Alma Mater Studiorum · University of Bologna

**School of Science**
**Department of Physics and Astronomy**
**Master Degree in Physics**

# Prediction of cancer trajectories by statistical learning on radiomic features

**Supervisor:**
**Prof. Gastone Castellani**

**Co-supervisor:**
**Dr. Claudia Sala**

**Submitted by:**
**Gianluca Carlini**

Academic Year 2019/2020

**Abstract**

Radiomics refers to the analysis of quantitative features extracted from medical images including Positron Emission Tomography (PET), Computerized Tomography (CT), Magnetic Resonance Imaging (MRI), and other medical imaging techniques. Radiomic features can be used to build models providing diagnostic, prognostic, and predictive information.

The aim of this work was to build a machine learning model able to predict survival probability in 85 cervical cancer patients, using the radiomics features extracted from CT and PET medical images. A thorough feature selection process was conducted employing different techniques to select the best predictors among the original features in the dataset. In particular, the Genetic Algorithm revealed to be the best of the feature selection methods employed, with promising applications in the field of radiomics.

Two different survival models have been developed, a Cox proportional hazard model and a Random Survival Forest. A Decision Tree Classifier was also implemented as a further model to evaluate.

All the models were trained on 80% of the available data and tested on the remaining 20%. The Concordance Index (CI) was used as the evaluation metric for the two survival models, while the area under the Receiver Operating Characteristic curve (ROC AUC) was used as the evaluation metric for the classifier.

The Cox model trained using 9 selected CT features was superior to all the other models tested. It achieved a Concordance Index score of 0.71 on the test set, showing promising predictive capabilities on external data.

Finally, the recurrence outcome was used as an additional feature, producing a general improvement of all the models.

# Contents

# Chapter 1

# Introduction

## 1.1 Radiomics

Radiomics refers to the field of medicine and data science in which a large number of quantitative imaging features are extracted from medical images and successively analyzed to build models providing diagnostic, prognostic, and predictive information.
One of the most interesting aspects of this research field resides in the fact that the features we mentioned, extracted from Computerized Tomography (CT), Positron Emission Tomography (PET), and Magnetic Resonance Imaging (MRI), are completely meaningless for a human observer, while a machine learning model can find useful patterns in them.
The radiomics workflow can be structured in the following five phases (Kumar et al., 2012; Lambin et al., 2017):

(i) Image acquisition and reconstruction

(ii) Image segmentation and rendering

(iii) Feature extraction

(iv) Exploratory analysis

(v) Model development

### Image acquisition and reconstruction

The first step in radiomic analyses is the choice of an imaging protocol, the volume of interest (VOI), and a prediction target, e.g., recurrence or survival.
In routine clinical imaging, there are a lot of acquisition parameters that can widely vary, such as slice thickness, image resolution, the intensity of the beam, and so on. Moreover, the reconstruction algorithms themselves can be largely different, introducing additional variations. This broad set of possible configurations makes the comparison of images obtained in different institutions very difficult and represents one of the most important challenges to overcome in the field of radiomics.

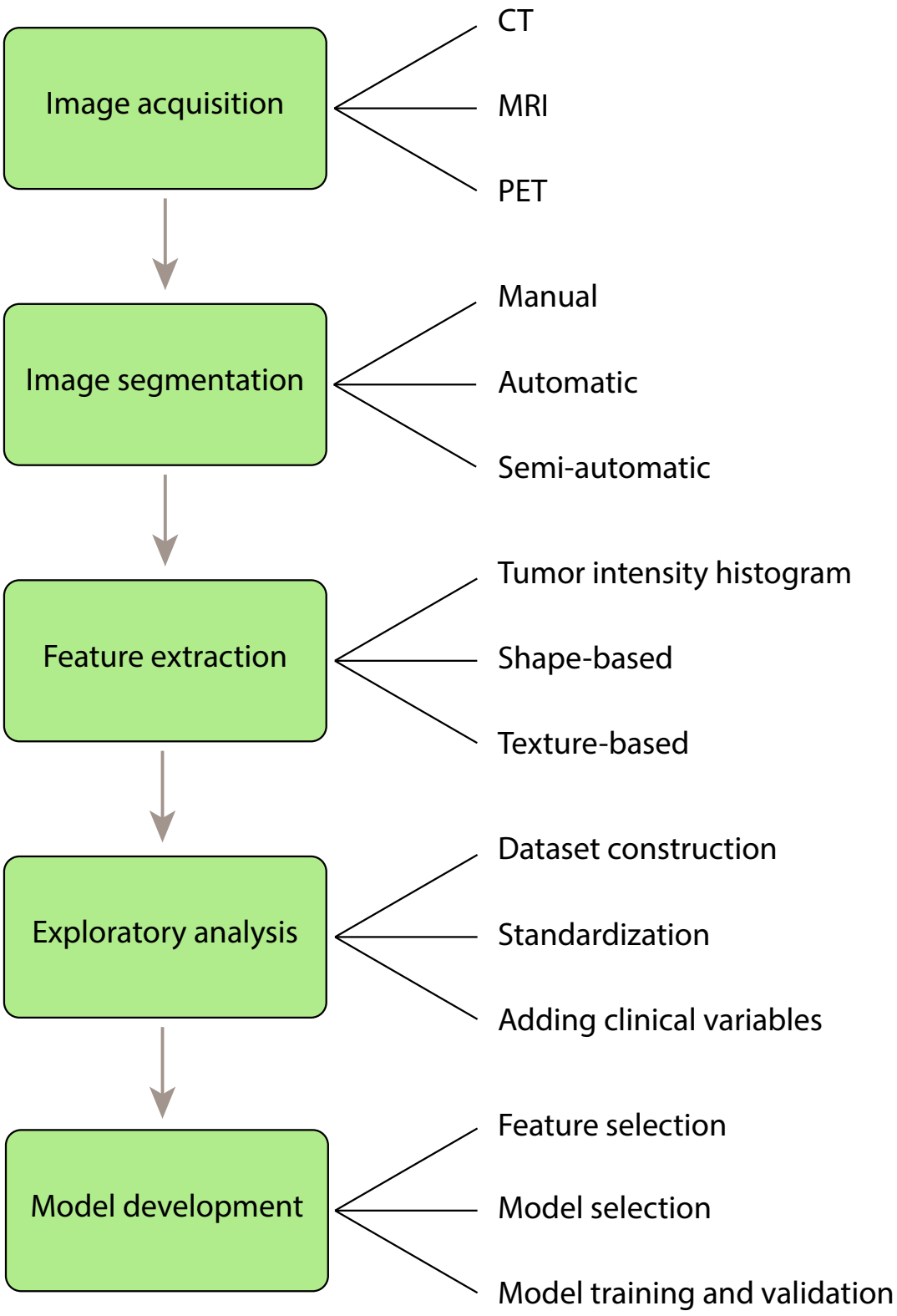



Figure 1.1: Schematic representation of the radiomics workflow

### Image segmentation and rendering

The image acquisition is followed by the image segmentation step, in which the VOIs are identified. The segmentation process is a crucial step in the radiomic workflow since it directly affects the features that will be extracted from the image. Images can be segmented either manually or by using segmentation algorithms. Manually segmented images are often considered as the ground truth, however, the manual segmentation process is very time consuming, therefore it becomes infeasible when a great number of images is acquired; moreover, it is also highly dependent on the operator who performs the segmentation.
Many segmentation algorithms, like region-growing methods or *snake* algorithms, are commonly employed in medical imaging. They are much more rapid than the human counterpart, but the final result depends on the algorithm used, and it is also influenced by the image noise and initialization parameters of the algorithm.

### Feature extraction

An incredibly high number of radiomic features can be extracted from a medical image. Starting from the *tumor intensity histogram*, it is possible to obtain a wide number of explanatory variables, such as *minimum*, *maximum*, *kurtosis*, *skewness*, and so on.
From the 3D representation of the tumor, *shape-based* features can be extracted, like *volume* and *surface*, from which is possible to calculate other informative variables, as the *surface-volume ratio* or the *sphericity* of the tumor.
Finally, a huge number of *texture-based* features can be extracted, such as *energy*, *entropy*, *contrast*, *gray level non-uniformity*, and many more.
It is not unusual to end up with a number of features that largely exceed the number of available samples. Some of those features are likely to be redundant, and others may not be informative for the problem at hand; for this reason, it is important to define a strategy to identify and keep the useful features only.

### Exploratory analysis

Once the features have been extracted, they can be used to build a radiomic dataset. To do this, we first need to organize the extracted data in a format that can be used in a machine learning model, i.e., typically, a matrix with features as columns and samples as rows. Successively, the data have to be combined with the target outcomes. When possible, non-radiomic features, like age or sex, can be added to the dataset since they could provide additional information and improve the overall performance of the model. It is also important to examine the goodness of the data, for example, by checking the presence of null values and outliers. Moreover, depending on the model we want to use, it can be helpful to standardize the data.

### Model development

The last step of the radiomic workflow is the construction of the model to analyze the data. Usually, as we said earlier, the first thing to do is to use one or more feature selection techniques to identify the most informative feature subset. The feature selection process is crucial since the performance of a machine learning model strongly depends on the features used. In particular, including a large number of features in a model will probably cause it to overfit the training set, compromising its prediction capabilities on new data.

Once we identified the candidate feature subset, it is important to carefully choose the optimal machine learning model for the problem at hand. When possible, it can be helpful to build more than one model and compare their performance.
Finally, the model has to be validated using an adequate score function or metric. It is possible to verify the model performance on the training data using methods like cross-validation, however, it is preferable to also validate the model on a test dataset containing unseen data. The performance obtained for the training and the test sets should be coherent, showing that the employed model is solid.

### 1.1.1 Applications and Challenges of Radiomics

#### Applications of Radiomics

In the last few years, radiomic features were revealed to be promising predictors in several cancer-related studies (Yip and Aerts, 2016). Those features provide quantitative measures of tumor characteristics that would otherwise be inaccessible.
In an MR study conducted on patients with head-and-neck cancer, a significantly greater increase in average ADC (apparent diffusion coefficient) was observed in tumors that responded badly to treatment when compared to tumors with a better response (King et al., 2013).
Lucia et al. (2018) showed that radiomic features extracted from PET/CT and MR images, such as *Grey Level Non Uniformity* and *Entropy*, have been found to be significant predictors of recurrence in patients with cervical cancer, with higher prognostic power than usual clinical variables.
Also, features extracted from CT images of 182 lung cancer patients were revealed to be predictive for the risk of developing distant metastasis (Coroller et al., 2015).
Moreover, in some situations, machine learning models trained on radiomic features showed better prediction capabilities than expert physicians. In particular, Mattonen et al. (2016) conducted a study on lung cancer patients comparing physician assessment to a radiomic model for the detection of recurrence. They found that when determining early prediction of recurrence, within 6 months before the treatment, physicians judged the majority of lesions as benign, i.e., no recurrence, with a false-negative rate of 99%. The radiomic model, instead, demonstrated better prediction capacities, with a false-negative rate of 24%, suggesting that statistical models can detect early changes associated with local recurrence that are usually neglected by physicians.
The aforementioned researches show that radiomic features can be employed for the prediction of treatment response and outcomes. These results open to the possibility of developing personalized treatment plans based on the radiomic features extracted from medical images.

Furthermore, various studies revealed that radiomic features can also be employed to investigate tumor staging and to differentiate between malignant and non-malignant tissues. For instance, Mu et al. (2015) showed that machine learning models can be built using radiomic features to classify early-stage and advanced-stage tumors. The early identification of the tumor stage provides important information that can be used to accordingly adapt the treatment plan. Nie et al. (2008) built a neural network that, utilizing multiple radiomic features extracted from MR images of breast lesions, was able to discriminate between malignant and benign tumors.

Finally, radiomic variables have also been related to tumor genetics. A famous study

by Aerts et al. (2014) showed that some radiomic features extracted from CT images of lung tumors could be associated with underlying gene-expression patterns.
Gutman et al. (2015) also found that MRI volumetric radiomic features were strongly associated with mutation status and could predict several DNA mutations in glioblastoma. Similarly, in a very recent study, Jacob et al. (2021) showed that MRI radiomic variables were associated with specific gene-expression profiles in endometrial cancer patients. The encouraging connection between features extracted from medical images and tumor genetics may help in developing powerful prognostic models and personalized treatment strategies.

### Radiomics limitations

Several factors can influence the radiomic features extracted from medical images, and consequently, the models developed using those features.
First of all, the acquisition processes suffer from high variability. For instance, the image resulting from a CT acquisition depends on a wide range of parameters, such as the beam intensity and energy, or the slice thickness. The final result is also strongly affected by the reconstruction algorithm used by the specific CT scanner. Modern CT machines use *iterative reconstruction algorithms* while, until lately, the vast majority of CT scanners used *filtered backpropagation*. Moreover, reconstruction artifacts due to the presence of metallic objects, respiratory motion, and beam hardening are often produced.
Several studies (Galavis et al., 2010; Yan et al., 2015; Pfaehler et al., 2018) showed that the extracted radiomic features can be very sensitive to acquisition and segmentation modalities, and it is still not clear why some radiomic variables are more robust than others. In addition, before the feature extraction, the voxel intensities within tumor volumes have to be discretized to a limited grey level range, and the number of gray level values can strongly influence the extracted variables.

Another significant obstacle, especially for PET acquisitions, is the scanner resolution. For small-size tumors, indeed, textural features fail to quantify the intratumoral heterogeneity, due to the resolution limits. For instance, Hatt et al. (2015) conducted a study on textural features extracted from 555 PET images, showing that, for tumors with volumes less than 10 $cm^3$, the textural features were highly correlated with the tumor volume, adding almost no further information. For bigger tumors, instead, volume and textural features were found to be independent predictors. CT acquisitions are less affected by tumor dimension because CT scanners generally have a better resolution.

A further impediment in radiomics is the small amount of available data. Generating a radiomic dataset is an extremely time-consuming and expensive task. For instance, years are typically required for patient follow-up. Moreover, sharing data between different institutions is often impossible due to the high variability in the data acquisition process, as we said before, and also because of privacy limitations. As a consequence, it is common to end up with datasets having many more features than samples. The main issue resulting from having a small and highly-dimensional dataset is that, even with a careful process of feature selection, it is possible to find features that accurately predict the outcome by purely random chance. For this reason, it would always be preferable to validate the model using unseen data.

Hopefully, in the next future, we will witness a progressive standardization of the processes of image acquisition, segmentation, and feature extraction, which will lead to easier data sharing between institutions and better reproducibility of the results. The availability of large and reliable radiomic datasets will be the basis for the construction

of powerful machine learning models that will assist physicians in planning personalized treatment plans, improving the general quality of healthcare.

## 1.2 Purpose of the Work

The purpose of this work was to build a machine learning model to predict survival probability in cervical cancer patients using radiomic features extracted from PET/CT images as predictors. Cervical cancer is both the fourth most common cause of cancer and the fourth most common cause of death from cancer in women. For this reason, it is essential to develop novel techniques to cope with this type of disease.
The research was conducted in collaboration with the S. Orsola University Hospital of Bologna to develop machine learning systems that will help physicians in planning better treatment plans. In particular, the feature selection process employed in this work has never been adopted before within the S. Orsola research group.
Since the dataset used in the work had a number of features that greatly exceeded the number of samples, particular attention was put in the feature selection process, aiming to obtain a substantially smaller set of predictive features. Various techniques have thus been employed for feature selection, ranging from the well-known **LASSO** regression to the more sophisticated and powerful **Genetic Algorithms**.
The modeling process consisted of the development of three different types of machine learning models, two survival models, and a classifier. The first survival model was the famous **Cox proportional hazard model**, which is one of the most used models in survival analysis due to its robustness and simple interpretability.
The second survival model was a **Random Survival Forest**, used primarily to explore the capability of these kinds of models of intrinsically performing feature selection.
Finally, since the outcome could also be considered a binary variable (survival or death), a **Decision Tree Classifier** was built. The choice of the Decision Tree Classifier was made mainly because of its superior interpretability; in fact, we did not expect any classifier to perform well in this case since a classifier does not take into account the time of the event and the censoring of the data. However, the Decision Tree graph can provide valuable information to understand the relations between features and data.
All the models have been trained on the 80% of the data and tested on the remaining 20%. The survival models were evaluated using the **concordance index**, as it is one of the most commonly used metrics in survival analysis. The classifier, instead, was evaluated using the area under the receiver operating characteristic curve (**ROC AUC**).

The entire analysis was conducted in parallel for CT and PET features, meaning that the feature selection process was performed for CT and PET features separately, and so was the modeling. Thus, in the end, for each type of model, a version was built using the PET features and a version using the CT features.

# Chapter 2

# Materials and Methods

This chapter is divided into two main parts: the first part will be a theoretical introduction and explanation of the techniques used in the work, while, in the Methods section, their actual implementation will be shown and discussed. Following the pipeline of the project, we will first introduce the dimensionality reduction procedures, then the Machine Learning models will be presented, and, finally, we will discuss the evaluation metrics and processes.

## 2.1 Dimensionality Reduction

The number of features that can be extracted from a medical image, like a CT scan, is extraordinarily high; it is not uncommon to end up with a database having more features than samples. When building a machine learning model, this can be a serious issue, known as the *curse of dimensionality*.
The problem is that there is plenty of space in high dimensions. For example, if we randomly pick two points in a unit square, the distance between these two points will be, on average, roughly 0.52. If we do the same in a unit three-dimensional cube, the average distance becomes roughly 0.66, and things get worse and worse as the number of dimensions increases.
As a result, in high-dimensional datasets, most of the training instances are likely to be far away from each other, making those kinds of datasets very sparse; consequently, new instances will likely be far away from any training instance, yielding predictions much less reliable than in lower dimensions.
Increasing the size of the dataset could be a solution, but, usually, acquiring new data, especially labeled data, is a very expensive, time-consuming task. Furthermore, the number of instances required to reach a given density grows exponentially with the number of dimensions.

The field of dimensionality reduction can be divided into two macro areas: **Feature Selection** and **Feature Extraction**. The main difference between them is that the former tries to reduce the dimensionality of the dataset by finding a subset of the original feature set, while the latter creates a new set of features capable to describe the data.
For the purpose of this work, we mostly used feature selection techniques, but, for visualization reasons, a feature extraction method has been employed, and it will be presented first.

### 2.1.1 t-distributed Stochastic Neighbor Embedding (t-SNE)

**t-SNE** is a powerful manifold learning technique, proposed by Van der Maaten and Hinton (Maaten and Hinton, 2008), that converts the affinities of data points to probabilities. The affinities in the original space are represented by Gaussian joint probabilities and the affinities in the embedded space are represented by Student's t-distributions. When compared to other existing techniques, t-SNE shows several advantages:

- It is particularly sensitive to local structure
- Reveals the structure at many scales on a single map
- Reveals data that lie in multiple different manifold or clusters
- Reduces the tendency to crowd points together at the center

The ability of t-SNE to group samples based on local structures makes it one of the best candidates for visualizing high-dimensional data in two or three dimensions.

To understand how the algorithm works we first define our conditional probabilities $p_{j|i}$ and $q_{j|i}$.
$p_{j|i}$ is the conditional probability that $x_i$ would pick $x_j$ as its neighbor if neighbors where picked in proportion to their probability density under a Gaussian centered at $x_i$. Mathematically, it is defined as:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)} \tag{2.1}$$

It represents the similarity of the point $x_j$ to the point $x_i$. $\sigma_i$ is of course the variance of the Gaussian centered on $x_i$.
A similar conditional probability can be defined for the lower-dimensional counterparts of $x_j$ and $x_i$, $y_j$ and $y_i$; we will call this conditional probability $q_{j|i}$. The variance of the Gaussian that is employed to calculate the conditional probabilities $q_{j|i}$ is set to $1/\sqrt{2}$, which will only produce a rescaled version of the final map. In this way, $q_{j|i}$ takes the form:

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k\neq i}\exp(-\|y_i - y_k\|^2)} \tag{2.2}$$

Now, the purpose of the algorithm is to minimize the sum of **Kullback-Leiber divergences** of the joint probabilities in the original space and the embedded space by gradient descent. The Kullback-Leiber divergence is a nonsymmetric measure of the difference between two probability distributions P and Q. More specifically, it is a measure of the information lost when Q is used to approximate P.
So, finally, the cost function of the algorithm is given by:

$$C = \sum_i D_{KL}(P_i||Q_i) = \sum_i \sum_j p_{j|i}\log\frac{p_{j|i}}{q_{j|i}} \tag{2.3}$$

in which $P_i$ represents the conditional probability distribution over all other data points given data point $x_i$, and $Q_i$ represents the conditional probability distribution over all other map points given map point $y_i$.
Note that the Kullback-Leiber divergence is not convex, so multiple restarts with different initialization will end up in a local minimum of the KL divergence.
Even if t-SNE is capable to produce almost miraculous mappings, it has some important drawbacks that should be considered:

- It is computationally expensive; its complexity is $O(dN^2)$, where d is the number of output dimensions and N is the number of samples.
- The algorithm is stochastic, so multiple restarts lead to different embeddings.
- Global structure is not explicitly preserved.

Furthermore, it is not straightforward to interpret the maps produced by t-SNE and they can also be misleading, as they strongly depend on the parameters of the algorithm.
Among those parameters, one of the most important is *perplexity*, which loosely says how to balance attention between local and global aspects of the data.
Perplexity can in some sense be thought of as an estimation of the number of close neighbors each point has.

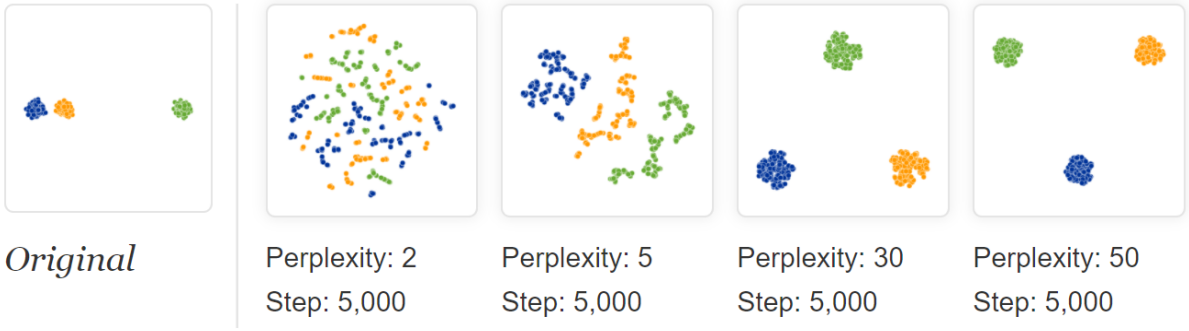



Figure 2.1: t-SNE diagrams for three Gaussian clusters with 200 point each, showing how the produced maps are affected by the perplexity value (Wattenberg et al., 2016)

To have an idea of how this parameter affects the resulting map, we can take a look at Figure (2.1), from which we can also infer another important characteristic of the t-SNE algorithm: the distance between the identified clusters can be meaningless; as we can see, in fact, the original distances are not preserved in any of the produced maps.

### 2.1.2 Correlation

Usually, when we have to reduce the dimensionality of the data, the first thing to do is to check the correlations between features.
A feature that is highly correlated to another, in fact, does not really provide new information about the data; instead, it will add a certain amount of noise to the data and will likely end up penalizing the performance of the predictive model.
There are several types of correlation coefficients, but the **Pearson's correlation coefficient**, usually called the **Pearson's r**, is by far the most famous one.
Given a set of $n$ pairs of data $\{x_i, y_i\}$, with $i = 1, \ldots, n$ the Pearson's correlation coefficient $r_{xy}$ is defined as follows:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.4}$$

where $\bar{x}$ and $\bar{y}$ are the average $x$ and $y$ values.
The value of $r$ is between -1 and 1, with -1 being a total negative linear correlation, 0 being no linear correlation and 1 being a total positive linear correlation.
Notice that Pearson's correlation *only measures the linear correlations between data.* This

is a rather important point because it means that the $r$ coefficient gives us no information about more complicated correlations, as we can see in Figure (2.2).

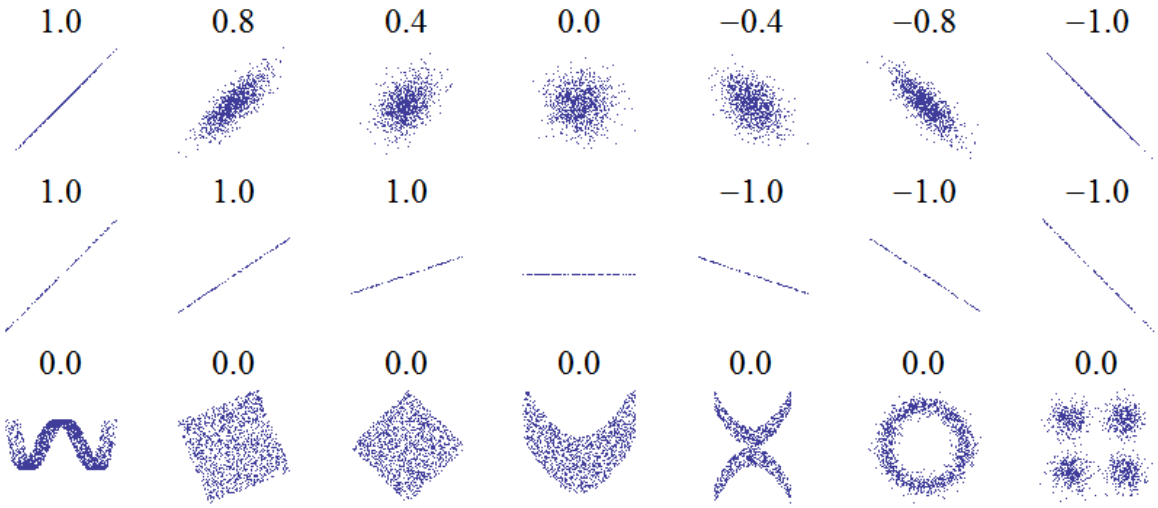



Figure 2.2: Multiple datasets with their own Person's correlation coefficients. The bottom row shows how Pearson's correlation does not detect non-linear correlations. The middle row, instead, shows how the value of $r$ reflects the strength and direction of the linear correlation, but not its slope.

As a prior step in a feature selection process, thus, it is possible to calculate the Pearson's correlation coefficient for all the features and then discard the ones with a coefficient higher than a certain threshold. The main issue with this procedure is that, among the two correlated features, it is not possible to know which one is the most informative, we can only discard one of the two arbitrarily.

### 2.1.3 LASSO Regression

LASSO is the acronym of Least Absolute Shrinkage and Selection Operator; it performs two main tasks: regularization and feature selection (Fonti and Belitser, 2017).
To understand how it works, we introduce the method for the simplest case: the linear regression.
A linear model is a model that describes the relationship between the response $Y_i$ and the explanatory variables $X_{ij}$. The case of one explanatory variable is known as *Simple Linear Regression*, while the case with two or more explanatory variables is called *Multiple Linear Regression*.
A linear model can be represented as follows

$$Y_i = \beta_0 + x_{i1}\beta_1 + \ldots + x_{ik}\beta_k + \epsilon_i \quad , \quad i = 1, \ldots, n \tag{2.5}$$

where the parameters $\beta_0, \ldots, \beta_k$ are the regression coefficients, k is the number of explanatory variables, and $\epsilon_i$ represents the error. We can also write the (2.5) in vector form:

$$\vec{Y} = \beta_0 + \mathbf{X}\vec{\beta} + \vec{\epsilon} \tag{2.6}$$

where here $\vec{Y}$ is the $n \times 1$ vector of responses, $\mathbf{X}$ is the $n \times k$ matrix of the $n$ occurrences with $k$ explanatory variables, $\vec{\beta}$ is the $k \times 1$ coefficient vector, and $\vec{\epsilon}$ is the $n \times 1$ error vector.

The goal of linear regression is to fit a straight line to a set of points minimizing some objective function, for example, the mean squared error (MSE).
The LASSO regression also minimizes the mean squared error, but it adds a constraint on the model parameters; mathematically:

$$\hat{\beta} = \min_{\vec{\beta}} \left( \frac{\|\vec{Y} - \mathbf{X}\vec{\beta}\|_2^2}{n} + \lambda \|\vec{\beta}\|_1 \right) \tag{2.7}$$

where $\| \cdot \|_p$ is the $L_p$ norm and $\lambda \geq 0$ is the parameter that controls the strength of the penalty, the larger the value of $\lambda$, the greater the amount of shrinkage. With $\lambda = 0$ the problem becomes an ordinary least square optimization.
When we solve the optimization problem (2.7), some of the $\vec{\beta}$ coefficients are shrunk to zero and, in this way, the features with coefficients equal to zero are excluded from the model.

### Geometric Interpretation

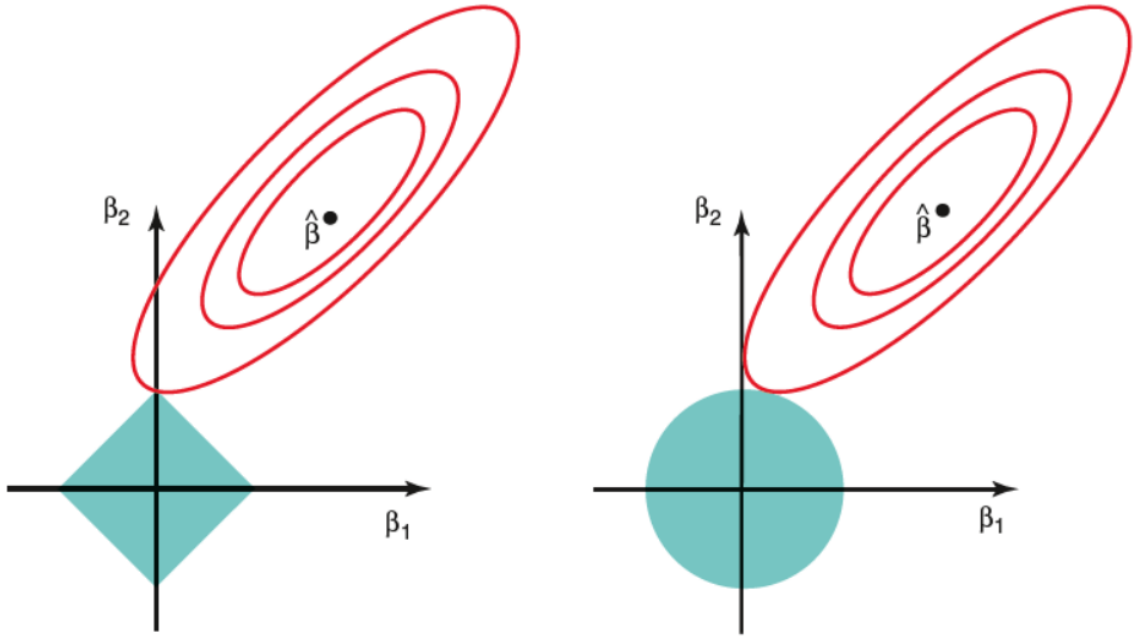



Figure 2.3: Difference between the LASSO regression (left) and the Ridge regression (right).

To better visualize how the feature selection is performed, it is useful to consider the geometrical point of view. In Figure (2.3) we can see a comparison between the famous *Ridge Regularization* and the LASSO regularization.
The constraint region for the Ridge regression is given by the condition: $\beta_1^2 + \beta_2^2 \leq t$, corresponding to the equation of a circumference. On the other hand, the LASSO condition is $\beta_1 + \beta_2 \leq t$, yielding a constraint region equal to a diamond (here $t = 1/\lambda$). The ellipses in the figure are the borders of the mean squared error function, which has its minimum in correspondence with the point $\hat{\beta}$. The minimum of the overall objective function is found on the intersection between the ellipse and the constraint region.
Hence, the fundamental difference between these two methods is given by the shape of the constraint region; in particular, the diamond shape of the LASSO method has

corners, while, of course, the circular shape of the Ridge method has not. When the intersection point is found on one of the corners of the diamond region, one of the two coefficients will be equal to zero, and this cannot happen in the circular region case. The Lasso method, however, has some limitations:

- When the number of features $p$ is greater than the number of samples $n$, LASSO can select at most $n$ features.
- If there are grouped features (i.e. highly correlated features), LASSO tends to select one feature from each group ignoring the others.

### 2.1.4 Genetic Algorithms

Genetic Algorithms (GA) are a class of search algorithms inspired by the famous Darwin's theory of evolution. By imitating the process of natural selection, they can be used to solve various problems in the field of artificial intelligence, and they are also optimal candidates to perform feature selection.
In short, the Darwinian evolution principle states that, in a population of individuals, the ones who better adapt to the environment are more likely to survive, reproduce, and pass their characteristic traits to their offspring. In this way, as generations go by, species become more and more adapted to live in their environment.
In the case of Genetic Algorithms, each individual represents a candidate solution for the problem in exam. All the individuals are iteratively evaluated and those who better solve the problem get a higher chance of being selected and pass their characteristics to the next generation of individuals. So, like in Darwin's theory, as generations go by, candidate solutions will get better and better in solving the problem at hand. The function used to evaluate the individuals is called the **Fitness** function.

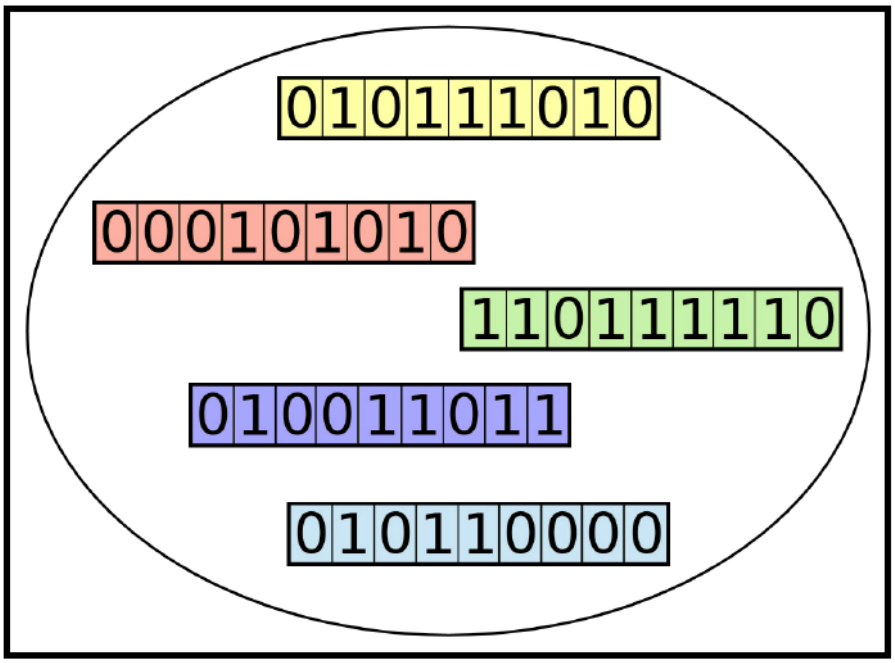



Figure 2.4: Example of a population of chromosomes with binary genes (Wirsansky, 2020)

Once again, trying to mimic natural processes, we say that an individual is represented by a **chromosome**, which, in turn, is just a sequence of **genes**. So, for example, a chromosome could be a binary string, where each gene is either a zero or a one. It follows that our collection of candidate solutions, i.e., the population, can be thought of as an ensemble of chromosomes.
The general workflow of a genetic algorithm is shown in Figure (2.5).

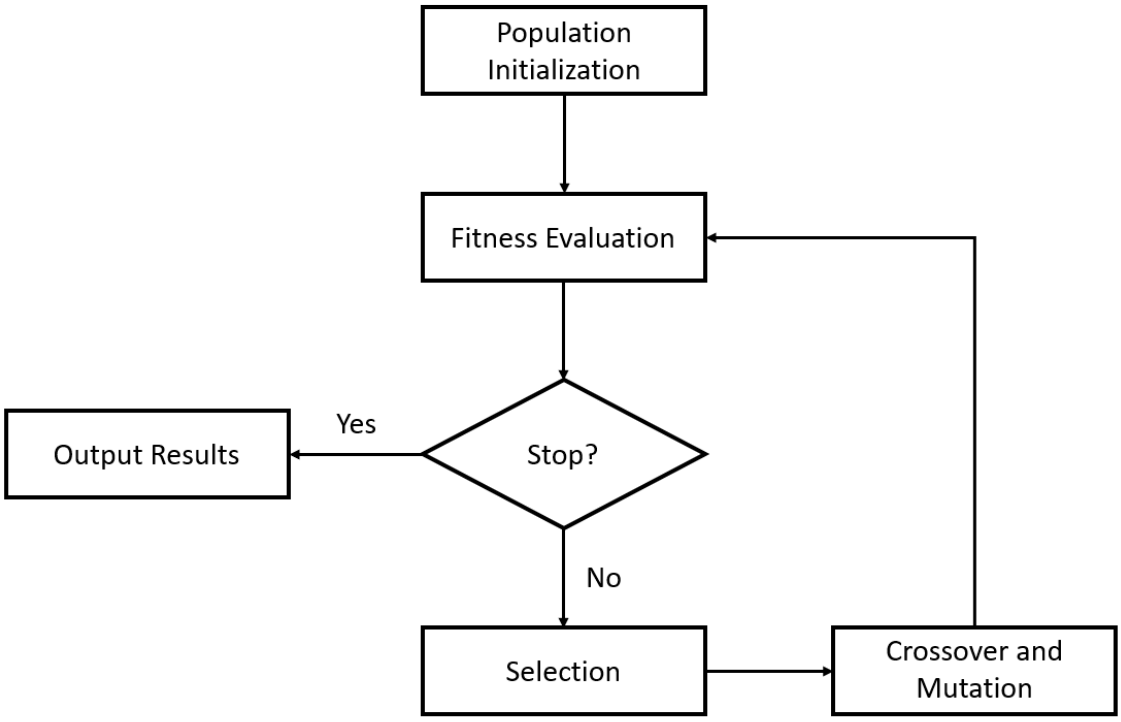



Figure 2.5: Genetic Algorithm flowchart. At first, the population is initialized, then the fitness of each individual is evaluated. Successively, if the stop condition is satisfied, the algorithm outputs the results, else, it performs crossover and mutation operations.

### Fitness Function

The *Fitness* function gives us a measure of the goodness of the solution to the problem at hand, so it is the function we want to optimize to obtain the best possible solution. We can either minimize or maximize it, depending on the specific situation.
One of the greatest advantages of using a fitness function is that the genetic algorithm needs the fitness value only, so it does not require derivatives or other information. This means that genetic algorithms can also optimize functions that are hard to differentiate, or that cannot be differentiated at all.
Moreover, relying on fitness, we can solve problems that do not have a mathematical representation, as long as we can obtain a fitness score. For instance, imagine we have a driving simulator in which we can tune some parameters of the car. We could use the genetic algorithm to find the parameter setting that produces the fastest car, using as fitness the time of the lap.
As an example, we can see in Figure (2.6) the minimization of one of the functions we used as fitness for our genetic algorithm.

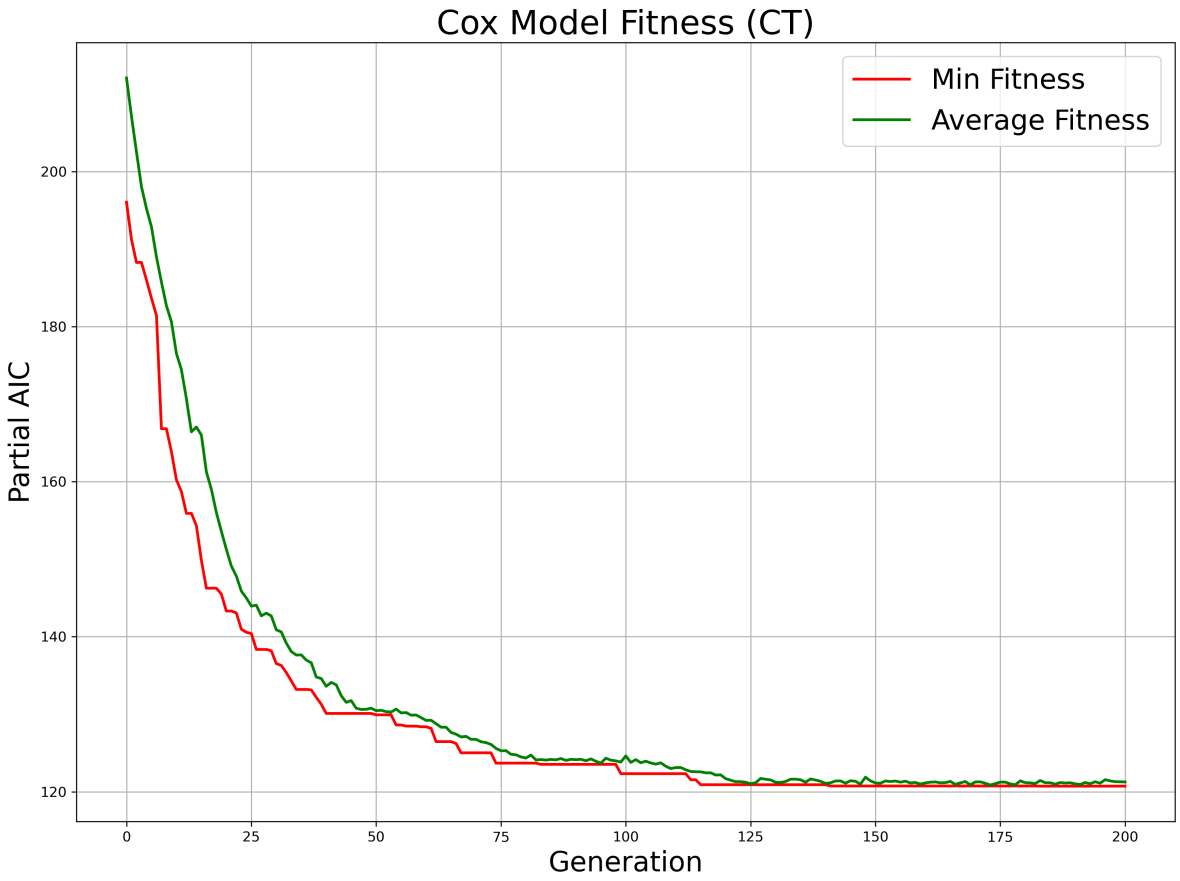



Figure 2.6: Evolution of the fitness value over the generations, with the average fitness value in green and the minimum fitness value in red.

The fitness is independently calculated for each individual at the beginning of the generation; the individuals with the best fitness value will have the greatest chance to be selected for the next generation.

### Selection

Right after the fitness calculation of every individual, a selection process is used to pick the individuals that will get to reproduce and create the next generation of candidate solutions. As we said, the individuals with the higher (or lower, depending on the problem) fitness value will have a greater probability of being selected; however, individuals with a low fitness value can still be chosen, in order to encourage genetic variety.
There are many possible selection strategies (Goldberg and Deb, 1991; Blickle and Thiele, 1996), inspired by both mathematical principles and similarities in nature.

We will only focus on the selection scheme used in this work: the **Tournament Selection**. In tournament selection (Fang and li, 2010), $k$ individuals are randomly chosen from the population, then the one with the highest fitness score wins and gets selected. The process is repeated $N$ times to obtain the parents population. The tournament size, i.e., the number of individuals picked for the tournament, strongly influences the result of the process, in fact, increasing it decreases the probability of weaker individuals being selected.

The most common implementation of the tournament selection is the *binary tournament*, in which two individuals are picked to compete.

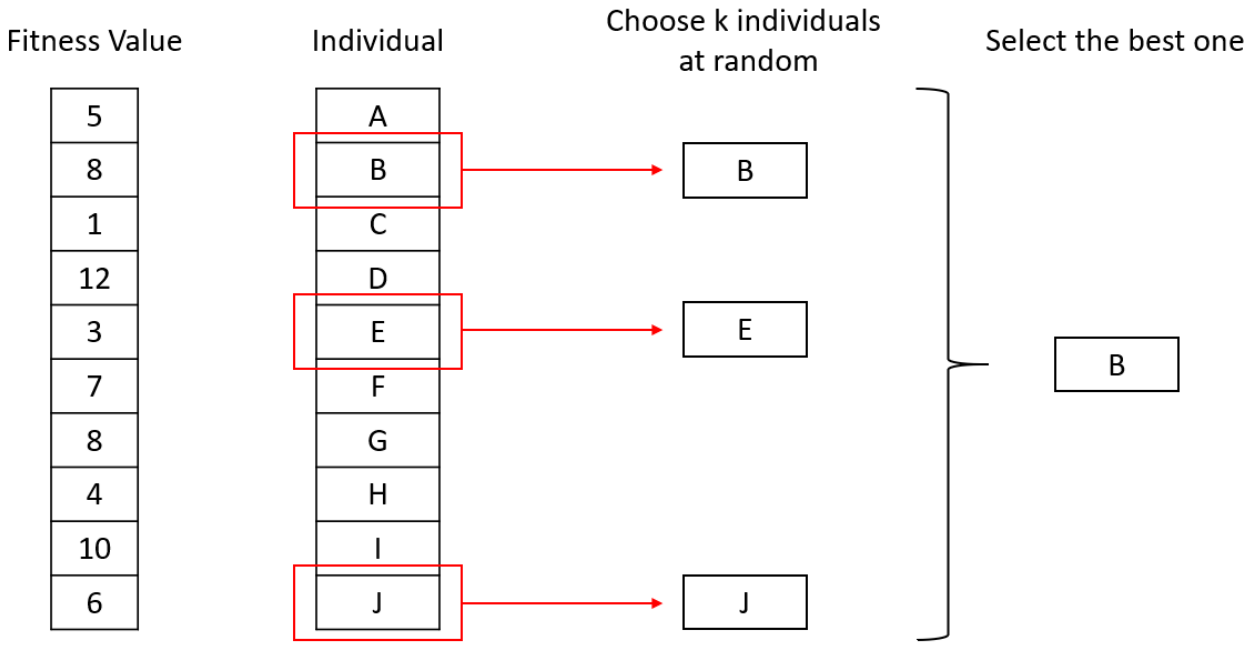



Figure 2.7: Example of Tournament Selection with tournament size equal to three. Three individuals, namely B, E and J, are randomly chosen, then, the one with the highest fitness, B, is selected.

The tournament selection does not require any population sorting, so it can be implemented very efficiently: its time complexity is $O(N)$, where $N$ is the number of repetitions.
Another interesting property of this selection method is that, as long as it can compare the individuals and determine which one is better, it does not require the actual fitness values. This may seem trivial but it is in fact an advantage; some selection techniques, for example, cannot work with negative values of the fitness function.

### Crossover

Once the selection process is ended and we have our population of parents, we are ready to let them reproduce to create the next generation.
The crossover step is quite simple: two parents are chosen from the current selected population and parts of their chromosomes are exchanged to create two new chromosomes representing the offspring. Typically, crossover is not always applied, it just happens with a high probability. When crossover is not applied, both parents are directly copied to the next generation.

Once again, there are many crossover techniques, but we will focus only on the one used in this work, the **two-point crossover**.
In the two-point crossover, as the name suggests, two crossover points are randomly selected on the chromosomes of both parents, then, the genes in the middle of those two points are swapped between the two parents. The process is shown in Figure (2.8).
This method can be generalized to $n$ crossover points; the easiest one, with $n = 1$ is known as **single-point** crossover.

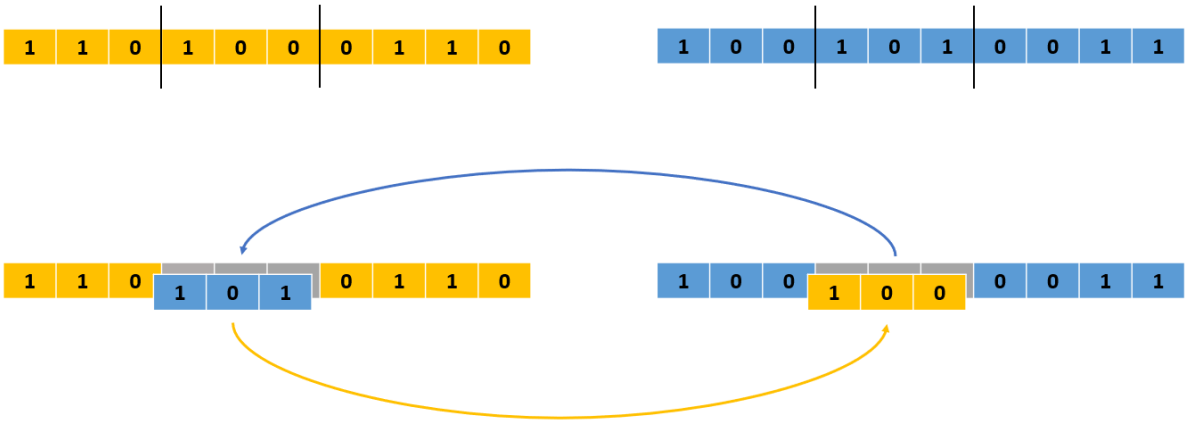



Figure 2.8: Example of a two-point crossover on binary chromosomes. Here the first crossover point is between the third and the fourth genes, while the second point is between the sixth and the seventh genes.

### Mutation

The last of the operators in the genetic algorithm is the mutation one. The mutation operator is applied to the individuals resulting from the previous selection and crossover operations.
The purpose of mutation is to provide a further element of randomness, in order to promote the exploration of the solution space. As opposite to the crossover operation, however, mutation is usually applied with a very low probability, because a high mutation rate could prejudice the results of the genetic algorithm, making it similar to a random search.


Figure 2.9: Example of a flip bit mutation. The values of the the second and eighth genes are flipped.

The mutation method used in the work is called **flip bit mutation**, and it is quite simple. When the flip bit mutation is applied to a binary chromosome, each gene of the chromosome has a certain probability of being flipped, i.e., of becoming the boolean negation of itself, as we can see in Figure (2.9).
Notice that the process just described can be subdivided into two steps, each with its own probability. First, there is the *probability of mutation*, that is the probability for a chromosome of being mutated; then, there is the *probability of flipping*, i.e., the probability for a gene of being flipped. These two probabilities are independent and can be controlled separately. As an example, consider the case in which the probability of mutation is equal to 1 and the probability of flipping is equal to 0. In a similar situation, all the chromosomes would be mutated, but the mutation would not have any effect.
It is also worth to explicit the fact that the flip bit mutation can only be used with binary chromosomes, as it would make no sense to use it with other types of chromosomes.

#### Stopping Conditions

The easiest way to stop the genetic flow is to simply define a maximum number of generations after which the algorithm must stop.
The other common solution involves the fitness function. If there is no substantial improvement of the fitness value over a certain number of generations, then the algorithm stops. A threshold can be used to define the minimum value improvement required.
In this implementation, we used the first of these two stopping conditions.

#### Elitism

The selection process we described has an intrinsic, important consequence: it does not guarantee that the best individual in the population will be selected.
Consider a population of 50 individuals and a tournament size of 2. In this case, if we repeat the tournament 50 times, the probability for the best individual not being selected is around 13 percent, so it is not negligible at all. Note that we are implicitly assuming that the same individual cannot be picked two times in a single tournament.
This is not really a big problem, because, even if the best individuals are lost during the selection process, it is probable that they will be reintroduced in the next generations. However, if we do not want to lose our best individuals during the evolution process, we can apply an **elitism** strategy.
The idea is very simple: before the selection operation takes place, the $n$ best individuals are selected and duplicated into the next generation. Those $n$ individuals can still be picked in the selection process, so $n$ should be a reasonably small number because otherwise we risk to prematurely saturate the population with sub-optimal solutions.
Applying an elitism strategy can really improve the performance of the genetic algorithm; in particular, it can reduce the time needed to find the best solutions.

## 2.2 Models

In the field of Machine Learning, we can distinguish two main types of algorithms: **supervised** and **unsupervised**. The principal difference between these two classes of algorithms is that the former need labeled data to learn, while the latter do not. **Clustering** is an example of unsupervised learning because it can learn relations in data without needing for labels.
When labels are available for the problem at hand, generally, supervised techniques perform better, so, since we had labeled data, we only used supervised algorithms.
The two major applications of supervised learning are **classification** and **regression**. Classification is the task of associating a categorical value, i.e., a class, to the input sample; regression, instead, is the action of mapping the input sample to a real number, or, more in general, to a vector of real numbers.
The data used in this work were suitable for both classification and regression.

### 2.2.1 Decision Trees

Decision Trees are versatile supervised machine learning algorithms that can perform both classification and regression. A tree is made up of a *root node*, *internal nodes*, *branches*, and *leaf nodes*. The root node is the starting point of the tree and both root

and internal nodes contain questions or criteria to be answered (e.g., is x greater than y?); branches are just connections between nodes and, finally, leaf nodes are the terminal nodes of the tree. Unlike real trees, Decision Trees grow upside down, with the root node at the top.
Each internal node of the tree, as well as the root node, can have two or more child nodes, depending on the algorithm implementation, however, we will focus on binary trees, i.e. trees in which each internal node has exactly two children because it is the implementation used for this work.

Decision Trees have various great qualities that contribute to their popularity and broad usage in machine learning.
First of all, they are *white box models*, meaning that the decision made by a tree is easily and clearly explainable by boolean logic; they are thus optimal candidates when the interpretability of the model is crucial. In contrast, machine learning models like neural networks are *black box models*, because the interpretation of a decision made by the model is not straightforward.

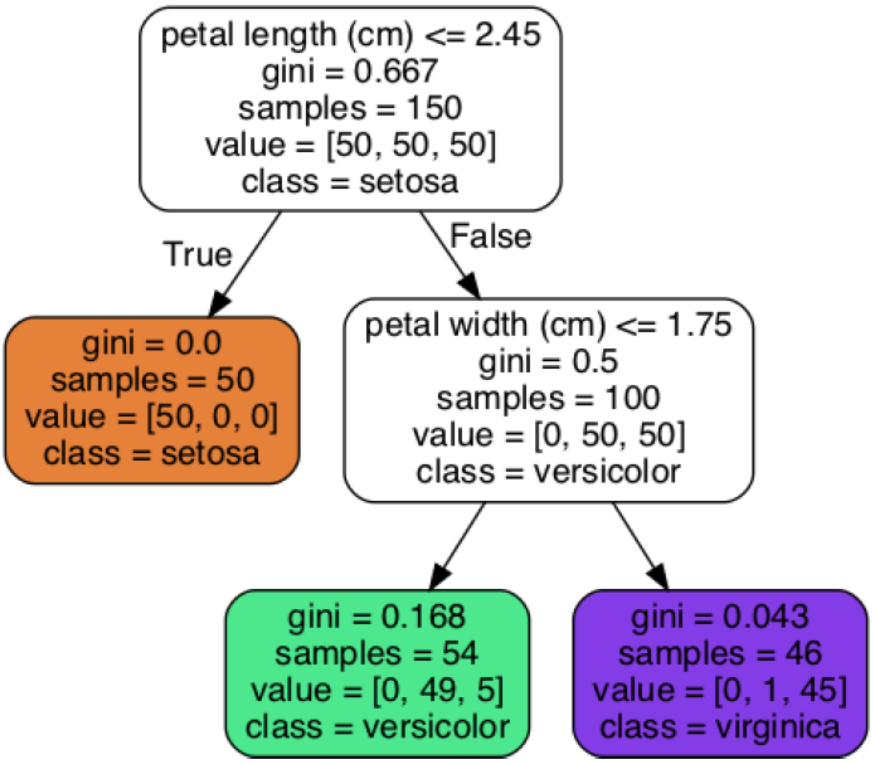



Figure 2.10: Decision Tree of the famous *Iris Dataset* (Géron, 2019).

Another major advantage is that Decision Trees require little data preparation, or even no data preparation at all. In fact, they can handle both numerical and categorical data and, thanks to their boolean nature, it is not required to normalize or scale the data.
Finally, the computational cost of using a tree is logarithmic in the number of data points used to train the tree.
To understand how Decision Trees make their decisions, we refer to Figure (2.10). As we can see, the root node contains the whole data, 150 samples equally partitioned in three classes: *setosa*, *virginica* and *versicolor*. The class associated with a node is the one with the greatest number of instances in that node. The dataset is divided into two subsets

by the condition petal length $\leq 2.45$, producing the corresponding child nodes. The left node is a terminal one, i.e., a leaf, since it holds only samples of the *setosa* class, so no further splitting is needed. The node on the right, instead, is an internal node, which, using the condition petal width $\leq 1.75$, generates the two final leaf nodes.
Moreover, all the nodes in the tree have an attribute in common, named *gini*. The gini attribute measures the node impurity, also known as **Gini Impurity**, defined as follows:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \tag{2.8}$$

where $i$ indicates the $i^{th}$ node and $p_{i,k}^2$ is the ratio of the class $k$ instances among the training instances in the $i^{th}$ node. So, for example, the leaf node on the lower right has a Gini Impurity equal to $1 - (0/46)^2 - (1/46)^2 - (45/46)^2 \simeq 0.043$.

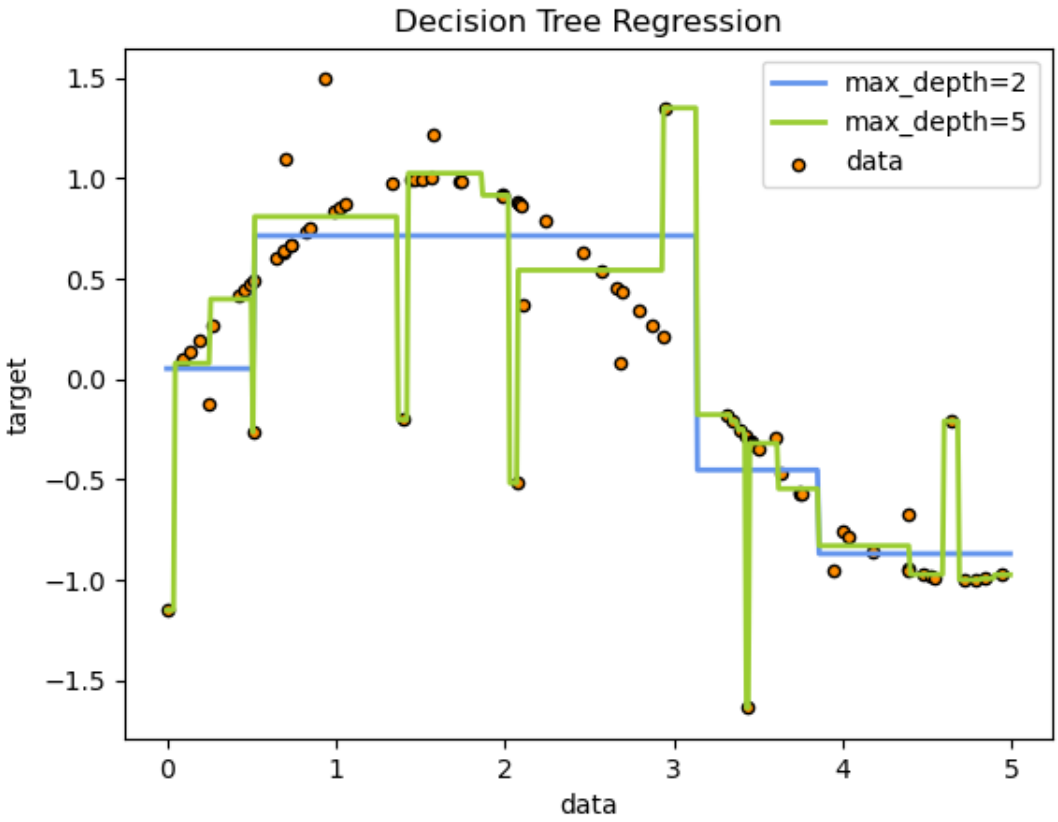



Figure 2.11: Decision Trees used to fit a sinusoidal curve with some noisy points. It can be seen how the hyperparameter *max_depth* influences the final result (Pedregosa et al., 2011).

A Decision Tree is built using the **CART** (*Classification and Regression Tree*) algorithm, which performs a greedy search trying to minimize a cost function. Given the feature set $\vec{k}$, the algorithm works by searching for the pair $(k_i, t_{k_i})$ that produces the purest subsets, where $t_{k_i}$ is a threshold value. To do this, at each node the algorithm tests various pairs and chooses the one that minimizes the following cost function:

$$J_{class}(k_i, t_{k_i}) = \frac{n_l}{n} G_l + \frac{n_r}{n} G_r \tag{2.9}$$

where $G_l$ and $G_r$ are the gini impurities of the left and the right subsets respectively, $n_l$ and $n_r$ are the numbers of samples in the left and right subsets, and $n$ is the total number of samples. In other words, the (2.9) is a weighted sum of the gini impurities of the subsets.
Note that the (2.9) is the cost function used when the Decision Tree is a classifier; if we are performing a regression task the CART algorithm will minimize the *Mean Squared Error* (MSE):

$$J_{reg}(k_i, t_{k_i}) = \frac{n_l}{n} MSE_l + \frac{n_r}{n} MSE_r \tag{2.10}$$

Note also that, due to its greedy nature, the CART algorithm will find a reasonably good solution, but it is not guaranteed that it will find the optimal one. This is a consequence of the fact that finding the optimal tree is known to be an NP-Complete problem, so it would be infeasible even for small datasets.
Once the tree is trained, it can make predictions in a very simple way. Given a new instance, the tree uses the thresholds found during training to assign the instance to one of the leaf nodes, then, the predicted class will be the one associated with that node. In the same way, it is possible to determine the class probabilities. For example, referring to figure (2.10), the class probabilities associated with the lower right node are: 0 for *setosa* (0/46), 0.02 for *versicolor* (1/46) and 0.98 for *virginica* (45/46).

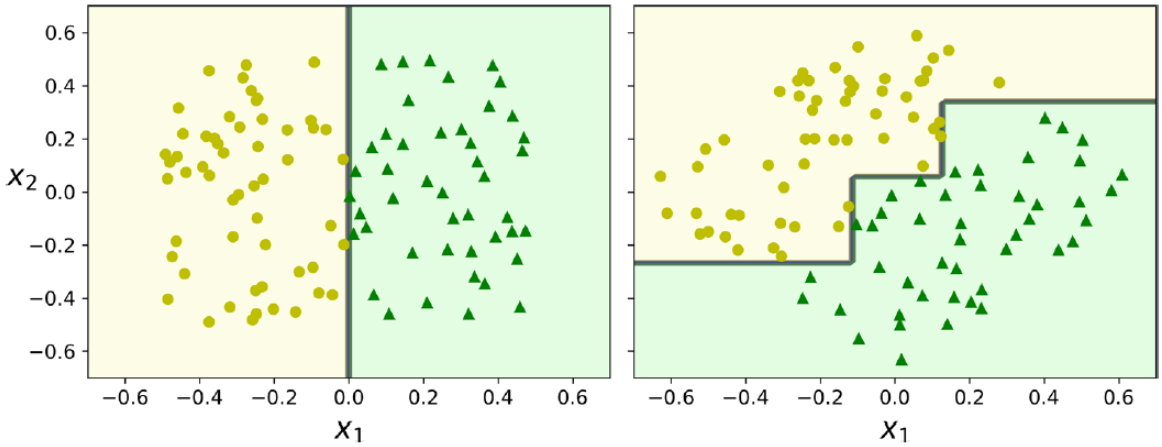



Figure 2.12: Rotating the training set by 45° causes an evident modification of the decision boundaries of the tree (Géron, 2019).

Decision Trees have many remarkable properties: they are simple, powerful, and versatile; however, they also have some important drawbacks that must be considered.
First of all, if the tree has no constraints, the CART algorithm will continue the splitting until it cannot find a split that further reduces the impurity. As a consequence, a tree that can grow without restrictions will almost certainly overfit the data. We can clearly see this behavior in Figure (2.11), where two Decision Trees are used to fit a sinusoidal wave. Here the hyperparameter *max_depth* is used to stop the trees from growing when a certain depth is reached. The tree with *max_depth* equal to five is obviously overfitting the data, while the tree with *max_depth* equal to two is not.
In general, Decision Trees are naturally prone to overfitting, so it is very important to carefully choose the regularization hyperparameters to restrict their freedom during training. The *max_depth* is one of the most used, but there are also some alternatives, like *min_samples_split*, which identifies the minimum number of samples required to split

a node, and *max_leaf_nodes*, that restricts the maximum number of leaf nodes the tree can have.
Another major issue of Decision Trees is their sensitivity to small variations in the training data; for example, removing a single instance may result in the generation of a completely different tree. On the left side of Figure (2.12) we can see a linearly separable dataset. The right side of the figure shows the same dataset, but rotated by 45°. In both cases, the Decision Tree can easily separate the two classes, however, the decision boundaries for the rotated set are clearly more complicated, and the correspondent tree will not generalize as well as the one trained on the original set.
The instability of Decision Trees can be reduced by using them in an ensemble.

### 2.2.2 Random Forests

Random Forests (Breiman, 2001) are among the most powerful Machine Learning algorithms currently available; they are a so-called **ensemble method** because they aggregate the predictions of a group of predictors. Specifically, a Random Forest is an ensemble of Decision Trees: each tree is trained on a bootstrap sample of the original training set, then the predictions of the trees are combined to give the final prediction.
Bootstrapping (Horowitz, 2001) is a sampling technique in which new data sets are built by randomly drawing samples from the original set of data, allowing the same instance to be sampled more than once; in this way, each tree in the forest is built from a different data set. Typically, the size of the bootstrap sample is the same as the original dataset.

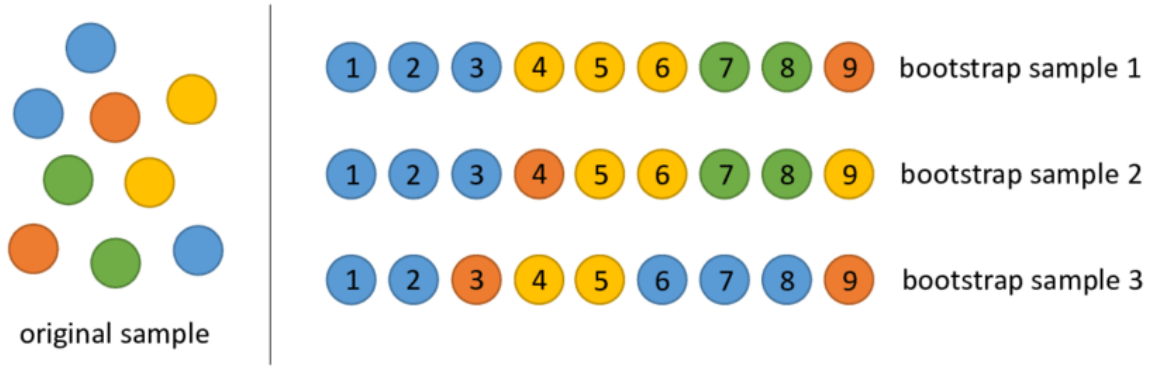



Figure 2.13: Example of Bootstrap sampling (Galdi and Tagliaferri, 2018).

Moreover, when splitting the nodes during the construction of the trees, the best feature for the division is found among a random subset of the original features.
Random Forests usually perform better than a single Decision Tree, being less prone to overfit, less sensitive to data variation and having an overall smaller variance. On the other hand, a Random Forest is not a *white box model* anymore: explaining why it gives a certain prediction is not straightforward. As it was for Decision Trees, Random Forests can be used both for classification and regression, and they also have some other notable properties.

First of all, Random Forests have the intrinsic ability to perform feature selection. In fact, by looking at the features that, on average, over the entire forest, mostly reduce the impurity of the dataset, we can obtain the relative importance of the features.
Although this can be very useful, it has to be used carefully, in fact, the feature impor-

tance is calculated using the training instances, so it does not necessarily apply to the test data too. Furthermore, the process favors the features with many unique values.
Another interesting advantage of Random Forests is the possibility of evaluating the model using the **Out-of-Bag** evaluation. A consequence of bootstrap sampling, in fact, is that, on average, only about 63% of the training instances are sampled for each predictor. The remaining portion of the data, called *out of bag samples*, can thus be used as a test set for that predictor. It follows that the ensemble itself can be evaluated by averaging the out-of-bag scores of each predictor in the ensemble.

#### Random Survival Forests

Random Survival Forests (Ishwaran et al., 2008) are an extension of regular Random Forests capable of handling right-censored data and performing survival analysis. They are a non-parametric alternative to other popular parametric and semi-parametric survival models, such as the famous **Cox proportional hazards model**.
Random Survival Forests share many of their properties with standard Random Forests; one of the main differences between them is in the splitting criteria used to grow the trees. Several splitting rules have been proposed over the years (Bou-Hamad et al., 2011); a common implementation uses the **log-rank** statistic to compare the two groups formed by the child nodes (LeBlanc and Crowley, 1993). The retained split is the one with the largest significant test statistic value. The log-rank test indicates whether survival between the two groups is significantly different. The null hypothesis tested by the log-rank is that the two groups have identical hazard functions, i.e., equal event-time distribution. The rejection of this hypothesis thus means that the event rates differ between the two groups. The use of the log-rank test yields a split that assures the best separation of the median survival times in the two children nodes.

### 2.2.3 Cox Proportional Hazards Model

The Cox proportional hazards model (Cox, 1972) is by far one of the most used models in survival analysis. The Cox model gives an expression for the hazard at time $t$ for an individual with a given set of explanatory variables $\vec{x} = \{x_1, \ldots, x_n\}$:

$$h(t, \vec{x}) = h_0(t) \exp\left[\sum_{i=1}^{n} \beta_i x_i\right] \tag{2.11}$$

where $h_0$ is the **baseline hazard** and $\vec{\beta} = \{\beta_1, \ldots, \beta_n\}$ are the parameters of the model. We can also rewrite the (2.11) in vector form:

$$h(t, \vec{x}) = h_0(t) e^{\vec{\beta}\vec{x}} \tag{2.12}$$

The first noticeable feature of the (2.11) is the fact that the baseline hazard depends only on $t$, while the expression inside the exponential is time independent: this is the **proportional hazards assumption**. This assumption essentially states that, while hazards can vary over time, the *ratio* of the hazards for any two individuals remains constant. Another important property of the Cox model is that the baseline hazard function $h_0(t)$ is not specified. This makes the Cox model a **semi-parametric** model, in fact, in a parametric model the functional form is completely specified.
Its semi-parametric nature is one of the reasons why it is so popular. Even if the baseline

hazard is not specified, the results obtained using the Cox model will closely approximate the ones obtained by the correct parametric model (Kleinbaum and Klein, 2010); for this reason, the Cox model is said to be a *robust* model.
A further implication of equation (2.11) is that the hazards given by the model will always be non-negative, as they should be.
Finally, another remarkable property of the Cox model is the possibility to measure the effect of the explanatory variables without actually estimating the baseline hazard function; all we need are the estimated of the $\beta_i$ parameters. The measured effect of an explanatory variable is known as the **Hazard Ratio** (HR).

### Estimating the parameters: the Partial Likelihood

The Cox model parameters are derived by maximizing a likelihood function, as it happens for other models, like logistic regression. The likelihood function gives us a measure of the goodness of the fit of a statistical model to a sample of data for given values of the unknown parameters. In general, the formulation of a likelihood function is built upon the distribution of the outcome, e.g., if we toss a coin, we expect the outcomes to be Bernoulli distributed.
For the Cox model, however, this is not possible, because there is no assumed distribution for the outcome. The approach used to define the Cox likelihood is thus based on the observed order of the events. Since it only considers the probability of the subjects who experience a failure, the Cox likelihood is usually called **partial likelihood**, and we will now derive its mathematical formulation.

Suppose we observe $(t_i, \delta_i, \vec{x}_i)$ for the i-th patient, where $t_i$ is a failure or censoring time, $\delta_i$ is the failure indicator (1 if failure, 0 if censored) and $\vec{x}_i$ is the set of features. We then define the *risk set* as the set of individuals who are at risk at time $t$: $\mathcal{R}(t) = (i \,|\, t_i > t)$. Suppose now that the patient $j$ experiences a failure at time $t_j$. The probability for this to happen is equal to the conditional probability for an individual $j$ of being chosen from the risk set to fail:

$$\begin{aligned} L_j(\vec{\beta}) &= P(\text{individual j fails} \,|\, \text{one failure from} \mathcal{R}(t_j)) \\ &= \frac{P(\text{individual } j \text{ fails} \,|\, \text{at risk at} \, t_j)}{\sum_{l \in \mathcal{R}(t_j)} P(\text{individual } l \text{ fails} \,|\, \text{at risk at} \, t_j)} \end{aligned} \tag{2.13}$$

If we then substitute the probabilities in (2.13) with the hazard identified in the (2.12), we get:

$$L_j(\vec{\beta}) = \frac{h(t_j, \vec{x}_j)}{\sum_{l \in \mathcal{R}(t_j)} h(t_j, \vec{x}_l)} = \frac{h_0(t_j) e^{\vec{\beta}\vec{x}_j}}{\sum_{l \in \mathcal{R}(t_j)} h_0(t_j) e^{\vec{\beta}\vec{x}_l}} \tag{2.14}$$

A noticeable thing of the (2.14) is that the baseline hazard is not needed to calculate the likelihood, if fact, by factoring $h_0(t_j)$ in the denominator, we can cancel it out from the equation. This explains why we do not need the baseline hazard to estimate the parameters.
Finally, suppose we have $N$ events at times $t_1 < t_2 < \ldots < t_N$, where an event can be either a failure or a censoring, and let $L_i$ be the contribution to the likelihood corresponding to the i-th event time. Recall that $\delta_i$ is equal to 1 if the event is a failure and to 0 if the event is a censoring. Then, the Cox likelihood will be the product of all the

$L_i$ corresponding to a failure:

$$L_{cox} = (L_1)^{\delta_1} \times (L_2)^{\delta_2} \times \ldots \times (L_N)^{\delta_N} = \prod_{i=1}^{N} (L_i)^{\delta_i} \tag{2.15}$$

which, using the (2.14), becomes:

$$L_{cox} = \prod_{i=1}^{N} \left( \frac{e^{\vec{\beta}\vec{x}_i}}{\sum_{l \in \mathcal{R}(t_i)} e^{\vec{\beta}\vec{x}_l}} \right)^{\delta_i} \tag{2.16}$$

Once we have obtained the formula for the Cox likelihood, we can derive the regression parameters by setting the partial derivatives of the natural logarithm of the likelihood to zero and solving the resulting system of equations:

$$\frac{\partial \ln L_{cox}}{\partial \beta_i} = 0 \quad , \quad i = 1, 2, \ldots, p \tag{2.17}$$

with $p$ being the number of parameters of the model.

**Hazard Ratios and Survival curves**

When doing survival analysis, two quantities of particular interest are the estimated Hazard Ratios (HR) and the estimated survival curves.
The hazard ratio is defined as the hazard for one individual divided by the hazard for a different individual. Mathematically, the hazard ratio for two individuals $i$ and $j$ will be:

$$HR_{i,j} = \frac{h(t, \vec{x}_i)}{h(t, \vec{x}_j)} = \frac{h_0(t) e^{\vec{\beta}\vec{x}_i}}{h_0(t) e^{\vec{\beta}\vec{x}_j}} \tag{2.18}$$

so, finally

$$HR_{i,j} = \exp\left[\vec{\beta}(\vec{x}_i - \vec{x}_j)\right] \tag{2.19}$$

As we can see, according to what we said earlier, the hazard ratio in (2.19) is constant over time.

A survival curve, instead, defines the probability of surviving after time $t$. One interesting result of using a Cox model to fit the data is that we can obtain survival curves adjusted for the explanatory variables, called **adjusted survival curves**. We can use the hazard function of the Cox model to obtain the corresponding survival curve (Kleinbaum and Klein, 2010):

$$S(t, \vec{x}) = \left[S_0(t)\right]^{e^{\vec{\beta}\vec{x}}} \tag{2.20}$$

where $S_0(t)$ is the baseline survival function. Baseline hazard and survival functions are usually estimated using the **Breslow's method** (Lin, 2008).

## 2.3 Model Evaluation

When building and training a model, it is extremely important to choose the right metric to evaluate its performance. To have a better idea about the importance of the evaluation metric, we will first consider one of the most used to evaluate classification models: the

**accuracy**. Accuracy is very simple to define, it is just the fraction of predictions that our model got right, or, more formally:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{2.21}$$

Now, imagine we have a dataset made of one thousand samples belonging to two different classes, denoted with 0 and 1. Imagine then in the aforementioned dataset 900 instances belong to class 0, and the other 100 to class 1. In a similar situation, we could simply build a model that always predicts 0 to achieve a 90% accuracy.

### 2.3.1 Precision and Recall

An interesting metric to look at is called **Precision**, it is the accuracy of the positive predictions and it is defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.22}$$

where TP are the *True Positives* and FP are the *False Positives*. An easy way to get a perfect precision is to make one single positive prediction and ensure it is correct; that's why precision is typically used along with another metric, known as **Recall**. The Recall is defined as the ratio of positive instances that are correctly detected by the classifier:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.23}$$

These two metrics are often combined to make another metric, called $\mathbf{F_1}$ **Score**, which is the harmonic mean of Precision and Recall:

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} = 2\frac{P \cdot R}{P + R} = \frac{TP}{TP + \frac{FN+FP}{2}} \tag{2.24}$$

The $F_1$ Score can be a really solid metric since it will have a high value only if both Precision and Recall values are high. On the other hand, however, the major criticism of the $F_1$ Score is that it gives the same importance to Precision and Recall, favoring classifiers that achieve similar values of those two metrics. In many real-life scenarios, however, one of the two can be much more important than the other. For example, if you have trained a classifier that detects if a certain component of an airplane is defective, it would probably be preferable to have a very high Recall, making sure that all the defective component are detected, even if the precision is low, i.e. some of the good components are labeled as defective. Unfortunately, increasing Precision reduces Recall, and vice versa; this is known as the *Precision/Recall trade-off*.

### 2.3.2 The ROC Curve

Another famous instrument to evaluate the performance of a binary classifier is the **Receiver Operating Characteristic** (ROC) curve. It displays the *true positive rate*, i.e., the Recall, against the *false positive rate* (FPR) for various threshold values. The FPR is the ratio of negative instances that are incorrectly classified as positive, namely:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{2.25}$$

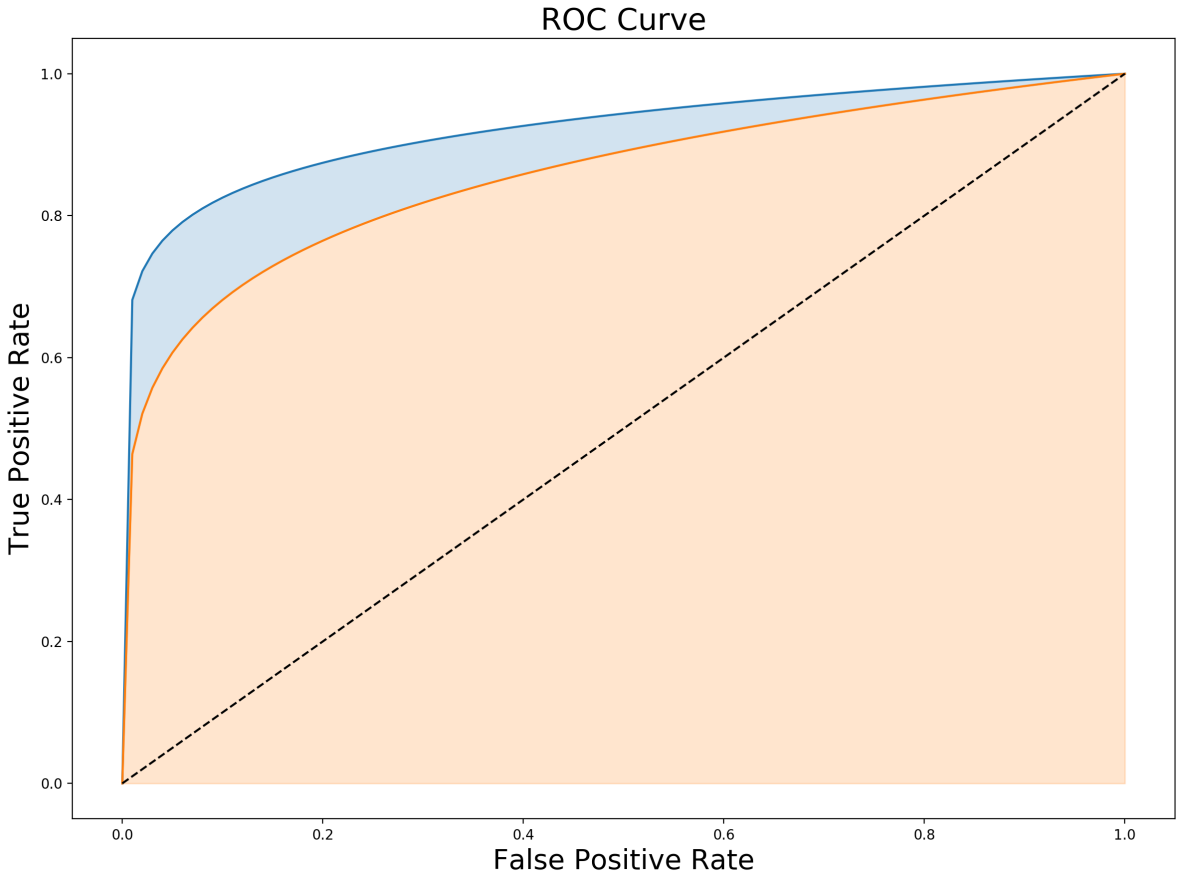



Figure 2.14: Plot of two dummy ROC curves with the associated area under the curve (AUC). The dotted line represents the ROC curve of a random classifier.

To have a better idea about the general shape of a ROC curve we can refer to Figure (2.14) in which two ROC curves are depicted; note that they are not derived from real data and serve us only as an example. In the figure, the dotted line represents the ROC curve of a random classifier; a good classifier stays as far away as possible from that line, toward the top left corner. For example, looking again to figure (2.14), a classifier producing the blue curve would be better than a classifier producing the orange curve, which, in turn, would of course be better than a random classifier.
The Area Under the Curve (AUC) of the ROC curve is a commonly used metric to compare two classifiers; a perfect classifier would have a ROC AUC equal to 1, while a random classifier would have a ROC AUC equal to 0.5.

### 2.3.3 Concordance Index

The metrics we have seen so far are used with classifiers. For survival analysis, we use regression models, thus, we have to rely on other kinds of metrics.
One of the most used performance measures for survival models is the **Concordance Index** (CI). It is a generalization of the ROC AUC we just introduced (Uno et al., 2011; Steck et al., 2008). The concordance index corresponds to the fraction of all pairs of subjects whose survival times are correctly ordered among all the subjects that can be compared. A pair of subjects is comparable if we can determine which of them was the first to experience an event. For example, consider three subjects $i$, $j$ and $k$ with $i$

experiencing a failure at time $t_i$ and $j$ and $k$ being censored at times $t_j > t_i$ and $t_k < t_i$. In this case, we can compare $i$ and $j$ because we are sure that $i$ experienced the event *before* $j$, but we cannot compare $i$ and $k$, because $k$ was censored before $t_i$.
Mathematically, the concordance index can be defined as follows (Longato et al., 2020):

$$\text{CI} = \frac{\sum_{i=1}^{N} \delta_i \sum_{j=i+1}^{N} I(t_i^{obs} < t_j^{obs}) \cdot I(h_i > h_j)}{\sum_{i=1}^{N} \delta_i \sum_{j=i+1}^{N} I(t_i^{obs} < t_j^{obs})} \tag{2.26}$$

where:

- I(x) = 1 if x is true, 0 otherwise
- $\delta_i = 1$ if subject $i$ experienced a failure, 0 if censored
- $t_i^{obs}$ and $t_j^{obs}$ are the observed survival times for subjects $i$ and $j$ respectively
- $h_i$ and $h_j$ are the predicted hazards for subjects $i$ and $j$ respectively (note that an higher predicted hazard corresponds to a lower predicted survival time)

Similarly to the ROC AUC, the value of the concordance index is between 0 and 1; the higher the value, the better the model. Again, a concordance index of 1 indicates a perfect predictor, while a concordance index of 0.5 means that the model is as good as a random predictor.

### 2.3.4 Akaike Information Criterion (AIC)

The Akaike Information Criterion (AIC) (Akaike, 1998) was first introduced by Hirotogu Akaike as an extension to the method of maximum likelihood estimation. The likelihood function, in fact, gives us a measure of the goodness of the fit, but it does not take into account the number of parameters of the model. The problem is that, in general, the higher the number of parameters, the higher is the risk for the model to overfit. AIC, instead, considers both the goodness of the fit and the simplicity of the model by introducing a penalty that is directly proportional to the number of parameters. It is formulated as follows:

$$\text{AIC} = 2k - 2\ln(\hat{L}) \tag{2.27}$$

where $k$ is the number of parameters of the model and $\hat{L}$ is the maximum value of the likelihood function for the model.
It follows that AIC can be used for model selection: given a set of candidate models we prefer the one with the lowest AIC value. Note that AIC can be used to compare different models, but gives no information about the absolute quality of a model.

### 2.3.5 Cross-Validation

To evaluate a model, usually, the entire dataset is divided into two mutually exclusive sub-sets: the **train** set and the **test** set, with the latter being generally smaller, typically 20-30 percent of the whole data. This procedure is sometimes referred to as **holdout**. As their names suggest, the train dataset is used to fit the model and the test dataset is used to evaluate it. However, the *train-test split* approach alone may not be sufficient, and in some situations it can also lead to erroneous conclusions, especially when used incorrectly. In particular, the problem arises when we want to fine-tune the hyperparameters of our

model. In fact, if we tweak the hyperparameters of the model to achieve the best possible performance on a given test set, we are implicitly giving to it information about the actual test data. This will probably produce a model that does not generalize well to new, unseen data.
To partially solve this problem, another part of the original data can be held out, ending up with three datasets, the usual train and test ones and the new one, called **validation set**. The issue, however, is that in this way we are drastically reducing the number of available samples that we can use to train the model. Assuming that the model performance increases as more instances are seen, it is clear that the holdout method is a pessimistic estimator because only a portion of the data is given to the classifier for training.

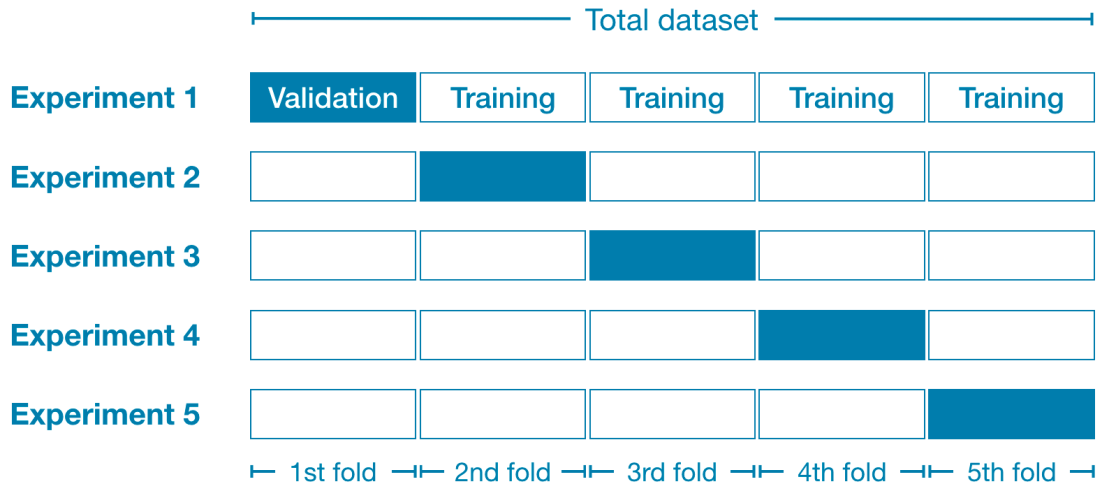



Figure 2.15: *k -fold* Cross-Validation. The dataset is divided into $k$ folds, then $k-1$ folds are used for training and the one left is used for testing. The process of training and testing is repeated k times.

An alternative, famous way to evaluate a model: is the **Cross-Validation**. The basic approach, named *k-fold Cross-Validation*, consists in splitting the original data into $k$ subsets of equal dimension (if possible), then $k-1$ folds are used to train the model, and the remaining one is used to test it. The process is repeated $k$ times, each time using a different fold as the test set, and the remaining data as the training set, in this way, all the folds are used exactly once as test data. The entire operation can be visualized in Figure (2.15). The final performance score of the cross-validation procedure is the average of the values obtained at each iteration. We can thus write the cross-validation estimate of the prediction error as follows:

$$CV(f) = \frac{1}{K}\sum_{i=1}^{K} L(y_i, f_i(x_i)) \tag{2.28}$$

where $K$ is the number of folds, $f_i(x_i)$ are the predicted values for the samples in the *i-th* fold, $y_i$ are the true values associated to those samples, and L is the metric.
The greatest advantage of cross-validation is that it allows to effectively test the model without wasting data, and this can be crucial in situations in which very few training instances are available, as it was in this work. However, even if we use cross-validation to tune and test the model, it is a good practice to hold out a certain amount of data to use as a final estimate of the model performance.

Recommended choices of $k$ are usually 5 or 10 (Kohavi et al., 1995; Kim, 2009); the case in which the number of folds is equal to the number of samples in the dataset is known as **leave-one-out** cross-validation. In this scenario, the cross-validation estimator is approximately unbiased for the true, expected, prediction error, but the variance is likely to be very high because the training sets are similar to one another; also, the computational burden is not negligible.
Both 10-fold and 5-fold cross-validation have a lower variance than the leave-one-out, but bias can be a problem, depending on how the performance of the learning method varies with the size of the training set.

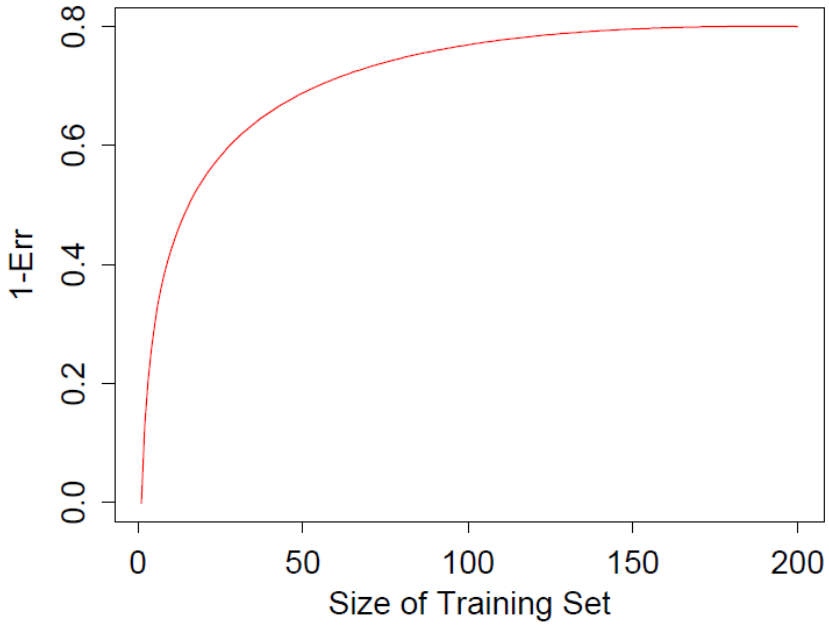



Figure 2.16: Hypothetical learning curve for a model on a given training set (Hastie et al., 2009)

Referring to Hastie et al. (2009) we consider Figure (2.16), showing a hypothetical learning curve for a model trained on a given dataset. The learning curve rises quickly until a certain number of training samples is reached, around 150, then increasing the number of training data points only produces a slight improvement of the performance. So, if we had a dataset with 200 observations, with a 5-fold cross-validation, the model would be trained on 160 samples; its score would be nearly identical to the one obtained by the same model trained on the whole dataset, yielding an almost unbiased cross-validation result. On the other hand, if the dataset had just 50 samples, a 5-fold cross-validation would train the model on 40 samples, producing an underestimate of $1 - Err$, i.e., the cross-validation estimate of the error would be biased upward.

That said, in some cases, as shown by Rao et al. (2008), the performance estimated by cross-validation is no longer an effective estimate of generalization. For example, consider a scenario in which we have a dataset with a small number of instances and a great number of features, and we want to perform feature selection using cross-validation. Under these conditions, probably, the process of feature selection will eventually find a lucky subset of features yielding a good cross-validation result, but the selected features could have no predicting power on the test set. The same can happen when we use cross-validation to perform model selection over a large number of models.

## 2.4 Methods

### 2.4.1 Data Acquisition and Preliminary Analysis

#### Data Acquisition

The data acquisition, which was not performed by me, was organized in the following way. First, all the patients underwent a PET/CT scan. The acquired images were then segmented in a semi-automatic procedure using a threshold algorithm and the PET Edge$^{TM}$ algorithm. To extract the radiomic features, the Pyradiomics python library (Van Griethuysen et al., 2017), based on *3D Slicer*, was used. In this way, *shape-based* features, *first-order* features, and *texture-based* features were extracted.

#### Data Preparation

The very first step in data analysis is of course the data preparation. The dataset used in this work contained radiomic features extracted from CT and PET images of 99 patients. Five of them had a number of features that was too low to include them in the study, so they were dropped. Each remaining patient had a set of 114 CT features and a set of 114 PET features. Also, each patient had a set of 4 outcomes: *recurrence*, *time to recurrence*, *survival* and *survival time*.
First of all, the *info* features have been removed since they could not be used in the machine learning model, resulting in two sets of 105 features each. All the null values of the features were filled with 0. Nine patients had feature values about ten orders of magnitude higher than the others, so they were excluded as outliers, leading to a total of 85 remaining patients.
Successively, all the features were standardized by removing the mean $\mu$ and dividing by the standard deviation $\sigma$:

$$z_i^n = \frac{x_i^n - \mu}{\sigma} \tag{2.29}$$

where $x_i^n$ is the value of the feature $n$ for sample $i$, and $z_i^n$ is the correspondent standardized value.
Survival and the associated survival times were used as labels. The former was a binary variable with value 1 for the event *death* and value 0 for *censored* event. Among the 85 subjects, 24 experienced a failure, the others were censored. Survival time was of course the time until death or censoring, it was measured in months.
Finally, the dataset was divided into a *training* and a *test* set; 80% of data (68 samples with 19 deaths) was used for training and the remaining 20% (17 samples with 5 deaths) for testing.
We wanted to predict the hazards of the patients. Recall that a higher predicted hazard corresponds to a lower predicted survival time.

### 2.4.2 Feature Selection

The core part of the work was the feature selection process. Correlation, LASSO regression, and Genetic Algorithms were used and compared to find the best feature subsets; also, t-SNE was used for visualization purposes. Note that all the following processes were applied to CT and PET features separately. Note also that all the operations of feature selection were performed *using the train data only*.

### Correlation

To remove the correlated features, first of all, the correlation matrix was built using the *Pandas* library (Team, 2020). Successively, for each feature, the correlation coefficients with respect to the other features were evaluated, if one of those coefficients was higher than a certain threshold, the feature was dropped.
The threshold was set to 0.9 so that only the highly correlated features were dropped.
It gives in output the dataframe of the uncorrelated features and the list of the correlated ones, i.e, the features that were dropped. The central part of the algorithm is shown in the following code snippet:

```
correlation_matrix = features.corr() # calculate the correlation matrix

correlated_features = set()

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)

correlated_features = list(correlated_features)

features = features.drop(correlated_features, axis = 1)
```

This technique was the easiest to implement and the less computationally expensive. It is also quite versatile since it allows to directly act on the number of selected features by controlling the threshold value. The main weakness is that between the two features being compared, we cannot know which of the two is more informative.

### LASSO Regression

To compute the correlation matrix we only need the feature values, to perform feature selection using LASSO, instead, we actually need to fit a model to the data. The model used was a Cox model with LASSO penalty, also known as $\boldsymbol{l_1}$ **penalty**. We introduced the LASSO technique for the linear regression case to make it clearer; the only difference here is that, instead of using the Mean Squared Error as objective function, we use the log-likelihood defined for the Cox model:

$$\hat{\beta} = \min_{\vec{\beta}} \left[ -\ln(L_{cox}(\vec{\beta})) + \lambda \|\vec{\beta}\| \right] \tag{2.30}$$

Note that the minus sign before the log-likelihood comes from the fact that usually the log-likelihood is maximized to find the best parameters, but the problem of maximizing it is the same as minimizing the negative log-likelihood.
The number of parameters $\beta$ of the model that will be shrunk to 0 depends on the penalization parameter $\lambda$: higher values of $\lambda$ will result in fewer features being selected.

Finding the optimal value for the penalization parameter is not straightforward. The process adopted in this work was the following: 100 Cox models, each with a different penalization value, have been evaluated using a *5-fold cross-validation repeated 5 times*; the parameter producing the best performance was selected as the optimal one. In a repeated cross-validation, at each repetition, before performing cross-validation, the data are randomly shuffled, and the final score is the average of the scores obtained for each

cross-validation. This should give a more solid estimate of the model performance and should also slightly mitigate the risk of overfitting.
Regarding the values of the penalization parameters, they were 100 evenly spaced real numbers ranging from $10^{-4}$ to 1.
Once the best penalization parameter was identified, a Cox model was built using that parameter, then the model was fitted to the training data to get the $\beta$ values. The features selected were the ones with an associated nonzero value of *beta*.
The Cox model was implemented using the *scikit-survival* library (Pölsterl, 2020), while the repeated cross-validation was built upon *scikit-learn* (Pedregosa et al., 2011).
This algorithm is way more computationally intensive than the correlation one; here, in fact, each of the 100 models must be trained and tested 25 times. The code used to find the best penalization parameter is shown below:

```
# generate the penalization values
alphas = 10 ** np.linspace(-4, 0, 100)

# define the Cox model with LASSO penalization
coxLasso = CoxnetSurvivalAnalysis(l1_ratio=1)

scores = [] # store the scores

# define the repeated Cross-Validation
cv = RepeatedKFold(n_splits=5, n_repeats=5, random_state=0)

for alpha in alphas:
    # set the penalization parameter
    coxLasso.set_params(alphas=[alpha])

    # calculate and append the scores
    score = np.mean(cross_val_score(coxLasso, X_scaled, Y, cv=cv))
    scores.append(score)

# select the best penalization parameter
best_alpha = alphas[np.argmax(scores)]
```

### Genetic Algorithm

The Genetic Algorithm was the most sophisticated technique implemented in the work. It was based on the DEAP library (Fortin et al., 2012).
To perform feature selection, binary chromosomes were used. A chromosome had a number of genes equal to the number of features in the dataset. The features associated with genes with value 0 were dropped, whereas the features associated with genes with value 1 were kept. In this way, each chromosome identified a subset of the original set of features. Successively, for each chromosome, a Cox model was built using the subset of features selected by that chromosome. The resulting models were then evaluated using different metrics; the scores of the models were used as fitness values so that the best chromosomes were the ones producing the best models, i.e., the ones selecting the best features.

The prior step of feature selection was made using the **partial AIC** as the metric to evaluate the models (Note that "partial" comes from the fact that the Cox model uses a *partial likelihood*).

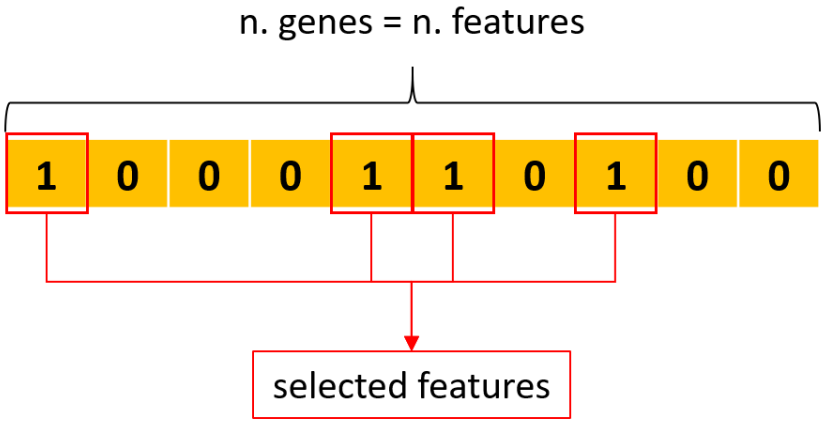



Figure 2.17: Example of a chromosome used for feature selection. The number of genes is equal to the number of features. The features corresponding to a gene 1 are selected.

As we saw, the AIC penalizes models with a higher number of parameters, so it is a natural candidate to use for feature selection. The Genetic Algorithm architecture was the following: *binary tournament* was used as selection method, *two-point crossover* as crossover method and *flip-bit mutation* as mutation method. The probabilities of crossover and mutation were 0.9 and 0.2 respectively, with the probability of the single gene being flipped equal to 1/(n. genes). An elitism strategy was used to always keep in the population the 5 best individuals. The population was made by 50 individuals and the evolution process lasted for 200 generations.

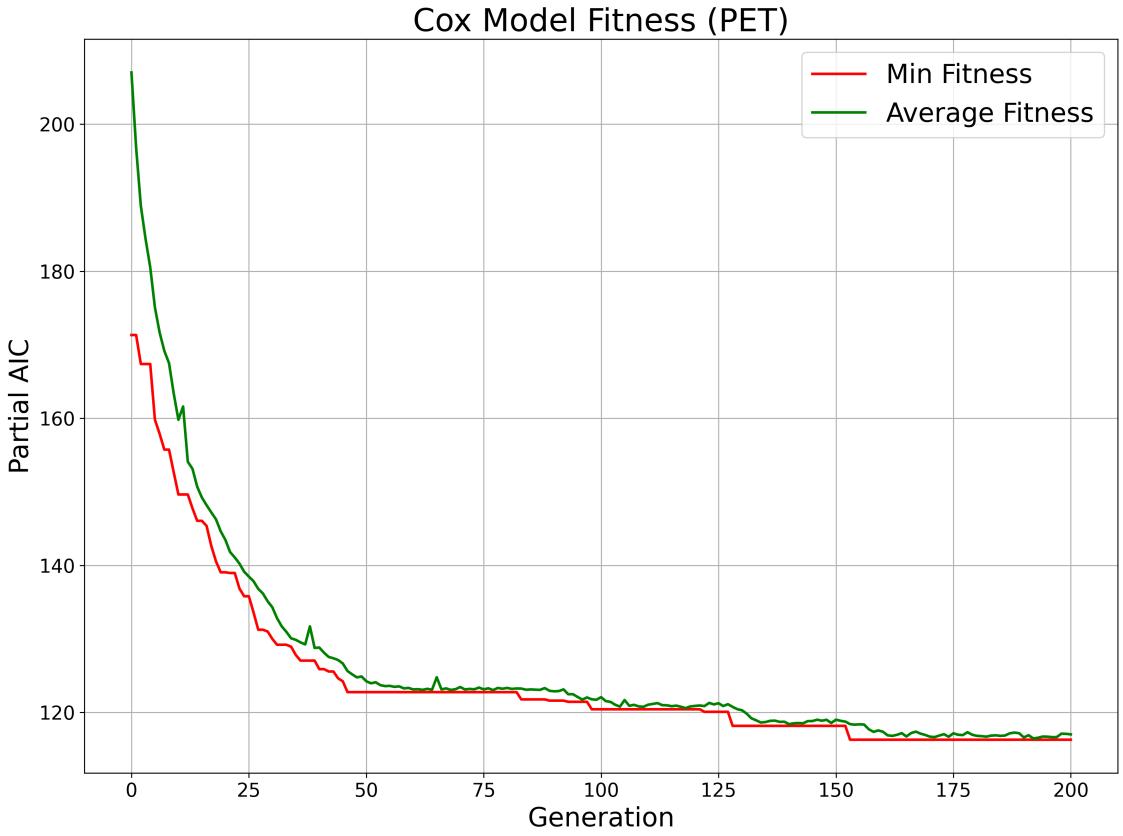



Figure 2.18: Partial AIC minimization using PET features. The minimum fitness and the average fitness are showed in red and green respectively.

The library used to implement the Cox model, in this case, was *lifelines* (Davidson-Pilon et al., 2020), because it provided direct access to the partial AIC associated with the model, so it was easier to use in the Genetic Algorithm. The process of minimization of the partial AIC is shown in Figure (2.18). In particular, by looking at the average fitness, we can see that the partial AIC of all the models was substantially reduced.

The second and last step of feature selection was performed on the features selected in the previous step, in order to get an even smaller subset of informative features. The main difference with the previous process was in the fitness function; this time, we wanted to maximize the following:

$$\text{Fitness} = \mu(CV_5) - \sigma(CV_5) \tag{2.31}$$

with $\mu(CV_5)$ being the average score obtained in a *5-fold cross-validation* repeated 5 times and $\sigma(CV_5)$ being its standard deviation. As it was in the first part, for each individual, a Cox model was built using the features selected by that individual. For each of those models, a *5-fold cross-validation* repeated 5 times was performed, using the concordance index as metric, and the average score of the repeated cross-validation and its standard deviation were computed. The best individuals were the ones who maximized equation (2.31), i.e., the difference between the average score and the standard deviation. The purpose of that fitness function was to maximize the score obtained in the overall repeated cross-validation, and, at the same time, to achieve a similar score for all the folds, by minimizing the standard deviation.

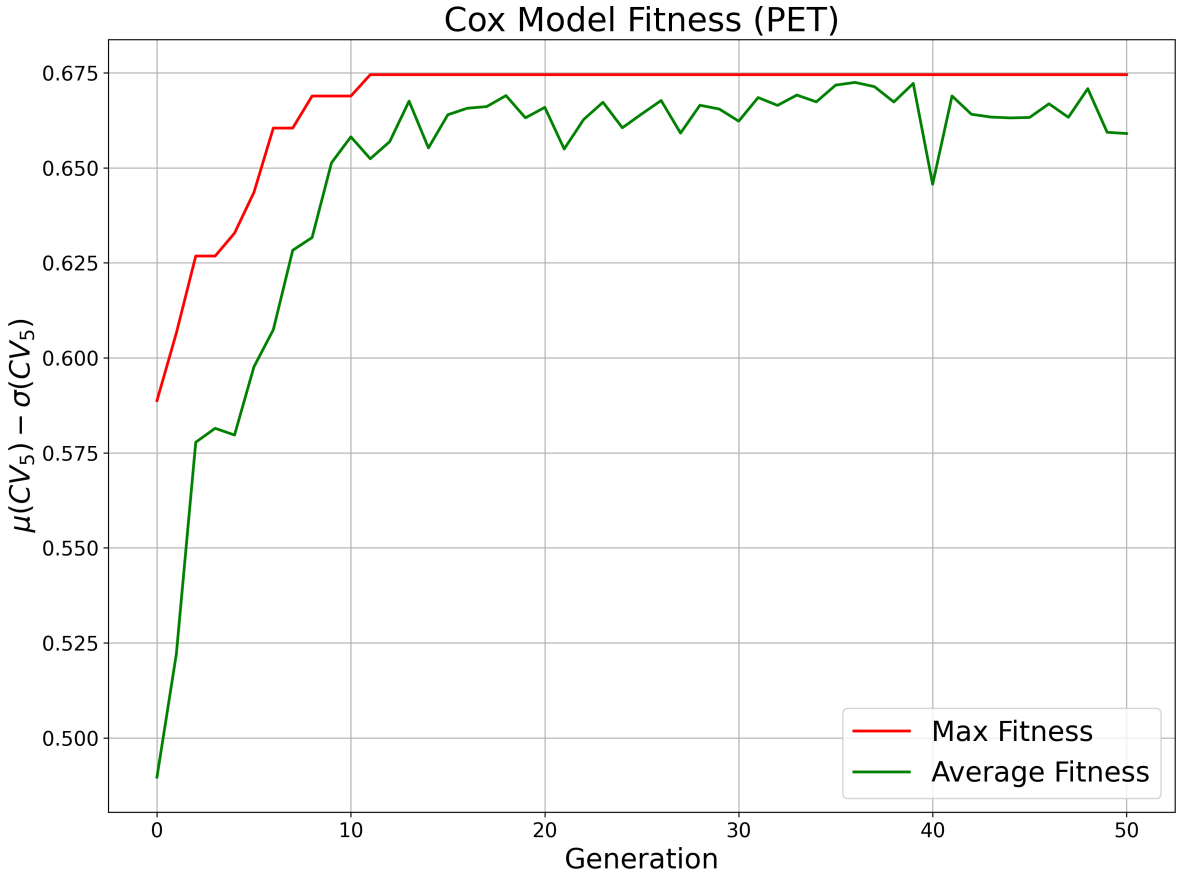



Figure 2.19: Maximization of the fitness function (2.31) for PET features. The maximum fitness and the average fitness are showed in red and green respectively.

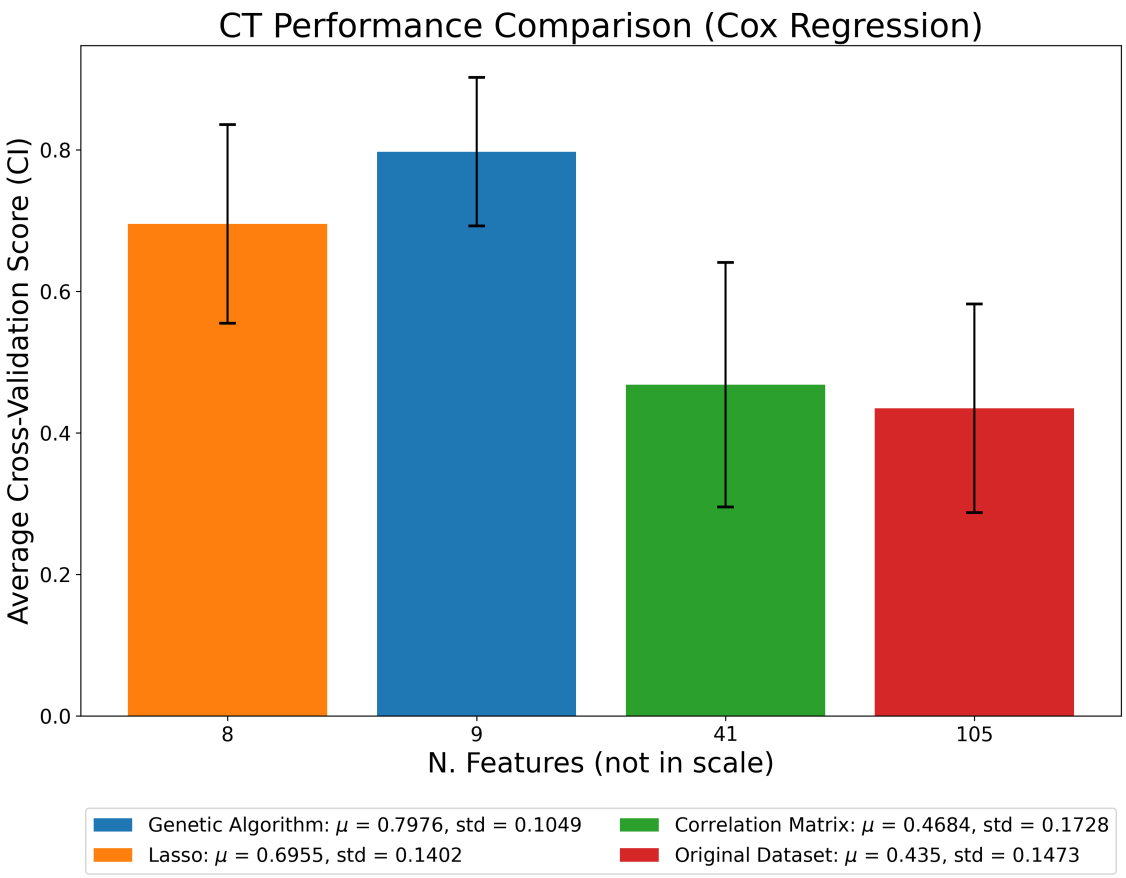
The general architecture of the Genetic Algorithm was almost the same, the only difference was in the number of generations. This time, the maximum number of generations was reduced to 50, for two main reasons. The first is that the computational burden of the process is not negligible, in fact, in a single generation, each of the 50 Cox models is trained and tested 25 times. The second is that, this time, the features were selected among a smaller number of initial features, so it was easier for the algorithm to find a good solution. The process of maximization of the (2.31) for PET features is shown in Figure (2.19); as we can see, the best solution was found in just 11 generations.
The functions written to calculate the fitness (2.31) in the genetic algorithm are the following:

```
# repeated cross-validation calculation
def Repeated_CrossValidation(X, Y, estimator, ZeroOneList, folds=5,
                             rand_state=1, std=False):

    # if no feature is selected, return 0.5
    if sum(ZeroOneList) == 0:
        return 0.5
    else:
        # drop features corresponding to a gene 0
        zeroIndices = [i for i, n in enumerate(ZeroOneList) if n == 0]
        X_deleted = np.delete(X, zeroIndices, 1)

        cv = RepeatedKFold(n_splits=5, n_repeats=5,
                           random_state=0)
        score = cross_val_score(estimator, X_deleted, Y, cv=cv)

        if std:
            return score.mean(), score.std()
        else:
            return score.mean()

# actual fitness of an individual
def CrossValidation(individual):

    avg, std = Repeated_CrossValidation(X_genetic_scaled, Y, skCox,
                                        individual, folds=5, std=True)
    return (avg - std),
```

### Best Features Subset

To decide which of the feature selection techniques used was the best, each of the subsets of features found with the three methods was used to build a Cox model, then those models were evaluated using again a *5-fold cross-validation* repeated five times with concordance index as the metric. The models that achieved the best results were considered the candidate ones, and, consequently, the features used in those models were considered the candidate predictors.
The results are shown in Figure (2.20), with the average cross-validation score in the y-axis and the number of selected features in the x-axis. As we can see, the features selected by the Genetic Algorithm produced the best results, followed by the ones selected by the LASSO regression. Features selected with the correlation method, instead, gave very poor results; in particular, for PET features, the fit did not converge.

(a)




(b)

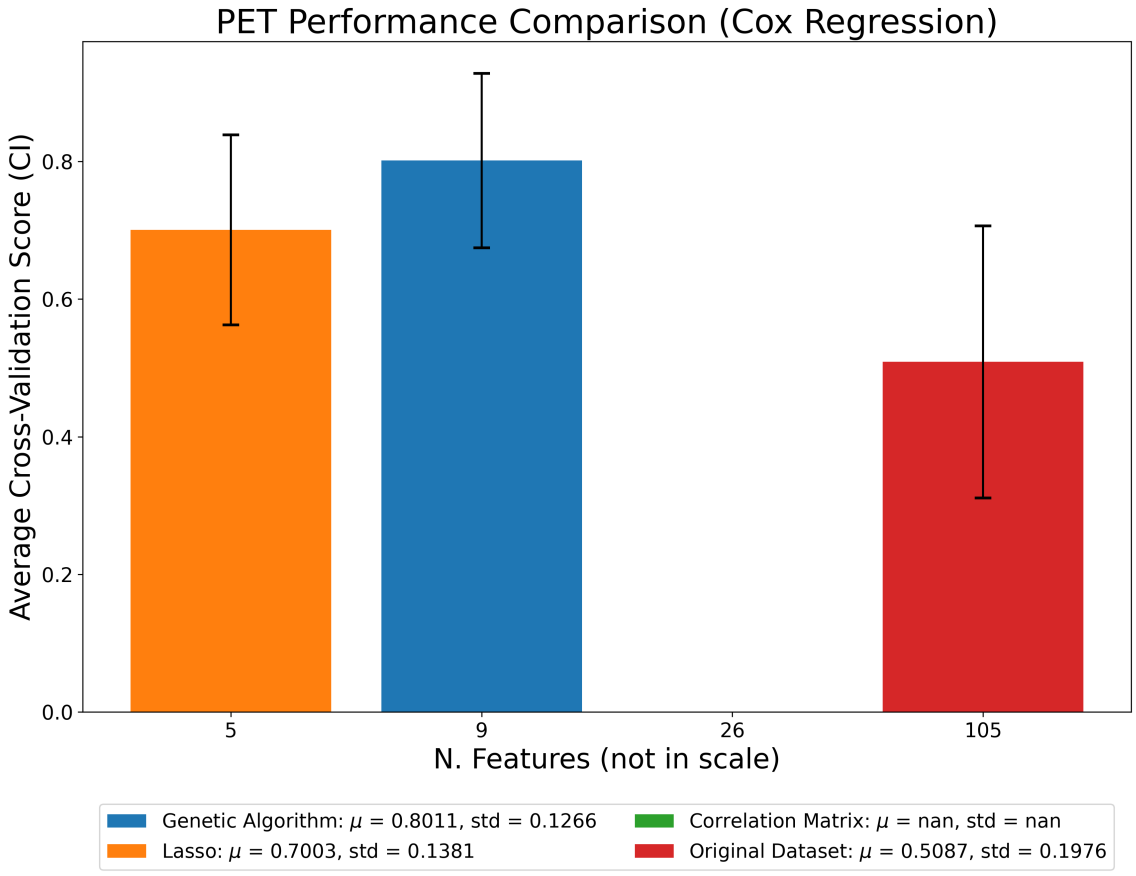



Figure 2.20: Repeated 5-fold cross-validation performance comparison for Cox models built on different features subsets. Results for CT and PET features are showed in (a) and (b) respectively.

As a reference, the baseline score, i.e., the one obtained using all the features, was calculated. It is depicted in red in Figure (2.20) and clearly shows the substantial improvement obtained with feature selection.
In the end, considering the results obtained with the different methods, the subsets of features selected by the Genetic Algorithm, both for PET and CT, were kept as the candidate ones and used for the rest of the work.

### 2.4.3 Model Selection and Fine Tuning

Once the subsets of candidate predictors were identified, the following step was to fine-tune the parameters of the Cox models and also to build the Random Survival Forest models and the Decision Tree Classifiers as alternative models to test.

#### Cox Model Fine Tuning

The parameter we wanted to tune for the Cox models was the famous Ridge regularization parameter, also known as $\boldsymbol{l_2}$ **penalty**. The procedure adopted to find the best regularization was almost the same used previously with the LASSO regression. The only difference was that, for LASSO, we searched for the parameter that maximized the cross-validation average score, while in this case, we wanted to find the parameter maximizing the (2.31), i.e., the difference between the average score obtained in the repeated 5-fold Cross-Validation and its standard deviation.

```
# generate the penalization values
alphas = 10 ** np.linspace(-3, 0, 100)

# define the Cox model without penalty
cox = CoxPHSurvivalAnalysis()

scores = []  #store the scores

# define the repeated Cross-Validation
cv = RepeatedKFold(n_splits=5, n_repeats=5, random_state=0)

for alpha in alphas:
    # set the penalization parameter
    cox.set_params(alpha = alpha)

    # calculate and append the scores
    score = (np.mean(cross_val_score(cox, X_scaled, Y, cv=cv)) -
             np.std(cross_val_score(cox, X_scaled, Y, cv=cv)))
    scores.append(score)

# select the best penalization parameter
best_alpha = alphas[np.argmax(scores)]
```

The same process, partially shown in the code above, was employed for both the CT and the PET Cox models. The best values of the regularization parameters were found to be 0.05 for the CT features and 0.06 for the PET features.

### Random Survival Forest

As we saw, Random Forests have the intrinsic property of performing feature selection, in fact, each tree is built using random subsets of the original set of features, then, the predictions of the trees are combined to give the final prediction.
Therefore, the idea was to build two Random Survival Forests, one for PET and one for CT, utilizing all the features in the dataset and compare its performance to the Cox models built with the selected features. The advantage with this approach is that we do not have to worry about finding the best predictors, because the model should be able to handle that. On the other hand, however, it is extremely important to use the right hyper-parameters, because the performance of a Random Forest deeply depends on them.

To find the best parameters for the Random Survival Forests, a *Grid-Search* strategy (Pedregosa et al., 2011) was used. Basically, in a grid search, we define the set of parameters we want to tune for the model, and, for each of those parameters, we identify a series of possible values. The grid-search algorithm, then, tests all the value combinations for those parameters and selects the one that produces the best result. Once again, the models are evaluated using a 5-fold cross-validation repeated five times, with the CI as the metric, hence, the model with the best parameters is the one that maximizes the score in the repeated cross-validation.
The parameters, with their associated values, tested in the grid-search were the following:

- *n_estimators* = {100, 200, 500}: the number of trees in the forest
- *min_samples_split* = {8, 10, 12, 14}: the minimum number of samples required to split a node
- *max_features* = {5, 6, 7, 8, 9, 10}: the number of features to consider when looking for the best split

A part of the code is shown below:

```
# define the set of parameters and associated values
param_grid = {'n_estimators' : [100, 200, 500],
              'min_samples_split' : [8, 10, 12, 14],
              'max_features' : [5, 6, 7, 8, 9, 10]}

# repeated 5-k Cross-Validation
cv = RepeatedKFold(n_splits=5, n_repeats=5, random_state=0)

# grid search
grid = GridSearchCV(rsf, param_grid=param_grid, cv=cv)
grid.fit(X_scaled, Y)

results = grid.cv_results_ # results
best = grid.best_params_   # best parameters
```

The process was repeated for both CT and PET features. Notice that the grid-search process can be quite slow. In this case, for example, 72 models were evaluated ($3 \times 4 \times 6$), and each of them had to be trained and tested 25 times.
For CT features, the best parameters combination found was: *n_estimators* = 100, *min_sample_split* = 12 and *max_features* = 6. The respective cross-validation average

score, with the concordance index as metric, was 0.737, with a standard deviation of 0.107.
For PET, instead, slightly different parameters were found: *n_estimators* = 200, *min_sample_split* = 12 and *max_features* = 5. The respective cross-validation average score was 0.721, with a standard deviation of 0.138.
The cross-validation scores of the Random Survival Forest were promising, in fact, it performed slightly worse than the Cox model built using the features selected by the Genetic Algorithm, but better than all the Cox models built with the other subsets of selected features.

#### Decision Tree Classifier

Since the data were also suitable for a classification study, two Decision Tree Classifiers were built, one for PET and one for CT, using the features selected for the Cox model, to evaluate if it was able to effectively discern among the patients who experienced a failure and the ones who did not. The choice of a Decision Tree as a reference classifier was made because it is easy to interpret and the actual tree can be plotted for visualization. As it was for the Random Survival Forest, a grid-search was used to find the combination of parameters that maximized the cross-validation repeated five times. This time, however, because the algorithm is a classifier, the area under the ROC curve was used as the metric in the cross-validation. With respect to Random Forests, Decision Trees have fewer parameters to adjust, since we mostly want to control the depth of the tree. The parameters, with their associated values, defined in the grid-search were the following:

- *min_samples_split* = {8, 10, 12, 14, 16, 18}
- *class_weight* = {balanced, None}

The *class_weight* parameter was introduced because with classifiers it is important to pay attention to the balancing of the data. In fact, the presence of a dominant class in the training set could cause the classifier to be biased toward that class. This is exactly the case of this project, since 49 samples of the training set are associated with the class 0, and the remaining 19 with the class 1.
When the *class_weight* is set to balanced, it automatically adjusts the weights of the classes in the following way:

$$w_i = \frac{n_{samples}}{n_{classes} \times n_i} \tag{2.32}$$

where $w_i$ is the weight associated with class $i$, $n_{samples}$ is the total number of samples in the dataset, $n_{classes}$ is the number of classes, and $n_i$ is the number of samples of the class $i$. If *class_weight* is *None*, no weight is applied.
The optimal configuration for both PET and CT features was found to be: *min_samples_split* = 12 and *class_weight* = *None*. It gave an average cross-validation score of 0.675 with a standard deviation of 0.148 for CT features and an average score of 0.616 with a standard deviation equal to 0.158 for PET features.

### 2.4.4 Data visualization and Model Evaluation

Various techniques were used to visualize the data and the results. First of all, t-SNE was used to obtain a two-dimensional representation of the data. It was applied to the features already selected by the genetic algorithm because it generally produces better

results with lower dimensional data. Various *perplexity* values were tried, but they had no substantial impact on the generated maps, so, at the end, a *perplexity* of 20 was chosen both for PET and CT features.
Data distributions were generated using the **kernel density estimation** provided by the *Seaborn* library (Waskom et al., 2020). The kernel density estimation is very useful to visualize the distributions of the variables for the two groups of patients, i.e., the patients who survived and the ones who died. A snippet of the code used to plot the distributions of the features selected by the genetic algorithm is shown next.

```
# generate a plot for each feature in the dataset
for i in range(len(dataArray[0])):

    fig = plt.figure(figsize=(12, 9))
    ax = fig.add_subplot(1, 1, 1)

    # kernel density estimation plot for the two groups
    ax = sns.kdeplot(dataOnes[:, i], color='orange', label='Death',
                     shade=True)
    ax = sns.kdeplot(dataZeros[:, i], color='green', label='Survival',
                     shade=True)

    ax.legend(fontsize=16)

    # set the x-axis label according to the name of the feature plotted
    ax.set_xlabel(Data.columns[i], fontsize=20)
    ax.set_ylabel('Probability Density', fontsize=20)
    ax.set_title('Data Distribution (Standardized)', fontsize=22)
    ax.tick_params(axis='both', labelsize='x-large')

    # save the plot with the name of the feature and the distinction
    PET/CT
    plt.savefig('{}_Distribution_{}'.format(Data.columns[i], WHAT), dpi
    =300)
```

Moreover, the *lifelines* library was employed to plot the **adjusted survival curves** obtained with the Cox model, both for PET and CT features. Note that all the plots and adjusted survival curves have been produced using the training data only.

Finally, the Cox models and the Random Survival forests were evaluated on the test sets (PET and CT) that were held out at the beginning of the study, using the concordance index as metric. The Decision Tree Classifier, instead, was evaluated using the area under the ROC curve.
After the testing, the *recurrence* outcome was added to the models as an additional feature to determine if it could improve the models' performance.

# Chapter 3

# Results

## 3.1 Visualization

Initially, we had two feature sets, PET and CT, with 105 features each. Using the Genetic Algorithm, it was possible to reduce the number of features to 9, for both CT and PET features. The plots produced with the *kernel density estimation* showed a clear difference in the distribution of most of the selected features for the two groups of patients. Below, three distribution plots of CT features and three of PET features are displayed. The green distributions are associated with the patients who did not experience a failure during the study time, while the orange ones are associated with the patients who did.

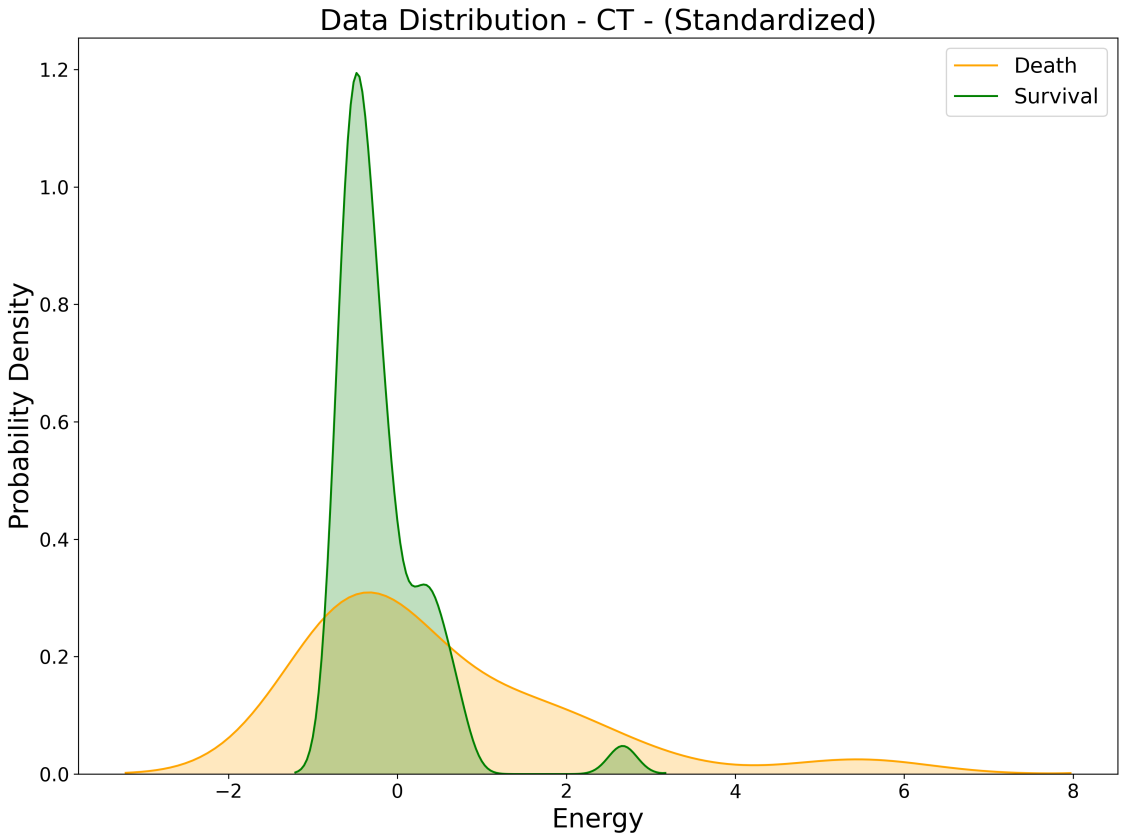
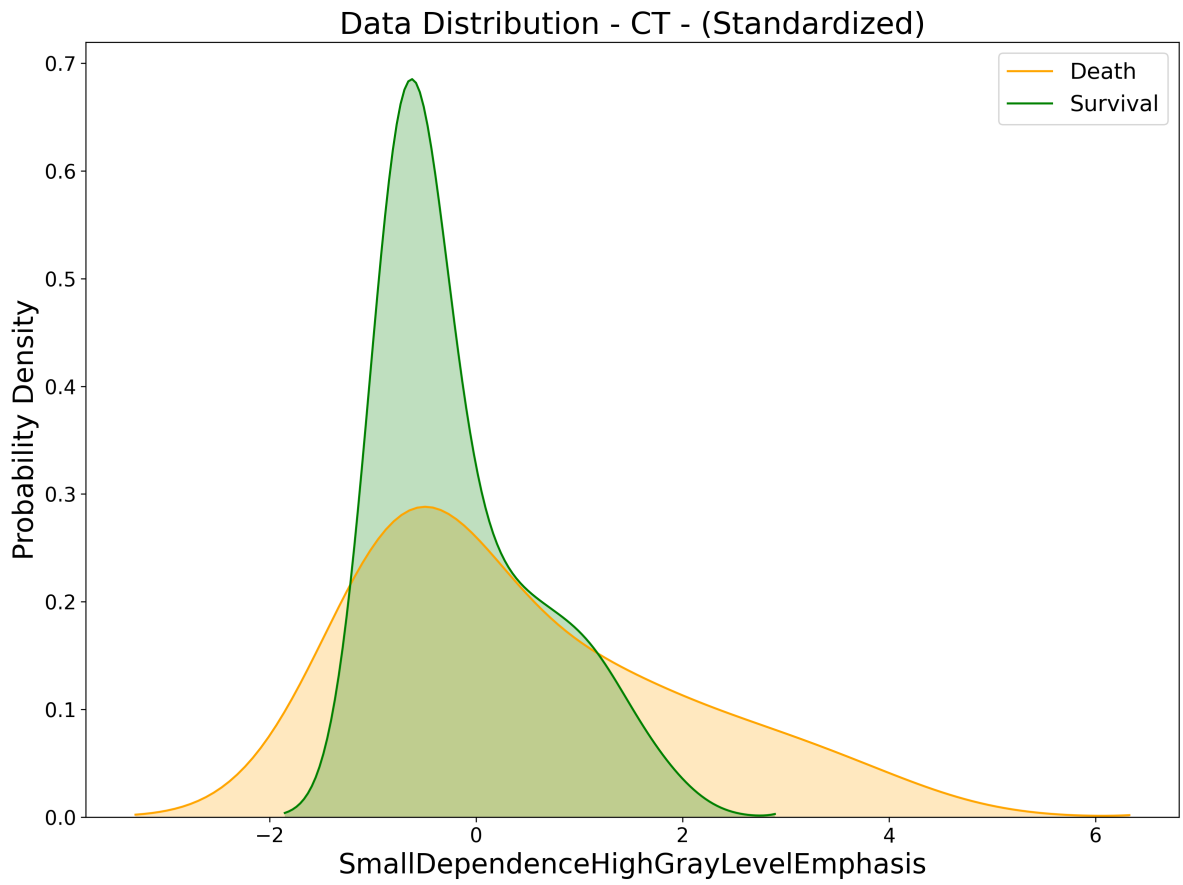



Figure 3.1: Distribution plot of the *Energy* CT feature for the two groups of patients.

(a)




(b)

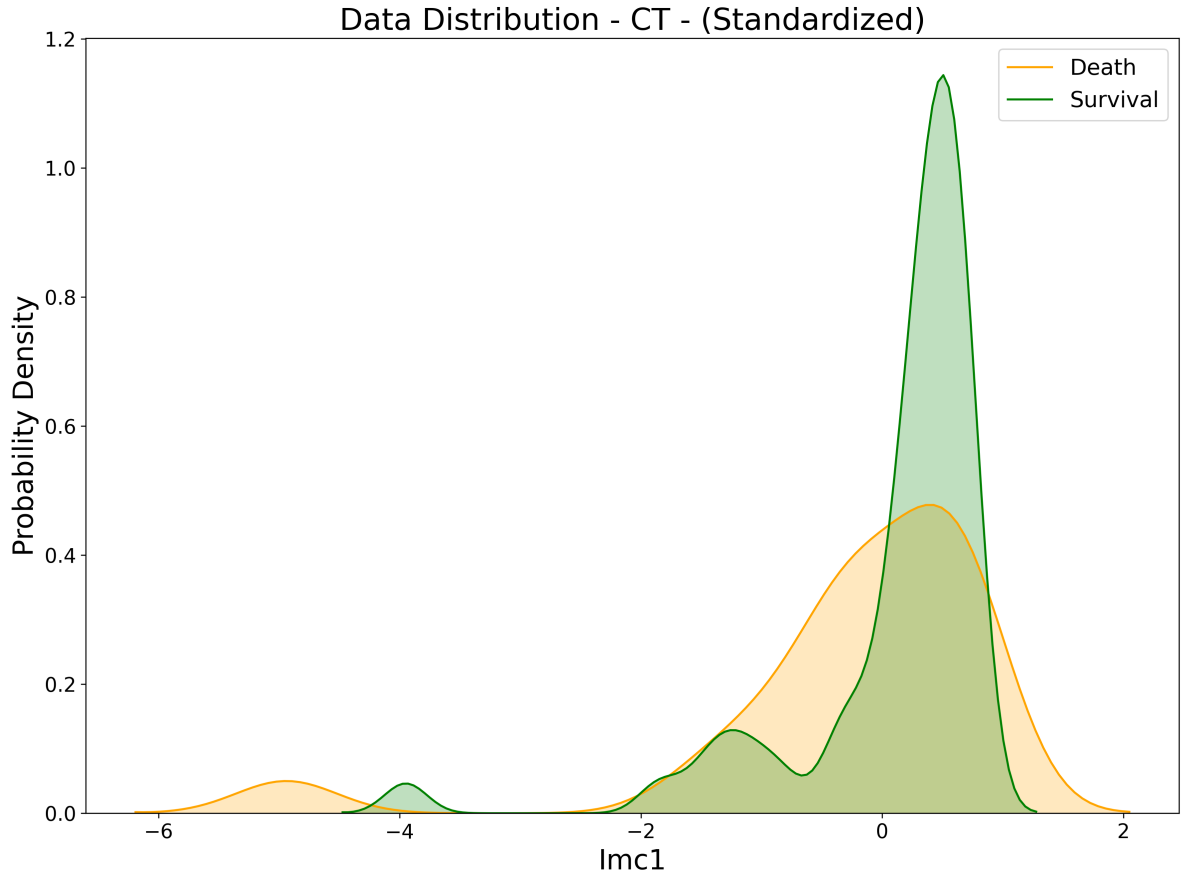
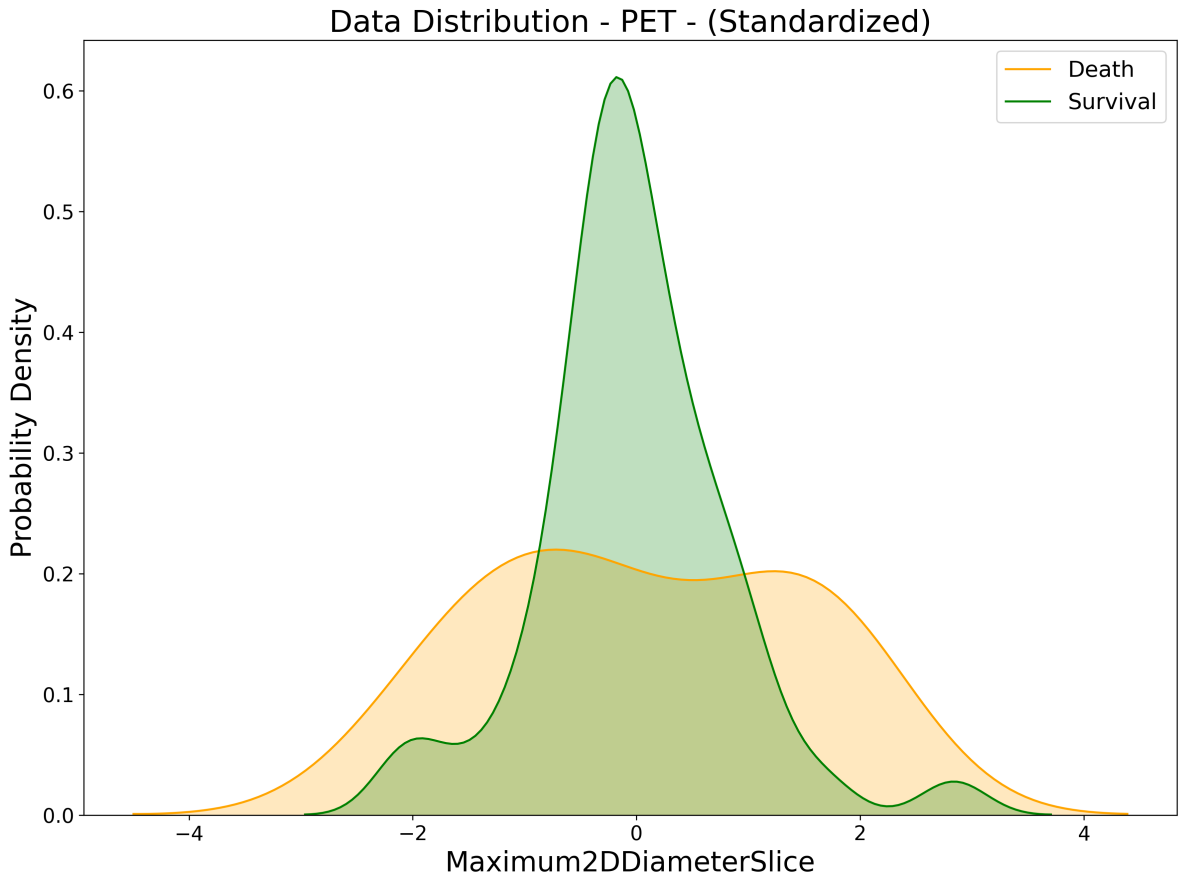



Figure 3.2: Distribution plots of the *SmallDependenceHighGrayLevelEmphasis* (a) and *Imc1* (b) CT features for the two groups of patients.

(a)




(b)

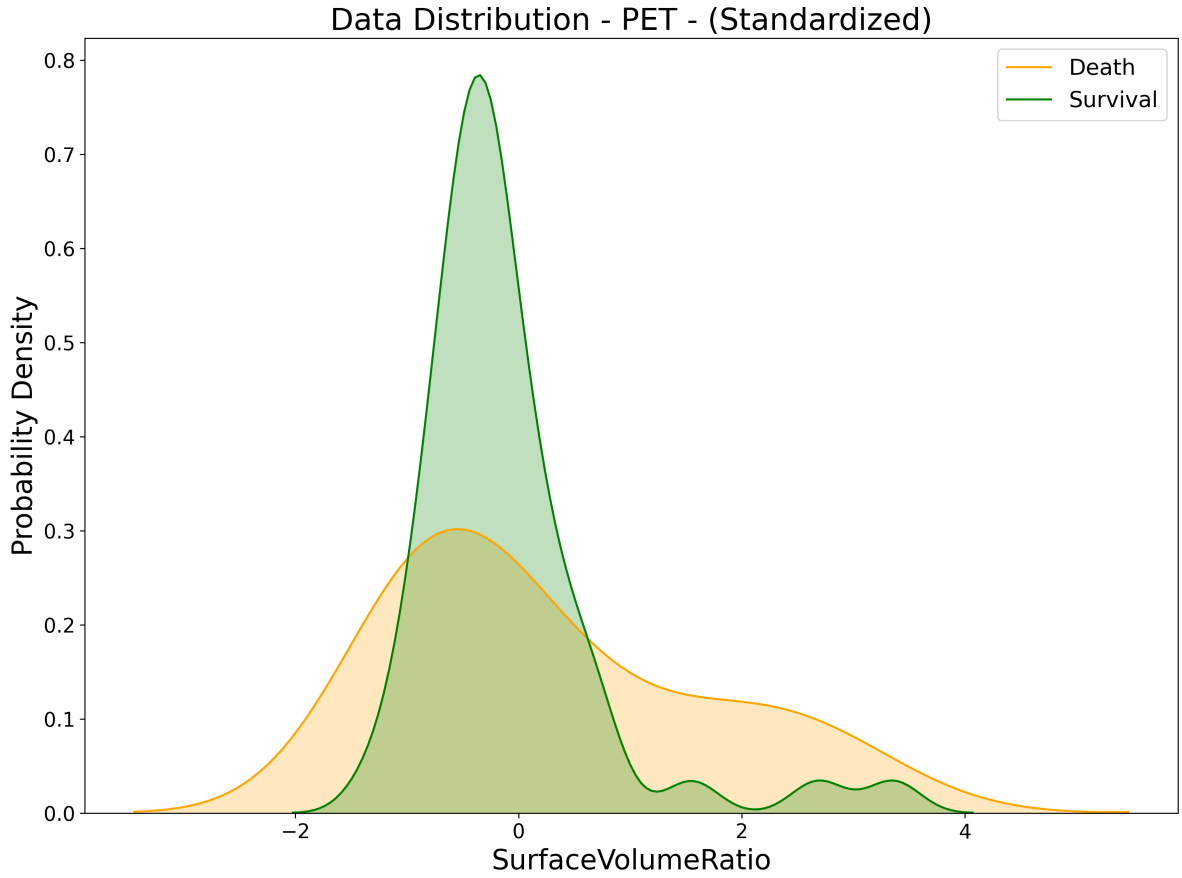



Figure 3.3: Distribution plots of the *Maximum2DDiameterSlice* (a) and *SurfaceVolumeRatio* (b) PET features for the two groups of patients.

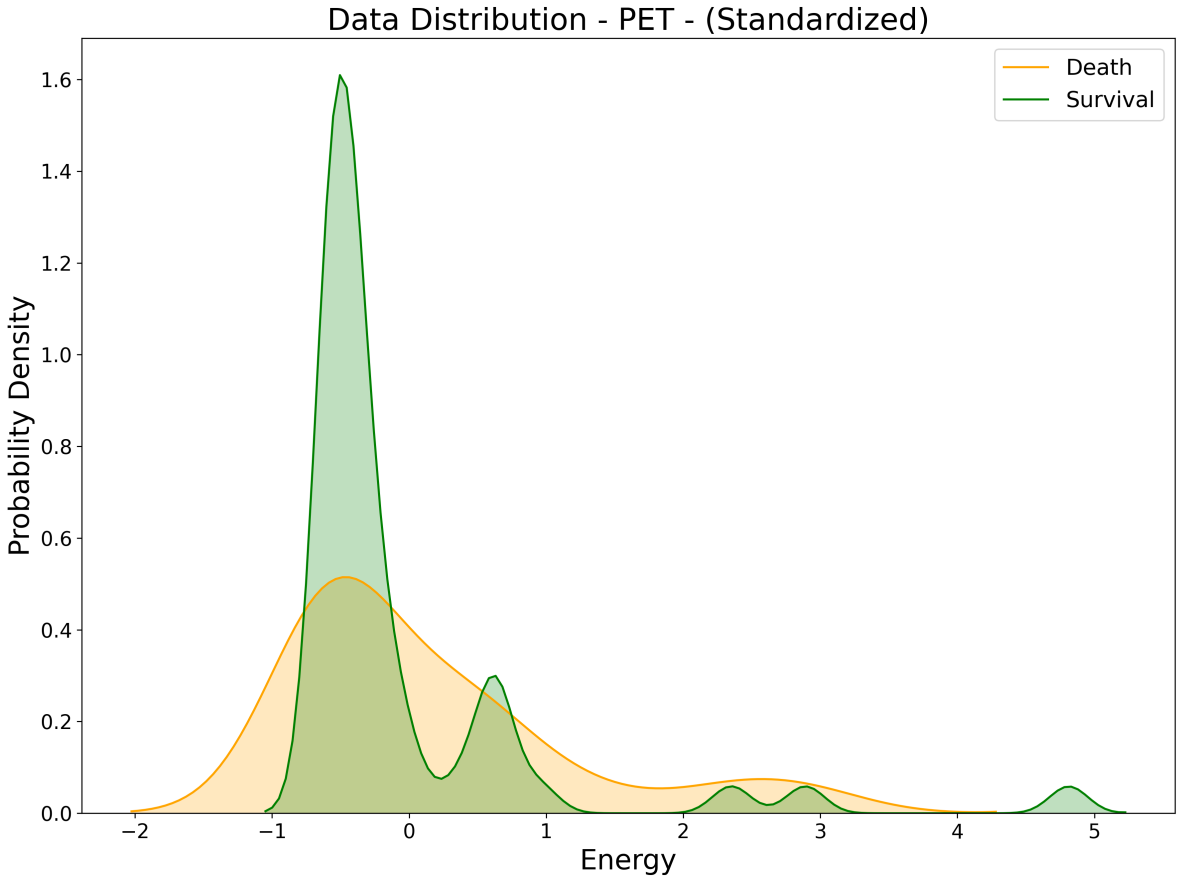
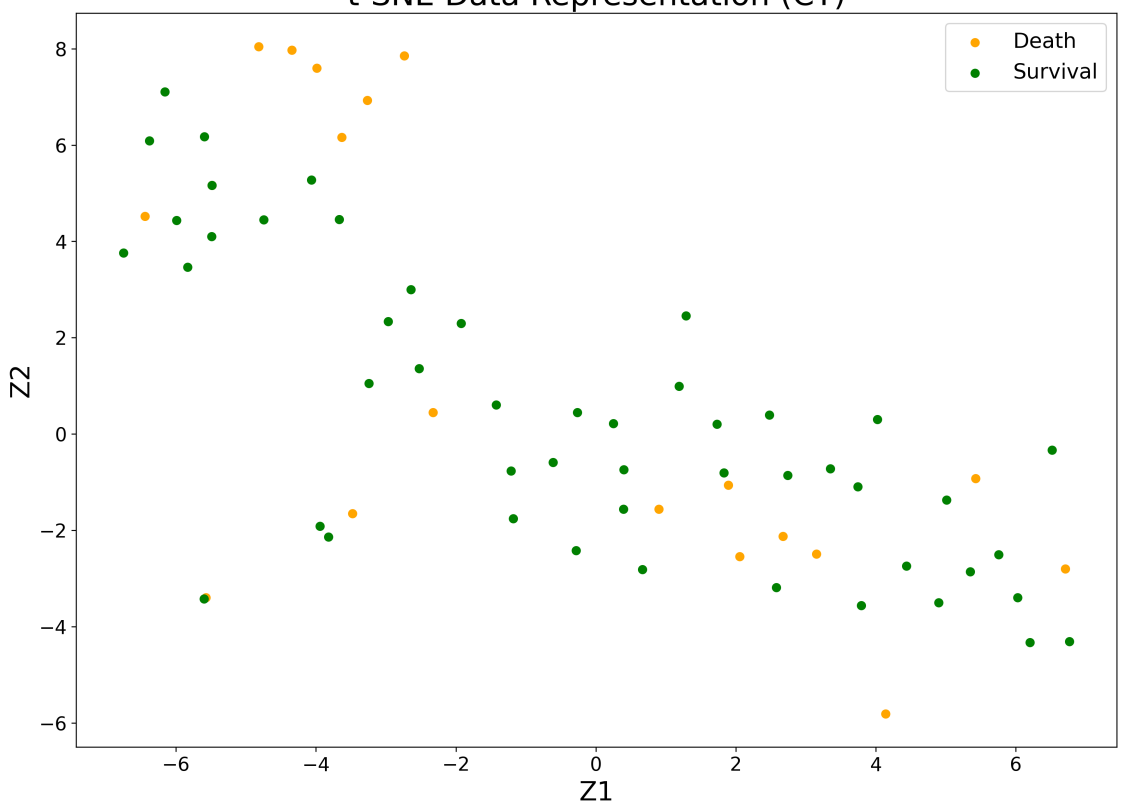



Figure 3.4: Distribution plot of the *Energy* PET feature for the two groups of patients.

For what concerns the t-SNE maps, instead, the algorithm was unable to effectively divide the two groups of patients. Various *perplexity* values have been used, but they gave similar results. In particular, no significant structure or cluster arises from the plots produced by t-SNE. Also, no noticeable difference can be found between the image produced with the CT features and the one produced with the PET features.
In my opinion, the poor results of the t-SNE mapping can be attributed to the combination of two main factors. The first one is the noise. CT and PET acquisitions are likely to be rather noisy, and t-SNE is particularly affected by the noise in the data. In fact, noise can act as a sort of bridge connecting two parts of the manifold that otherwise would be well separated. The second reason is the censoring of the data. The probability of experiencing an event, and thus of belonging to one of the two classes, indeed, strongly depends on the time of the study. In other words, two patients with similar feature values may belong to two different classes just because one of them has not experienced the event yet.
That said, however, the result of the t-SNE mapping does not mean that there are no relations between the features and the probability of the event, it simply suggests that the data are not suitable for a two-dimensional representation.
The t-SNE plots obtained for CT and PET features are shown next. The green points are the subjects who did not experience a failure, while the orange points are the subjects who did.

(a)




(b)

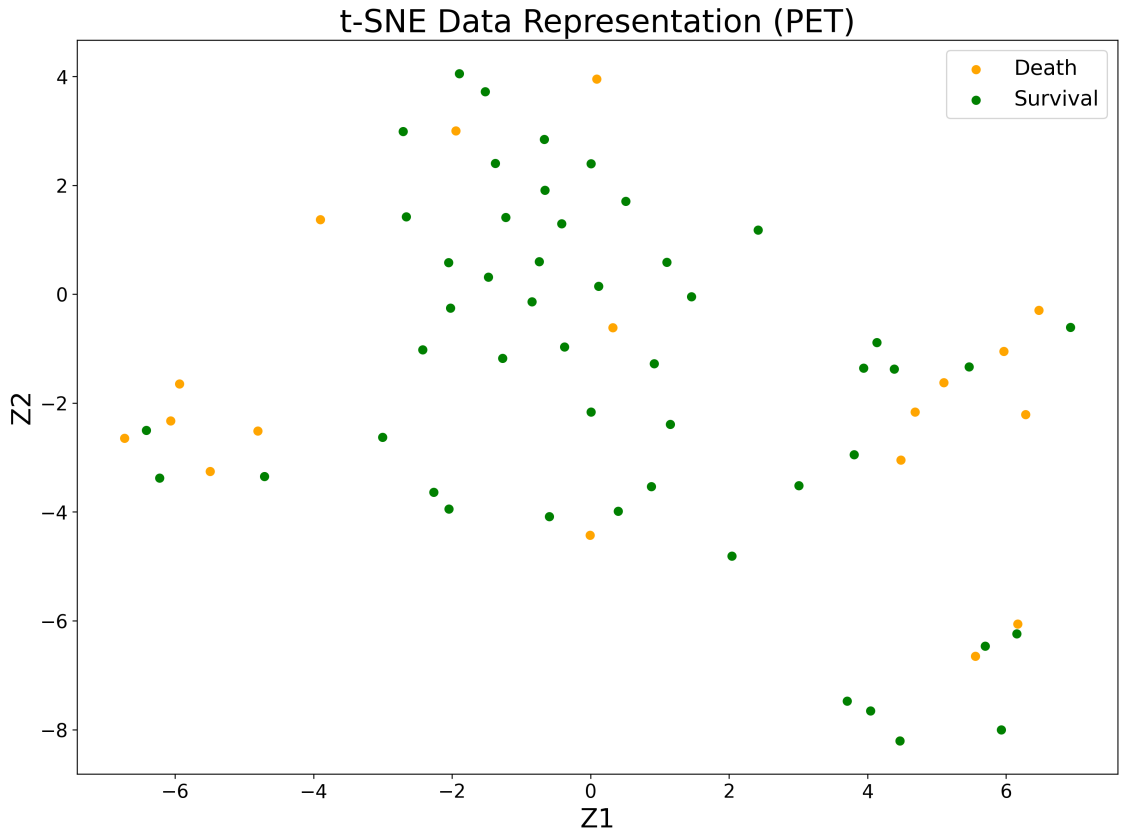



Figure 3.5: Two-dimensional representations of the data, obtained applying the t-SNE algorithm to the CT (a) and PET (b) selected features. Note that all the features had been previously standardized.

## 3.2 Models

### 3.2.1 Cox Model

First of all, we present the results of the fits of the Cox models to the training data, because they give us useful information about the features found using the genetic algorithm. Starting from the CT features, we refer to Table (3.1).

| Covariate | coef | HR | se(coef) | coef lower 95% | coef upper 95% | z | p |
|---|---|---|---|---|---|---|---|
| SmallDepHighGLEmph | 2.579 | 13.178 | 0.781 | 1.048 | 4.109 | 3.303 | 0.001 |
| LargeDepLowGLEmph | -1.069 | 0.343 | 0.546 | -2.139 | 0.001 | -1.957 | 0.050 |
| JointAverage | -4.234 | 0.014 | 1.240 | -6.665 | -1.803 | -3.414 | 0.001 |
| DifferenceEntropy | -1.124 | 0.325 | 0.527 | -2.157 | -0.091 | -2.132 | 0.033 |
| Imc2 | -1.258 | 0.284 | 0.911 | -3.044 | 0.529 | -1.380 | 0.168 |
| Imc1 | -2.679 | 0.069 | 1.339 | -5.304 | -0.054 | -2.000 | 0.045 |
| Skewness | -1.458 | 0.233 | 0.603 | -2.640 | -0.276 | -2.417 | 0.016 |
| Energy | 1.054 | 2.868 | 0.295 | 0.476 | 1.632 | 3.573 | 0.000 |
| Contrast1 | -2.093 | 0.123 | 1.122 | -4.292 | 0.106 | -1.865 | 0.062 |

Table 3.1: Results of the Cox model fitted to the training data using the subset of candidate CT features. Note that the names of the first two features have been shortened for a better visualization.

By looking at the last column of the table, we can see that most of the explanatory variables are highly significant, in fact, only three of them have an associated *p-value* greater than 0.05. The other interesting quantities obtained from the fit are the hazard ratios (HR), which give us the point estimates of the effects of the variables. However, it is also important to consider the confidence intervals of the hazard ratios. We can obtain the confidence intervals for the hazard ratios by exponentiating the two limits obtained for the confidence intervals for the regression coefficients. For instance, the hazard ratio of the *SmallDependenceHighGrayLevelEmphasis* is roughly 13.2, but its 95% confidence interval is given by the range of values 2.85-60.9, which is very broad. This means that the point estimate of the effect of that variable is somewhat unreliable.

Similar considerations can be made for the results obtained by fitting the Cox model to the training data using the PET features. In this case, however, by looking at the *p-values* in Table (3.2), we can see that only four of the candidate PET features can be considered highly significant. It is also interesting to notice that the *Energy* feature is present in both the CT and PET subsets of selected features, and the coefficient associated with it in the two models is almost identical. Moreover, the *Energy* feature can be considered highly significant both for the PET and the CT models. The similarity between the CT and PET *Energy* feature, indeed, can also be seen in the distribution plots (3.1) and (3.4).
An important consideration has to be made for two of the selected PET features: *ZoneEntropy* and *Correlation*. They have a very low variance, and this could mean that they have been selected because the model uses them to overfit the data, achieving a good score. As a consequence, it is possible that these two variables will not really have predictive power on the test data.

| Covariate | coef | HR | se(coef) | coef lower 95% | coef upper 95% | z | p |
|---|---|---|---|---|---|---|---|
| Max2DDiamSlice | 0.884 | 2.421 | 0.621 | -0.333 | 2.101 | 1.424 | 0.155 |
| SurfaceVolumeRatio | 1.270 | 3.562 | 0.480 | 0.329 | 2.212 | 2.645 | 0.008 |
| Max2DDiamRow | 1.052 | 2.864 | 0.598 | -0.119 | 2.223 | 1.761 | 0.078 |
| Correlation | -2.408 | 0.090 | 1.238 | -4.835 | 0.019 | -1.945 | 0.052 |
| Energy | 1.046 | 2.846 | 0.474 | 0.118 | 1.974 | 2.208 | 0.027 |
| Maximum | 1.560 | 4.758 | 0.861 | -0.128 | 3.248 | 1.811 | 0.070 |
| Minimum | 1.815 | 6.140 | 0.832 | 0.185 | 3.445 | 2.182 | 0.029 |
| 10Percentile | -3.731 | 0.024 | 1.360 | -6.397 | -1.065 | -2.743 | 0.006 |
| ZoneEntropy | -3.393 | 0.034 | 1.812 | -6.945 | 0.158 | -1.873 | 0.061 |

Table 3.2: Results of the Cox model fitted to the training data using the subset of candidate PET features. Note that the names of the first and third features have been shortened for a better visualization.

As we said, from a fitted Cox model we can obtain the survival curves adjusted for the explanatory variables. Below, four plots of adjusted survival curves are shown, two for the CT model and two for the PET model. They are generated by varying the value of a single feature while the others are kept constant.

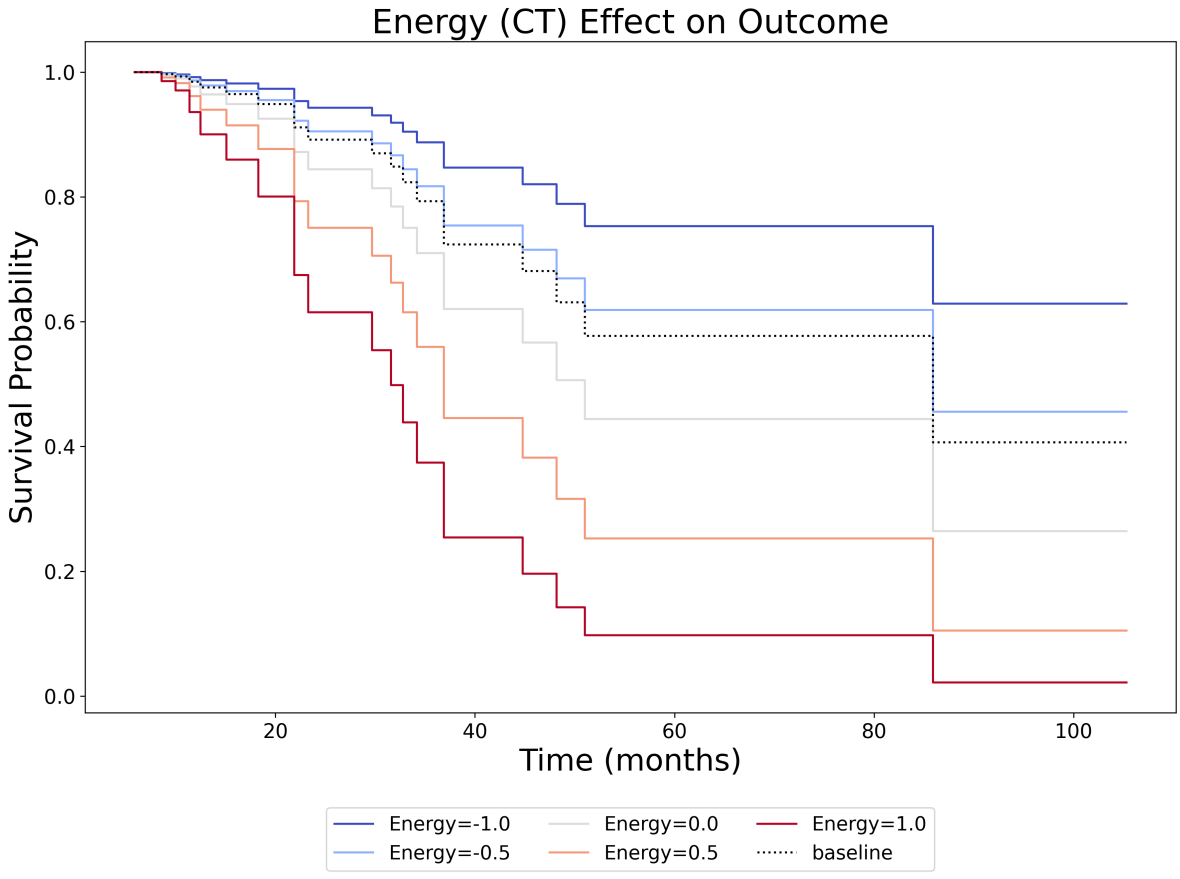
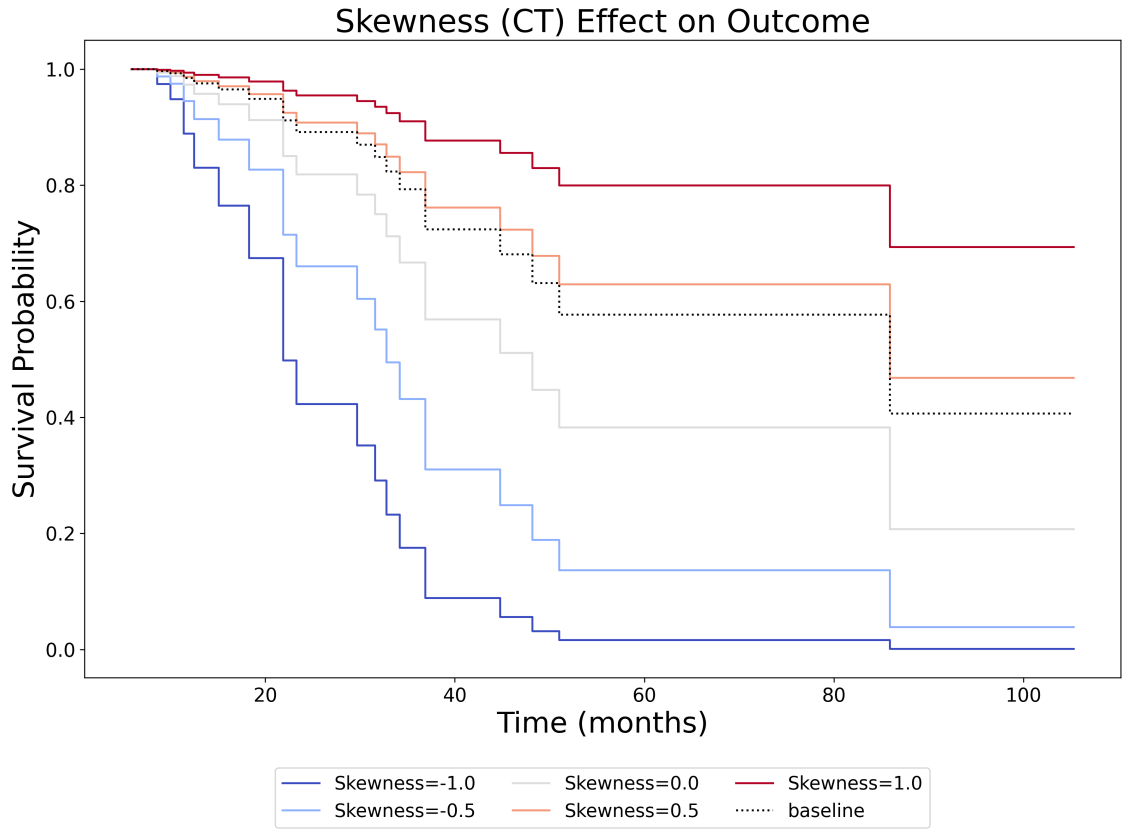



Figure 3.6: Effect of the variation of the *Energy* CT feature on the predicted survival.

(a)




(b)

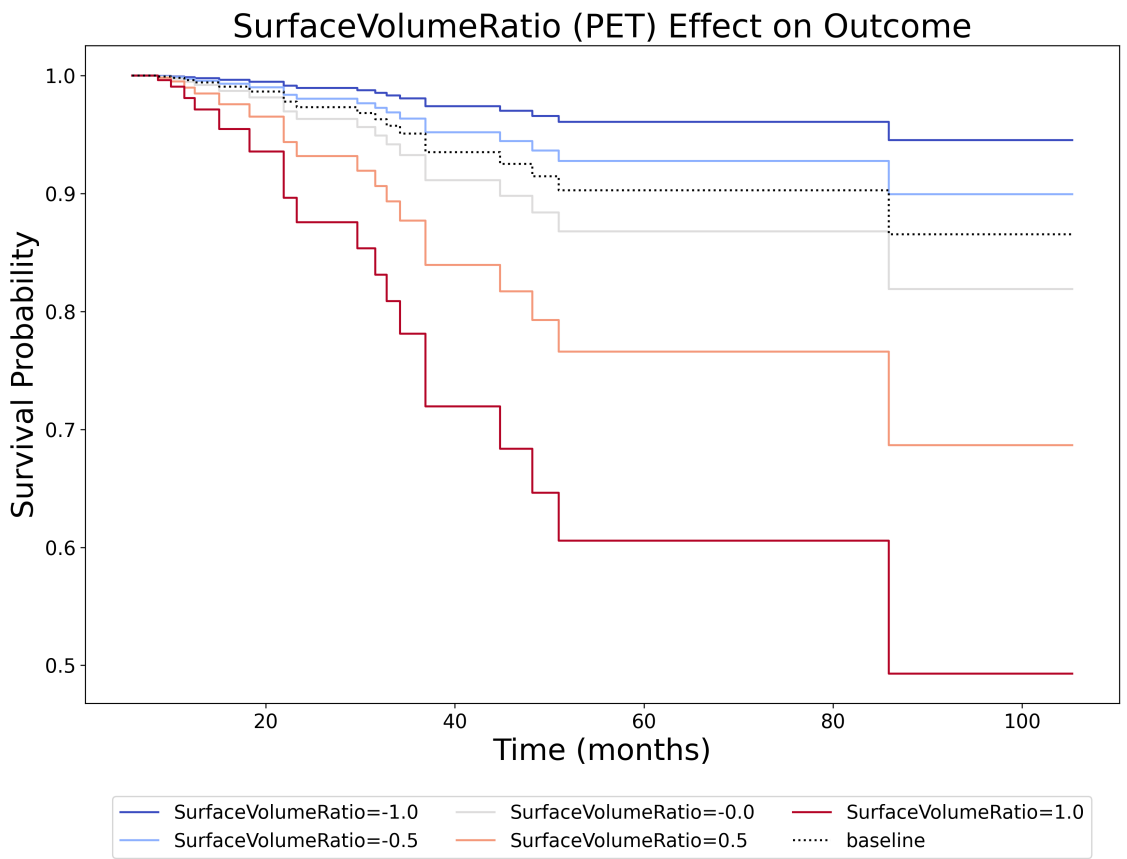



Figure 3.7: Effect of the variation of the *Skewness* CT feature (a) and the *SurfaceVolumeRatio* PET feature (b) on the predicted survival.

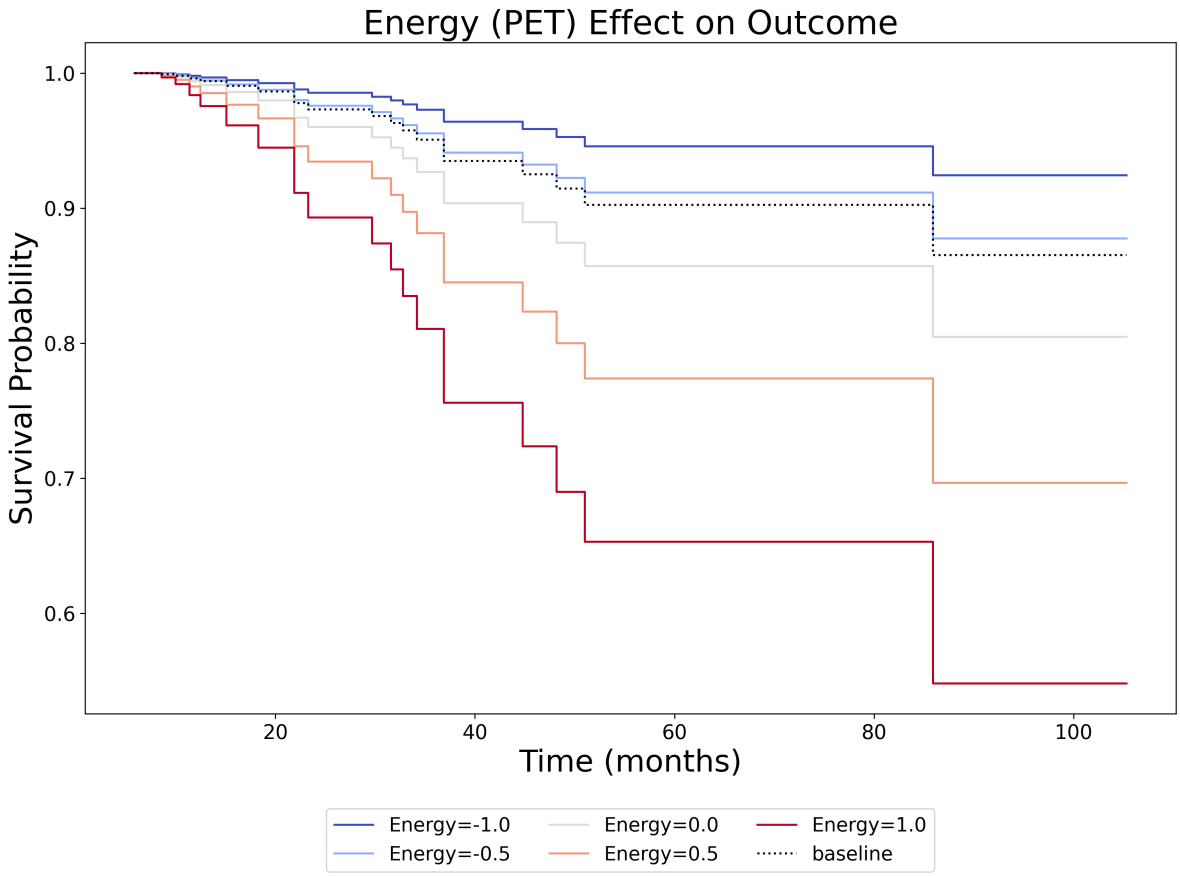



Figure 3.8: Effect of the variation of the *Energy* PET feature on the predicted survival.

Here, the *baseline* curve is equal to the predicted curve generated using the mean value of each feature.

### Cox Models Evaluation

Finally, to assess the quality of the Cox models, we evaluated them on the test sets using the concordance index as the metric. Recall that each of the two test sets (PET and CT) contains 17 samples, with 5 failures. The results are summarized in Table (3.3).

| Model | n. params | $l_2$ penalty | concordant pairs | discordant pairs | CI |
|---|---|---|---|---|---|
| Cox CT | 9 | 0.05 | 41 | 17 | 0.707 |
| Cox PET | 9 | 0.06 | 17 | 41 | 0.293 |
| Cox CT (base) | 105 | 0.05 | 36 | 22 | 0.621 |
| Cox PET (base) | 105 | 0.06 | 33 | 25 | 0.569 |

Table 3.3: Results of the CT and PET Cox models evaluated on the test sets.

As we can see in the table above, the CT Cox model achieves a very good result, correctly ordering 41 of the 58 possible pairs of subjects, reaching a 0.707 concordance index score.

This is perfectly in agreement with the cross-validation score obtained by the model, which was $0.798 \pm 0.105$ (Figure (2.20a)).
The PET Cox model result, instead, is exactly the opposite. It correctly orders only 17 pairs of subjects, achieving a concordance index score of 0.293. This is somewhat strange since it shows that the PET Cox model is *anti-concordant*, meaning that it predicts the risks at the inverse of how it should.
Referring to (Harrell et al., 1982), a concordance index near to 0, in principle, could also be considered good, because we could flip the risks predicted by the model to achieve a good concordance. However, I deem that, in this case, it wouldn't be legit to flip the model predictions. The reason is that the best features were explicitly picked in order to maximize the concordance index score obtained in a cross-validation, and, in fact, the PET Cox model achieved a cross-validation score of $0.801 \pm 0.127$ (Figure 2.20b). This means that by flipping the predictions, we would be using information deriving from the test data; information that we shouldn't have.
As a reference, in Table (3.3) are also showed the baseline results of the CT and PET Cox models, trained and tested using all the available features. The baseline PET model performs better than the one built using the selected features, but its concordance index score is still too low to be considered significant. The baseline CT model, instead, performs fairly well, but its score is lower than the one obtained by the CT Cox model built with the selected features.

### 3.2.2 Random Survival Forest

The results of the Random Survival Forest CT and PET models, evaluated on the respective test sets, are showed in Table (3.4).

| Model | n. estimators | min samples split | max features | concordant pairs | discordant pairs | CI |
|---|---|---|---|---|---|---|
| RSF CT | 100 | 12 | 6 | 16 | 42 | 0.276 |
| RSF PET | 200 | 12 | 5 | 30 | 28 | 0.517 |

Table 3.4: Results of the CT and PET Random Survival Forest models evaluated on the test sets.

From the previous table, it is clear that none of the two Random Survival Forest models is able to correctly predict the risk of the patients. The CT RSF model correctly orders only 17 of the 58 possible pairs of subjects, achieving a concordance index score of 0.276, substantially worse than the score obtained by the CT Cox model. This is also in contrast with the cross-validation score obtained by the CT RSF, which was $0.737 \pm 0.107$
The PET RSF model, instead, correctly orders 30 pairs of subjects, reaching a score of 0.517, similar to the score obtained by the baseline PET Cox model. This result can be considered compatible with the cross-validation score obtained by the PET RSF, which was $0.721 \pm 0.138$.

### 3.2.3 Decision Tree Classifier

First, we start by showing the graph of the Decision Tree Classifier built using the selected PET features. In the graph, the nodes of the tree are substituted with the stacked histograms of the class occurrences.

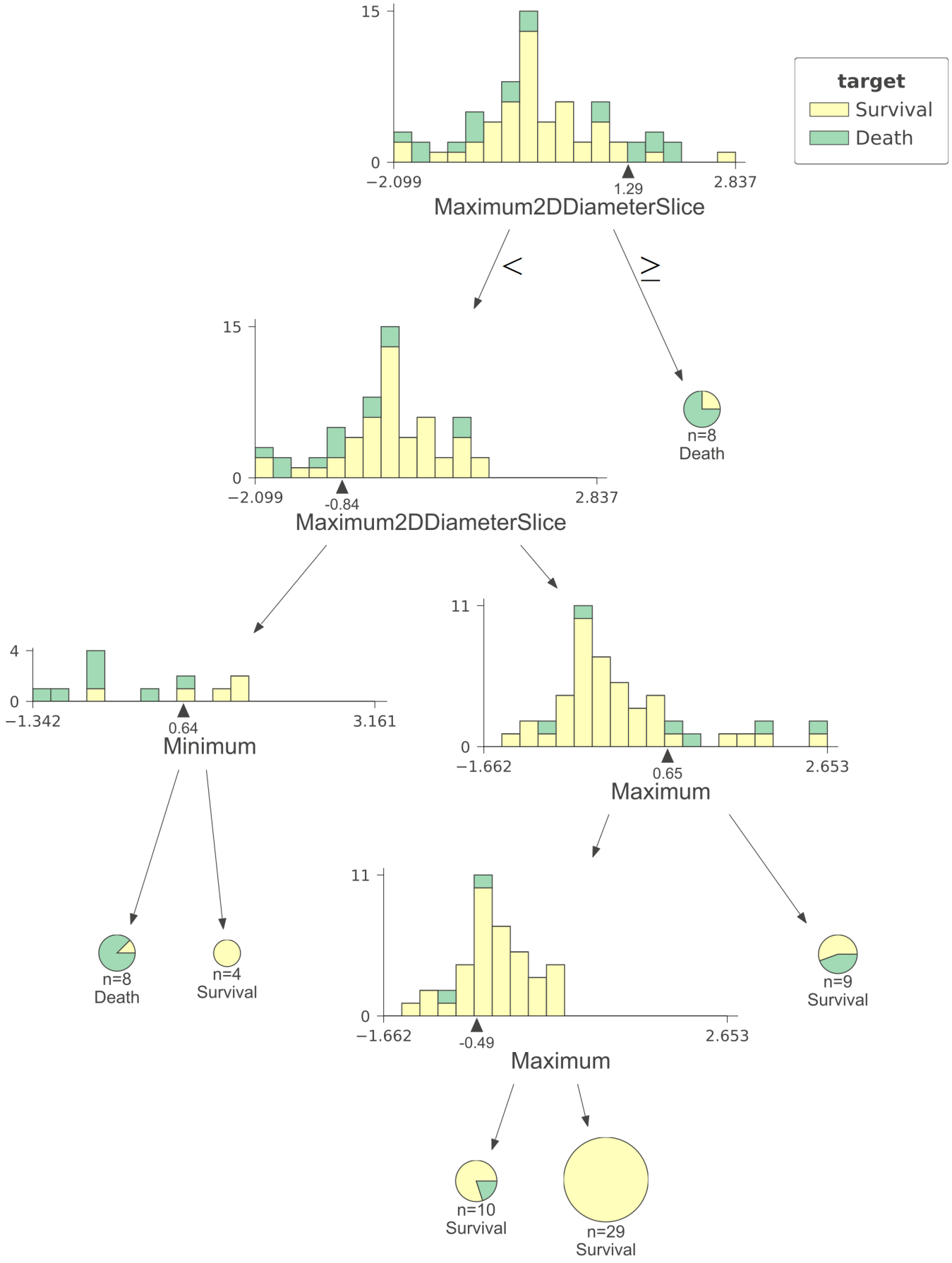



Figure 3.9: PET Decision Tree Classifier graph. The Survival class is depicted in yellow, while the Death class in green. Note that to generate this graph, the tree was fitted to the training data only.

From Figure (3.9) we can see that the tree uses only 3 of the 9 available features to split the data. Moreover, it is clear that the data are not easily separable, in fact, none of the leaf nodes of the Death class is pure. When evaluated on the test data, the PET Decision Tree achieves a ROC AUC score of 0.275, meaning that it is not able to correctly classify the samples.
For the CT Decision Tree the result is almost the same; it reaches a ROC AUC score of 0.292 on the test set.

### 3.2.4 Adding Recurrence

Recurrence should be a good predictor for the risk of death of a patient, therefore, we used it as an additional feature in the models. The results obtained are summarized in Table (3.5).

| Model | concordant pairs | discordant pairs | CI |
|---|---|---|---|
| Cox CT | 45 | 13 | 0.776 |
| Cox PET | 36 | 22 | 0.621 |
| RSF CT | 26 | 32 | 0.448 |
| RSF PET | 30 | 28 | 0.517 |

Table 3.5: Results of the CT and PET Cox and RSF models evaluated on the test sets with recurrence added as a feature.

As expected, the performance of all the models improved, except for the PET RSF model, which performed the same. The greatest improvement is achieved by the PET Cox model. The CT Cox model obtains an exceptionally good result, confirming to be the best of the models. Note that all the hyper-parameters of the models remained the same.
Regarding the Decision Tree Classifiers, adding the recurrence as a feature drastically changes the shape of the produced trees, in particular for the PET one, which is shown in Figure (3.10). As we can see, recurrence is used as the first feature for the splitting, meaning that it is very important to separate the two groups of subjects. Moreover, the tree now has more pure leaf nodes and uses more features to split the data.
The ROC AUC score achieved by the PET Decision Tree is now 0.612, way higher than the score obtained without the recurrence feature, but still far from being a good classifier. The CT Decision Tree, instead, has no improvement when recurrence is added as a feature; it obtains a ROC AUC score of 0.283, slightly lower than its score without recurrence, however, the difference is negligible.

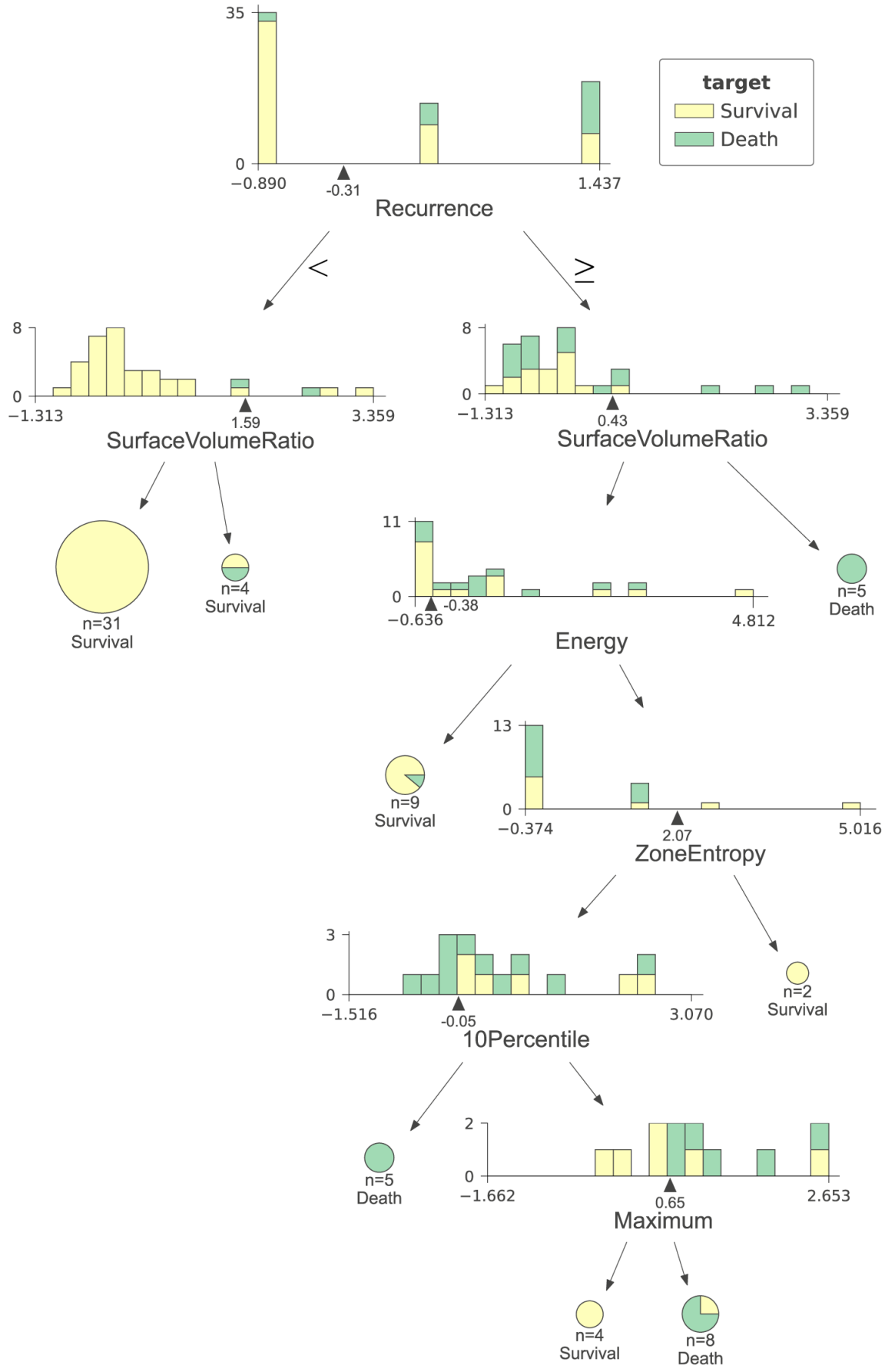



Figure 3.10: PET Decision Tree Classifier graph, with recurrence added as a feature. Note that the recurrence is a real number because the features were standardized.

## 3.3 Discussion of the Results

Starting from the results obtained using the radiomic features only, we saw that the best of the models built was the CT Cox model, which achieved a very solid score when evaluated on the test set. The PET Cox model, instead, was not capable to correctly predict the risks of the patients. The fact that, in general, CT features revealed to be better predictors can be explained considering that CT images have a greater resolution than the PET ones, so they can naturally provide more information. This is true especially for small-size tumors.
The fact that the PET Cox model performed very well in cross-validation and poorly when evaluated on the test set is a clear sign of overfitting. The 9 PET features found by the genetic algorithm, indeed, allowed the Cox model to fit the training data almost flawlessly, but they revealed to be bad predictors when applied to unseen data. This shows that, even if two steps of feature selection have been performed with the genetic algorithm, first using AIC and successively cross-validation, the risk of overfitting is still concrete when the dataset has few samples and many features.

Regarding the Random Survival Forest models, we showed that they both were not able to make correct predictions on the test set. This is probably due to the fact that, even if tree ensembles are capable of performing feature selection internally, the number of features was too high, especially considering the small number of subjects in the dataset. In particular, it is reasonable to believe that many of the available features could have no predicting power at all, hence, adding them to a model would only make its performance worse.
The Decision Tree models clearly showed that classifiers are not suited for a survival analysis, since they cannot take into account the censoring of the data. However, the tree graphs can be useful to visualize the relations between the features and the groups of patients. For instance, the fact that the *SurfaceVolumeRatio* feature is used to split both the shallower child nodes (Figure ((3.10)) indicates that it could be an important predictor.

Using recurrence as an additional feature improved the performance of almost all the models tested. The models that benefit most from the recurrence addition were the CT and PET Cox models. This is easily explainable by the fact that both the Cox models were built using small subsets of selected features, so the addition of a strong predictor actually resulted in a great performance improvement.
Conversely, adding recurrence to the Random Survival Forest model resulted in an almost negligible difference, because the sets of features used for those models were way larger, so the recurrence predictor had a minor effect.
The important result arising from the comparison of the Cox models built with the selected features and the Random Survival Forest models built with all the features is that a good process of feature selection can allow a simple model to perform better than a more powerful and sophisticated one.
A great advantage of using a simpler model with a subset of selected features is the interpretability of the result, which is of primary importance in studies like the one conducted in this work. From the Cox model, in fact, we can obtain information about the relations between the features and the predicted outcomes. For example, according to the CT Cox model, a higher value of the Energy feature corresponds to an increasing of the hazard. Likewise, the model fitted using recurrence as a feature reveals that patients who experienced recurrence have a higher death probability, as we could expect. Similar

information can be obtained only using *white box* models.

In conclusion, of all the models tested, the only one that showed clear predictive properties was the CT Cox model, both with and without the addition of the recurrence feature. In particular, the best CT Cox model, the one with recurrence, achieved a concordance index score of 0.776, which is exceptionally high, and clearly demonstrates the robustness of the model. The result also suggests that the CT radiomic features may be more descriptive than the PET ones, however, this could also depend on the feature selection process and not on the features themselves.

# Chapter 4

# Conclusions

The achieved results show that it is possible to develop machine learning models to predict the risk of death in cervical cancer patients using radiomic features as predictors. The performance of the CT Cox model, indeed, was good both for the training data and for the test data, showing that the model is reliable. This is true both with and without the recurrence added as a feature. In particular, the CT Cox model trained on the selected features was clearly superior to all the other models tested. This result indicates that the feature selection process is a crucial step of the analysis, especially when few data are available, as it usually happens in radiomics.
Concerning the feature selection, we showed that the best of the feature selection techniques employed was the Genetic Algorithm. Genetic Algorithms are rarely used in radiomics, however, they have several advantages with respect to other common methods. First, Genetic Algorithms can be used with any type of model and can optimize any kind of function, so they are much more versatile than methods like the LASSO regression. Moreover, it is possible to tune a wide range of parameters, allowing a profound control of the feature selection process.
The main weakness of the Genetic Algorithm feature selection process implemented in this work is that it does not provide any measure of the feature importance.

As future work, a process to measure the feature importance could be implemented in the algorithm. This could be accomplished in different ways. For example, consider that after several generations, usually, the mean fitness value of the population is close to the maximum fitness value achieved by the best individual, meaning that most of the individuals in the population are good candidate solutions. Consequently, it could be possible to count how many times a feature has been selected by the individuals of the population and use the result as a metric of the feature importance. However, this should be investigated further because it is possible that, at the end of the evolution process, many of the individuals are almost identical, so they would not provide reliable information.
Another way to measure the feature importance could be to run the Genetic Algorithm several times with different initialization and count how many times each feature is selected by the best individual in the population; similarly to the procedure used to empirically calculate the p-value. This should provide a more trustworthy measure because Genetic Algorithms are stochastic in nature, so it is unlikely to obtain the same solution more than once, especially if the initial number of features is high. The main drawback of a similar process is the computational burden. In fact, depending on the method used to calculate the fitness function, the number of individuals, and the number of generations,

a single evolution process, i.e., a single execution of the Genetic Algorithm, could last hours. With the proposed method, to have a significant measure of the feature importance, the algorithm should be executed tens of times, hence the entire process would be very time-consuming, especially if compared to a simple LASSO regression.

As further research, the algorithms and procedures developed to predict the survival probability could be easily adapted to predict the probability of recurrence. Recurrence outcome has a higher number of positive events, and this can benefit the learning process of survival models as the Cox model. The construction of the Cox partial likelihood, indeed, is made using the positive events only. However, before using recurrence as the outcome, it has to be discussed how to include subjects that experienced recurrence more than once.

Moreover, the recurrence could be used as a *time-varying* feature. We said that one of the assumptions of the Cox proportional hazard model is that the features do not depend on time, however, it is possible to extend the Cox model to take into account time dependencies. In particular, if $t_r$ is the time at which the patient experiences recurrence, we can consider the patient to be in the state 0, i.e., no recurrence, from the initial time of the study to $t_r$, then, from time $t_r$ to the final time of the study, the patient would be in state 1. A Cox model built in this way is called *multi-state Cox model*. Again, the implementation in the case of patients experiencing recurrence more than once should be further discussed.

The dataset used in this work also contains information about the tumor staging and the tumor type. As a future study, thus, it would be possible to investigate the relation between radiomic features and the aforementioned tumor characteristics. For example, machine learning models could be built to classify tumors into *early-stage* and *advanced-stage*. A great advantage of a classification study would be the possibility to use all the images acquired from a patient. For the survival analysis, in fact, only the first image of a patient has been employed, being it the image corresponding to the initial time of the study. For classification purposes, instead, also the follow-up images could be included since the only goal would be to predict the tumor type or staging. The resulting dataset would have more than twice the number of samples of the one used for the survival analysis. This would make the feature selection process more manageable, reducing the risk of overfitting.

Finally, the entire analysis could also be performed without dividing CT and PET features, i.e., selecting the best predictors out of the dataset containing both CT and PET features and ultimately developing a model with the selected features.

Also, clinical features could be added to the ones already selected in this work, to investigate if they are capable of improving the models' performance.

# Bibliography


Aerts, H. J., Velazquez, E. R., Leijenaar, R. T., Parmar, C., Grossmann, P., Carvalho, S., Bussink, J., Monshouwer, R., Haibe-Kains, B., Rietveld, D., et al. (2014). Decoding tumour phenotype by noninvasive imaging using a quantitative radiomics approach. *Nature communications*, *5*(1), 1–9.

Akaike, H. (1998). Information theory and an extension of the maximum likelihood principle. *Selected papers of hirotugu akaike* (pp. 199–213). Springer.

Blickle, T., & Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, *4*(4), 361–394.

Bou-Hamad, I., Larocque, D., Ben-Ameur, H., et al. (2011). A review of survival trees. *Statistics surveys*, *5*, 44–71.

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

Coroller, T. P., Grossmann, P., Hou, Y., Velazquez, E. R., Leijenaar, R. T., Hermann, G., Lambin, P., Haibe-Kains, B., Mak, R. H., & Aerts, H. J. (2015). Ct-based radiomic signature predicts distant metastasis in lung adenocarcinoma. *Radiotherapy and Oncology*, *114*(3), 345–350.

Cox, D. R. (1972). Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, *34*(2), 187–202.

Davidson-Pilon, C., Kalderstam, J., Jacobson, N., Reed, S., Kuhn, B., Zivich, P., Williamson, M., AbdealiJK, Datta, D., Fiore-Gartland, A., Parij, A., WIlson, D., Gabriel, Moneda, L., Moncada-Torres, A., Stark, K., Gadgil, H., Jona, Singaravelan, K., ... Golland, D. (2020). *Camdavidsonpilon/lifelines: V0.25.7* (Version v0.25.7). Zenodo. https://doi.org/10.5281/zenodo.4313838

Fang, Y., & li, J. (2010). A review of tournament selection in genetic programming, 181–192. https://doi.org/10.1007/978-3-642-16493-4_19

Fonti, V., & Belitser, E. (2017). Feature selection using lasso. *VU Amsterdam Research Paper in Business Analytics*, *30*, 1–25.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, *13*, 2171–2175.

Galavis, P. E., Hollensen, C., Jallow, N., Paliwal, B., & Jeraj, R. (2010). Variability of textural features in fdg pet images due to different acquisition modes and reconstruction parameters. *Acta Oncologica*, *49*(7), 1012–1016. https://doi.org/10.3109/0284186X.2010.498437

Galdi, P., & Tagliaferri, R. (2018). Data mining: Accuracy and error measures for classification and prediction. https://doi.org/10.1016/B978-0-12-809633-8.20474-3

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow, 2nd edition*. O'Reilly Media, Incorporated. https://books.google.it/books?id=O2VJzQEACAAJ

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms* (pp. 69–93). Elsevier.

Gutman, D. A., Dunn, W. D., Grossmann, P., Cooper, L. A., Holder, C. A., Ligon, K. L., Alexander, B. M., & Aerts, H. J. (2015). Somatic mutations associated with mri-derived volumetric features in glioblastoma. *Neuroradiology*, *57*(12), 1227–1237.

Harrell, F. E., Califf, R. M., Pryor, D. B., Lee, K. L., & Rosati, R. A. (1982). Evaluating the yield of medical tests. *Jama*, *247*(18), 2543–2546.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction*. Springer Science & Business Media.

Hatt, M., Majdoub, M., Vallières, M., Tixier, F., Le Rest, C. C., Groheux, D., Hindié, E., Martineau, A., Pradier, O., Hustinx, R., Perdrisot, R., Guillevin, R., El Naqa, I., & Visvikis, D. (2015). 18f-fdg pet uptake characterization through texture analysis: Investigating the complementary nature of heterogeneity and functional tumor volume in a multi–cancer site patient cohort. *Journal of Nuclear Medicine*, *56*(1), 38–44. https://doi.org/10.2967/jnumed.114.144055

Horowitz, J. L. (2001). The bootstrap. *Handbook of econometrics* (pp. 3159–3228). Elsevier.

Ishwaran, H., Kogalur, U. B., Blackstone, E. H., Lauer, M. S., et al. (2008). Random survival forests. *The annals of applied statistics*, *2*(3), 841–860.

Jacob, H., Dybvik, J. A., Ytre-Hauge, S., Fasmer, K. E., Hoivik, E. A., Trovik, J., Krakstad, C., & Haldorsen, I. S. (2021). An mri-based radiomic prognostic index predicts poor outcome and specific genetic alterations in endometrial cancer. *Journal of Clinical Medicine*, *10*(3). https://doi.org/10.3390/jcm10030538

Kim, J.-H. (2009). Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational statistics & data analysis*, *53*(11), 3735–3745.

King, A., Chow, K., Yu, K., Mo, F., Yeung, D., Yuan, J., Bhatia, K., Vlantis, A., & Ahuja, A. (2013). Head and neck squamous cell carcinoma: Diagnostic performance of diffusion-weighted mr imaging for the prediction of treatment response. *Radiology*, *266*(2), 531–538. https://doi.org/10.1148/radiol.12120167

Kleinbaum, D. G., & Klein, M. (2010). *Survival analysis*. Springer.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai*, *14*(2), 1137–1145.

Kumar, V., Gu, Y., Basu, S., Berglund, A., Eschrich, S. A., Schabath, M. B., Forster, K., Aerts, H. J., Dekker, A., Fenstermacher, D., et al. (2012). Radiomics: The process and the challenges. *Magnetic resonance imaging*, *30*(9), 1234–1248.

Lambin, P., Leijenaar, R. T., Deist, T. M., Peerlings, J., De Jong, E. E., Van Timmeren, J., Sanduleanu, S., Larue, R. T., Even, A. J., Jochems, A., et al. (2017). Radiomics: The bridge between medical imaging and personalized medicine. *Nature reviews Clinical oncology*, *14*(12), 749–762.

LeBlanc, M., & Crowley, J. (1993). Survival trees by goodness of split. *Journal of the American Statistical Association*, *88*(422), 457–467. http://www.jstor.org/stable/2290325

Lin, D. (2008). On the breslow estimator. *Lifetime data analysis*, *13*, 471–80. https://doi.org/10.1007/s10985-007-9048-y

Longato, E., Vettoretti, M., & Di Camillo, B. (2020). A practical perspective on the concordance index for the evaluation and selection of prognostic time-to-event models. *Journal of Biomedical Informatics*, 103496.

Lucia, F., Visvikis, D., Desseroit, M.-C., Miranda, O., Malhaire, J.-P., Robin, P., Pradier, O., Hatt, M., & Schick, U. (2018). Prediction of outcome using pretreatment 18f-fdg pet/ct and mri radiomics in locally advanced cervical cancer treated with chemoradiotherapy. *European Journal of Nuclear Medicine and Molecular Imaging*, *45*. https://doi.org/10.1007/s00259-017-3898-7

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(Nov), 2579–2605.

Mattonen, S. A., Palma, D. A., Johnson, C., Louie, A. V., Landis, M., Rodrigues, G., Chan, I., Etemad-Rezai, R., Yeung, T. P., Senan, S., et al. (2016). Detection of local cancer recurrence after stereotactic ablative radiation therapy for lung cancer: Physician performance versus radiomic assessment. *International Journal of Radiation Oncology* Biology* Physics*, *94*(5), 1121–1128.

Mu, W., Chen, Z., Liang, Y., Shen, W., Yang, F., Dai, R., Wu, N., & Tian, J. (2015). Staging of cervical cancer based on tumor heterogeneity characterized by texture features on18f-FDG PET images. *Physics in Medicine and Biology*, *60*(13), 5123–5139. https://doi.org/10.1088/0031-9155/60/13/5123

Nie, K., Chen, J.-H., Hon, J. Y., Chu, Y., Nalcioglu, O., & Su, M.-Y. (2008). Quantitative analysis of lesion morphology and texture features for diagnostic prediction in breast mri. *Academic radiology*, *15*(12), 1513–1525.

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

Pfaehler, E., Beukinga, R., Jong, J., Slart, R., Slump, C., Dierckx, R., & Boellaard, R. (2018). Repeatability of 18 f-fdg pet radiomic features: A phantom study to explore sensitivity to image reconstruction settings, noise, and delineation method. *Medical Physics*, *46*. https://doi.org/10.1002/mp.13322

Pölsterl, S. (2020). Scikit-survival: A library for time-to-event analysis built on top of scikit-learn. *Journal of Machine Learning Research*, *21*(212), 1–6. http://jmlr.org/papers/v21/20-729.html

Rao, R. B., Fung, G., & Rosales, R. (2008). On the dangers of cross-validation. an experimental evaluation. *Proceedings of the 2008 SIAM international conference on data mining*, 588–596.

Steck, H., Krishnapuram, B., Dehing-Oberije, C., Lambin, P., & Raykar, V. C. (2008). On ranking in survival analysis: Bounds on the concordance index. *Advances in neural information processing systems*, 1209–1216.

Team, P. D. (2020). *Pandas-dev/pandas: Pandas 1.2.0* (Version v1.2.0). Zenodo. https://doi.org/10.5281/zenodo.4394318

Uno, H., Cai, T., Pencina, M. J., D'Agostino, R. B., & Wei, L.-J. (2011). On the c-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. *Statistics in medicine*, *30*(10), 1105–1117.

Van Griethuysen, J. J., Fedorov, A., Parmar, C., Hosny, A., Aucoin, N., Narayan, V., Beets-Tan, R. G., Fillion-Robin, J.-C., Pieper, S., & Aerts, H. J. (2017). Computational radiomics system to decode the radiographic phenotype. *Cancer research*, *77*(21), e104–e107.

Waskom, M., Gelbart, M., Botvinnik, O., Ostblom, J., Hobson, P., Lukauskas, S., Gemperline, D. C., Augspurger, T., Halchenko, Y., Warmenhoven, J., Cole, J. B., de Ruiter, J., Vanderplas, J., Hoyer, S., Pye, C., Miles, A., Swain, C., Meyer, K.,

Martin, M., ... Brunner, T. (2020). *Mwaskom/seaborn: V0.11.1 (december 2020)* (Version v0.11.1). Zenodo. https://doi.org/10.5281/zenodo.4379347

Wattenberg, M., Viégas, F., & Johnson, I. (2016). How to use t-sne effectively. *Distill*. https://doi.org/10.23915/distill.00002

Wirsansky, E. (2020). *Hands-on genetic algorithms with python - applying genetic algorithms to solve real-world deep learning and artificial intelligence problems*. Packt Publishing.

Yan, J., Chu-Shern, J. L., Loi, H. Y., Khor, L. K., Sinha, A. K., Quek, S. T., Tham, I. W., & Townsend, D. (2015). Impact of image reconstruction settings on texture features in 18f-fdg pet. *Journal of Nuclear Medicine*, *56*(11), 1667–1673. https://doi.org/10.2967/jnumed.115.156927

Yip, S. S., & Aerts, H. J. (2016). Applications and limitations of radiomics. *Physics in Medicine & Biology*, *61*(13), R150.