Random Forests
An Application To Tumour Classification

BSc Aleksi Kanervo

MSc thesis
May 2022

Reviewers:
Prof. Ion Petre
Docent Yury Nikulin

DEPARTMENT OF MATHEMATICS AND STATISTICS

UNIVERSITY OF TURKU
Department of Mathematics and Statistics

KANERVO, ALEKSI: Random Forests
MSc Thesis, 43 pages, 2 appendix pages
Mathematics
Information Security and Cryptography
May 2022

In this thesis, machine learning approaches, namely decision trees and random forests, are discussed. A mathematical foundation of decision trees is given. It is followed by discussion of the advantages and disadvantages of them. Further, the application of decision trees as a part of random forests is presented. A real life study of brain tumours is discussed regarding usage of random forests. The data consists of six different types of brain tumours, and the data is acquired by Raman spectroscopy. After the data has been curated, a random forest model is utilised to classify the class of the tumour. At the current point, the results seem optimistic, but require further experimentation.

Keywords: Machine learning, decision tree, random forest, Raman spectroscopy

Contents

1 Introduction

Lately, healthcare and medicine has seen a huge increase in implementations of machine learning in order to be more efficient in noticing patterns in huge datasets. The idea itself is not new, but current methods and equipment have brought the process to a new level. Artificial intelligence can be utilised to spot malignant tissue in brain. This is supported by previous studies (Ellwaa et al., 2016; Sotoudeh et al., 2019) on identifying tumour tissue.

Artificial intelligence has ties to computer science and mathematics. Mathematics is the main focus of this thesis, and it lays the foundation for various algorithms that are used. First, the concept of decision trees is presented. Also, the history behind the motivation for such concept is explained. They are the basis for random forests, which are the main interests regarding the glioma study discussed in this thesis.

The study discussed in this thesis is on glioma data. The goal is to find out whether a random forest model could correctly classify brain tumours with significant accuracy. A baseline model architecture is introduced, and the goal is to optimise it to produce better classification results. The results are acquired by constructing several different random forest model architectures which are applied on the data that has initially been gathered by using Raman spectroscopy.

2 Decision Trees

Decision trees are used to find new information based on some given attributes. Blockeel and De Raedt (1998) define decision trees to utilise the divide-and-conquer strategy. The divide-and-conquer strategy divides given input into several smaller parts, and then proceeds to find a solution to each of those parts. The solutions of these smaller parts are combined into a single solution for the original problem. Decision trees are heavily used to generalise information, however, the task of generalisation becomes an issue with huge trees (Kotsiantis, 2013).

Decision trees can be used to solve either classification or regression problems. Classification and regression differ from each other by their predicted values. The output of a decision tree defines whether the problem falls under classification or regression. If the output is continuous or quantitative, the problem is defined as a regression problem. If the output is discrete or nominal, the problem can be labelled as classification problem.

There are three key parts to decision trees: a root node, decision nodes and terminal nodes (Njoku, 2019). The root node is the initial node that starts the decision process. Decision nodes are all the nodes between the terminal nodes and the root node. Their purpose is to further make decisions. Terminal nodes are the last nodes in the tree. Their values are the possible outcomes of the tree, and when reached, it is the end decision.

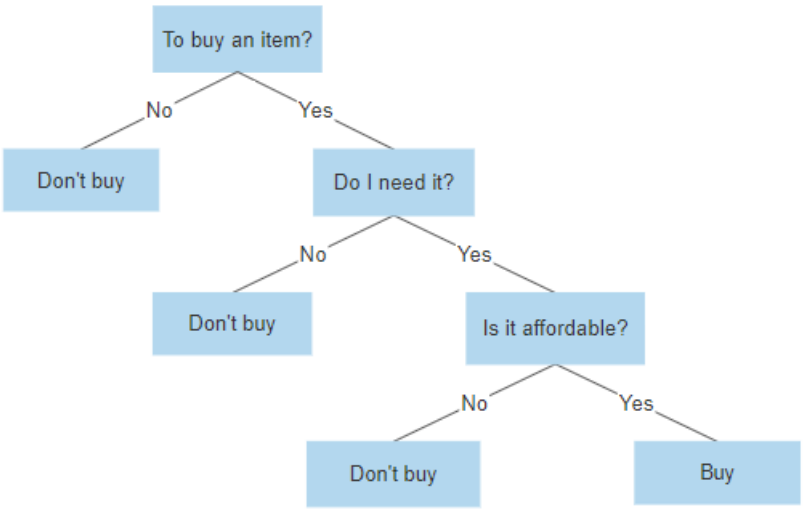

Figure 1: Classification: an example of a decision.

2.1 History of decision trees

In the last 60 years, there have been many groundbreaking publications in the field of decision making and decision trees. One of the first essential publications is the introduction of the first regression tree that includes the usage of impurity (Morgan and Sonquist, 1963). They pointed out many issues about data that should be kept in mind when handling it. One of the biggest concerns is the nature of the data: it can be continuous, ordinal or categorical. This created the need to have flexibility in how to analyse data. Analysing was not the only issue and despite having measures to split the data in two, having a criterion to do so was a more difficult task.

In 1977 a method to implement both classification and regression trees was created, namely CART (Breiman et al., 1984). The model was able to deduce which type the variables were and split them accordingly. The nodes are split recursively until no improvements to make a better decision are made. Seven years after the first model of the CART algorithm, the model was published and is widely used even nowadays.

The year 1986 brought light to decision trees that were not only binary (Quinlan, 1986). An algorithm called iterative dichotomiser 3 (ID3) was an introduction to multivariate decision trees. An information gain was calculated in order to decide which split was to be made on each node. This algorithm had its problems and there were optimisations to be made.

Quinlan (1993) continued to develop the ID3 algorithm and after almost a decade, a new, revised algorithm, C4.5, was published. Key improvements included handling missing values and dealing with noise.

Despite both CART and ID3 being published around the same time, they were independently developed. This reflects the need of such algorithms to model real life decision making, and the phenomenon can be seen nowadays in the constantly increasing need to predict outcomes in the business world.

2.2 Applications of decision trees

There are many use cases for decision trees in real life. They can be used to decide day-to-day problems such as buying groceries or even planning holiday destinations. They are also used in scientific research.

Decision trees are widely used in data mining. For example, data mining is used in school environments in order to help students get better grades, to work on their studying habits, and to generally have better quality of work (Al-Radaideh et al., 2006). Overall, this process can be seen as making education systems better: those

in charge are able to change course contents to focus on topics that the students are less successful in. Also, it helps course creators and student counsellors to see what kind of courses are needed to create comprehensive curricula.

Exploratory data mining, which is a method which uses decision trees, has been utilised frequently in psychological research in behavioural sciences (McArdle and Ritschard, 2013). Decision trees are used in psychological research because they are easy to interpret. Despite being straightforward to interpret, they lose generalisability because even small changes can alter the results tremendously (Jacobucci, 2018). Especially in psychology, it is important that the results would be replicable when handling information about human behaviour.

Further usage of decision trees as a data mining technique can be seen in medicine, for instance, in oral medicine (Khan et al., 2009). Medical databases contain massive amount of information so gathering information by hand is not a feasible option. Using decision trees can provide insight from the data, and find patterns. This can help professionals deduce appropriate operations for patients.

The sensitive nature of medical actions requires accurate and well-informed foundation and data. Decision trees have been shown to provide such reliability in medicine (Podgorelec et al., 2002). Due to the simple interpretation of a single decision tree, professionals are able to interpret the results of the classification efficiently.

Another application of decision trees is in online learning by utilising demographic data (age, income, level of education, etc.) Rizvi et al. (2019) have conducted a thorough analysis of demographic data used in decision trees, and how the decisions can be used in creating courses for certain demographics. Furthermore, it can provide more guided monetary decisions when funding online education. The pitfall of this approach is that the decision trees tend to be biassed towards the majority demographic. Despite the possible bias, the results are sufficient for decision making.

2.3 Common logical operators

Decision trees represent a logical way to reach conclusions. It is clear that they can be used to express basic logical operators. Equivalently, same operators can be expressed with truth tables. Basic logical operators AND, OR, and XOR can be seen in Figure 2 below.

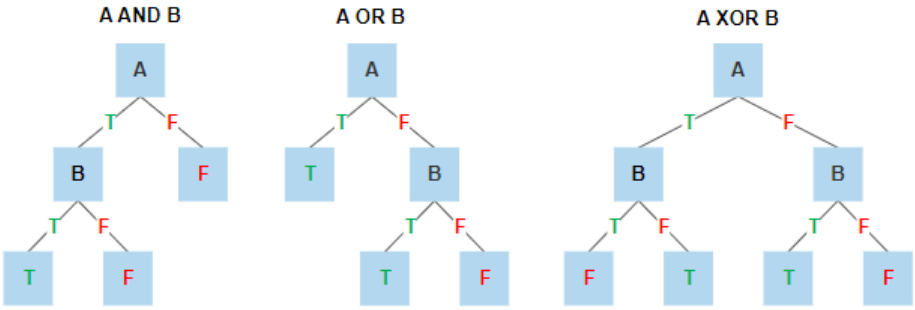

Figure 2: Basic logical operators.

These decision trees can be interpreted in a similar manner as truth tables. To consider the OR tree, the value for A is evaluated first. If A has a value True, then the whole tree would branch to the left and output the value True. If the value for A is False, then the whole process is repeated on the next node, B, until some output is reached.

2.4 Supervised Learning

Both aforementioned classification and regression are forms of supervised learning. The goal of a supervised learning algorithm is to predict an outcome from some input variables. A key thing in supervised learning is that the labels (results) are known initially. An example of a supervised learning problem is a simple image classification: is this creature in a photo a dog or a cat? To formulate this idea on a general level, let $\mathcal{X}$ and $\mathcal{Y}$ be the input and the output respectively. A predictor function f, also known as the classifier, has some input and its result is the desired prediction, i.e., the output:

$$f : \mathcal{X} \to \mathcal{Y}.$$

The goal in supervised learning is to find the predictor function f. Crisci et al. (2012) propose that in order to create a good predictor, the predictor function should be selected from a set of all predictors: $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$, but it is also needed to define what makes a predictor good. The predictor should be as accurate as possible, but it is borderline impossible to have a perfect predictor, therefore there will be some loss. Let L be a loss function that takes f, $\mathcal{X}$ and $\mathcal{Y}$ as inputs. As mentioned before, there is a best predictor $f \in \mathcal{F}$. Denote this best predictor that minimises the loss function L as $f^* \in \mathcal{F}$ such that

$$f^* = \arg\min_{f \in \mathcal{F}} L(f, \mathcal{X}, \mathcal{Y}) := \{f \in \mathcal{F} : L(f', \mathcal{X}, \mathcal{Y}) \geq L(f, \mathcal{X}, \mathcal{Y}) \text{ for all } f' \in \mathcal{F}\}.$$

However, Crisci et al. (2012) point out that the set of all predictors $\mathcal{F}$ can be massive. Thus, it's not feasible to consider the whole set. Hence, a new set $\mathcal{C}$ is defined as a set of predictors that can be computed in a feasible manner, not, for example, exponential models. Now, defining the predictor that minimises the loss function L can be stated as follows:

$$f^{**} = \arg\min_{f \in \mathcal{C}} L(f, \mathcal{X}, \mathcal{Y}) := \{f \in \mathcal{C} : L(f', \mathcal{X}, \mathcal{Y}) \geq L(f, \mathcal{X}, \mathcal{Y}) \text{ for all } f' \in \mathcal{C}\}.$$

It is important to note that f^{**} might not be the same as the original best predictor f^*. Formally this can be expressed as

$$f^{**} = f^* \text{ if and only if } f^{**} \in \mathcal{F}.$$

A problem occurs since f^{**} is not practical in minimising L. This is because the amount of all elements is too large for L. Hence, there is no concrete evidence of a function minimising the loss function on $\mathcal{C}$. As a result, the loss function is adapted to correspond to some sample size n (Crisci et al., 2012; Devroye et al., 2013) in a way that the predictor minimising this is also dependent on the sample size n:

$$\hat{f}_n = \arg\min_{f \in \mathcal{C}} \hat{L}_n(f, \mathcal{X}, \mathcal{Y}) := \{f \in \mathcal{C} : \hat{L}_n(f', \mathcal{X}, \mathcal{Y}) \geq \hat{L}_n(f, \mathcal{X}, \mathcal{Y}) \text{ for all } f' \in \mathcal{C}\}.$$

This definition focuses on the conditions necessary to make it possible for the sample dependent predictor, $\hat{f}_n$, to converge to the best choice for predictor, f^*, when the sample size gets arbitrarily big.

A risk for a function can be defined with the help of the loss function:

$$R(\hat{f}_n) = \int \hat{L}_n(f, \mathcal{X}, \mathcal{Y}) \mathrm{d}P(\mathcal{X}, \mathcal{Y}),$$

where $f \in \mathcal{C}$ and $P(\mathcal{X}, \mathcal{Y})$ is the joint probability distribution. In reality, knowing the value for this distribution is not feasible. Thus, an empirical risk minimisation is done by evaluating the integral using Monte Carlo integration (Kong et al., 2003) (will be omitted), and reducing it to more a feasible form. Let $\mathcal{T}_n \in (\mathcal{X} \times \mathcal{Y})^n$ be a training set randomly chosen from n records. Now empirical risk minimisation is

done by averaging loss over the training set can be denoted as (Cunningham et al., 2008):

$$R_{emp}(\hat{f}_n, \mathcal{T}_n) = \frac{1}{n} \sum_{i=1}^{n} \hat{L}_n(f, x_i, y_i),$$

where x_i and y_i are the ith values of a record's features and targets in the training set.

In very general terms, supervised learning is about modelling dependency between features and the targets, i.e., predicting target values for new data from given features.

2.5 Mathematics of decision trees

Classification and regression trees have similar goals but different types of outcomes. This is why there are different methods for the different trees to split the nodes. Splitting a node is essential for decision trees. Splitting aims to achieve pure nodes. Pure nodes are nodes that have all of their data belonging to one class. Whether splitting a node in a regression or a classification tree, the split with the least impurity is chosen.

2.5.1 Splitting a regression tree

When the target is a continuous variable (a real number), variance can be reduced to make a better split. Variances of nodes determine the best one to split. That is, the node with the lowest variance is the one to choose. Variance can be reduced in two different ways: averaging and pruning. (Geurts and Wehenkel, 2000) state that pruning increases bias but reduces complexity. Averaging does not increase bias but loses in accuracy. Variance can be expressed as

$$Var(X) = \frac{1}{n} \sum (X - \mu)^2,$$

where n is the sample size, X is the sample, and μ is the mean of the sample. Starting from the root node, variance is calculated for each branching decision or terminal node, i.e., child node. After choosing the split that has the lowest variance, the variance is calculated again on the child nodes. This iterative process is carried out until a decision is made, that is, a terminal node has been reached.

2.5.2 Splitting a classification tree

For splitting a classification tree, there are two options: information gain and Gini impurity.

Gini impurity (GI) is only an applicable method to split a node if the targets are categorical. It describes how likely it is to incorrectly label a random element (Yuan et al., 2021). Consider a learning sample $L = \{(x_1, c_1), (x_2, c_2), \cdots, (x_n, c_n)\}$, where each c_i represents a class and each x_i represents a measurement. Let $P = (p_1, p_2, \cdots, p_n)$ be the set of probabilities for each class, where each $p_i = \frac{c_i}{L}$ (Tangirala, 2020). The GI can be expressed as

$$GI(\mathcal{L}) = \sum_{i=1}^{n} p_i(1 - p_i),$$

where indices $i \in \{1, 2, \cdots, n\}$ denote classifications, n denotes the amount of different classes, and p_i denotes the probability of ith classification.

Some formatting can be made, influenced by Raileanu and Stoffel (2004):

$$\begin{aligned} GI(\mathcal{L}) &= \sum_{i=1}^{n} (p_i - p_i^2), \\ GI(\mathcal{L}) &= 1 + \sum_{i=1}^{n} -p_i^2, \\ GI(\mathcal{L}) &= 1 - \sum_{i=1}^{n} p_i^2. \end{aligned}$$

Here, $\sum_{i=1}^{n} p_i = 1$ because summing all probabilities for all possible values for a class will total in probability of 1.

When choosing a criteria to split, a weighted Gini impurity can be calculated. This means calculating the GI for all splits of a node, and then comparing the amount of samples to the total amount of samples in the origin node that is being split, and multiplying those proportions with the GI value (Liu et al., 2018):

$$\sum_{i=1}^{k} \frac{s_i}{s_p} * GI_i,$$

where $i \in \{1, 2, \cdots, k\}$ are the splits, k is the amount of splits, s_i is the amount of nodes in the ith split, s_p is the amount of nodes in the node being split, and GI_i is the Gini impurity of a ith split.

Since the GI measures how likely it is to incorrectly label random elements, it is

the optimal case when the value for it is 0, i.e., there is no impurity.

To utilise Gini impurity in splitting a decision tree, its value is calculated for each decision node including the root node. To compare different split criteria, weighted GI is calculated, and the one with the lowest value is chosen. The process is repeated until a terminal node is reached.

Information gain (IG) is another method to decide the best split for a decision tree. The method is especially useful when the target is categorical. The idea behind IG is based on entropy. Entropy can be expressed as the probability of a class and multiplying it with the base 2 logarithm of that certain class. This is done for each class and they are summed together (Tangirala, 2020).

$$\begin{gathered} Entropy(L) = \frac{c_1}{L}\log_2(\frac{c_1}{L}) + \frac{c_2}{L}\log_2(\frac{c_2}{L}) + \cdots + \frac{c_n}{L}\log_2(\frac{c_n}{L}), \\ Entropy(L) = p_1\log_2 p_1 + p_2\log_2 p_2 + \cdots + p_n\log_2 p_n, \\ Entropy(L) = -\sum_{i=1}^{n} p_i\log_2 p_i. \end{gathered}$$

There is a reason why there is a negative sign in front of the entropy summation. Since for p_i for any given $i \in \{1, 2, \cdots, n\}$ holds $0 \leq p_i \leq 1$, we have that $\log_2 p_i \leq 0$. Thus, $p_i\log_2 p_i \leq 0$, and even further $\sum_{i=1}^{n} p_i\log_2 p_i \leq 0$. It is more feasible to concentrate on positive entropy, hence adding the negative sign in front of the summation creates a non-negative number, that is:

$$Entropy(L) = -\sum_{i=1}^{n} p_i\log_2 p_i \geq 0.$$

Entropy directly reflects the purity of the node. The closer the entropy value $Entropy(P)$ is to 0, the purer the node is. Information gain can be expressed as

$$IG(L) = 1 - \sum_{i=1}^{n} p_i\log_2 p_i.$$

It is important to note that the probability for any class i can not equal 0. This is because $\lim_{x\to 0^+} log_2 n = -\infty$.

Weighted entropy is the value for the whole split decision. After calculating entropies for the child nodes of a split, the values are multiplied with the proportional

amount of samples compared to the parent node's samples. This is done for each child node of the split and these are summed together:

$$\sum_{i=1}^{k} \frac{s_i}{s_p} * E_i,$$

where similarly as in weighted GI, $i \in \{1, 2, \cdots, k\}$ represents each child node, k is the amount of child nodes in the split, s_i is the amount of samples in i, s_p is the amount of total samples in the parent node, and E_i is the entropy of ith child node.

To implement the information gain model to choose the split of a decision tree, similar steps as variance reduction and Gini impurity methods are carried out. Entropy is calculated for each split's resulting child nodes. To choose the criteria on which to split by, weighted entropy is calculated, and the lowest one of those is chosen. This process is repeated until a terminal node is reached. *Note:* also highest IG value can be chosen instead of lowest entropy.

2.6 Implementing decision trees

There exists several algorithms to implement decision trees. Iterative Dichotomiser 3 (ID3) is an algorithm to classify binary problems. Classification And Regression Tree (CART) can also be used for binary classification but are not limited to. CART also works on more complex tasks.

2.6.1 ID3

ID3 is an algorithm that creates a decision tree out of the dataset. It utilises entropy, $-\sum_{i=1}^{n} p_i log_2 p_i$, and information gain, $1 - \sum_{i=1}^{n} p_i log_2 p_i$. The algorithm creates the decision tree's each node (Hssina et al., 2014). To choose the root node, the algorithm chooses the feature that has the best result in classifying the training data. The choice is made by calculating the entropy or information gain for each feature (Quinlan, 1986). The one with the lowest entropy or the highest IG is chosen. This will be the root node. New dataset is created that excludes the decision made on the root node. In other words, the feature on which the decision is made, can be dropped from future calculations. This very decision is a split where a clear classification can be made, say, where a set of samples can be chosen from the group without choosing any other kind of sample. That is, the algorithm prefers splits where one subset is much smaller compared to others. The process is repeated for the next decision node to choose the best feature to split the dataset by.

Since there may be several features, the tree can branch off several times from

the root node. The process is the same to decide how the split is made. Each tree's branch ends in a terminal node with a decision.

The ID3 algorithm has its advantages and disadvantages. It creates an easy to understand tree purely based on the training data. The tree is short and it's built quickly (Slocum, 2012). Each time a terminal node is reached, the data can be pruned to reduce the amount of features and hence making each following decision quicker.

A negative side to ID3 is that small samples are easily overfit. Despite being able to make decisions based on multiple features, the algorithm is only able to consider them one at a time.

2.6.2 CART

CART algorithm is another algorithm that forms a decision tree out of the dataset. Where ID3 bases its decision on entropy or information gain, CART utilises Gini impurity, $1 - \sum_{i=1}^{n} p_i^2$. CART creates a binary tree which means that from each decision node including the root node, a node is split into two instead of many possible splits (Denison et al., 1998). Consider a dataset $\mathcal{D}$ split on some feature F into $\mathcal{D}_1$ and $\mathcal{D}_2$. To calculate the GI value for the split on feature F, it can be done in parts for each subset of data:

$$GI_F(\mathcal{D}) = \frac{|D_1|}{|D|} GI(\mathcal{D}_1) + \frac{|D_2|}{|D|} GI(\mathcal{D}_2),$$

where $|\mathcal{D}|, |\mathcal{D}_1|$ and $|\mathcal{D}_2|$ are the sizes of respective datasets. Define reduction in impurity as:

$$\Delta GI(F) = GI(\mathcal{D}) - GI_F(\mathcal{D}).$$

The feature that has the smallest GI will be chosen to be the splitting feature as it will create the biggest reduction in impurity. To create a whole tree, this process is repeated until the tree has reached its maximum size (Questier et al., 2005). The tree of maximum size, however, is prone to overfitting. The tree is optimised by pruning branches that affect the accuracy the least. Questier et al. (2005) consider pruning the trees by dividing the maximum size tree into subtrees and then calculating a cost-complexity measure for it. Let $R_\alpha(T)$ be the cost-complexity measure for a tree T (Timofeev, 2004):

$$R_\alpha(T) = R(T) + \alpha|T|,$$

where $|T|$ corresponds to the amount of terminal nodes in the tree, α is a measure that decreases the tree's size. Increasing the value of this variable creates smaller and smaller trees, all of which are optimal to their own size. For the whole tree, the goal is to find the value that reduces the complexity of the tree the most while losing as little accuracy as possible. $R(T)$ can be defined in two ways depending on whether classification or regression tree is in question. For classification trees, it is the rate of misclassification, which can be represented in terms of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN):

$$MCR = \frac{(FP + FN)}{(TP + TN + FP + FN)},$$

where MCR stands for misclassification rate. For regressions trees, $R(T)$ is defined as residual sum of squares:

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where y_i is the ith variable considered by the algorithm and $\hat{y}_i$ is the predicted value of y_i.

To choose the best value for α, a method called cross validation is utilised. Cross validation is a method to evaluate performance of an estimator. It is used to reduce overfitting. This is done by using some test set which is a part of the data, and all remaining data is used on training. In optimising the CART algorithm, the test set is one of the subtrees in the tree structure. The cost-complexity measure is calculated for each tree in the training set, and the one with the lowest value is chosen to be the best (Breiman et al., 1984). This results in a pruned decision tree that has reduced complexity compared to the maximum sized tree.

The CART algorithm has several positives and negatives. Similarly as ID3, CART is easy to interpret. Visualisations are clear and informative as there are only two branches from each decision node. The algorithm is able to handle numerical and categorical features and is able to prune the tree to have only the most important features (Singh and Gupta, 2014).

As a general decision tree issue, changes in data can affect how the created tree looks, i.e., change in data increases variance. The algorithm favours classes that are more represented in the data, thus creating biased trees.

2.6.3 Implementing a decision tree classifier in Python

Implementing a decision tree classifier in Python is simple by using ready-made packages. Below is an example of how easily a simple decision tree classifier is implemented in Python using scikit-learn, which provides machine learning focused tools for data analysis.

Iris dataset is a popular dataset to display basic functionalities in different classifiers. It consists of 3 different species of Iris, and includes 4 different features for each record: sepal length, sepal width, petal length and petal width. With the loaded dataset come the actual classes for each record. Dataset is split into training and testing sets using built-in functions. Then, the accuracy of predicted classes will be calculated. The code used to run this example can be found in Appendix A.

Scoring a simulation run resulted in 0.94736..., which means that the predictions were nearly 95% accurate. To showcase how the previously described classifier makes the splits, a visualisation of the classifier in action can be seen in Figure 6 in Appendix B.

As it can be seen, when the Gini impurity value (Gini in the graph) is 0, a classification has been made. The classification process can get complex quickly when the amount of features is increased. Based on the Figure 6, the iris species setosa clearly separates from the two other species.

2.7 Advantages and disadvantages

The nature of decision trees proposes several advantages. However, with advantages come disadvantages.

2.7.1 Advantages

The driving factor for usage of decision trees is the simplicity to interpret decision trees. They split data in a clear, understandable manner whether it's based on some categorical variable (e.g., is it raining: yes, no), or a continuous variable in a range (e.g., age < 30). Another reason why decision trees are so widely used is that pre-processing of data does not require much effort (Al-Radaideh et al., 2006). The data does not require standardisation or even processing of missing values, although, handling missing values may improve performance (Saar-Tsechansky and Provost, 2007).

Missing values can be handled in different ways. Sometimes the whole record for which the missing value occurs can be deleted, or the values can be deduced from other variables by, for example, taking a mean value of the feature and using

that value. Another way to handle missing data is to address that somehow in each decision node. This way it can be ensured that a certain decision is always reached for missing data. It is important to note that a data point can be missing on purpose.

Decision trees are able to handle both regression and classification tasks. This makes them versatile and an option for many tasks. In both cases, a visualisation of a tree is highly encouraged. Depending on the given parameters, the trees can get quite large to be visualised. In any case, each node of a decision tree provides information regarding a certain feature. Decision nodes contain a mixture of possible classes, and further splitting the node shows exactly which feature decided the splitting, eventually yielding pure classifications.

Decision trees are used as a basis for other, more advanced algorithms, for instance, random forests (Parmar et al., 2018). Random forests will be discussed in more detail in a later chapter.

2.7.2 Disadvantages

One of the disadvantages becomes prevalent when the data gets more complex with more features. Rokach and Maimon (2005) state that because decision trees utilise divide-and-conquer methods, they work extremely well on few features but lose accuracy with many complex features. However, despite decision trees not being so accurate for one complex classification, many decision trees can be used together to split the classification tasks into smaller parts in order to make more accurate classifications. In other words, decision trees are not able to generalise data well from large and complex datasets.

Overfitting is a problem, which means that the model is able to fit the training data really well but fails to generalise any other data (Kotsiantis, 2013). Overfitting happens easily if there is noise in the data. There are several ways to mitigate overfitting, for example, feature selection and, in general, pre-processing of data.

Feature selection is about measuring how good a certain selection of a feature is. The idea is to find features that have minimal effect on the outcome and then drop those features for the selection. Feature selection creates a subset of the original data with reduced complexity and is overall more generalisable. Feature selection can be performed by, for example, ranking the features. A way to rank features is to calculate Pearson correlation coefficient, which measures linear correlation (Chandrashekar and Sahin, 2014):

$$r_i = \frac{cov(x_i, Y)}{\sigma_{x_i}\sigma_Y},$$

where x_i represents the ith feature, Y is the labels of the classes, $cov(x_i, Y)$ is the covariance of the ith feature and the class label. Covariance describes how two variables vary together. Positive covariance means that increase in one variable implies the other variable also increases. Negative covariance means that increase in one variable implies the other variable decreases. Variables σ_{x_i} and σ_Y are the standard deviations of respective variables. The closer the absolute value of r_i is to 0, the less linearly correlated the two variables are. It is more interesting to consider correlated variables as they can help generalise the data.

3 Random Forests

Random forests are essentially a group of random decision trees. Simply, out of all the decision tree classifiers, the most common prediction is chosen to be the prediction of the random forest. Ho (1995) is credited to be the first to introduce the general idea of random forests. She was able to show that using a collection of random trees can help to increase accuracy of predictions. This is because the error for each prediction can be averaged between the trees. Random forests follow divide-and-conquer methods like decision trees. The random forests are based on bagging, also known as bootstrap aggregation. This method aims to have low variance and low bias by repeatedly randomly sampling with replacement using training data. Randomly sampling with replacement allows multiple samples to be used rather than just one.

3.1 Bagging (bootstrap aggregation)

Bagging is used to create many versions of some predictor after which these predictors are aggregated to be one predictor (Breiman, 1996). Let $\mathcal{L} = \{(y_i, \mathbf{x}_i), i = 1, \cdots, n\}$ be a learning set, where y_i is an ith true label, and $\mathbf{x}_i$ is the ith input that is used for a prediction. Denote a predictor by $f(\mathbf{x}, \mathcal{L})$. Consider a collection of learning sets:

$$\mathcal{T} = \{\mathcal{L}_i, i = 1, \cdots, k\},$$

where k is the amount of learning sets. We have that $|\mathcal{L}_i| = n$ for all i since each learning set in the collection $\mathcal{T}$ consists of data from $\mathcal{L}$ with the same proportion of learning data. For each learning set in $\mathcal{L}_i \in \mathcal{T}$, there is a predictor $f(\mathbf{x}, \mathcal{L}_i)$. These predictors create a set of predictors:

$$\mathcal{T}_p = \{f(\mathbf{x}, \mathcal{L}_i), i = 1, \cdots, k\}.$$

Considering the predictor, the type of task matters: is classification or regression in question? For classification, the resulting prediction is made by taking the most frequent prediction from the predictions. Let $j \in \{1, ..., J\}$ be a class that can be predicted, and J the amount of all classes. Assume j is a prediction of the predictor $f(\mathbf{x}, \mathcal{L})$. Consider the amount of different predictions. The amount of j predictions from all predictors can be denoted as:

$$Q_j = \#\{i : f(\mathbf{x}, \mathcal{L}_i) = j\}.$$

The previous process is the aggregation process. The Q_j with the highest value will be chosen to be the final prediction:

$$f_A(\mathbf{x}) = \underset{j \in \{1,\cdots,J\}}{\arg\max}\, Q_j := \{j \in \{1,\cdots,J\} : f(\mathbf{v}) \leq f(\mathbf{x}) \text{ for all } \mathbf{v} \in \{1,\cdots,J\}\}.$$

For regression tasks where the predictions are numerical, an average value of the predictions can be considered as the final prediction after aggregation (Quinlan et al., 1996). Denote the aggregation of the predictor function as the expected value of a prediction from a learning set:

$$f_A(\mathbf{x}) = \mathbb{E}_{\mathcal{L}}[f(\mathbf{x}, \mathcal{L})].$$

Let B be a bootstrap, i.e., a random sample with a replacement. This means that some learning set has been selected from the available data but now some sample will be replaced with another value from the data, possibly changing the set slightly. The same datapoint can appear several times in the bootstrap. A bootstrap is taken from the learning set repeatedly. From then is formed a set of predictors that use the bootstrap samples instead of the learning sets. For classification, the most common prediction is chosen from $\{f(\mathbf{x}, \mathcal{L}^{(B)})\}$.

For regression, the values of $\{f(\mathbf{x}, \mathcal{L}^{(B)})\}$ can be averaged. Let a_i be the value of a prediction from an ith bootstrap sample $\mathcal{L}_i^{(B)}$. Now, the average value of the predictions is:

$$f_B(\mathbf{x}) = \frac{1}{J}\sum_{i=1}^{J} a_i,$$

where J is the size of the predictor set. This sets the basis for random forests and how they make predictions utilising a collection of decision trees.

3.2 Random forest training algorithm

The random forest algorithm relies on decision trees that are uncorrelated. This is achieved by using bagging and random feature selection to create different trees (Parmar et al., 2018). Initial algorithm by Breiman (2001) and further clarification by Parmar et al. (2018) has provided an easy way to state the steps of the algorithm for a classification task.

The training set is divided into subsets. Each subset contains a certain amount of features, denoted by hyperparameter M. That is, each subset contains the same

amount of features, but the features vary. Let k denote the amount of subsets, and θ_i the ith subset. Each subset of features is chosen randomly from all the available features. Randomness ensures that the subsets $\theta_i, i \in \{1, \cdots k\}$ are independent and not dependent on each other. The data is now trained with each of the feature subsets, creating a decision tree classifier. For an input $\mathcal{X}$, the classifier can be denoted by $h(\mathcal{X}, \theta_i)$ for the ith feature subset. The classifier is created for each k feature subsets. Each classifier outputs a classification result, and after all results are gathered, the most common of them is chosen.

To put the creation of a random forest classifier simply, first a training set is randomised and split into subsets of features. From those subsets, decision trees are made that act as a test set. Each decision tree results is a classification, and the end result is chosen by voting the most common classification.

3.2.1 Hyperparameter tuning

The algorithm is affected by hyperparameter tuning. Different selection of hyperparameters can yield very different results but the algorithm should run in reasonable time and perform efficiently. Probst et al. (2019) mentions several parameters that have an important role in the random forest algorithm. They include but are not limited to: amount of features in each sample and the number of trees in the forest.

In a comprehensive study by Fernández-Delgado et al. (2014) a significant amount of 179 classifiers were analysed with various hyperparameters. They concluded that increasing the number of trees in the forest mostly improved the accuracy of the classifier. However, increasing the amount of trees arbitrarily isn't the most optimal solution. In general, it is thought that $2^6 - 2^7$ trees should yield good enough results. Oshiro et al. (2012) show in their study that on various data, having more than 2^7 trees does not increase the results significantly. Running time wise increasing the amount of trees does not provide good enough improvement, so the amount of trees is worthwhile optimising when handling lots of data.

Selecting only a few features for each subset will create trees with almost no correlation but in the long term might lose accuracy in predictions (Probst et al., 2019). This is a result of the likelihood of selecting only irrelevant features that don't contribute much to the prediction accuracy. However, if choosing almost all the features there are in the whole data, the trees are more correlated and thus less stable, but provide better prediction accuracy. When implementing a random forest algorithm for classification, usually the default value of chosen features is $\sqrt{p}$, where p is the amount of features in the whole data. Probst et al. (2019) evaluates that increasing the amount of features increases the running time of the algorithm

linearly.

Another criteria to consider when tuning hyperparameters isn't necessarily a hyperparameter but resembles one. To choose which splitting criteria to use when splitting decision trees can yield vastly different results. To optimise the splitting rules, Geurts and Wehenkel (2000) suggest that randomising the splitting criteria for each tree would yield more efficient results. However, when considering just a single tree regarding the results, no significant difference can be seen between splits; randomising the split creates purely computational advantage.

3.2.2 Feature selection and importance

As in decision trees, not considering irrelevant features can improve results of the algorithm. First, some way to determine the most relevant features has to be implemented.

One way of considering whether a variable is relevant is to evaluate whether some feature has correlation with the target label (Rogers and Gunn, 2005; Roobaert et al., 2006). However, John et al. (1994) suggest that instead of evaluating relevance of a single feature at a time, it is more beneficial to consider feature subsets.

As random forests utilise bagging, the features used in each decision tree are randomly chosen with equal likelihood. Furthermore, as each decision tree aims to maximise information gain for given features, a correlation of a feature to the target label can be evaluated as this information gain. Rogers and Gunn (2004) propose a node complexity measure to weight the IG as an optimisation method. Consider a binary classification (0 or 1) with n samples and i classifications that result in 1. Using binomial coefficient, the amount of possible orders for data is:

$$C_i^n = \binom{n}{i} = \frac{n!}{i!(n-i)!}.$$

This itself is not enough to evaluate information gain as the order is reflective to itself, that is, each node would contain two ways to order the data. However, to get insight from this approach, both of the orders should be the same. Denote the amount of symmetrical ordering by A_u. Those that are not symmetrical can be calculated by $C_i^n - A_u$. There are two possible options: symmetrical and non symmetrical orderings. A probability for a symmetrical ordering, U_x, is $P(U_x) = \frac{1}{C_i^n}$, and the probability for a non-symmetrical ordering, N_x, is $P(N_x) = \frac{2}{C_i^n}$. The value of numerator is 2 because there are two ways to order the data at those nodes. With the help of definition of information gain, the node complexity measure, NCM, can be defined (John et al., 1994) for a single done s as

$$NCM(s) = -\frac{A_u}{C_i^n}\log_2 P(U_x) + (-\frac{C_i^n - A_u}{C_i^n}\log_2 P(N_x)),$$
$$NCM(s) = \log_2 C_i^n - (1 - \frac{A_u}{C_i^n}).$$

This estimates the average value for the IG on a single node. Also, this method does not allow any overlapping for the data, thus in this form it is not suitable for random forests. For random forests, the data is overlapping due to bagging. Because duplicate data points are possible in each node, possible orderings of data points are reduced. Rogers and Gunn (2005) point out that due to this fact, the expected IG can have a lower and an upper bound. The middle point of these thresholds give a realistic estimation for the average expected IG. When the imbalance is the greatest, i.e., a node is split in a way where only one record of of each feature is chosen, Rogers and Gunn (2004) state that the lower bound for the estimated IG in regard to the node size, n, is:

$$E[IG] \geq \frac{1}{n} - \frac{n-1}{n}\log_2\frac{n-1}{n}.$$

For the upper bound in the most balanced case, i.e., the amount of features is equal on the node, can be estimated to be

$$E[IG] \leq (\frac{n}{2})^{-0.82},$$

which is a visual approximation of a graph of a simulation (Rogers and Gunn, 2004). When choosing the average information gain, the middle point of these two estimations is considered.

An average over many trees helps deducing whether a feature is relevant or not. Those features with high average IG would be kept and those with low average IG could be scrapped to make the overall splitting process more efficient and the results more stable. It depends on a task at hand what is redeemed a suitable threshold to count a feature as relevant.

3.3 Usages

In real life, random forests have several use cases and they are used widely in various fields. For example, banks use random forests to evaluate whether to give a customer a loan. They are interested in the customer's prior behaviour in terms of whether they have paid previous loans back in time, do they have steady income, or even

credit score if that is applicable. The evaluation could result in whether the customer would be likely to pay a loan back and based on this, a human makes a decision to grant it or not. Random forests can also be used to detect credit card frauds when customer name, transactions, credit card number and other related information (Jebaseeli et al., 2021).

In medical research, random forests can be used to make classifications based on medical data Alam et al. (2019). They can be implemented to predict whether a patient has, for instance, a diabetes (Zou et al., 2018) or breast cancer (Nguyen et al., 2013) based on their existing patient data. It is important to note that these predictions might not be accurate but rather give insight to doctors if some cases are worth looking into.

3.4 Advantages and disadvantages

Random forests provide increased prediction accuracy compared to decision trees. As a random forest is a collection of decision trees, each tree counts and the end result is the most common decision made among the trees. On top of that, the prediction error can be averaged between trees. Similarly as decision trees, random forests can be utilised for both classification and regression, and handles missing values well. In general, random forests incorporate many advantages of decision trees such as different types of variables and the splitting of data.

Comparing random forests to decision trees, it is clear that the amount of resources needed for random forest algorithms is a lot more than for decision tree algorithm. This is due to the fact each decision tree needs to traverse through. A key improvement over decision trees regarding random forests is that overfitting is not as significant issue. Even if individual trees may overfit, a forest of trees mitigates that issue when the amount of trees is increased. Where decision trees are easily interpreted, random forests are not. It is easy to figure out the outcome of one decision tree but doing so for possibly hundreds of different trees simultaneously is not as simple a task.

4 Raman Spectroscopy

Raman spectroscopy is a technique to study chemicals based on their fingerprints without destroying the chemicals. The procedure can used on any substance. The fingerprints can be thought as identifiers for the chemicals, and they are unique (Rostron et al., 2016). The technique is based on inelastic scattering of photons. In the process a continuous wave (or a pulsed) laser is pointed to a photon which scatters the light. The scatter is considered in two parts. The first part is called Rayleigh scatter which is not useful as the scattering is the same wavelength as the incoming light. This is because there is no change in energy which lets the light to reform back into its original form after the initial scattering happens. The other part is called Raman scatter. It is the other part of the scattering which is a minimal part of the light source. This part is of some other wavelength than the source. The different wavelength is dependent on the chemical that is targeted, thus creating unique, identifiable feature. The Rayleigh scatter is filtered out and the Raman scatter is collected with some detector, for instance, a spectrograph, that stores data in a form of different wavelengths of light.

The actual process to create Raman spectra involves hitting a sample with a precise laser. A sample is hit in different spots without killing the cell, thus making this method non-destructive. As the laser hits the tissue of the sample, the molecules inside the cell vibrate. The actual laser that scatters depends on how the molecules vibrate. It is substance dependent how the molecules vibrate. Previously mentioned spectrograph captures the vibrations resulting in a spectrum of intensities.

The method of Raman spectroscopy provides a more efficient, cheaper and time saving way to study chemicals. The data provided by this method is not difficult to understand for those not specialised in such data.

4.1 Data offered by Raman spectroscopy

The data created by Raman spectroscopy is due to vibrations in a molecule, and it provides a lot of data about chemicals. Information about chemical structure and chemical identity can be gathered using Raman spectroscopy (Movasaghi et al., 2007). Their research shows that the technique is very suitable to find structural properties in biological tissues. There is previous literature on usage of Raman spectroscopy on brain tumours. (Gajjar et al., 2013) and (Kowalska et al., 2020) have shown that Raman spectroscopy is an efficient way to separate tumour from normal tissue. The vibrations caused by the laser create differences in the tissue so that this separation is possible.

The technique can be used to analyse particles in a solution. This is especially relevant in polymorphism, which means that there are several possibilities for different forms in a single compound, and these forms can be distinguished by characterisation (Tuschel, 2019).

Raman spectroscopy is used in microbiology in order to identify microorganisms. Jarvis and Goodacre (2004) provide a comprehensive analysis how it can be utilised to quickly characterise and fingerprint bacteria with repeatable results. Generally, there are many use cases for Raman spectroscopy in biology (Butler et al., 2016), but also in medical field. It can be used to spot and correctly identify uncontrollable growth of tissues (neoplasia) in different parts of a human body such as brain (Meyer et al., 2011) and lungs (Huang et al., 2003).

4.2 Hyperspectral imaging

Hyperspectral imaging provides two-dimensional images of the spectrum of electromagnetic radiation. Based on the absorption of light, it can be utilised to define different characteristics of a photon (Lu and Fei, 2014). With different amounts of absorption occurs different amount of scattering of the light. Raman spectroscopy can be thought as a basis for hyperspectral imaging when thinking about efficiency and mapping of spectra (Maybury et al., 2018). The images created by hyperspectral imaging are combined into a hypercube which consists of each wavelength from ultraviolet to near-infrared. Humans perceive colour in three bands: red, green and blue. The wavelengths of the hypercube can be seen as a continuous range of values, and the human scale as discrete values. Raman spectroscopy and hyperspectral imaging can be performed on a same sample, and Raman spectroscopy is a driving factor whether hyperspectral imaging has provided a correct identification. That is, if identification of hyperspectral imaging differs from identification of Raman spectroscopy, the identification can be faulty (Maybury et al., 2018). Hyperspectral imaging works more efficiently than Raman spectroscopy but the latter is thought to have more reliable results in that sense.

5 Random forest application: brain tumour classification

The project at hand is on glioma data. The results from this project would be to help detect tumours early enough to increase the chances of a successful surgery and prolong the life expectancy of a patient.

The data is provided by the National Institution of Health located in The United States. A sample tissue is cut off of a patient's tumour. It is scanned using Raman spectroscopy. A common sample size is $60 * 60 * 1738$ (width, height, frequency), but the size varies from sample to sample; only the frequency is constant. The frequency of 1738 is the result of modifying the original data which can contain even 5000 frequencies. The tumour has frequencies over a certain threshold. Using the Raman spectroscopy, a laser is used to hit the tumour in different position horizontally and vertically. In the case of a 60 x 60 size tumour, it is hit in 60 positions in both directions. Each spot that is hit with the laser is fingerprinted. Scanning samples is time consuming. A single sample can even take 24 hours to scan.

The data consists of 58 samples, and some samples can be from the same patient. In terms of the number of patients, it might sound like a low number. Regarding the project, the data seems sufficient. The data of a sample is curated before it is used in making predictions utilising a random forest classifier. A sample might not be only tumour, but can also contain healthy tissue, blood vessels, dead tissue or even plastic from a container the sample is placed in.

A question arises: can machine learning be applied to separate tumour from non tumour parts? It is a difficult question to answer but at the current point of the project, the separation can be done. A hypothesis regarding the whole project is that Raman fingerprints characterise a tumour type. In other words, is it possible to learn a tumour type from Raman spectra?

The data consists of 6 different categories, namely LGm-1 to LGm-6. LGm-1 includes data from 5 different patients. LGm-2 type of tumour includes data from 11 different patients. Type LGm-3 has data from 4 different patients, however, there are 9 different data points from them. This means that there are several samples from the same patient. This is counted as the same tumour from a patient. LGm-4 type tumour includes 10 patients with a total of 13 samples. Type LGm-5 contains 15 sample points from 11 different patients. Lastly, LGm-5 type has 5 samples from a total of 4 patients.

Currently, the classification task can be performed on 2 or 3 categories with decently high scores but as the optimal goal would be to accurately classify 6 cate-

gories, the model falls short. It is entirely possible that with more data at hand the classification would perform better.

When processing the sample with deep learning techniques, it is important to note that a part of the tumour might look like a different tumour than it actually is. Hence, using a random forest classifier to get a prediction by a vote is our approach to tackle that issue. However, the deep learning approach for tumour classification should not be neglected in general, and it is, in fact, a method that is being worked on in parallel to my approach. In literature, there are deep learning applications in classifying brain tumours (Nazir et al., 2021) as well as in Raman spectroscopy (Jinadasa et al., 2021). Even though deep learning can be used in analysing brain tumours, the current techniques are redeemed lacklustre due to their complexity, and hence losing efficiency. Deep learning is successfully used in our processing of the Raman spectra when separating the malignant tissue from the healthy tissue.

At the current moment, our random forest approaches are yielding similar results to the deep learning methods. Deep learning methods will not be discussed in more detail as they exceed the scope of this thesis.

As each sample consists of 1738 frequencies, they can be visualised in a graph. The original data is three dimensional. In order to visualise it in a two dimensional graph, some form of combination of dimensions has to be done. As each sample's dimensions contain information about width, height and frequency, for the plotting width and height are combined by multiplying them together. Below is a plot of the whole spectra for a sample from patient HF-868.

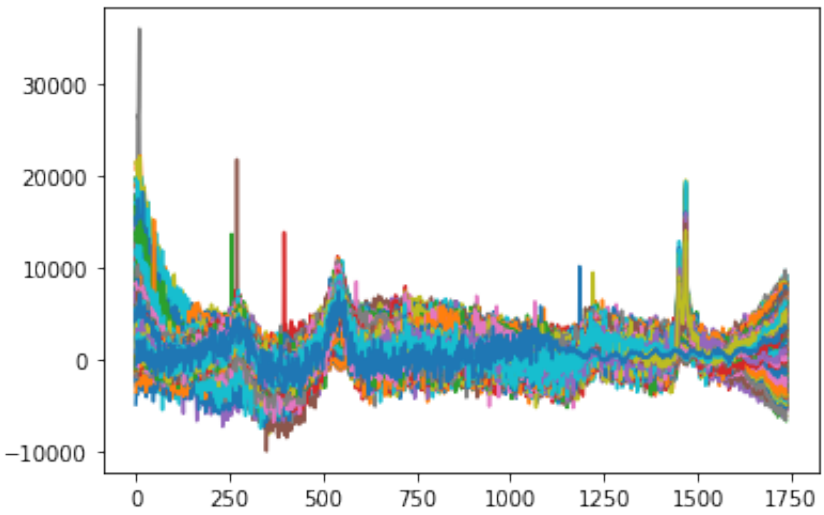

Figure 3: Patient HF-868: original spectra.

Each sample contains 3 clusters, labelled as -1, -2 and -3. The label -1 is for the

majority cluster, which is kept for the purposes of further analysis. Clusters with labels -2 and -3 are discarded. To discard clusters -2 and -3, their values are set to 0 while -1 is kept unchanged. It is known from the providers of data that the high peaks on the left side of the plot are not interesting to analyse. I do not have further explanation for this. Following is a plot that contains only the majority cluster. Minority clusters have been discarded. It can be seen that there are still a couple high peaks on the left, so further cleaning is probably required. Compared to the original spectra, this is much cleaner.

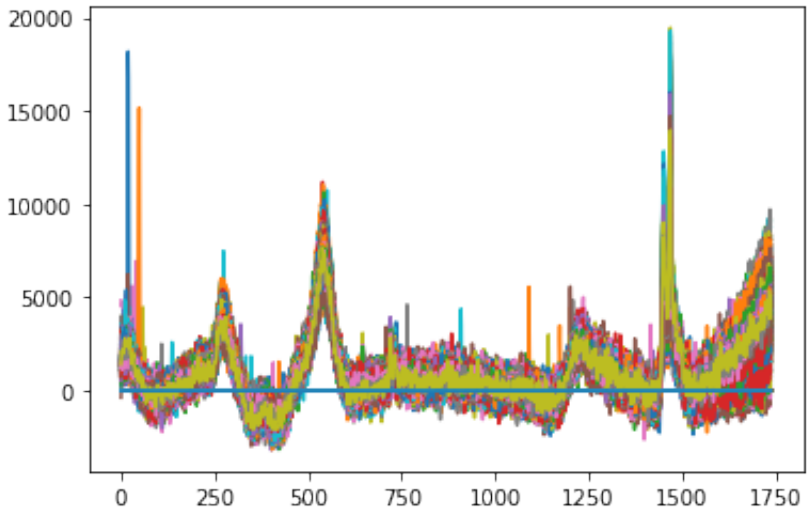

Figure 4: Patient HF-868: spectra with label -1.

Similarly, the sample can be visualised as an image. It shows clearly where there are clusters. The goal is to keep the yellow area, and to get rid of the darker, smaller cluster in the middle. This helps to understand more precisely what removing the certain clusters in the sample does.

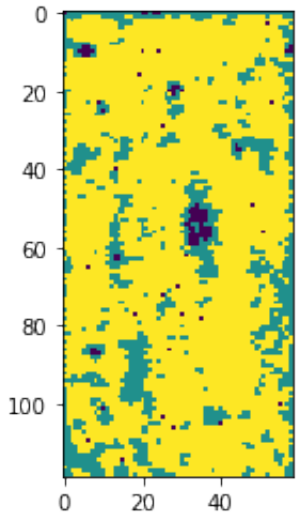

Figure 5: HF-868: cluster.

5.1 Model architecture

The progress discussed along the way will be mainly focusing on classification into 6 categories. Most of the testing is done by utilising only one training cycle instead of proper leave-one-out cross validation cycle to gain insight into how training and validation accuracies change as different hyperparameters are changed for the random forest model. The parameters defined for the model are: *bootstrap, class_weight, criterion, max_depth, min_samples_leaf, min_samples_split, n_estimators, random_state, warm_start, n_job,* and *max_features*.

Bootstrap parameter defines whether whole data is used to build each individual tree. Setting this value to *true* allows the usage of bootstrap samples.

Parameter **class_weight** describes whether different classes have different weights. Setting class_weight to *balanced_subsample* determines the used weights for every tree based on a bootstrap sample.

The random forest model utilises one of the two splitting **criteria**: gini or entropy. These criteria are discussed more in detail in an earlier chapter. In short, gini chooses the split based on Gini impurity, and entropy bases the split on information gain.

Max_depth defines the maximum depth of a tree in a forest. If a value is given, all the leaves might not end up being pure, but with a big dataset this simplifies the model and reduces the running time.

Min_samples_split simply determines what is the minimum number of samples on any given node in order to split it. This parameter can be met before the *max_depth*, which would stop expansion of the leaves further.

Min_samples_leaf determines the minimum amount of samples in a leaf. If at any point the amount of samples in a leaf after a split is smaller than *min_samples_leaf*, no further splits will be done.

Parameter **n_estimators** defines the amount of trees in the forest. Considering the size of our dataset, this value can be sufficiently large while keeping the usage of the model feasibly computable.

Random_state is defined to have consistent results every run. This predetermines randomness for reproducibility.

Warm_start determines whether previous fit is used to expand it, or whether a new forest is fit altogether.

N_job defines how many jobs are allowed to run simultaneously. This essentially defines how many CPUs are allowed to be used.

Max_features is used to limit the amount of maximum features that are used when deciding on a split. This value can be base 2 logarithm, square root of the amount of features, or it is possible to utilise all features.

5.1.1 Our model

Our random forest classifier model architecture involves previously mentioned parameters. The architecture is used for all the training done for the data. When looking into how different values affect the training and validation, several parameters are kept the same over different individual runs of training.

The parameters that won't be changed when testing different values are: *bootstrap, class_weight, random_state, warm_start,* and *n_jobs*. *Bootstrap* is set to True, as we are utilising bootstrap samples. *Class_weight* is balanced_subsample. This allows each tree created from a bootstrap sample to have a different weight. *Random_state* is set to be a constant to control randomness so the results are reproducible over many runs. *Warm_start* is set to False. This forces a new tree to be fit on each round rather than expanding the previously fit tree. *N_job*'s value is set to be -1. This makes it possible to utilise all CPUs while running the code.

The following parameters' values are applicable for balanced training using all available features.

The parameters that are changed over different runs to find the best values are: *criterion, max_depth, min_samples_leaf, min_samples_split, n_estimators*, and *max_features*. *Criterion* is chosen to be either *gini* or *entropy*. Values for *max_depth* range from 8 to 18. For *min_samples_leaf* values ranging from 50 to 150 are considered. *Min_samples_split* has a value at least twice as great as *min_samples_leaf* so after a split is made, the resulting leaf will have enough samples to be kept. The amount

of trees in the forest, *n_estimators*, is between 50 and 1500. Because of the large amount of features in the dataset, the *max_features* parameter is defined to have a subset of the features contained in the dataset. The value is set to be either *sqrt* to take the square root of the amount of features, or *log2* to take a base 2 logarithm of the amount of features.

5.2 Progress

Before discussing the results that I gathered during the process, few terms need to be defined. Results are derived from confusion matrices. The metrics found in a confusion matrix are accuracy, precision, recall, and f1-score. Accuracy is the total amount of correctly predicted samples over all data. This is, the amount of true positives and true negatives over the amount of true positives, true negatives, false positives and false negatives:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

Precision is defined as the amount of true positives divided by the combined amount of true positives and false positives:

$$Precision = \frac{TP}{TP + FP}.$$

Recall is defined as the amount of true positives divided by the combined amount of true positives and false negatives:

$$Recall = \frac{TP}{TP + FN}.$$

In this context, true positive value reflects the amount of correctly predicted positive class, i.e., whether a sample is correctly predicted to its real class. False positive value describes that a sample is incorrectly predicted to a certain class. False negative means that a sample has been classified to a class which is not its true class. F1-score considers both precision and recall. It is defined as double of the multiplication of the two over addition of the two:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}.$$

This part of model experimenting was performed on a remote cluster, Dione, which is a super computer. This was utilised due to lack of local resources. Due to behaviour of random forests, the running will not be significantly affected by using this cluster as it is ran on CPUs. Each time the training is carried out, it is

submitted to a node within the cluster.

When I was given the initial model, the changeable parameters were as follows: splitting was determined to be made using entropy, maximum depth of a tree is 10, each leaf has a minimum of 100 samples, and at least 250 samples in a leaf are required to make a split. The initial model had 1000 trees in the forest, and the maximum amount of features was determined by base 2 logarithm. This run of a balanced training took approximately 30 minutes, and it yielded a R^2 training score of 0.73. Classification report on training results in accuracy of 0.82, precision of 0.80, recall of 0.83, and f1-score of 0.81. Results for validation only result in accuracy of 0.20, precision of 0.24, recall of 0.18, and f1-score of 0.19. Let this run and these results be referred as **run1**.

5.2.1 Optimisation of *criterion* and *max_depth*

My aim in the following balanced training runs is to optimise the various parameters by increasing the results step-by-step. Each consecutive run mentions only the changed parameters compared to the previous run. My approach to this was to only consider parameters *criterion* and *max_depth*. I tried different combinations of these two values while keeping other parameters the same as in the initial model. For each tested value of *max_depth* both gini and entropy criteria are evaluated.

Run2 introduces only one change. The *criterion* changes from entropy to gini, and the *max_depth* stays the same with a value of 10. The different criterion does not provide much change to the R^2 training score or classification reports. In fact, the R^2 training score decreased from 0.73 to 0.72, and there is no change in accuracy, precision, recall or f1-score values when considering them to two decimal places.

Parameter *max_depth*'s increase to 12 in **run3** introduces a slight improvement in training accuracy. The R^3 training score for **run3** is 0.74, which is a slight improvement compared to **run1**. Training accuracy increases to 0.84, precision to 0.82, recall to 0.85, and f1-score to 0.83. However, results for validation don't see any change and stay the same.

Run4 keeps the *max_depth* at 12, but sees the change of *criterion* to entropy. This change doesn't provide any visible changes to classification or validation metrics, but the R^2 training score increases to 0.75. I regard this as the best run so far in terms of numbers, and running time is only approximately 5 minutes longer than on **run1**.

For the fifth run, namely **Run5**, the *max_depth* is increased to 14. This yields no visible changes in the classification reports as R^2 training score, and all the metrics are the same as in **run4**.

The next run, **run6**, keeps the *max_depth* of 14 but incorporates the gini splitting rule. A little improvement can be seen in training precision and recall as they increase to 0.83 and 0.86 respectively.

Run7 keeps the gini splitting rule but has increased in *max_depth*; the new value is 16. No changes compared to previous can be noticed. The next run, **run8**, incorporates entropy splitting criterion. This is the first time the results get a little bit worse compared to previous rounds while only increasing the *max_depth* parameter and changing *criterion* from gini to entropy and vice versa. For **run8** the training precision decreases back to 0.82, but otherwise all the values are the same.

As increasing the *max_depth* further didn't seem to improve the results, I tested if decreasing the value from **run1** could yield better results. Having *criterion* as entropy and *max_depth* as 8, **run9** yields so far the worst results. With a R^2 training score of 0.67 and training accuracy of 0.78, other metrics are weaker too. Training precision is 0.77, recall is 0.79, and the f1-score is 0.77.

Run10 shows another similar decrease in results when using gini splitting *criterion* compared to **run1**. With R^2 training score of 0.64 and training accuracy of 0.77 and similar other metrics as in **run9**, it is sufficient to say that decreasing the parameter *max_depth* from 10 is not a suitable option.

5.2.2 Optimisation of *n_estimators* and *max_features*

For the next part, I selected the model architecture from **run4**, even though **run6** yielded slightly better results. I do not think the choice of architecture plays a significant role at this point, so I chose the slightly faster option. The architecture looks as follows for the next runs that aim to optimise parameters *n_estimators* and *max_features*: *criterion* is set to entropy, *max_depth* is 12, *min_samples_leaf* is 100, *min_samples_split* is 250, *n_estimators* is 1000, and *max_features* is set to be base 2 logarithm.

Run11 aims to see how decreasing the amount of trees in the forest affects the training and validation accuracies. Decreasing the amount of trees, *n_estimators*, to 750 shows no change in either training or validation or accuracies or any other metrics. The very same results were yielded when *n_estimators* was set to 500 on **run12**. This gave me some insight that this many trees might not be necessary for the model, and can, in fact, be lowered even further.

To further test this, on **run13** I tried with 1250 trees in the forest. This, indeed, did not yield any better results than 500 trees. In fact, the results were on par with the results from **run4**. Hence, lowering the amount of trees in the forest is something I want to keep in mind when I combine different results, and test the

model further.

Since the last few rounds proved to be useful in terms of reducing the parameter *n_estimators*, next I looked into changing the amount of maximum features in a considered split. **Run14** determines *max_features* to be sqrt, i.e., a split is done based on the square root of the amount of features. This yields the first bigger change in metrics. With a R^2 training score of 0.80 and accuracy of 0.89, this seems to be the best so far. Training precision is 0.86, recall is 0.89, and f1-score is 0.88. However, validation results decrease a little. Validation accuracy is now 0.17, precision is 0.22, recall is 0.17, and f1-score is 0.17.

As **run14** shows the change of *max_features* affects positively to training score, I wanted to check whether now using gini criterion would show a similar trend as entropy. As a result, R^2 training score is 0.79 and accuracy is 0.88. It is not quite as good as on **run14**. Nevertheless, it is better than the results on **run4**.

5.2.3 Optimisation of *min_samples_leaf* and *min_samples_split*

To continue looking for changes in parameters that improve the training score, the baseline will be as follows, according to **run14**: *criterion* is set to entropy, *max_depth* is 12, *min_samples_leaf* is 100, *min_samples_split* is 250, *n_estimators* is 1000, and *max_features* is *sqrt*. Parameter *n_estimators* is kept at 1000 despite decreasing it, it shows improvements, but I want to utilise many trees, so many splits can be observed when the amount of samples required in a leaf and to do a split is changed.

For **run16** I kept the *min_samples_split* as 250 but decreased *min_samples_leaf* to 50. Letting the leaves expand with a smaller threshold this first experiment shows positive results as the metrics improve a little. The R^2 training score is 0.81 with accuracy of 0.89. The training precision increased to 0.87, recall to 0.90, and f1-score remains 0.88. There is no change in validation metrics.

Run17 introduces a decrease in the parameter *min_samples_split*. It is reduced to 200, so the maximum depth of a tree is more likely reached. This change further improves the training score to 0.82. The accuracy for the model with this change is 0.90. Compared to **run16**, the only other change in metrics is that the f1-score for training is 0.89 instead of 0.88.

On the **run18** the *min_samples_split* is further reduced to 175. Still, without changes in validation metrics, the R^2 training score increases to 0.83. Now, with 0.90 accuracy, the precision is 0.88, recall is 0.91, and the f1-score is 0.89. After these several sounds of experimenting of decreasing the threshold when a leaf is split further, there is a clear indicator that the training score improves.

As a test, I was curious to see whether the previous conclusion that decreasing

the number of trees improves the score still holds. **Run19, run20** and **run21** saw an increase in *n_estimators*. Each result did not yield any better results. However, the results didn't get worse either. For the next runs, **run22** and **run23**, I decreased the amount of trees in the forest to 750 and 500 trees respectively. To my surprise, these results decreased the training accuracy by a little. My assumption for this behaviour is that there are not enough trees to get enough votes from in order to form the final classification.

5.2.4 Further optimisation and testing

Going forward, I tried some modifications to **run18**, as well as testing some arbitrary values. **Run24** introduced a drastic decrease in the amount of trees in the forest. The original value of 1000 for this run is 50. The biggest change is in the running time. Where **run18** took over 2 hours to perform the balanced training, **run24** took under 10 minutes. There is barely any difference in the classification reports. Comparing **run24** to **run18**, the R^2 training score drops from 0.83 to 0.82. Accuracy remains 0.90 as well as precision of 0.88. Another slight decrease can be seen in the recall metric as it drops from 0.91 to 0.90. The f1-score remains the same on both runs: 0.89.

For **run25** I changed parameters *min_samples_leaf* and *n_estimators* to 75 and 100 respectively. This change yielded no differences in any of the metrics. Further increasing the *max_depth* parameter to 14 for **run26** also provided no changes. Out of these three runs, **run24** was the quickest with a time of just under 10 minutes whereas **run26** was nearly 15 minutes.

Run27 introduces the change of *criterion* from entropy to gini, while leaving *max_depth* at 14. Parameters *min_samples_leaf* and *min_samples_split* are changed to 50 and 150 respectively. This run has 100 trees in the forest. No bigger changes in metrics can be seen, however, training recall score increases from 0.90 to 0.91.

Run28 can best be compared to **run24**. *Criterion* is changed from entropy to gini, and *min_samples_split* is reduced from 175 to 100. The R^2 training score increases to 0.83, but none of the metrics change. The key change in this run is the running speed. The **run28** took just over 4 minutes to run.

This is all the testing I did for the different values, and along the way I proceeded to implement certain models to the whole cross-validation training cycle.

5.2.5 Full cross-validation cycle

The full cross-validation cycle is a time consuming process. It takes several days to fully run. It implements leave-one-out cross-validation for 6-class classification.

To be more specific, the form of cross-validation is leave-one-subject-out cross-validation. This means that given an iteration of the cross-validation, data from a certain tumour does not appear in training, validation and testing sets at the same time. If this happened, there would be data leakage. This would create highly optimistic results and would not be accurate for our objective.

The leave-one-subject-out cross-validation, more generally leave-cluster-out cross-validation, works in similar ways as leave-one-out cross-validation, and has shown relevancy in regards to proteins (Kramer and Gedeck, 2010). Essentially, let n be the amount of patients in the data set. The data is split into n clusters such that each cluster contains only data from one patient. Note, that there can be several occurrences of the same tumour from the same patient. Therefore, these need to be validated at the same time. A model is trained on all data but some ith subject that is left out for testing. Training data is further split into a new training set and a validation set. Then the output is predicted for the test data. Lastly, the model's performance is evaluated.

5.2.6 Full cross-validation results

The first full cross-validation run acts as a basis for the classification. This process is tedious to run and can take even days. However, the model architecture plays a role in the running time and having a smaller architecture can significantly affect the time to run the cross-validation cycle.

The first full run, labelled **CV1**, was given to me when I received the code. The parameters to this are as follows: *criterion* is set to gini, *max_depth* is 6, *n_estimators* is 50, and *max_features* is set 0.25, i.e., to one fourth of the total features. Parameters *min_samples_leaf* and *min_samples_split* are omitted, thus using the default values 1 and 2 respectively. Classification report on full data yields results that are comparable to purely guessing the type of tumour. The accuracy is 0.17, precision is 0.12, recall is 0.18, and f1-score is 0.13. This architecture has the best precision to classify LGm-2 type of tumours. The run **CV1** took around 40 hours to run, which I think purely is caused by each tree reaching the maximum depth due to no amount of samples were defined to be in a leaf, or to be split.

The second run, **CV2**, incorporated some results that I gathered from the balanced training. The *criterion* parameter is gini, *max_depth* is 12, *min_samples_leaf* is 100, *min_samples_split* is 225, *n_estimators* is 50, and *max_features* is sqrt. The running time is slightly less than on **CV1**, but the results are worse. In fact, the accuracy is 0.14, precision is 0.09, recall is 0.11, and f1-score is 0.10.

The next full run, **CV3**, sees few more changes including results from balanced

training. The *criterion* parameter is set to entropy, *max_depth* is 12, *min_samples_leaf* is 50, *min_samples_split* is 100, *n_estimators* is 50, and *max_features* is sqrt. This run took around 20 hours to run, which is significantly less than the two previous runs. The classification metrics are quite similar to run **CV1**. The accuracy is 0.16, precision is 0.12, recall is 0.14, and f1-score is 0.13.

Next, I tried for curiosity whether decreasing the depth of each tree would affect the results in a similar way as in balanced training. That is, in balanced training the results got worse when the depth of the trees is reduced. The run **CV4** sees only two changes to run **CV3**: the parameter *max_depth* is reduced to 8 from 12, and *criterion* is set to gini. Compared to 20 hours of running time, this took around 7 hours with more consistent metrics. The accuracy is 0.17, precision is 0.16, recall is 0.17, and f1-score is 0.16.

The run **CV5** sees a change of only one value compared to **CV4**. The amount of trees in a forest, *n_estimators*, is increased from 50 to 75. This improves the metrics a little bit, but also decreases the running time. The running time for this run is around 4 hours. The accuracy is 0.19, precision is 0.17, recall is 0.18, and f1-score is 0.17.

What is common in all these runs is that the tumour types LGm-1, LGm-2 and LGm-5 are the most accurately predicted. The LGm-3 type is never classified correctly in any of the runs.

5.2.7 Discussion about balanced training and cross-validation results

The balanced training runs provided promising scores overall. The training classification metrics varied quite a bit depending on the model architecture. Approximately 0.10 increase was present in all of the metrics as the model got smaller. The model saw several key modifications along the way. The *criterion* parameter switched between gini and entropy. During the balanced training, entropy seemed to be the slower option compared to gini on otherwise similar models. However, this wasn't the case for full leave-one-subject-out cross-validation. The cross-validation cycle didn't seem to have much difference in gini or entropy *criterion*.

The full cross-validation yielded the best results with *max_depth* of 6 or 8, which wasn't the case for the balanced training. The balanced training provided the best results with slightly deeper trees: either 12 or 14. Along with the parameters *min_samples_leaf* and *min_samples_split*, this value could increase the running time by a significant amount. If the latter two parameters allowed the splitting to happen in the leaves until the maximum depth, the results did not improve or worsen. Only the running time increased for no real gain. Thus, limiting the depth of the trees

creates more optimal results in a reasonable time.

The amount of trees in the forest, *n_esimators*, works in a similar way in both cases, the balanced training and the cross-validation. A common factor in both cases is that the results did not seem to change when the amount of trees exceeded the value of 100. Too large values seemed to only complicate the model. Therefore, most optimistic results were yielded by values between 50 and 100.

The parameter *max_features* set to sqrt seemed to work the best for both cases. Value of base 2 logarithm seemed to perform worse on each run I tried. For full cross-validation, selecting one fourth of the features gave decent results but were barely surpassed by the square root amount of maximum features.

In conclusion, the model architecture from the balanced training does not fully reflect into the full cross-validation cycle. Some further testing is required. The main difference in results came from the *max_depth* parameter. Smaller values work better for the full cross-validation. It is a possibility that with more available data deeper trees would provide more realistic results.

When it comes to our original hypothesis, these results leave quite a lot of room for improvement, whether machine learning could be applied to consistently separate tumour tissue from non tumour tissue. Again, more data, even synthetic data, could help to tackle this problem. At the time of writing this, some ideas revolving around synthetic data are being contemplated.

6 Conclusion

Decision trees are a strong tool to make decisions. They are simple to interpret, and they split data in a well-informed manner. A split is made using either Gini impurity or entropy. Decision trees are prone to overfitting. Where decision trees fall short, random forests try to find improvement. Random forests are less easy to interpret, but they yield more accurate results in classification opposed to a single decision tree. Due to the voting aspect of the random forests, overfitting of the data is less likely.

Random forests show potential as a method of classification for brain tumours. In other words, applying machine learning algorithms on Raman spectra is a promising approach in detecting malignant tumours. Decision trees evidently provide a strong basis for this method. In a six class tumour classification problem, improvement can be made by tuning the hyperparameters of the model. A lot is yet to be done, but this is a beginning.

During the process of the medical study on six class classification of brain tumours, I found out optimising a random forest model in a balanced training environment does not necessarily reflect a cross-validation cycle. However, optimising the balanced training provided gives insight to which hyperparameters affect the results.

Currently, a possible issue regarding classification can be related to the amount of data. Hence, creating synthetic data could provide more optimistic results. This is out of the scope of this thesis but the approach is being worked on.

As a result, I was able to improve the efficiency of the random forest model. When applied on the complete data and running the whole cross-validation cycle, a massive time save was achieved. The optimisation resulted in achieving merely one tenth of the original running time with similar results.

Despite no proper improvements regarding the classification results were made, I count this as a success. In the long run, the time saved makes further classification much more efficient.

References

Abdelrahman Ellwaa, Ahmed Hussein, Essam AlNaggar, Mahmoud Zidan, Michael Zaki, Mohamed A Ismail, and Nagia M Ghanem. Brain tumor segmantation using random forest trained on iteratively selected patients. In *International workshop on Brainlesion: glioma, multiple sclerosis, stroke and traumatic brain injuries*, pages 129–137. Springer, 2016.

Houman Sotoudeh, Omid Shafaat, Joshua D Bernstock, Michael David Brooks, Galal A Elsayed, Jason A Chen, Paul Szerip, Gustavo Chagoya, Florian Gessler, Ehsan Sotoudeh, et al. Artificial intelligence in the management of glioma: era of personalized medicine. *Frontiers in oncology*, 9:768, 2019.

Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1-2):285–297, 1998.

Sotiris B Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013.

Obinna Chilezie Njoku. Decision trees and their application for classification and regression problems. *MSU Graduate Theses*, 3406, 2019.

James N Morgan and John A Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American statistical association*, 58(302):415–434, 1963.

Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. The Wadsworth statistics / probability series. Routledge, 1984.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

J Ross Quinlan. *C4.5: programs for machine learning*. The Morgan Kaufmann series in machine learning. Morgan Kaufmann Publishers, 1993.

Qasem A Al-Radaideh, Emad M Al-Shawakfa, and Mustafa I Al-Najjar. Mining student data using decision trees. In *International Arab Conference on Information Technology (ACIT'2006), Yarmouk University, Jordan*, 2006.

John J McArdle and Gilbert Ritschard. *Contemporary issues in exploratory data mining in the behavioral sciences*. Routledge, 2013.

Ross Jacobucci. Are decision trees stable enough for psychological research? *SN Applied Sciences*, 1(6), 2018.

Fahad Shahbaz Khan, Rao Muhammad Anwer, Olof Torgersson, and Göran Falkman. Data mining in oral medicine using decision trees. *International Journal of Biological and Medical Sciences*, 4(3), 2009.

Vili Podgorelec, Peter Kokol, Bruno Stiglic, and Ivan Rozman. Decision trees: an overview and their use in medicine. *Journal of medical systems*, 26(5):445–463, 2002.

Saman Rizvi, Bart Rienties, and Shakeel Ahmed Khoja. The role of demographics in online learning; a decision tree based approach. *Computers & Education*, 137: 32–47, 2019.

Carolina Crisci, Badih Ghattas, and Ghattas Perera. A review of supervised machine learning algorithms and their applications to ecological data. *Ecological Modelling*, 240:113–122, 2012.

Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

A Kong, P McCullagh, X-L Meng, D Nicolae, and Z Tan. A theory of statistical models for monte carlo integration. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(3):585–604, 2003.

Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

Pierre Geurts and Louis Wehenkel. Investigation and reduction of discretization variance in decision tree induction. In *European Conference on Machine Learning*, pages 162–170. Springer, 2000.

Ye Yuan, Liji Wu, and Xiangmin Zhang. Gini-impurity index analysis. *IEEE Transactions on Information Forensics and Security*, 16:3154–3169, 2021.

Suryakanthi Tangirala. Evaluating the impact of gini index and information gain on classification using decision tree classifier algorithm. *International Journal of Advanced Computer Science and Applications*, 11(2):612–619, 2020.

Laura Elena Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, 2004.

Haoyue Liu, MengChu Zhou, Xiaoyu Sean Lu, and Cynthia Yao. Weighted gini index feature selection method for imbalanced data. In *2018 IEEE 15th international conference on networking, sensing and control (ICNSC)*, pages 1–6. IEEE, 2018.

Badr Hssina, Abdelkarim Merbouha, Hanane Ezzikouri, and Mohammed Erritali. A comparative study of decision tree id3 and c4. 5. *International Journal of Advanced Computer Science and Applications*, 4(2):13–19, 2014.

Mary Slocum. Decision making using id3 algorithm. *Insight: River Academic J*, 8 (2), 2012.

David GT Denison, Bani K Mallick, and Adrian FM Smith. A bayesian cart algorithm. *Biometrika*, 85(2):363–377, 1998.

Frederik Questier, Raf Put, Danny Coomans, Beata Walczak, and Yvan Vander Heyden. The use of cart and multivariate regression trees for supervised and unsupervised feature selection. *Chemometrics and Intelligent Laboratory Systems*, 76 (1):45–54, 2005.

Roman Timofeev. Classification and regression trees (cart) theory and applications. *Humboldt University, Berlin*, pages 1–40, 2004.

Sonia Singh and Priyanka Gupta. Comparative study id3, cart and c4. 5 decision tree algorithm: a survey. *International Journal of Advanced Information Science and Technology (IJAIST)*, 27(27):97–103, 2014.

Maytal Saar-Tsechansky and Foster Provost. Handling missing values when applying classification models. 2007.

Aakash Parmar, Rakesh Katariya, and Vatsal Patel. A review on random forest: An ensemble classifier. In *International Conference on Intelligent Data Communication Technologies and Internet of Things*, pages 758–763. Springer, 2018.

Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, 2005.

Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

J Ross Quinlan et al. Bagging, boosting, and c4. 5. In *Aaai/iaai, Vol. 1*, pages 725–730, 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(3):e1301, 2019.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1):3133–3181, 2014.

Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. How many trees in a random forest? In *International workshop on machine learning and data mining in pattern recognition*, pages 154–168. Springer, 2012.

Jeremy Rogers and Steve Gunn. Identifying feature relevance using a random forest. In *International Statistical and Optimization Perspectives Workshop" Subspace, Latent Structure and Feature Selection"*, pages 173–184. Springer, 2005.

Danny Roobaert, Grigoris Karakoulas, and Nitesh V Chawla. Information gain, correlation and support vector machines. In *Feature extraction*, pages 463–470. Springer, 2006.

George H John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Machine learning proceedings 1994*, pages 121–129. Elsevier, 1994.

Jeremy D Rogers and Steve R Gunn. Ensemble algorithms for feature selection. In *International Workshop on Deterministic and Statistical Methods in Machine Learning*, pages 180–198. Springer, 2004.

T Jemima Jebaseeli, R Venkatesan, and K Ramalakshmi. Fraud detection for credit card transactions using random forest algorithm. In *Intelligence in Big Data Technologies—Beyond the Hype*, pages 189–197. Springer, 2021.

Md Zahangir Alam, M Saifur Rahman, and M Sohel Rahman. A random forest based predictor for medical data classification using feature ranking. *Informatics in Medicine Unlocked*, 15:100180, 2019.

Quan Zou, Kaiyang Qu, Yamei Luo, Dehui Yin, Ying Ju, and Hua Tang. Predicting diabetes mellitus with machine learning techniques. *Frontiers in genetics*, 9:515, 2018.

Cuong Nguyen, Yong Wang, and Ha Nam Nguyen. Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic. 2013.

Paul Rostron, Safa Gaber, and Dina Gaber. Raman spectroscopy, review. *laser*, 21: 24, 2016.

Zanyar Movasaghi, Shazza Rehman, and Ihtesham U Rehman. Raman spectroscopy of biological tissues. *Applied Spectroscopy Reviews*, 42(5):493–541, 2007.

Ketan Gajjar, Lara D Heppenstall, Weiyi Pang, Katherine M Ashton, Júlio Trevisan, Imran I Patel, Valon Llabjani, Helen F Stringfellow, Pierre L Martin-Hirsch, Timothy Dawson, et al. Diagnostic segregation of human brain tumours using fourier-transform infrared and/or raman spectroscopy coupled with discriminant analysis. *Analytical Methods*, 5(1):89–102, 2013.

Aneta Aniela Kowalska, Sylwia Berus, Łukasz Szleszkowski, Agnieszka Kamińska, Alicja Kmiecik, Katarzyna Ratajczak-Wielgomas, Tomasz Jurek, and Łukasz Zadka. Brain tumour homogenates analysed by surface-enhanced raman spectroscopy: Discrimination among healthy and cancer cells. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy*, 231:117769, 2020.

David Tuschel. Raman spectroscopy and polymorphism. *Spectroscopy*, 34(3):10–21, 2019.

Roger M Jarvis and Royston Goodacre. Discrimination of bacteria using surface-enhanced raman spectroscopy. *Analytical chemistry*, 76(1):40–47, 2004.

Holly J Butler, Lorna Ashton, Benjamin Bird, Gianfelice Cinque, Kelly Curtis, Jennifer Dorney, Karen Esmonde-White, Nigel J Fullwood, Benjamin Gardner, Pierre L Martin-Hirsch, et al. Using raman spectroscopy to characterize biological materials. *Nature protocols*, 11(4):664–687, 2016.

Tobias Meyer, Norbert Bergner, Christoph Krafft, Denis Akimov, Benjamin Dietzek, Jürgen Popp, Christiane Bielecki, Bernd FM Romeike, Rupert Reichart, and Rolf Kalff. Nonlinear microscopy, infrared, and raman microspectroscopy for brain tumor analysis. *Journal of biomedical optics*, 16(2):021113, 2011.

Zhiwei Huang, Annette McWilliams, Harvey Lui, David I McLean, Stephen Lam, and Haishan Zeng. Near-infrared raman spectroscopy for optical diagnosis of lung cancer. *International journal of cancer*, 107(6):1047–1052, 2003.

Guolan Lu and Baowei Fei. Medical hyperspectral imaging: a review. *Journal of biomedical optics*, 19(1):010901, 2014.

Ian J Maybury, David Howell, Melissa Terras, and Heather Viles. Comparing the effectiveness of hyperspectral imaging and raman spectroscopy: a case study on armenian manuscripts. *Heritage science*, 6(1):1–15, 2018.

Maria Nazir, Sadia Shakil, and Khurram Khurshid. Role of deep learning in brain tumor detection and classification (2015 to 2020): A review. *Computerized Medical Imaging and Graphics*, 91:101940, 2021.

MWN Jinadasa, Amila C Kahawalage, Maths Halstensen, Nils-Olav Skeie, and Klaus-Joachim Jens. Deep learning approach for raman spectroscopy. *Recent Developments in Atomic Force Microscopy and Raman Spectroscopy for Materials Characterization*, 2021.

Christian Kramer and Peter Gedeck. Leave-cluster-out cross-validation is appropriate for scoring functions derived from diverse protein data sets. *Journal of chemical information and modeling*, 50(11):1961–1969, 2010.

A Python code for a decision tree classification

```
# Imports.
import pandas as pd # To handle dataframes.
from sklearn.datasets import load_iris # Iris dataset.

iris = load_iris() # Load the iris dataset

# Creating column labels.
columns = ['sepal_length', 'sepal_width', 'petal_length', '
                                    petal_width']

# Creating a dataframe of iris data set with created column labels.
df = pd.DataFrame(iris.data, columns = columns)

# Adding a target column with known target data.
df['target'] = iris.target

# Makes creating training and testing splits easy
from sklearn.model_selection import train_test_split

# Creating training and testing data by splitting the data.

# Creating training and testing sets. 25\% data is used for testing
                                    .

X_train, X_test, y_train, y_test = train_test_split(df.drop(['
                                    target'], axis = 'columns'), iris
                                    .target, test_size = 0.25)

# Decision tree classifier.
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier() # Create a classifier.

# Fit the training set into the classifier.
clf.fit(X_train, y_train)


# Score the classifier and see how accurately it has predicted the
                                    targets.
clf.score(X_test, y_test)
```

B Iris dataset: decision tree splits

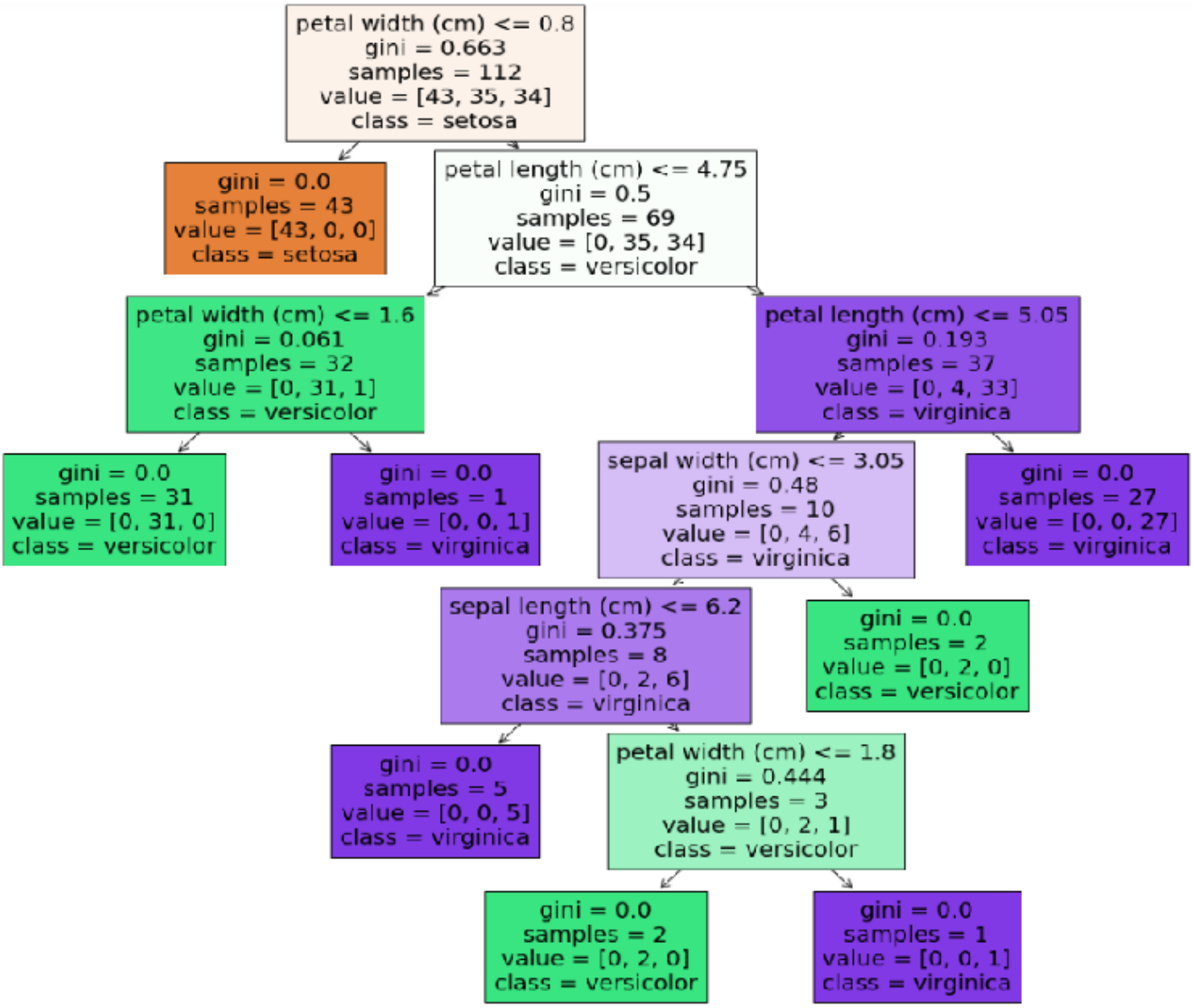

Figure 6: Iris dataset's decision tree splits.