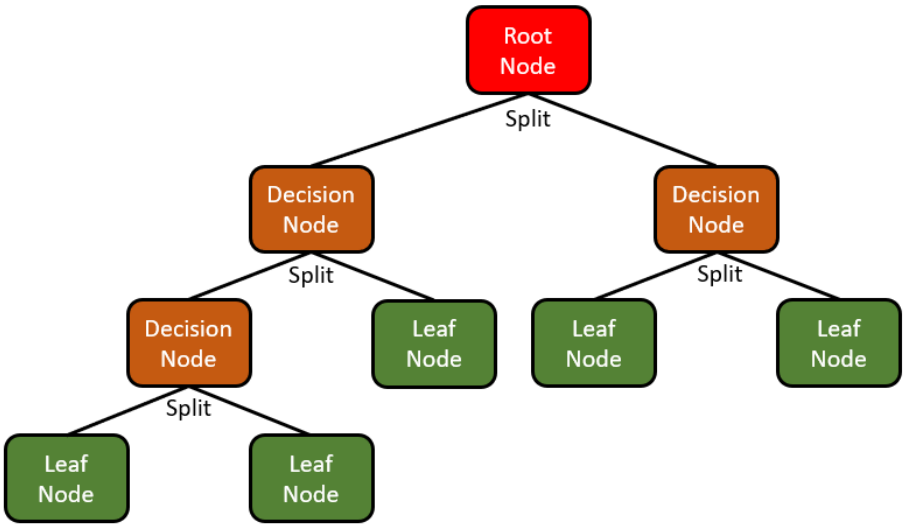


Figure 2.12: The building blocks (nodes) of a typical decision-tree model.

The building blocks of a decision–tree can be summarized in three kinds of nodes, and these building blocks form a hierarchy of nodes where a node is either a question regarding a feature or a prediction:

- **Root Node**: No parent node, where question regarding features give rise to child nodes.
- **Decision Node**: One parent node. Question regarding features give rise to child nodes, either new decision nodes or leaf nodes.
- **Leaf Node**: Node where predictions are realized.

Information gain (IG) (equation 2.26) is used as a criterion to measure impurity[4] of a node, and the information gain is based on the decrease in entropy after a dataset is

[4]The node impurity is a measure of the homogeneity of classes at a node, and is measured as the tree is grown recursively. A node is said to be pure, or homogeneous, if it only contains a single class. The more classes at the node, the more impure it is, and is given by a number between 0 and 1.

split on a feature. If Gini impurity or Entropy = 0, that is, all cases in a node fall into a single target category, it is declared a leaf node and can't be split further.

Algorithm 1 CART (*F*,*Y*,*D*) (Wu et al., 2020)

Require: *F*: feature set, *Y*: label set, *D*: sample set
Ensure: *T*: decision tree
if *prune conditions satisfied* **then**
classification: return leaf node with majority class
regression: return leaf node with mean label value
else
determine the best split feature *j* and value *s*
split *D* into 2 partitions D_l, D_r
return a tree with feature *a* that has two edges, call $CART(F-j, Y, D_l)$ and $CART(F-j, Y, D_r)$
end if

We assume there is a training dataset *D* with *n* observations $\{\mathbf{x}_1, .., \mathbf{x}_n\}$ each containing *d* features and has the corresponding output label set $Y = \{y_1, .., y_n\}$. For each tree node, it first decides whether some pruning conditions are satisfied, e.g., feature set is empty, tree reaches the maximum depth, the number of samples is less than a threshold. If any condition is satisfied, then it returns a leaf node with the class of majority samples for classification or the mean label value for regression. Otherwise, it determines the best split to construct two sub-trees that are built recursively. In order to find the best split feature and split threshold, CART uses Gini impurity or Entropy as a metric in classification. Let *c* be the number of classes and $K = \{1, .., c\}$ be the class set. Let $\tilde{D}$ be sample set on a given node, the Gini impurity and Entropy is calculated (Wu et al., 2020):

$$I^{Gini}(\tilde{D}) = 1 - \sum_{k \in K} (p_k)^2 \tag{2.24}$$

$$I^{Entropy}(\tilde{D}) = -(p_k) \cdot log_2(p_k) \tag{2.25}$$

where p_k is the fraction of samples in $\tilde{D}$ labeled with class k. Let F be the set of available features, given any split feature $j \in F$ and split value $\tau \in Domain(j)$, the sample set $\tilde{D}$ can be split into two partitions $\tilde{D}_l$ and $\tilde{D}_r$. Then, the *information gain* of the split is calculated as follows:

$$\begin{aligned} IG(\tilde{D}, j) &= I_{Parent}(\text{before split}) - I_{Child}(\text{after split}) \\ &= I(\tilde{D}) - (w_l * I(\tilde{D}_l) + w_r * I(\tilde{D}_r)) \end{aligned} \tag{2.26}$$

where $w_l = |\tilde{D}_l|/|\tilde{D}|$, $w_r = |\tilde{D}_r|/|\tilde{D}|$ and the impurity criterion I can be either the Gini impurity or Entropy. The split with the maximum information gain is considered the best split of the node (Wu et al., 2020).

One of the challenges with decision trees is that they are very good at learning relationships within any given data you train them on, but they tend to overfit the data you use to train them on and usually generalize poorly to new data. That is, they are low bias, high variance learning models as illustrated in figure 2.13.

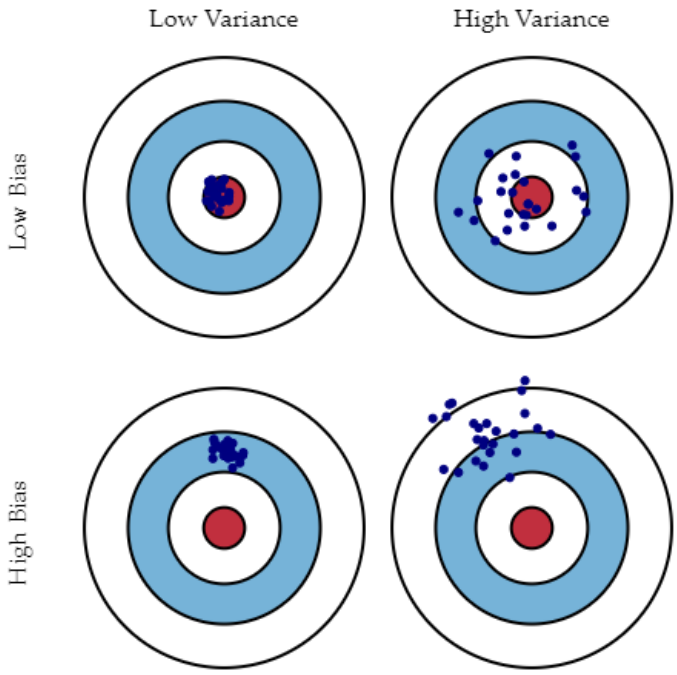


Figure 2.13: Graphical illustration of bias and variance. (Fortmann-Roe, 2012)

A workaround for this is to apply ensemble methods like boosting or bagging (short for bootstrap aggregating). Boosting is a technique, and not a specific ML algorithm, where a collection of weak learners[5] are converted into a strong learner[6], by sequentially training each new instance to the ensemble such that it correct the errors of the

[5]Weak learner: ML algorithm that is slightly better than chance.
[6]Strong learner: Any algorithm that can be tuned to achieve good performance.

previous tree.

In bagging, multiple subsets are created by randomly sampling observations with replacement from the original training set. Models are then fitted on these subsets. The final predictions are determined by majority voting (equivalent to majority voting for the kNN algorithm described by equation 2.23) on the predictions by all models in the ensemble.

Random forest is a type of bagging method, but in addition not all features from the original dataset are present in each subset. Only a predetermined number of randomly sampled features. Again, for prediction, just like for bagging, it is determined by majority vote.

2.4.4 Cross–Validation and Evaluation of ML Classification Models

For most ML classification algorithms, the initial step is to normalize the feature values (equation 2.2) in the dataset on which the algorithm is to be applied to. This is done because ML algorithms, for example the kNN algorithm which rely on some defined distance between a point with unknown label and labeled neighbouring points when classifying, tend to perform better when features can be evaluated on a similar scale.

Next, the dataset (D) is split into two randomly drawn training and testing subsets. The training dataset is used to train the ML algorithm, and the test dataset is used to evaluate the ML model afterward. This train–test procedure got one weakness: It is sensitive to the randomness of the split. A way to mend this to a large degree is to utilize *k–fold cross–validation* on the training set. This is done by further splitting the training set into k equal parts and in turn one part is withheld for testing, and the $k-1$ remaining parts are used for fitting the ML model. For each turn the resulting metrics are kept, and after k models have been fitted and tested, the metrics are averaged. This process is illustrated in figure 2.14. In the end, the initially withheld test set is used to validate the model.

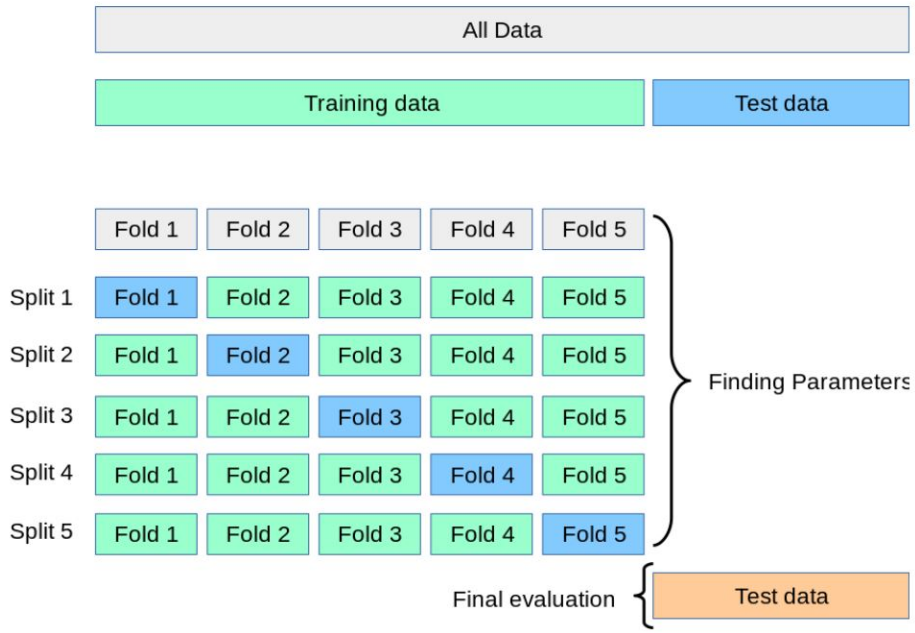


Figure 2.14: Example of the process of using cross–validation on training data when deciding on optimal ML parameters (scikit learn, n.d.).

Most of Scikit–learns library of ML algorithms have predefined hyperparameters[7], and often an algorithm will yield decent performing model when applied as is. Although, there may be a lot to gain in terms of for example accuracy by applying hyperparameter tuning on a algorithm. This is done by defining a range of values for each hyperparameter one wish to tune. The algorithm can then either be fitted with each combination of hyperparameters, known as grid search, or alternatively, if the number of possible combinations of hyperparameters get very large, one may opt for a randomized search. In a randomized search, a defined number of random hyperparameter combinations are tested. In both methods, a score is kept track on for each iteration, for example the accuracy of the algorithm, and the combination yielding the highest score is applied to the algorithm.

Confusion matrices are used to evaluate the classification models. It show the ways in which a classification model is confused when making predictions.
The confusion matrix based on four variables, shown in figure 2.15.

[7]In ML, hyperparameters are parameters whose value is used to control the learning process.

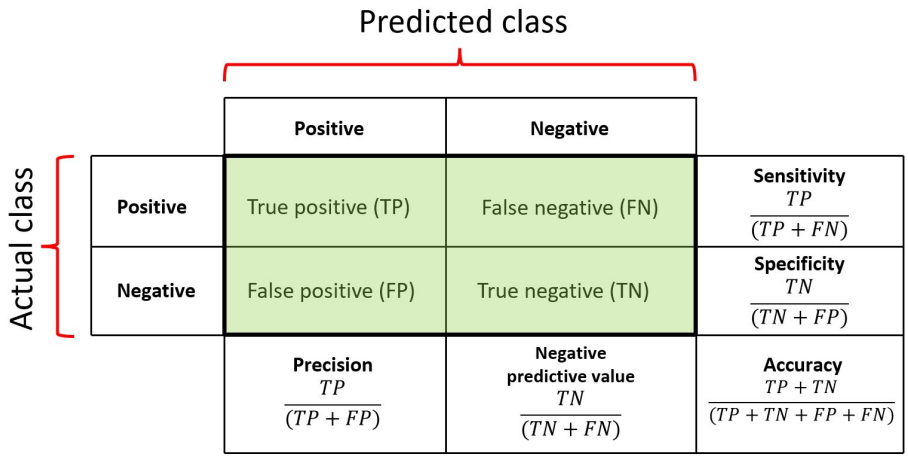


Figure 2.15: Example of a confusion matrix with a two–event outcome, positive or negative. After a model has been trained, the confusion matrix give metrics on how good that model is at correctly predict and where it struggles.
The output "**TN**" shows the number of negative examples classified accurately. "**TP**" indicates the number of positive examples classified accurately. "**FP**" shows the number of actual negative examples classified as positive; and "**FN**" is the number of actual positive examples classified as negative.

False positive (FP) is also known as "Type I error", and false negatives (FN) is also known as "Type II error".

The metrics most commonly kept track on is precision, sensitivity (also known as *recall*), F1 score (equation 2.27) and accuracy.

$$F1 - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.27}$$

2.4.5 Permutation Feature Importance

The permutation feature importance is the decrease in model score, for example accuracy, when observations in a single feature are randomly shuffled (Breiman, 2001). This break the relationship between the feature and the class. Thus, the more important the feature is the higher the decrease in score when shuffling. This shuffling in repeated K times for each feature.

Given a dataset D with $j \in [1, .., d]$ features, where s is the reference score of a model m fitted on dataset D, the importance i_j for feature x_j is given as:

$$i_j = s - \frac{1}{K} \sum_{k=1}^{K} s_{k,j} \tag{2.28}$$

Where $s_{k,j}$ is the score of model m on a corrupted dataset (a dataset with shuffled features) $\tilde{D}_{k,j}$ for each repetition $k \in [1, K]$.

3 | The Datasets

The source dataset for this thesis is the same as developed by Augustsson & Reker (2012). They analyzed more than 1000 quartz crystals from 58 samples of different plutonic, volcanic, metamorphic and pegmatitic rocks. I.e., one analyzed quartz crystal is the same as one observation in the source dataset. Therefore, the source dataset consist of more than 1000 observations.

In their paper, Augustsson & Reker (2012) proposed a three-group discrimination scheme based on origin of the the rocks from which these quartz crystals came:

1. volcanic quartz
2. low-temperature metamorphic and mafic plutonic quartz
3. felsic plutonic, high-temperature metamorphic, and pegmatitic quartz

The different rock species that are represented in each group are listed in table 3.1.

Table 3.1: Overview over which rock species are represented in groups 1, 2 and 3.

Group 1	**Group 2**	**Group 3**
Andesite	2-mica granite	2-mica orthogneiss
Igimbrite	2-mica tonalite	Amphibole-bearing gneiss
Rhyolite	Biotite-hornblende-gnejs	Biotite gneiss
	Biotite granite	Blueschist
	Charnockite	Diorite
	Granulite	Metadiorite
	Leucosome	Metagranodiorite
	Pegmatite	Metatonalite
	Tonalite	Orthogneiss
	Wollastonite-bearing clinopyroxene hornfels	

In the part of this thesis where I have applied ML classification algorithms, those algorithms were trained and tested on two different datasets. The first dataset, hereafter referred to as the ***full* dataset**, is a subset of the source dataset where observations with missing values are removed (statistics on this dataset is summarized in table 3.2).

Table 3.2: Summary statistics on the full dataset.

	Peak 1, nm	Peak 1, counts	Peak 2, nm	Peak 2, counts	Trough, nm	Trough, counts	Peak 1, rel. pos.	Peak 2, rel. pos.	Peak 1, rel. int.	Peak 2, rel. int.
count	841	841	841	841	841	841	841	841	841	841
mean	480,04	494,21	617,12	359,08	560,68	306,60	0,86	1,10	1,63	1,29
std	7,01	265,66	12,20	174,91	22,65	155,35	0,04	0,05	0,45	0,61
min	467,50	36,00	559,40	47,00	496,80	34,00	0,76	1,01	0,35	0,59
25 %	474,80	306,00	608,10	243,00	543,10	201,00	0,83	1,06	1,27	0,98
50 %	478,00	472,00	616,80	335,00	567,50	283,00	0,84	1,09	1,64	1,12
75 %	483,70	641,00	625,70	436,00	574,20	379,00	0,88	1,13	1,96	1,38
max	502,00	1387,00	652,30	1330,00	628,00	1082,00	0,98	1,28	3,01	6,17

The second dataset, hereafter referred to as the ***reduced* dataset**, is a subset of the source dataset where in addition any observations that have a third peak in their CL spectra are removed (statistics on this dataset is summarized in table 3.3). The reason for these third peaks are likely due to influence from adjacent plagioclase (at ∼560 nm) and Fe^{3+} in feldspar (at ∼705 nm) (Augustsson & Reker, 2012), as illustrated in figure 3.1. Therefore, the results from the classification should reflect to what degree the additional noise from feldspar in the quartz CL spectra affect accuracy. All "counts" features in both datasets are corrected for background noise on a by observation basis. This is done by subtracting the magnitude of background noise from these feature values.

Table 3.3: Summary statistics on the reduced dataset where observations with a third peak in the CL spectra have been removed.

	Peak 1, nm	Peak 1, counts	Peak 2, nm	Peak 2, counts	Trough, nm	Trough, counts	Peak 1, rel. pos.	Peak 2, rel. pos.	Peak 1, rel. int.	Peak 2, rel. int.
count	394	394	394	394	394	394	394	394	394	394
mean	481,38	398,55	621,97	338,64	556,55	243,30	0,87	1,12	1,63	1,50
std	7,34	226,66	10,45	213,84	22,83	123,52	0,04	0,05	0,48	0,78
min	467,50	44,00	596,30	53,00	496,80	38,00	0,77	1,02	0,49	0,63
25 %	475,30	225,50	615,65	210,25	536,90	167,00	0,83	1,08	1,21	1,08
50 %	480,15	370,50	622,15	288,50	565,40	223,00	0,85	1,11	1,60	1,25
75 %	486,00	515,75	629,30	389,50	573,20	294,00	0,90	1,16	1,97	1,64
max	501,10	1328,00	652,30	1330,00	621,40	772,00	0,98	1,28	3,01	6,17

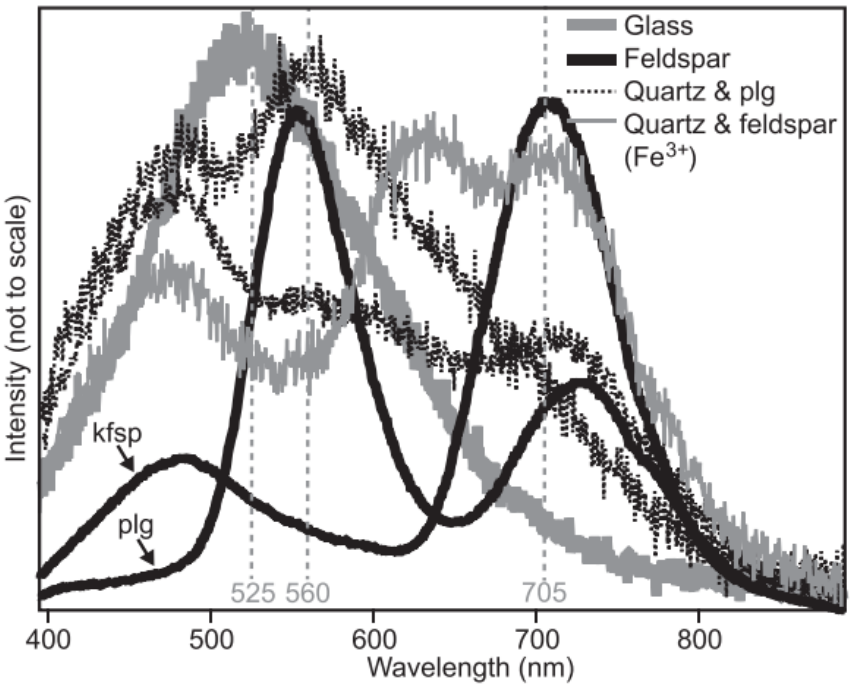


Figure 3.1: Source of a third peak in CL spectra of quartz: Spectra of plagioclase (plg), potassium feldspar (kfsp) and glass, as well as quartz spectra affected by Fe^{3+} in neighboring feldspar (Augustsson & Reker, 2012).

In both datasets, *relative peak intensity* is given by equation 3.1 and *relative peak position* is given by equation 3.2.

$$Peak_i,\ rel.\ int. = \frac{Peak_i,\ counts - Background\ Noise,\ counts}{Trough,\ counts - Background\ Noise,\ counts} \tag{3.1}$$

$$Peak_i,\ rel.\ pos. = \frac{Peak_i,\ nm}{Trough,\ nm} \tag{3.2}$$

Extreme outliers were also removed removed from the source dataset. The criteria for removing these observations was set to a *Peak 1, counts* > 1500, and a total of 40 observations were removed by this criteria. Of the 40 removed observations, 1 was from group 2 and 39 were from group 3. The cut–off value of 1500 for "peak 1 counts" was chosen because this removed the most extreme outliers, while still preserving the bulk of the observations as shown in the density plot in figure 3.2. The reason for

removing the outliers is to focus on how ML classifiers perform in the bulk of the observations, as extreme outliers in general are quite easy to classify correctly.

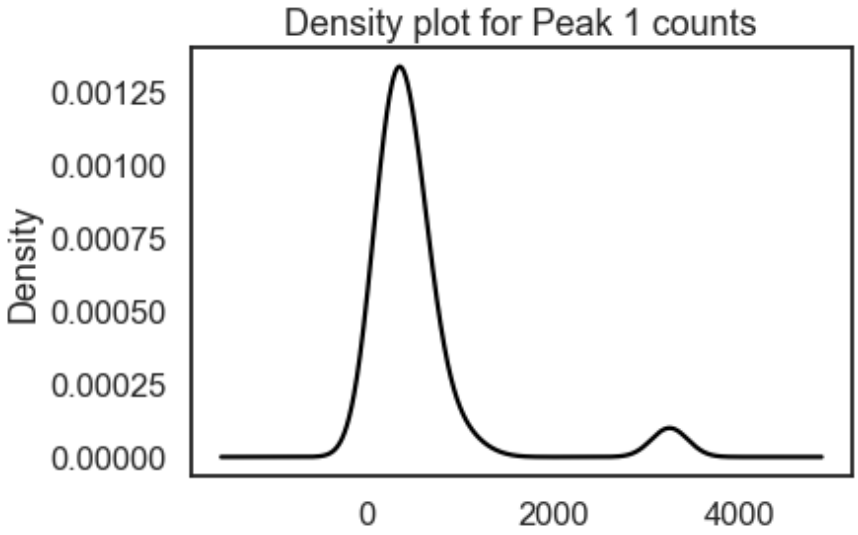


Figure 3.2: Density plot for "Peak 1 counts" in the source dataset.

Despite the full dataset being quite larger (n=841) than the reduced dataset (n=394), we see from table 3.4 that each groups relative representation in each dataset is close to similar.

Table 3.4: Total and relative group representation in each dataset.

	Full dataset		Reduced dataset	
Group	Number of obs. in dataset	Frac. of dataset	Number of obs. in dataset	Frac. of dataset
1	91	0,11	55	0,14
2	477	0,57	204	0,52
3	273	0,32	134	0,34

The histograms for the features in the full dataset, seen in figure 3.3, reveal that there is in general a high degree of overlap among the distribution of values attributed to the different groups. I.e., there are for the most part no clear ranges, for neither wavelength nor intensity, where solely values from a single group are present. The exception being the features *Peak 2 counts* and *Through nm* where Group 1 have some high-value observations not overlapped by observations from the other groups.

On the other hand, there is a tendency for separation in the bulk of the distribution of

values from group 2 and group 3. This is clearly seen in the features *Peak 1 counts* and *Trough nm*, which also carry over to the relative features, except *Peak 2 rel. int.*. Therefore, this is a early general indication that the features *Peak 1 counts*, *Peak 2 counts* and *Trough nm* are important when classifying quartz samples using ML models. Especially for three-based models which rely on value discrimination between classes on single feature when classifying.

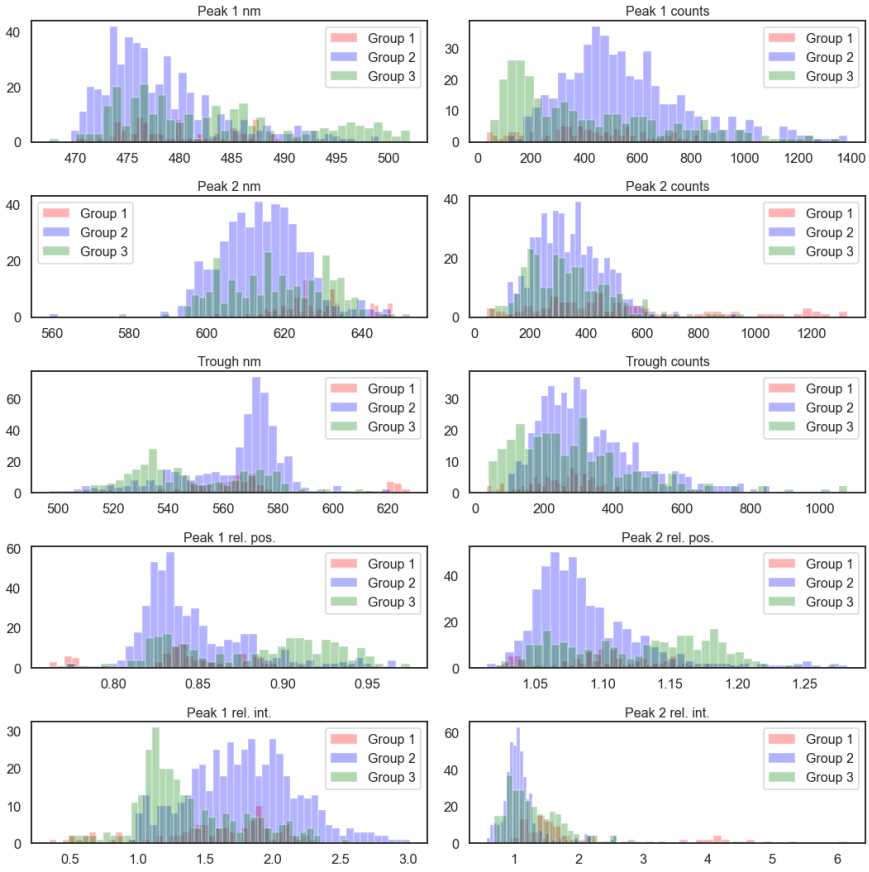


Figure 3.3: Histograms for all 10 features in the full dataset.

The histograms for the features in the reduced dataset, seen in figure 3.4, are similar to the histograms for the features from the full dataset, but show a comparably higher degree of separation among the distribution of values of the different groups. This increased separation is most evident between group 2 and 3. This show that removing all observations with influence from a third peak from the dataset, removed "noise". Therefore, this is an early indication that classification models based on the reduced

data set should do better in terms of accuracy than models based on the full dataset.

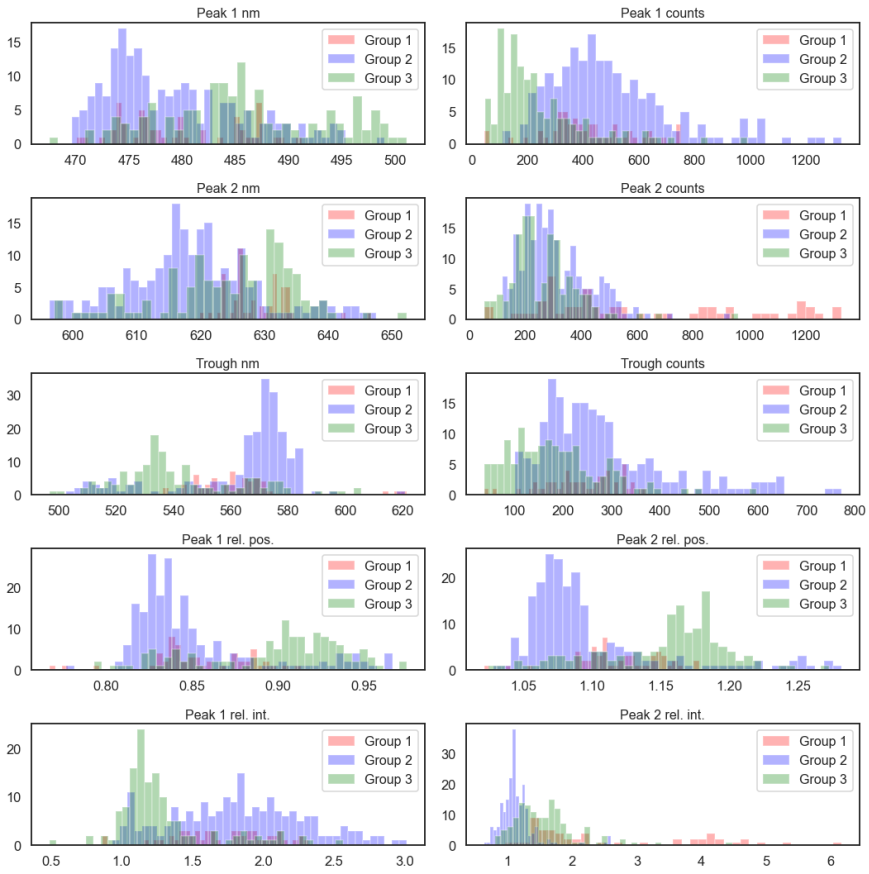


Figure 3.4: Histograms for all 10 features in the reduced dataset.

The PCA and ML classification algorithms are applied to four different feature subsets derived from each of the two datasets. The compositions of each feature subset is given in table 3.5.

Table 3.5: Features present in each the feature subsets the ML classification algorithms are trained on.

	Peak 1, nm	Peak 1, counts	Peak 2, nm	Peak 2, counts	Trough, nm	Trough, counts	Peak 1, rel. pos.	Peak 2, rel. pos.	Peak 1, rel. int.	Peak 2, rel. int.
Feature subset 1	●	●	●	●	●	●				
Feature subset 2							●	●	●	●
Feature subset 3		●		●		●				
Feature subset 4	●		●		●					

The reason for specifying these feature subsets give itself naturally as the features are of two types; features representing light spectrum (nm) and features representing intensity

(counts). By dividing features into subsets one can then examine whether one type is more important that the other to the models, and to what degree.

Further, and more important in this regard, is that it was shown in *Augustsson and Bahlburg* (2003) that (relative) height, or intensity, of the two major peaks in quartz CL spectra could be a good provenance indicator. This is tested if true for ML classification models as well.

4 | Principal Component Analysis (PCA) on the Datasets

The purpose of performing a principal component analysis on the datasets is to further examine the interrelations among the groups caused by the features. Histograms are good tool when examining the interaction, or overlap of values, between groups in single features. However, when features aggregate to data points in a higher dimension PCA may be used to examine group interaction in the dataset.

PCA is a dimensions reducing technique which allow one to observe the class interaction in a dataset in a dimension which can be visualized. This is done by transforming a dataset into principal components (PCs). The first two principal components account for most of the explained variance in a datasets, and creating a scatter plot from these two components result in what is known as a score plot (figures 4.2 and 4.5). The score plot can be used to assess the data structure of a dataset - to detect clusters, outliers, and trends among classes.

It may not be ideal to have too many features when applying classification algorithms as some features only add noise. Thereby reducing accuracy. In that regard, PCA can give an indication toward which features may be discarded, if any.

4.1 PCA on the Full Dataset (n = 841)

This dataset include observation where a third peak is present. Figure 4.1 give an overview over the cumulative explained variance with increasing number of principal components for each of the feature subsets. Noteworthy is that for feature subsets 1 the first two PCs explain a total of 69% of the variance of the dataset, and four PCs are needed to have >90% of the variance explained.
Further, for feature subset 3, the first two principal components explain a total of 96% of the variance in the dataset.

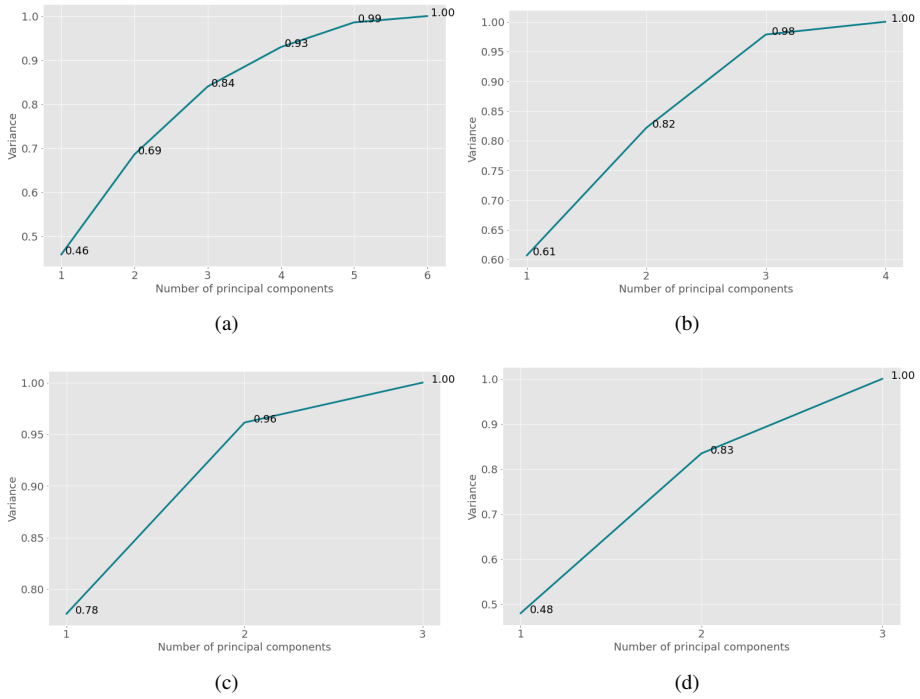


Figure 4.1: Cumulative explained variance as fraction of total explained variance, by principal components given PCA performed on (derived from the full dataset): **(a)** Feature subset 1: all features **(b)** Feature subset 2: relative features **(c)** Feature subset 3: only features representing intensity (counts) **(d)** Feature subset 4: only features representing wavelength (nm)

The score plots for each of the four feature subsets (figure 4.2) reveal that there exist a clear clustering tendency for observations belonging to the different groups, especially for groups 2 and 3. Though, observations from group 3 have a tendency to overlap with the clusters of group 2 observations.

Noteworthy is the score plot for feature subset 3 where not only is the the group clusters more concentrated than seen for the other feature subsets, but the border between group 2 and 3 is also sharper and less feathered with overlap. Even group 1 observations are to a degree segregated from the other group clusters. Since it is given by definition that the first two PCs for this feature subset account for 96% of the variance in the dataset, the resulting score score plot should be representative for the underlying group structure in this feature subset.

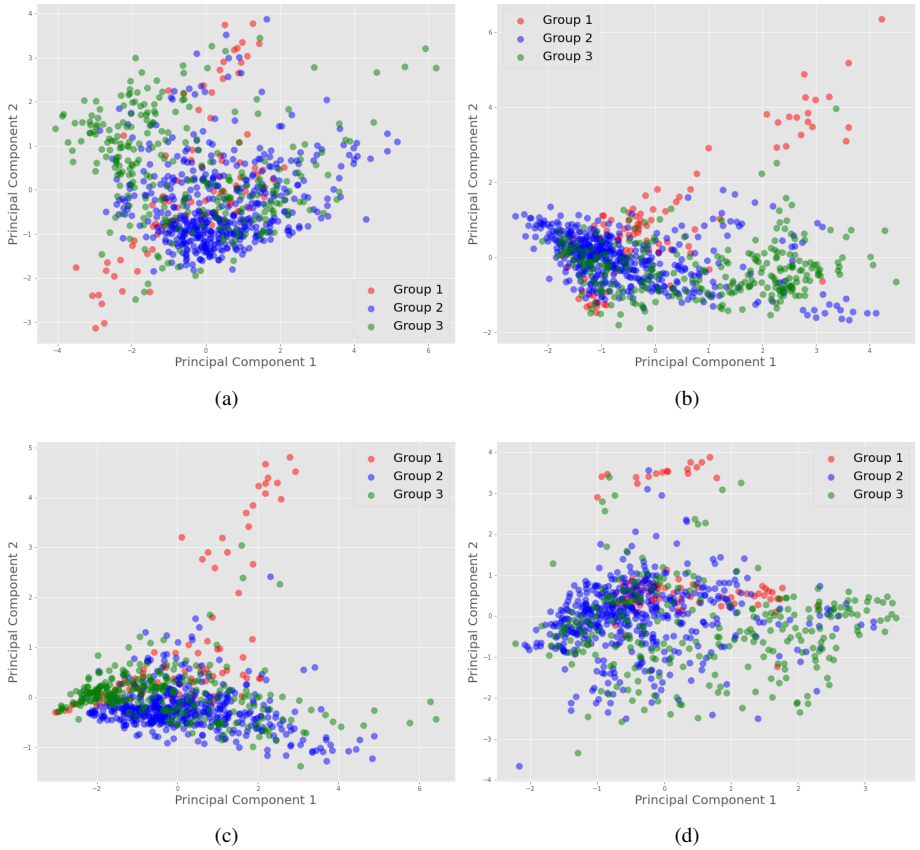


Figure 4.2: Projection of feature into 2 dimensions via features transformed into *Pricipal Component 1* and *Principal Component 2*, given PCA performed on (derived from the full dataset) **(a)** Feature subset 1: all features (explained variance by PC1 and PC2 is 69%) **(b)** Feature subset 2: relative features (explained variance by PC1 and PC2 is 82%) **(c)** Feature subset 3: only features representing intensity (counts) (explained variance by PC1 and PC2 is 90%) **(d)** Feature subset 4: only features representing wavelength (nm) (explained variance by PC1 and PC2 is 83%)

A bar plot of the factor loadings based on feature subset 1 is shown in figure 4.3. Loadings are important because they may be thought of as feature correlation with a given PC. Therefore, a large absolute loading for a feature onto PC1, which account for most of the explained variance of a dataset, translate into that feature accounting for a large proportion of the explained variance of the dataset.

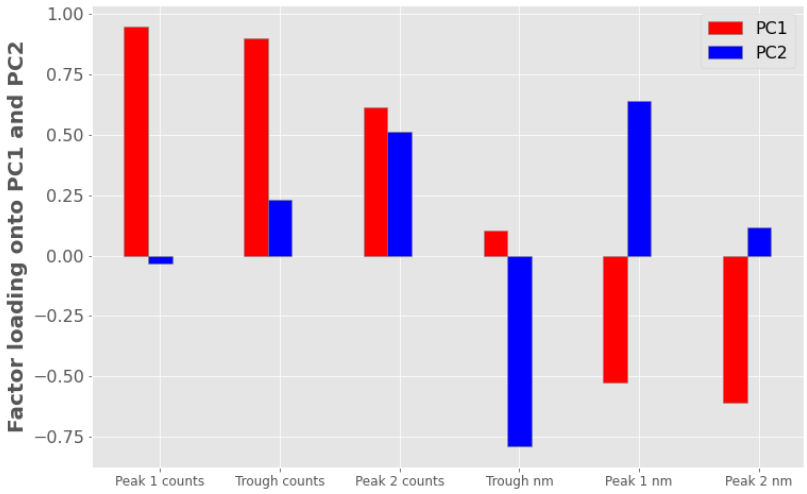


Figure 4.3: Factor loadings given PCA performed on feature subset 1 derived from the full dataset.

PC 1 is the most important PC as it explain the most variance. Almost all features show a relatively large absolute factor loading (> 0.5) onto PC1, with the exception being *Through nm*. Further, it is noteworthy is that the features representing intensity have positive loadings onto PC1, while features representing wavelength (except *Through nm*, which has insignificant correlation/loading with PC1) have negative loadings onto PC1. A principal component has one dimension, with a midpoint value of 0. Therefore, the sign of a loading tell which direction a features will push an observation on the line while the magnitude of the loading tell how much. With regard to classification, this indicate that features with significant factor load having the same sign, positive or negative, should be grouped and examined separated.

It has actually been suggested by some, for example *Hair et al.* (2014), to in general remove features from a dataset with factor loadings < 0.40. For this case, it would eliminate all features representing intensity.

4.2 PCA on the Reduced Dataset (*n*=384)

This dataset does not include observation where a third peak is present. Figure 4.4 give an overview over the cumulative explained variance with increasing number of principal components for each of the feature subsets. Again, it is noteworthy that for feature subsets 1 the first two PCs explain a total of 68% of the variance of the dataset, and the four first PCs are needed to have >90% of the variance explained.
For feature subset 3, the first two principal components explain a total of 95% of the variance in the dataset.

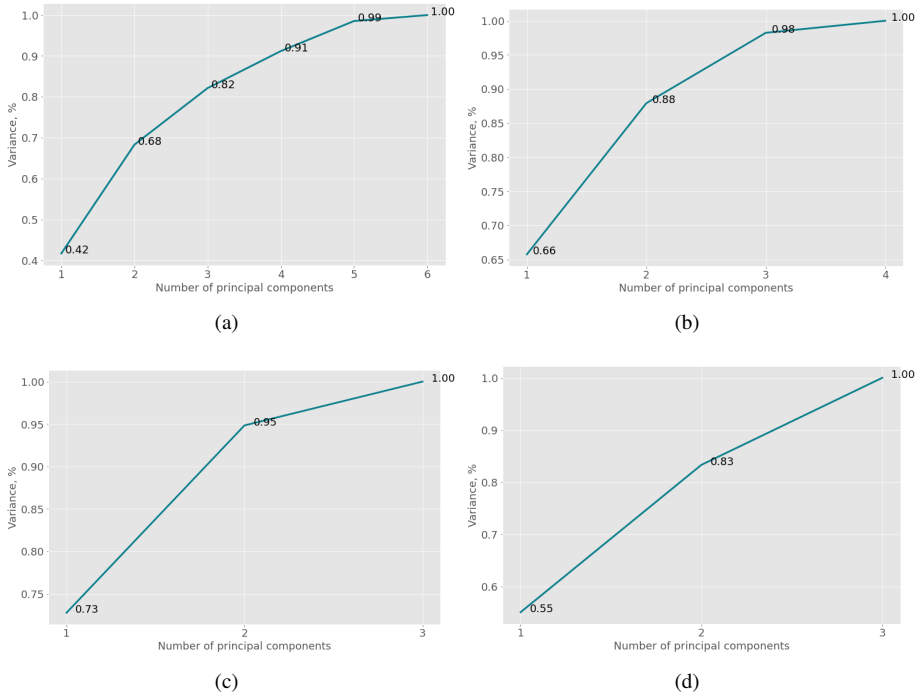


Figure 4.4: Cumulative explained variance as fraction of total explained variance, by principal components given PCA performed on **(a)** Feature subset 1: all features **(b)** Feature subset 2: relative features **(c)** Feature subset 3: only features representing intensity (counts) **(d)** Feature subset 4: only features representing wavelength (nm)

The score plots for each of the four feature subsets (figure 4.5) show much the same as did the score plots for all the feature subsets derived from the full dataset. What is different is less overlap among the group clusters, especially groups 2 and 3. The score plot for feature subset 2 now show a clearer segregation between groups 2 and 3, with less overlap among the clusters.

Just like the results from the histograms in chapter 3, PCA also indicate that removing observations from the dataset which have a third peak present in the CL spectra, reduce noise. This translate into an early indication that feature subsets from the reduced dataset should do better when fitted to ML classification models.

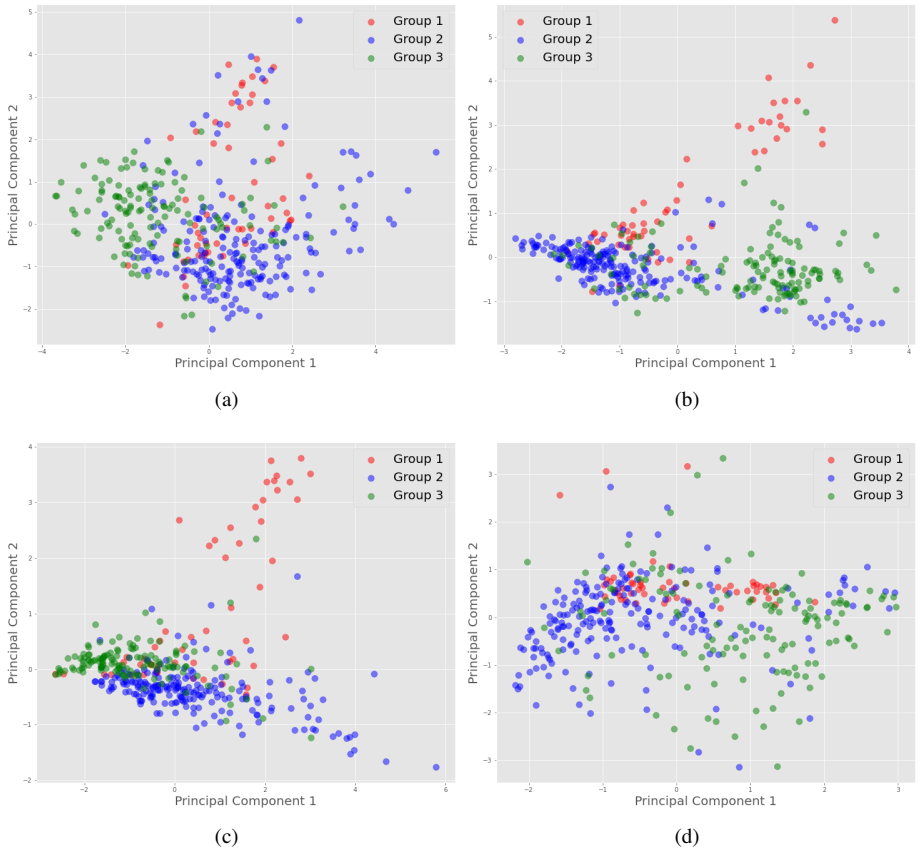


Figure 4.5: Projection of feature into 2 dimensions via features transformed into *Principal Component 1* and *Principal Component 2*, given PCA performed on **(a)** Feature subset 1: all features (explained variance by PC1 and PC2 is 68%) **(b)** Feature subset 2: relative features (explained variance by PC1 and PC2 is 88%) **(c)** Feature subset 3: only features representing intensity (counts) (explained variance by PC1 and PC2 is 95%) **(d)** Feature subset 4: only features representing wavelength (nm) (explained variance by PC1 and PC2 is 83%)

A bar plot of the factor loadings based on feature subset 1 is shown in figure 4.6. The result is much the same as seen for the full dataset in figure 4.3. The major difference now is that the feature *Through nm* have a significant positive factor loading (>0.5) onto PC1, while the factor loading for *Peak 2 counts* is now barely >0.4.

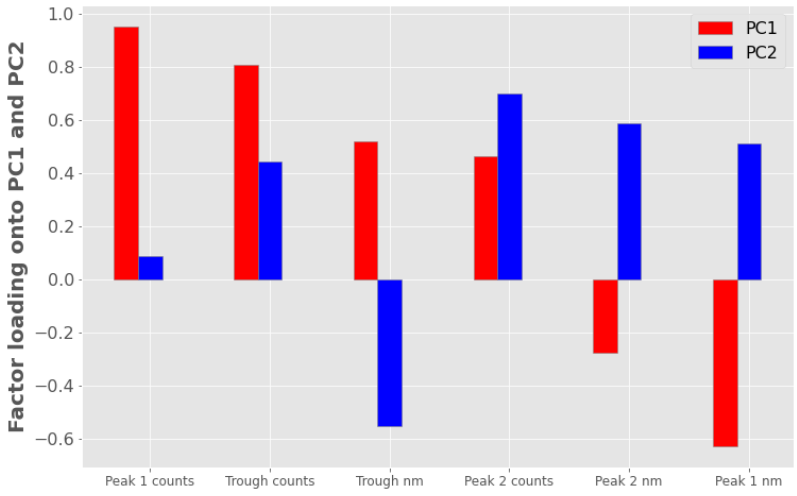


Figure 4.6: Factor loadings given PCA performed on feature subset 1 derived from the reduced dataset.

5 | Methods and Results

5.1 Filtering of Raw CL Spectra

When performing CL measurements, external light sometimes find its way into the area of measurement. An example of this is the large spike in intensity seen in figure 5.1 (a). Because the neural network base its curve fit on minimizing MSE relative to the CL spectra, it is thus necessary to remove such noise beforehand to ensure a good fit and accurate feature value extraction.

Initially, a few methods for noise filtering was evaluated. For example, one such method was to filter out data points from the CL spectra data that deviated a certain distance from a t–period rolling mean. The issue with this method being that a rolling mean is the average of the previous t–period data points, and for the rolling mean to not be too affected by the spikes one need the t–period to be a significant number. Therefore, the resulting curve given by the rolling mean would be shifted relative to the raw CL data points. This could potentially be corrected by back–shifting the rolling mean curve to fit with the CL data and then removing noise. The question was how robust this method would be as part of an automatic process. Likely, not very.

When exploring the method of filtering based on a rolling mean it was realized that neighbouring CL data points on average have a limited range of difference in intensity. This gave the idea to use the first–difference as a basis for filtering the raw CL data.

I propose to filtering the raw CL spectra data via the *first-difference* (equation 5.1) of the CL spectra data

$$\Delta(f)_i = f_i - f_{i-1}, \tag{5.1}$$

where f is a list of CL spectra intensity data with k data points, and $i \in [2, k]$.

The first-difference is the difference between one value and the next, and if that difference exceeds a user defined threshold S, the corresponding CL spectra data point are removed.

This method of filtering must be looped because often a spike can be comprised of several subsequent data points, and a single pass-through will only remove the first data

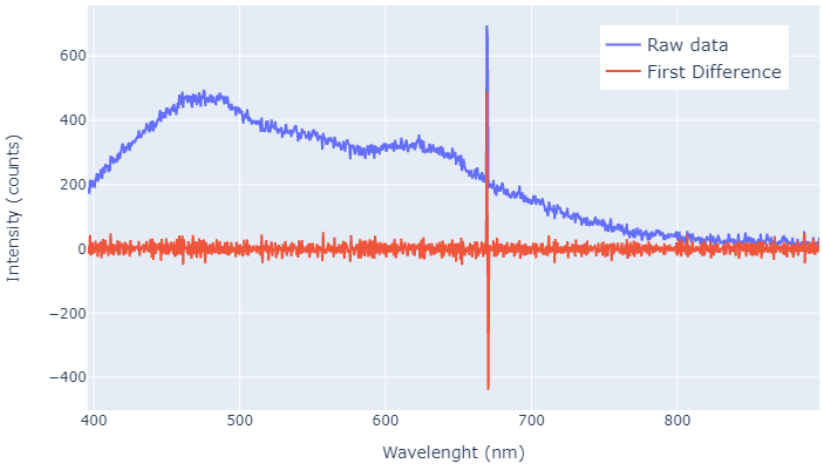
point of that spike.

Given a list f with CL spectra intensity data, algorithm 2 describe how noise if filtered out.

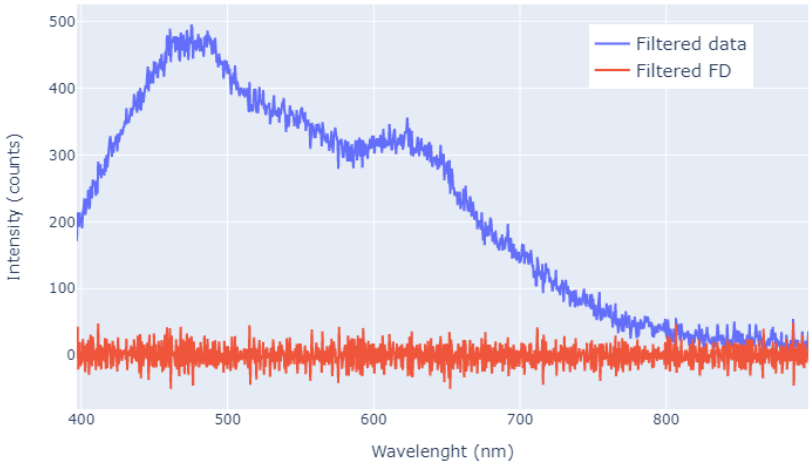
Algorithm 2 Filter (f)

Require: f: feature list, S: Sensitivity, k: $dim(f)$
Ensure: $\Delta(f)_i$ < S $\forall$ $i \in [2, k]$
  $m \leftarrow \{\}$
  $l \leftarrow dim(\{\Delta(f)_i | \Delta(f)_i > S\}_{i=2}^{k})$
  while $l > 0$ **do**
    $m \leftarrow \{f_i | \Delta(f)_i < S\}_{i=2}^{k}$
    $l \leftarrow dim(\{\Delta(m)_i | \Delta(m)_i > S\}_{i=2}^{k})$
  end while

This filtering algorithm does not replace the removed data points, because the number of data points removed will be very small compared with the total amount of data points in the CL data. Thus, removing a small number of data points will not affect the the result of the curve fitting via neural networks described in chapter 5.2.



(a)



(b)

Figure 5.1: (a) From an example dataset where a large spike in intensity caused by external light is seen at around 670 nm. Before the data can be fitted with a curve via neural network, any major noise must be removed. (b) After filtering the raw data. A filtering threshold of $S = 52$ was used in this case.

This method of filtering should be a robust as long as the first-difference of the CL

spectra data exhibit at least second-order stationarity (also called weak stationarity). This is when the series have, among other, a constant mean and variance. In the case that the criterion of second-order stationarity does not hold, one risk filtering out data points from the CL spectra that is not noise since the threshold value (S) a constant.

5.2 Curve Fitting Using Neural Network

When it comes to neural network architecture there are no clear rules as to how many hidden layers to apply, and how many nodes to put into each layer. In general, simpler models are more robust to noise in the inputs due to risk of over-fitting.
One tip, or rule of thumb, I've encountered multiple times on different ML forum online, is to start with one hidden layer and to err on the side of more rather than less nodes in the hidden layer - a few extra nodes in the hidden layer isn't likely do any any harm. On the other hand, too few nodes in the hidden layer can prevent convergence[1].
With this method on build a network progressively, keep adding layers and nodes, while keeping track on some relevant measurement. In this case, MSE.
We can see this method implemented in table 5.1. In the end, a neural network with 4 hidden layers and 30 nodes in each layer stood out as a sensible choice based on MSE.

The ELU activation function was chosen for this neural network due to its benefits over other activation functions (see section 2.4.1 for details).

[1] Convergence describes a progression towards a network state where the network has learned to properly respond to a set of training patterns within some margin of error.

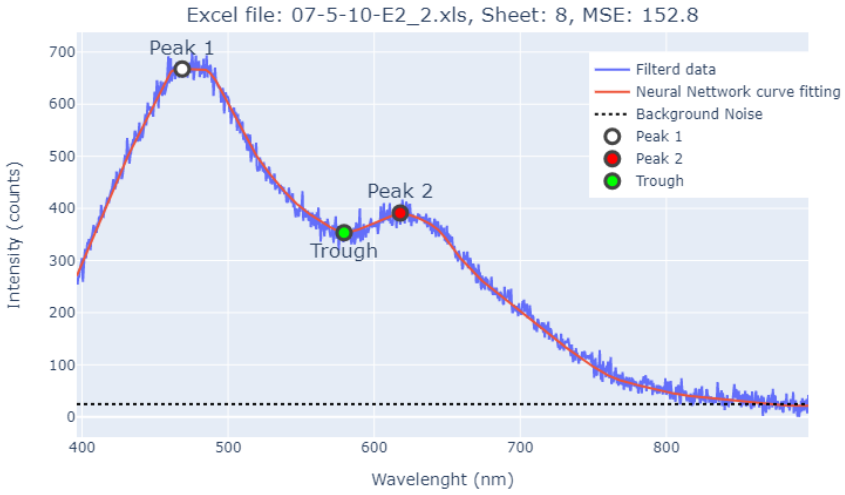
Table 5.1: Using MSE as a metric to decide the best architecture for the neural network. The tests were performed using 5000 epochs (the number of times the algorithm runs on the whole training dataset), a learning rate of 0.005 for the "Adam" optimizer and the ELU activation function. The values in blue are the final configuration of the neural network.

Hidden layers	**Nodes per layer**	**Mean square error**
1	10	536
1	50	159
1	75	185
2	50	163
3	50	140
3	40	153
4	50	151
4	40	142
4	**30**	**133**
4	25	132
4	20	145
5	25	156

The result for a curve fitted to a CL spectra by a neural network with the specified architecture (4 hidden layers, and 30 nodes per layer), is seen in figure 5.2. We observe that the neural network is just complex enough to capture the shape of the CL spectra without over-fitting. This neural network architecture has been tested on $\sim$10 different CL spectra and have yielded good results similar to the on seen in figure 5.2. This is an indication that the chosen network architecture is robust with respect to repeatability of result.

The total fit–time for the neural network given 5000 epochs is close to 100 seconds, but there was in place a clause in the algorithm, a stopping criteria, that the fitting should terminate if no improvement in loss was seen for 2500 epochs.

The number of epochs can likely be reduced significantly and thereby improving the fit–time. The reason for this claim is seen in figure 5.2 (b): We see that the neural network quickly converges toward the minimum loss value (MSE). After around 1000 epochs the loss value is very close to minimum, and only a slight gradual improvement is seen over the next 4000 epochs as the neural network continue to adjust parameters in search of a better fit. At the point of writing this thesis there have not been sufficient time to thoroughly test whether stopping at a lower number of epochs will yield robust results in the long run (explained in detail in section 5.3).



(a)

(b)

Figure 5.2: (a) Curve fitting using a neural network with 4 hidden layer and 30 nodes per layer. The architecture of this model is capable to capture the complexity of the underlying data without over fitting. (b) The loss plot show that the model with 4 hidden layer and 30 nodes per layer quickly converges toward a low level MSE. The loss plot also show that there is a small but steady improvement with increasing epochs.

5.3 Feature Value Extraction from Neural Network Fitted Curve

The purpose of fitting a cure by neural network to the CL spectra data is so that feature values used to classify the quartz sample on can be extracted. The features of interest are:

- **Peak 1 and 2**: intensity (counts) and wavelength (nm)
- **Trough**: intensity and wavelength
- **Background noise**: intensity

Background noise (BG) intensity is used to correct the value of the features *Peak 1 counts*, *Peak 2 counts* and *Through counts*. This to ensure the intensity of each CL spectra is evaluated with the same point of reference.

The CL curves are not always ideally shaped as seen in figure 5.3. Sometimes they may present as seen in figure 5.4.

When the CL curves are ideally shaped as in figure 5.3, the values of peak 1 and 2 are taken as the maximum value of the curve fitted by the NN, inside set intervals. These intervals are set based on where the location, the wavelength, for these peaks is expected to be. The values for these intervals may be updated inside the Jupyter notebook if needed by a user. The values for the through is taken to be the minimum of the NN cure inside an interval given by the maximum of peak 1 interval and the minimum of peak 2 interval. Lastly, the intensity of the background noise is defined to be the average intensity of the NN curve at wavelengths greater than 850 nm. This value of 850 nm is set as a default value because some tests on random CL spectra data indicate that this value yield a good general fit. This value may also be updated inside the Jupyter notebook if needed by a user.

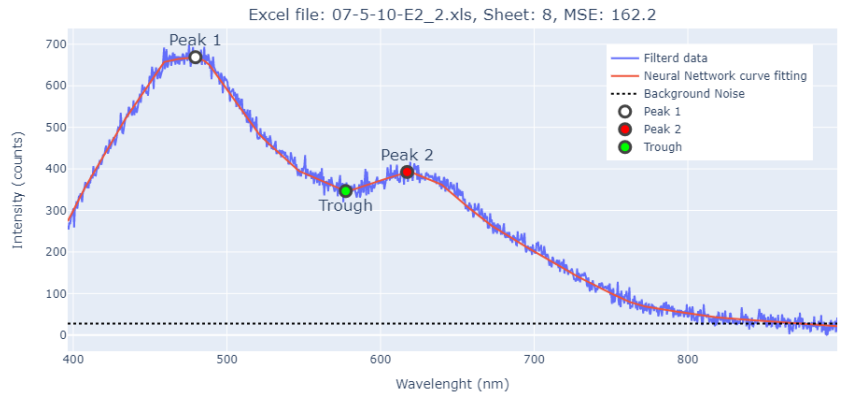


Figure 5.3: Ideally shaped CL spectra for feature value extraction, where values are extracted based on minimum and maximum intensity values inside certain wavelength intervals.

In the case when a CL curve has a shape similar to the one seen in figure 5.4, the method of automatically extracting the values for peak 1 and 2, and the through must be different than the one described previously. The reason being that these values are not defined by local minimum or maximum values inside set intervals.

Therefore, I propose to use the second derivative of the NN curve, still within set intervals, to decide these values. This method should work as long as NN curve does not overfit the CL data, but rather underfit to a small degree. This causes the NN curve to be made up of somewhat piece–wise linear elements in the intervals of interest, and the location of peak 2 (or peak 1) and the through is guided by spike–values of the second derivative of the neural network curve.

In figure 5.4, for example, the through wavelength is taken to be the point where rate of change of the NN cure suddenly increases inside the interval [570 nm, 600 nm] (user defined). This result in a $+ve$ spike in the second derivative. For peak 2 wavelength we look for a point in the NN curve where the rate of change suddenly decreases inside the [610 nm, 630 nm] (user defined). This result in a $-ve$ spike in the second derivative. The intensity of background noise is still defined as described previously.

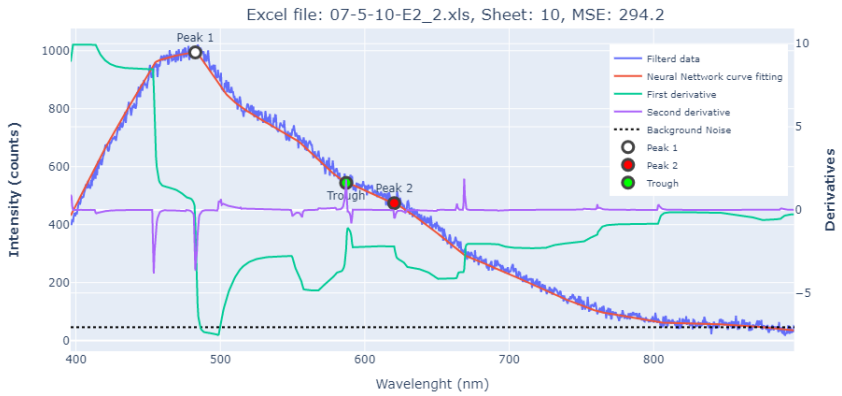


Figure 5.4: Example of a CL spectra shape that pose a challenge for extracting feature values, and it is proposed to use the second derivative of the curve fitted to the CL spectra to guide feature value extraction.

The Jupyter notebook include script for both methods for feature value extraction, and the user may choose method based on the shape of the CL spectra.

5.4 Group Prediction for Quartz Sample

Both the full and the reduced datasets were split into a training set and a testing set at a ratio of 1/3. I.e., 1/3 of a given dataset was used to test the model after it had been trained on the training set. A random state $= 6$ was sat for the split to ensure that the datasets were split the same way each time, and thus the results are comparable.

The random state for the split was chosen such that the fractional representation of each group in the testing sets was close to equal that of the initial datasets (see tables 5.2 and 5.3).

Table 5.2: Representation of each group in the training dataset of the full dataset.

	Observations in dataset before split	Observations in test set	Representation of groups in test set relative to the full dataset
Group 1	91	33	106,50 %
Group 2	477	159	100,00 %
Group 3	273	89	97,80 %
Sum	841	281	

Table 5.3: Representation of each group in the training dataset of the reduced dataset.

	Observations in dataset before split	Observations in test set	Representation of groups in test set relative to the reduced dataset
Group 1	55	18	94,70 %
Group 2	205	71	102,90 %
Group 3	134	43	95,60 %
Sum	394	132	

Table 3.4 in chapter 3 describing the datasets, revealed one major and important issue: The datasets are imbalanced. This mean that there is an unequal number of observations representing each group in the datasets. Group 1 has the lowest representation in both the full dataset and the reduced dataset with 11% and 14% of observations, respectively. Due to the choice of random state for the split into training and a testing datasets, these group representations are close to preserved after the split as seen in tables 5.2 and 5.3.

Most ML classification algorithms are designed around the assumption of balanced datasets (Lemaître, Nogueira, & Aridas, 2017), i.e., an equal fractional representation from each class in the datasets. When this is not the case, it may lead to biased accuracy results. For example, for a classification algorithm like the kNN algorithm a domination group will naturally have more potential neighbours represented than the other groups, thereby inflating the accuracy for that dominant group.

5.4.1 kNN Model

The kNN algorithm was trained using a 10–fold cross–validation, accompanied with a grid search over $k \in [1, 15]$ to find the optimal value for k.

Starting with the feature subsets derived from the **full dataset**, table 5.4 show the results for a kNN model trained and tested on each of these four feature subsets. The best performance was seen when the algorithm was trained on feature subset 1 and 2 (both had an accuracy of 77%), and in both cases without scaling of the features as well as uniform weights on the neighbour points.

Table 5.4: Accuracy and optimal number of *k* neighbours for the kNN algorithm trained on each of the four feature subsets derived from the full dataset.

	Weights	Accuracy, No scaling	Accuracy, With Scaling	k neighbours, No scaling	k neighbours, With scaling
F. subset 1	uniform	0,77	0,75	5	4
	distance	0,76	0,73	5	6
F. subset 2	uniform	0,77	0,76	6	9
	distance	0,74	0,73	7	3
F. subset 3	uniform	0,74	0,76	5	5
	distance	0,72	0,76	7	6
F. subset 4	uniform	0,7	0,69	6	8
	distance	0,7	0,68	9	10

When the algorithm was trained on feature subset 3, it achieved an accuracy of 76% when including feature scaling. This is almost as good as the best cases which achieved an accuracy of 77%. This conclude that the features representing wavelength does not add much in terms of predictive power to the model.

Permutation feature importance results shown in figure 5.5, which is based on feature subset 1 derived from the full dataset, reveal that the most important features in term of accuracy are the three representing intensity. I.e., feature subset 3. Some importance is also put on the wavelength of the trough (*Though nm*). Referencing the histograms in figure 3.3, one commonality seen in the histograms for these features are, despite a substantial overlap among the groups, that there is more separation between the groups compared to the other features. One can speculate that adding the feature *Through nm* to feature subset 3 may yield a best performing model. This was also indicated by the factor loadings from the PCA, not for this full dataset, but for the reduced dataset,

where *Through nm* showed a relatively large positive factor loading onto PC1.

Further, the score plot in figure 4.2 (c), which resulted from PCA performed on feature subset 3, show a high degree of group clustering without too much overlap among the groups.

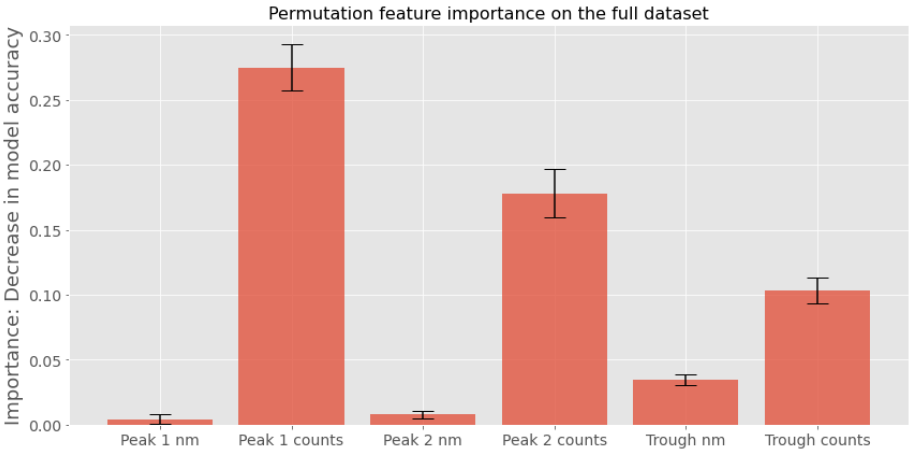


Figure 5.5: Permutation feature importance from the kNN algorithm fitted on the test set of feature subset 1 derived from the full dataset. The error bars show one standard deviation of uncertainty in the importance means.

The kNN algorithm was then trained and tested on each of the four feature subsets derived from the **reduced dataset**. The results are given in table 5.5, and show a clear improvement in the accuracy compared to when trained and tested on the full dataset. The best performance was now achieved when the algorithm was trained on feature subset 3 (accuracy of 84%), with no feature scaling and with uniform weights on the neighbour points. This is an improvement of 10% compared to the equivalent model configuration given the full dataset.

Table 5.5: Accuracy and optimal number of k neighbours for the kNN algorithm trained on each of the four feature subsets derived from the reduced dataset.

	Weights	Accuracy, No scaling	Accuracy, With Scaling	k neighbours, No scaling	k neighbours, With scaling
F. subset 1	uniform	0,80	0,83	9	3
	distance	0,83	0,83	10	3
F. subset 2	uniform	0,80	0,80	8	5
	distance	0,80	0,80	10	6
F. subset 3	uniform	0,84	0,81	4	4
	distance	0,82	0,82	10	10
F. subset 4	uniform	0,73	0,73	8	10
	distance	0,72	0,73	5	10

Permutation feature importance results shown in figure 5.6, which is now based on feature subset 1 derived from the reduced dataset, put close to all the importance on the three feature representing intensity. Almost none importance is now given to the feature *Though nm*.

An even higher importance is now put on the feature *Peak 1 counts* at close to 35% compared to about 27% from the full dataset. Referencing the histograms for the reduced dataset in figure 3.4, this result is not very surprising as the histogram for the feature *Peak 1 counts* show an increase in separation between group 2 and group 3 after observations with a third peak have been removed.

Histograms for the features alone is probably not the best indicator for feature importance for the kNN algorithm. The reason being that the dimension where the observations are plotted is equal to the number of features in the dataset. Therefore, one cannot look at individual features by themselves. For example, some of the features representing wavelength also show some separation between the values from the different groups, but close to no importance is given to them by the kNN model.

Because of this, a dimension reducing technique like PCA will likely give a better early indication of feature importance by testing different feature subset combinations, and observing class clustering and separation in either 2- or 3-dimensional score plots. The cumulative explained variance by the principal components must also be kept in mind, since this will dictate how representative what is observed in the score plot is for the actual structure in the dataset.

The high importance given to the feature *Peak 2 counts* is likely due to this feature

being the only one where group 1 have a relatively high degree of separation from the other groups.

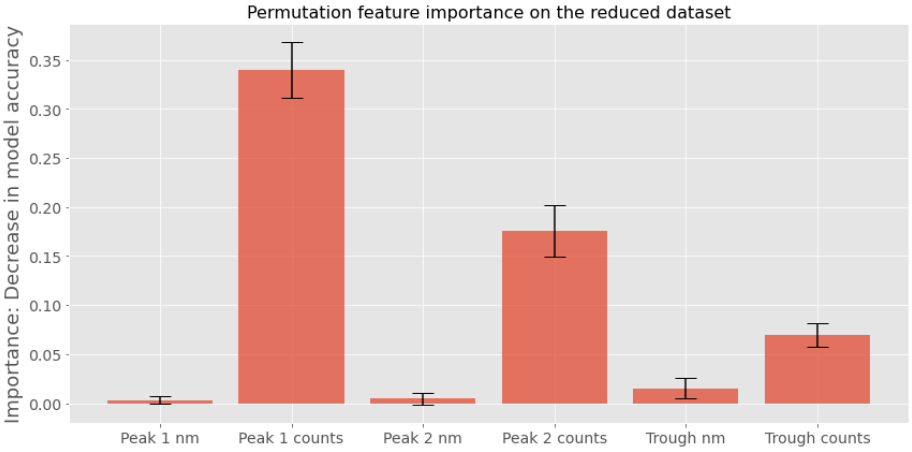


Figure 5.6: Permutation feature importance from the kNN algorithm fitted on the test set of feature subset 1 derived from the reduced dataset. The error bars show one standard deviation of uncertainty in the importance means.

The confusion matrix for the best performing model configuration, i.e., feature subset 3 derived from the reduced dataset, is shown in table 5.6. The highest f1–score was seen for group 2 at 87%, and the lowest f1–score seen for group 1 at 69%. These results must be interpreted with consideration to the datasets being imbalanced: The result show a correlation between the f1–score and the number of observation for a given group.

Table 5.6: Confusion matrix for the kNN model configuration with the highest accuracy: Feature subset 3 from the reduce dataset with uniform weights and no feature scaling.

	Precision	Recall	F1-score	Support
Group 1	0,79	0,61	0,69	18
Group 2	0,83	0,92	0,87	71
Group 3	0,88	0,81	0,84	43
Accuracy			0,84	132
Macro avg.	0,83	0,78	0,80	132
Weighted avg.	0,84	0,84	0,84	132

It is therefore reasonable to assume that the accuracy from the confusion matrix is not

truly representative, that it is biased due to an imbalanced source dataset. Best guess is that the accuracy may be a bit lower than it would be for a balanced dataset. The f1–score for group 1 could have been lowered due to few observations (n = 18), while on the other hand the f1–score for group 2 could be a bit elevated due to relatively many observations (n = 71). A more balanced dataset may therefore lead to improvement in both precision and recall, especially for group 1 and group 3.

In addition, 40 observations were excluded form the source dataset due to their relative extreme values. They were removed in order to better observe how the ML classification algorithms were able to predict group label in the bulk volume of the group observations. A total of 39 of these removed observations were from group 3, and if included should be quite easy for the kNN algorithm to classify correctly. Thereby increasing the scores for group 3 along with the overall accuracy.

5.4.2 Random Forest Model

For each feature subset the random forest algorithm was trained on, the hyperparameters were tuned using a random grid search. It was opted to use a random grid search instead of the standard grid search, as was applied to the kNN algorithm, due to the shear number of possible parameter configuration for the random forest classifier ($p = 38,880$) (see appendix A.2 for details on hyperparameter distributions). Meaning that the algorithm would have been trained and tested p-times in order to find the best hyperparameter settings using a grid search. Therefore, instead of performing an exhaustive and time consuming grid search over all possible p parameter configurations, a fixed number of parameter settings was sampled from the specified hyperparameter distributions. In the following, 500 random parameter configurations were randomly sampled and tested.

Just like for the kNN algorithm, a 10–fold cross–validation was also performed for the random forest algorithm. Therefore, should one have performed a standard grid search for hyperparameter tuning, one would have performed close to 400,000 iterative fits per instance of a feature subset the model was fitted on. By applying the random grid search approach, the number of iterative fits were reduced to 5,000.

Table 5.7 give the resulting accuracy score for each subset configuration fitted to the random forest algorithm. Just like for the kNN algorithm, the feature subsets derived

from the reduced dataset performed the best in terms of accuracy. Although, for feature subset 1 the difference between the full dataset and the reduced dataset is marginal.

The table also show that there is little difference between the three best resulting subsets fitted to the random forest algorithm: Feature subsets 1,2 and 3 derived from the reduced dataset. And again, this show that the most important features for random forest algorithm as well, are the ones representing intensity (*Peak 1 counts*, *Peak 2 counts* and *Through counts*), since one can remove all but these features without lowering the accuracy score. The likely reason that feature subsets 1, 2 and 3 derived from the reduced dataset give so close to the same accuracy score is because all have the features representing intensity in them. Either directly or indirectly. Further, the random forest algorithm is less sensitive to additional "noisy" features in a dataset than the kNN algorithm because it tries to classify on single features and not points in a higher dimension (dimension equal to number of features) represented by all features in a dataset.

Table 5.7: Accuracy score results for the random forest algorithm trained and tested on each of the four feature subsets derived from the full and the reduced dataset, respectively.

	Accuracy, full dataset	Accuracy, reduced dataset
F. Subset 1	0,79	0,80
F. Subset 2	0,76	0,81
F. Subset 3	0,72	0,81
F. Subset 4	0,70	0,76

The permutation feature importance results for feature subset 1 derived from the full dataset (figure 5.7) is not as decisive with regard to feature importance as the equivalent feature importance for the kNN models. Still, the highest importance is given to the feature *Peak 1 counts*. It is also important to notice the scale of importance. Despite the feature *Peak 1 counts* given to be the most important feature, it only accounted for an average of $\sim$11% decrease in accuracy.

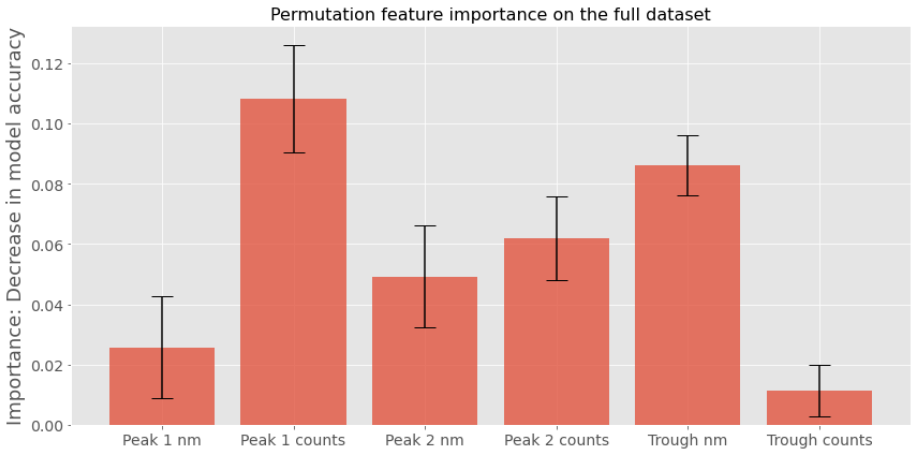


Figure 5.7: Permutation feature importance for the random forest algorithm fitted on the test set of feature subset 1 derived from the **full dataset**. The error bars show one standard deviation of uncertainty in the importance means.

The permutation feature importance results for feature subset 1 derived from the reduced dataset (figure 5.8) give almost the same result as for the full dataset, but now almost non importance is given to the features *Peak 1 nm* and *Though counts*. Also, the importance now given to feature *Peak 1 counts* is on average ∼20%, almost twice that for the full dataset. These results correlates with what can be observed in the feature histograms in figure 3.4, and they are not surprising given how a decision tree algorithm make its predictions based on individual features.

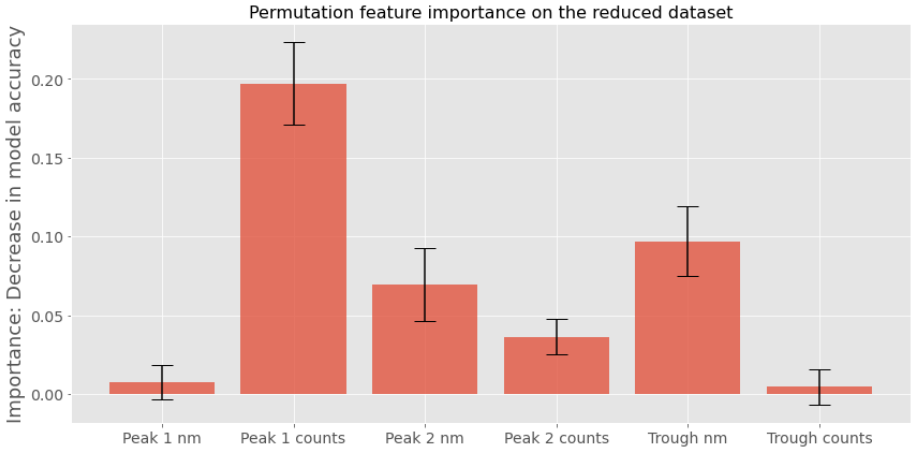


Figure 5.8: Permutation feature importance for the random forest model fitted on the test set of feature subset 1 derived from the **reduced dataset**. The error bars show one standard deviation of uncertainty in the importance means.

Although feature subset 2 and 3 from the reduced dataset had the same accuracy score, group 1 and 3 in feature subset 3 had a significant higher f1–score. Which came at a cost of a slightly lower f1–score for group 2. Therefore, feature subset 3 was selected as the best performer. This is convenient as it allows for a more direct comparison between the best performing kNN model and the best performing random forest model. The confusion matrix for this feature subset is given in table 5.8:

Table 5.8: Confusion matrix for the random forest model with the best performance and highest accuracy: Algorithm trained on feature subset 3 derived from the reduced dataset.

	Precision	Recall	F1-score	Support
Group 1	0,85	0,61	0,71	18
Group 2	0,82	0,87	0,84	71
Group 3	0,79	0,79	0,79	43
Accuracy			0,81	132
Macro avg.	0,82	0,76	0,78	132
Weighted avg.	0,81	0,81	0,81	132

Compared to the best performing kNN model the random forest model did quite a bit better at correctly classifying group 1 (precision $= 0.85$), but still have quite a few of group 1 observations wrongly classified as other groups (recall $= 0.61$). Still, the precision of 0.85 is an improvement compared to the kNN model which achieved a precision of 0.79 for group 1. This result can likely be linked to the feature *Peak 2 counts*, as group 1 have more extreme values in this feature compared to the two other groups. Since the random forest algorithm can attribute these extreme values directly to group 1, the error is reduced and the accuracy is not afflicted in the same way by an unbalanced dataset as the kNN algorithm.

The random forest model did a bit worse when it comes to classifying group 2 and 3 compared to the kNN model, witch can be blamed on the high degree of value overlap for these groups throughout the features in general. This is especially notable in the precision of group 3 which achieved a score of 0.79 in the random forest model, compared to 0.88 for the kNN model.

6 | Conclusions and Future Work

6.1 Conclusions

Machine learning has proven its potential to be a good tool for automating the process of evaluating CL spectra from quartz.

The filtering algorithm for raw CL spectra I propose is simple in its method and fast to run, but most important, robust with regard to repeatability of good results. The algorithm depend on one user input, S, which state the cut-off value for data points (noise) to filter out from the CL spectra. A method for automatically setting the value of S could quite possibly be found. This would then fully automate the filtering step.

The architecture of the neural network used to fit a curve to CL spectra has yielded good results. By good result is meant that a curve fitted has not been observed to over- or under-fit CL spectra, while at the same time achieving a low MSE measured on the fit. These are important criteria as they ensure that accurate feature values are extracted from CL spectra.
The current run-time for fitting a curve to CL spectra is $\sim$100 seconds. This time can likely be reduced significantly by reducing the number of epochs, which is currently set to 5000. Data on the fit show that the NN model quickly converges to a low MSE at around 1000 epochs, and only yield a small gradual improvement until 5000 epochs have been performed. It must be examined whether the fit given at around 1000 epochs is good enough for accurate feature values to be extracted.
The architecture of the neural network and hyperparameters could likely also be improved upon to further reduce the run-time.

I propose two methods for feature value extraction from a curve fitted by neural network to CL spectra. The first method is used when CL spectra is ideally shaped, i.e., it have two clearly defined peaks in intensity with a though in between them. The feature values are in this case extracted based on maximum and minimum intensity inside user defined wavelength intervals for the CL spectra.
The second method is used when a peak in a CL spectra is muted, almost non-existing. This also affect the through. In this case I propose to use the second derivative of the curve fitted by the neural network to find the feature values. This is an important reason

why it must be examined whether terminating a curve fit by the neural network after about 1000 epochs can be done. Ideally, the curve should be made up of piece-wise linear elements in the interval where the feature values are to be extracted, as the wavelength of a peak and the through is taken to be where there is a sudden change in the slope of the curve inside a set wavelength interval. Terminating the curve fit too early could cause the curve to under-fit the CL spectra such that the peak and the trough is inaccurately defined or not defined at all.

The ML classification algorithm which achieved the highest overall accuracy was the kNN algorithm with a model score of 84%. It achieved this when trained and tested on a feature subset of the reduced dataset containing only features representing intensity. Overall, when this algorithm was trained and tested on subsets from the reduced dataset it achieved 5-10 % higher accuracy compared to when trained and tested on subsets from the full dataset. This result was indicated prior to ML classification being performed by histograms for the feature values and PCA. Both the histograms and the score plots from the PCA showed significantly less overlap between feature values and observations among the groups for the reduced dataset. This conclude that CL spectra of quartz having been influenced by other minerals, e.g. feldspar, causing a third peak in the CL spectra, is negative for ML classification accuracy.
Further, the confusion matrices showed that the group that did worse overall in term of accuracy, was group 1. The group that did the best was group 2. Much of this can likely be attributed to imbalanced dataset. There is evident a correlation between the accuracy for a group (F1-score) and the fractional group representation in the dataset. Group 1 representation was 11% and 14%, while group 2 representation was 57% and 52% in the full dataset and the reduced dataset, respectively. Therefore, it is believed that the accuracy results are a bit reduced for this reason.

The accuracy results given by the best performing random forest models are quite similar to the results for the best performing kNN models. The dataset that, when the random forest algorithm was trained and tested on, yielded the models with the highest accuracy, again was feature subset 3 from the reduced dataset. This model achieved an accuracy of 81%, which is almost the same accuracy given by models based on feature subset 1 and 3 from the reduced dataset. The difference in accuracy for models based on subsets from the reduced dataset, compared to models based on subsets from the full dataset, is on average $\sim$5% higher.

The conclusion given by both ML classification algorithms is that these algorithms only need the features representing intensity to make their prediction on group for a quartz sample. This is evident in the difference in accuracy between feature subsets 1 and 3, as removing the features representing wavelength increase accuracy. Though, there are indications that including the feature *Through nm* to feature subset 3 may increase accuracy for both algorithms. Especially for the reduced dataset.

The accuracy results given by the ML classification models and the conclusion regarding only needing features representing intensity for making group prediction, are in line with what was reported by Augustsson & Reker (2012). They achieved a classification accuracy of 87% based on relative intensities of CL spectra (equation 3.1).

6.2 Future Work

I have next listed some moments that I believe can progress the work already done in this thesis further.

- Improve on the imbalanced dataset, i.e., add more group 1 and 3 observations to the source dataset.
- Improve run-time for the curve fitting to CL spectra by neural network.
- The score plots from the PCA show that observations from the same groups in general cluster together. Although, there is some overlap among the groups. Techniques, for example from unsupervised ML, could be used to examine if a different grouping of source rocks could yield better results with regard to classification.
- Examine whether adding the feature *Through nm* to feature subset 3 increase accuracy for the ML classification models.
- The classification accuracy may be increased by using other ML classification algorithms, or by including more algorithms and then putting them into an ensemble.
- Collect all the python code written for this thesis in an application.

Bibliography

Aggarwal, C. C. (2018). *Neural networks and deep learning*. Springer, Cham.

Augustsson, C., & Bahlburg, H. (2003). Cathodoluminescence spectra of detrital quartz as provenance indicators for paleozoic metasediments in southern andean patagonia. *Journal of South American Earth Sciences*, *16*(1), 15-26. Retrieved from https://www.sciencedirect.com/science/article/pii/S0895981103000166 (THE PACIFIC GONDWANA MARGIN SI.) doi: https://doi.org/10.1016/S0895-9811(03)00016-6

Augustsson, C., & Reker, A. (2012, 08). Cathodolumenescence spectra of quartz as provenance indicators revisited. *Journal of Sedimentary Research*, *82*, 559-570. doi: 10.2110/jsr.2012.51

Basodi, S., Ji, C., Zhang, H., & Pan, Y. (2020). Gradient amplification: An efficient way to train deep neural networks. *Big Data Mining and Analytics*, *3*(3), 196-207. doi: 10.26599/BDMA.2020.9020004

Boggs, J., Sam, Kwon, Y.-I., Goles, G. G., Rusk, B. G., Krinsley, D., & Seyedolali, A. (2002, 05). Is Quartz Cathodoluminescence Color a Reliable Provenance Tool? A Quantitative Examination. *Journal of Sedimentary Research*, *72*(3), 408-415. Retrieved from https://doi.org/10.1306/102501720408 doi: 10.1306/102501720408

Boggs, S., & Krinsley, D. (2006). *Application of cathodoluminescence imaging to the study of sedimentary rocks*. Cambridge University Press. doi: 10.1017/CBO9780511535475

Breiman, L. (2001, 10). Random forests. *Machine Learning*, *45*, 5-32. doi: 10.1023/A:1010950718922

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth & Brooks/Cole Advanced Books & Software.

Cheremisinoff, N. P. (1997). 1 - principles of geology. In N. P. Cheremisinoff (Ed.), *Groundwater remediation and treatment technologies* (p. 1-37). Westwood, NJ: William Andrew Publishing. Retrieved from https://www.sciencedirect.com/science/article/pii/B978081551411450003X doi: https://doi.org/10.1016/B978-081551411-4.50003-X

Clevert, D., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). In Y. Bengio & Y. LeCun (Eds.), *4th international conference on learning representations, ICLR 2016, san juan, puerto rico, may 2-4, 2016, conference track proceedings*. Retrieved from http://arxiv.org/abs/1511.07289

Demuth, T., Jeanvoine, Y., Hafner, J., & Ángyán, J. (1999, 05). Polymorphism in silica studied in the local density and generalized-gradient approximations. *J. Phys. Condens. Matter*, *11*, 3833-3874. doi: 10.1088/0953-8984/11/19/306

Fortmann-Roe, S. (2012). *Understanding the bias-variance tradeoff*. Retrieved from https://scott.fortmann-roe.com/docs/BiasVariance.html (accessed: 2022-03-10)

Frelinger, S. N., Ledvina, M. D., Kyle, J. R., & Zhao, D. (2015). Scanning electron microscopy cathodoluminescence of quartz: Principles, techniques and applications in ore geology. *Ore Geology Reviews*, *65*, 840-852. Retrieved from https://www.sciencedirect.com/science/article/pii/S0169136814002595 (Applications of Modern Analytical Techniques in the Study of Mineral Deposits) doi: https://doi.org/10.1016/j.oregeorev.2014.10.008

F.R.S., K. P. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559-572. Retrieved from https://doi.org/10.1080/14786440109462720 doi: 10.1080/14786440109462720

Glorot, X., Bordes, A., & Bengio, Y. (2011, 11–13 Apr). Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, & M. Dudík (Eds.), *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (Vol. 15, pp. 315–323). Fort Lauderdale, FL, USA: PMLR. Retrieved from https://proceedings.mlr.press/v15/glorot11a.html

Goldberg, P. (1966). *Luminescence of inorganic solids*. Academic Press.

Gorton, N., Walker, G., & Burley, S. (1997). Experimental analysis of the composite blue cathodoluminescence emission in quartz. *Journal of Luminescence*, *72-74*, 669-671. Retrieved from https://www.sciencedirect.com/science/article/pii/S0022231396002426 (Luminescence and Optical Spectroscopy of Condensed Matter) doi: https://doi.org/10.1016/

S0022-2313(96)00242-6

Götze, J. (2009). Chemistry, textures and physical properties of quartz - geological interpretation and technical application. *Mineralogical Magazine*, *73*(4), 645 - 671. doi: 10.1180/minmag.2009.073.4.645

Götze, J., Ploetze, M., & Habermann, D. (2001). Origin, spectral characteristics and practical applications of the cathodoluminescence (cl) of quartz; a review. *Mineralogy and petrology*, *71*(3-4), 225–250.

Hair, J., Hult, G. T. M., Ringle, C., & Sarstedt, M. (2014). *A primer on partial least squares structural equation modeling*.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*, 498-520.

Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, *28*(3/4), 321–377. Retrieved 2022-06-09, from http://www.jstor.org/stable/2333955

Jarrett, K., Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *2009 ieee 12th international conference on computer vision* (p. 2146-2153). doi: 10.1109/ICCV.2009.5459469

Kingma, D., & Ba, J. (2014, 12). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017, jan). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *J. Mach. Learn. Res.*, *18*(1), 559-563.

Müller, A., & Guido, S. (2016). *Introduction to machine learning with python: A guide for data scientists*. O'Reilly Media, Incorporated. Retrieved from https://books.google.no/books?id=qjUVogEACAAJ

Nair, V., & Hinton, G. (2010, 06). Rectified linear units improve restricted boltzmann machines vinod nair. In (Vol. 27, p. 807-814).

Nayak, A., Parui, K., Sharma, S., & Ratha, S. (2020, 11). Study and analysis of atomic spectra. *International Journal of Scientific and Research Publications (IJSRP)*, *10*, 946. doi: 10.29322/IJSRP.10.11.2020.p10787

Nelson, S. A. (2017, September). *Metamorphism and metamorphic rocks.* Retrieved from https://www.tulane.edu/~sanelson/eens1110/metamorphic.htm (accessed: 2022-03-04)

Nelson, S. A. (2018). *Types of metamorphism.* Retrieved 2022-05-17, from https://www2.tulane.edu/~sanelson/eens212/typesmetamorph.htm

Nielsen, M. (2013). *Neural networks and deep learning*. Retrieved from http://neuralnetworksanddeeplearning.com/

Ruder, S. (2016). *An overview of gradient descent optimization algorithms.* arXiv. Retrieved from https://arxiv.org/abs/1609.04747 doi: 10.48550/ARXIV.1609.04747

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 210-229. doi: 10.1147/rd.33.0210

scikit learn. (n.d.). *3.1. cross-validation: evaluating estimator performance.* Retrieved 2012-05-09, from https://scikit-learn.org/stable/modules/cross_validation.html

Vieira, S., Pinaya, W., & Mechelli, A. (2017, 01). Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience & Biobehavioral Reviews*, *74*. doi: 10.1016/j.neubiorev.2017.01.002

Wu, Y., Cai, S., Xiao, X., Chen, G., & Ooi, B. C. (2020, aug). Privacy preserving vertical federated learning for tree-based models. *Proceedings of the VLDB Endowment*, *13*(12), 2090–2103. Retrieved from https://doi.org/10.14778%2F3407790.3407811 doi: 10.14778/3407790.3407811

Yigit, H. (2013). A weighting approach for knn classifier. In *2013 international conference on electronics, computer and computation (icecco)* (p. 228-231). doi: 10.1109/ICECCO.2013.6718270

Zinkernagel, U. (1978). *Cathodoluminescence of quartz and its application to sandstone petrology*. Stuttgart, Germany: Schweizerbart Science Publishers. Retrieved from http://www.schweizerbart.de//publications/detail/isbn/9783510570089/Contributions_8_to_Sedimentology

Zou, C. (2013). Chapter 8 - oil and gas in metamorphic reservoirs. In C. Zou (Ed.), *Unconventional petroleum geology* (p. 275-305). Boston: Elsevier. Retrieved from https://www.sciencedirect.com/science/

article/pii/B9780123971623000086 doi: https://doi.org/10.1016/B978-0-12-397162-3.00008-6

A | Python Scripts

A.1 Filtering Algorithm

```
#Import libraries
import pandas as pd
import plotly.graph_objects as go

#Calculate First difference of Counts-total column in dataset "data"
data['First Difference']=data['CountsTotal'].diff() #Used for filtering

Sens_FD = 60 # S, Change the number to increase the filter threshold

init_len =len(data['CountsTotal'])
LenFilterFrame = len(data[data['First Difference']>Sens_FD])

while LenFilterFrame > 0:
    #Replace values based on sensitivity given above
    data = data[data['First Difference']<Sens_FD]
    #Calculate new First difference of Counts-total column
    data['First Difference']=data['CountsTotal'].diff() #Calculate first-difference
    LenFilterFrame = len(data[data['First Difference']>Sens_FD])
fin_len = len(data['CountsTotal'])

#Plot results
fig_data = go.Figure()
fig_data.add_trace(go.Scatter(x=data['Wavelength'].values[:],
y=data['CountsTotal'].values[:], mode='lines', name='Filtered data'))
fig_data.add_trace(go.Scatter(x=data['Wavelength'].values[:],
y=data['First Difference'].values[:], mode='lines', name='First-difference'))
fig_data.update_layout(title={'x':0.5,'y':0.85}, xaxis_title='Wavelenght (nm)', yaxis_title='Intensity (counts)')

fig_data.show()
print('Number of values filetered out: ', init_len-fin_len)
print('Length of filtered dataset is: ',fin_len)
```

A.2 Hyperparameter Tuning for the Random Forest model

```
# Range of hyperparameters
n_estimators = [50,100,200,500] # number of trees in the random forest
max_features = ['auto', 'sqrt'] # number of features in consideration at every split
max_depth = [i for i in range(2,11,1)] # maximum number of levels allowed in each decision tree
min_samples_split = [(i+1) for i in range(1,21,2)] # minimum sample number to split a node
min_samples_leaf = [i for i in range(1,19,2)] # minimum sample number that can be stored in a leaf node
bootstrap = [True, False] # method used to sample data points
classWeight = ['none', 'balanced', 'balanced_subsample']
crit = ['gini', 'entropy']

#Store hyperparameter ranges in dictionary
params_rf = {'n_estimators': n_estimators,
             'max_features': max_features,
             'max_depth': max_depth,
             'min_samples_split': min_samples_split,
             'min_samples_leaf': min_samples_leaf,
```

```
            'bootstrap': bootstrap,
            'class_weight': classWeight,
            'criterion': crit}
```

A.3 Fitting Curve to CL Spectra using NN

```
#Import libraries
import time
import pandas as pd
import plotly.graph_objects as go

import tensorflow as tf
import keras
from keras.models import load_model
from keras.callbacks import ModelCheckpoint, EarlyStopping


# choose the input and output variables
#  data is a dataset containing the CL spectra data

x, y = data['Wavelength'].values, data['CountsTotal'].values

tic = time.perf_counter()

#============ define model
model = keras.Sequential()
model.add(keras.layers.Dense(units = 1, activation = 'linear', input_shape=[1]))
model.add(keras.layers.Dense(units = 30, activation = 'elu'))
model.add(keras.layers.Dense(units = 30, activation = 'elu'))
model.add(keras.layers.Dense(units = 30, activation = 'elu'))
model.add(keras.layers.Dense(units = 30, activation = 'elu'))
model.add(keras.layers.Dense(units = 1, activation = 'linear'))

#============ setting of the model
model.compile(loss='mse', optimizer = tf.keras.optimizers.Adam(learning_rate=0.005)) # mean square error

# Display the model
#model.summary()

# Set callback functions to early stop training and save the best model so far
early_stopping_monitor = EarlyStopping(monitor='loss', patience=2500,
            mode = 'min', restore_best_weights = True)
model_checkpoint = ModelCheckpoint(filepath='best_model.h5', monitor='loss', save_best_only=True)

#========== training
history = model.fit(x, y, epochs=5000, verbose=0, callbacks=[early_stopping_monitor,model_checkpoint])

toc = time.perf_counter() #timing the training
print(f"Runtime was {toc - tic:0.4f} seconds")

fig_data = go.Figure()
fig_data.add_trace(go.Scatter(x=history.epoch, y=history.history['loss'], mode='lines'))
fig_data.update_layout(xaxis_title='epoch', yaxis_title='loss (MSE)')
fig_data.update_layout(title={'text':'Excel file: '+rawData_file_name+', '+'Sheet:'+str(sliderValue.value),
    'x':0.5,'y':0.85},
                       legend=dict(x=0.7,y=0.95,traceorder='normal',font=dict(size=15,)))
fig_data.update_yaxes(type="log")

fig_data.show()
```