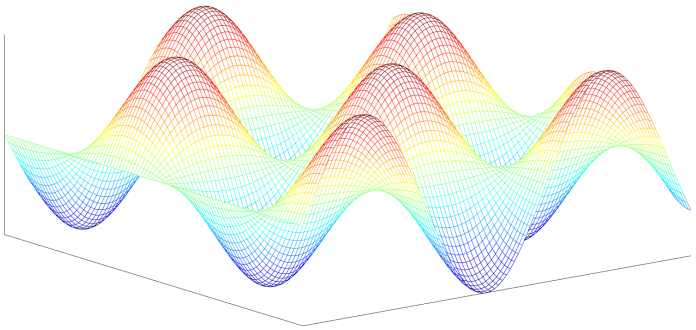
MACHINE LEARNING FOR SURVIVAL PREDICTION IN BREAST CANCER

BY

LEONARDO VANNESCHI

NOVA Information Management School (NOVA IMS)
Universidade Nova de Lisboa
Campus de Campolide, 1070-312 Lisboa, Portugal



Title

Machine Learning for Survival Prediction in Breast Cancer

Author

Leonardo Vanneschi

Publisher

Instituto Superior de Estatística e Gestão de Informação da Universidade Nova de Lisboa NOVA Information Management School (NOVA IMS)

ISBN

978-972-8093-18-1

Digital Version

http://hdl.handle.net/10362/110873

This work is supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., in the context of the project BINDER (PTDC/CCI-INF/29168/2017)

How to cite

Vanneschi, L. (2021). Machine Learning for Survival Prediction in Breast Cancer. Lisboa: Instituto Superior de Estatística e Gestão de Informação da Universidade Nova de Lisboa. NOVA Information Management School (NOVA IMS). ISBN: 978-972-8093-18-1. Link: http://hdl.handle.net/10362/110873

Contents

Chapter 1
Introduction

Current cancer therapies have serious side effects: ideally type and dosage of the therapy should be matched to each individual patient based on his/her risk of relapse. Therefore the classification of cancer patients into risk classes is a very active field of research, with direct clinical applications. Until recently, patient classification was based on a series of clinical and histological parameters. The advent of high-throughput techniques to measure gene expression led in the last decade to a large body of research on gene expression in cancer, and in particular on the possibility of using gene expression data to improve patient classification. A gene signature is a set of genes whose levels of expression can be used to predict a biological state (see [Nevins and Potti, 2007]): in the case of cancer, gene signatures have been developed both to distinguish cancerous from non-cancerous conditions and to classify cancer patients based on the aggressiveness of the tumor, as measured for example by the probability of relapsing within a given time.

While many studies have been devoted to the identification of gene signatures in various types of cancer, the question of the algorithms to be used to maximize the predictive power of a gene signature has received less attention. To investigate this issue systematically, one of the most established gene signatures is considered in this work, i.e. the 70-gene signature for breast cancer [van 't Veer et al., 2002]. This work proposes a comparison of the performance of four different machine learning algorithms in using this signature to predict the survival of a cohort of breast cancer patients. The 70-gene signature is a set of microarray features selected in [van 't Veer et al., 2002] based on correlation with survival, on which the molecular prognostic test for breast cancer "MammaPrint"^{TM} is based. While several machine learning algorithms have been used to classify cancer samples based on gene expression data [Chu and Wang, 2005, Deb and Reddy, 2003, Deutsch, 2003, Langdon and Buxton, 2004, Paul and Iba, 2005, Yu et al., 2007], in this a systematic comparison of the performance of four machine learning algorithms using the same features to predict the same classes is performed. In this comparison, feature selection is thus *not* explicitly performed as a pre-processing phase before executing the machine learning algorithms. The machine learning algorithms studied here are Genetic Programming (GP), Support Vector Machines, Multilayered Perceptrons

and Random Forests. These methods were applied to the problem of using the 70-gene signature to predict the survival of the breast cancer patients included in the NKI dataset [van de Vijver et al., 2002]. This is considered one of the gold-standard datasets in the field, and the predictive power of the 70-gene signature on these patients was already shown in [van de Vijver et al., 2002]. In this preliminary study all the studied machine learning methods were used in an "out-of-the-box" version, so as to obtain a first evaluation, as unbiased as possible, of the performance of the methods.

Many different machine learning methods [Lu and Han, 2003] have already been applied for microarray data analysis, like k-nearest neighbors [Michie et al., 1994], hierarchical clustering [Alon et al., 1999], self-organizing maps [Hsu et al., 2003], Support Vector Machines [Guyon et al., 2002, Hernandez et al., 2007] or Bayesian networks [Friedman et al., 2000]. Furthermore, in the last few years Evolutionary Algorithms (EA) [Holland, 1975] have been used for solving both problems of feature selection and classification in gene expression data analysis. Genetic Algorithms (GAs) [Goldberg, 1989] have been employed for building selectors where each allele of the representation corresponds to one gene and its state denotes whether the gene is selected or not [Liu et al., 2005]. GP on the other hand has been shown to work well for recognition of structures in large data sets [Moore et al., 2001]. GP was applied to microarray data to generate programs that reliably predict the health/malignancy states of tissue, or classify different types of tissues. An intrinsic advantage of GP is that it automatically selects a small number of feature genes during the evolution [Rosskopf et al., 2007]. The evolution of classifiers from the initial population seamlessly integrates the process of gene selection and classifier construction. In fact, in [Yu et al., 2007] GP was used on cancer expression profiling data to select potentially informative feature genes, build molecular classifiers by mathematical integration of these genes and classify tumour samples. Furthermore, GP has been shown a promising approach for discovering comprehensible rule-based classifiers from medical data [Bojarczuk et al., 2001] as well as gene expression profiling data [Hong and Cho, 2006]. The results presented in those contributions are encouraging and pave the way to a further investigation of GP for this kind of datasets, which is reported in this manuscript.

Chapter 2
Machine Learning

Machine Learning (ML) [Mitchell, 1997, Shalev-Shwartz and Ben-David, 2014] is a field of study whose objective is to program computers to automatically learn to solve a problem, or accomplish a task. ML is useful when manually programming a computer to carry out a task is either impractical or infeasible. Typical cases are either problems that are so complex to be beyond human capabilities, like the ones characterized by vast amounts of data, or tasks that living beings perform routinely, yet our introspection on how we do it is not sufficiently elaborated to allow us to extract a well defined algorithm, like for instance driving, speech recognition, image understanding or client categorization. Other tasks where ML is useful are the ones where adaptativity to changes in the environment is a necessary requirement, like for instance time series forecasting, handwritten text decoding or spam detection. In its most accepted definition:

> "Machine Learning is the study of algorithms that automatically improve by means of experience" [Mitchell, 1997].

In this definition, *learning* is intended as *improving by means of experience*. Even though the term "learning" can have several meanings and interpretations, it is clear that "improving by means of experience" is one of the most intuitive and close to our everyday experience. For instance, it includes the idea of "trial and error", that is very often implemented by many living beings when they are about to learn how to solve new tasks: learning often implies numerous consecutive attemps (or trials) of solving the task. If a trial gives a positive result, it will be rewarded by similar future trials; on the other hand, if a trial gives a negative result, it is customary to identify it with an erroneous behaviour, and thus not repeat it in the future attempts. Iterating the process, the trials should become more and more effective with time, until the task gets solved. A simple example consists in the method rats use to select food: when rats encounter food items with new look and smell, they will first eat a small amount of it. According to the flavour and the physiological effect of the food, the rats will later decide if eating more or not. If the food produces an ill effect, that food will be associated with illness, and not eaten again. If it tastes good and does not produce any negative effect on the health of the rat, it will probably be

eaten again. Also human beings use trial and error several times to learn tasks. For instance, when a person is learning how to play tennis, she will probably try to hit the ball by performing particular movements of the arms, shoulders and legs. Those movements will be identified as effective or erroneous, according to the result of the shot, and this result will affect the next attempts to hit the ball. As a last example of how much trial and error is used by humans for learning new tasks, the students that have recently attended a course of introduction to programming should agree on how many wrong attempts, with subsequent mistake identifications and adaptions, were needed before becoming able to write correct computer programs.

This learning process is what has inspired the introduction of the field of ML. But what do we exactly want machines to learn? Even though it is impossible to give general definitions to cover such a vast field as ML, it is possible to cover the large majority of the situations saying that one of the most frequent objectives of ML is the one of *learning a function*. In large part of the situations, ML is dealing with a problem that can be defined as follows. Given a set of data pairs:

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$$

the objective is to find (or approximate) a function (or relation) ϕ, such that:

$$\forall i = 1, 2, ..., n : \ \phi(\mathbf{x}_i) = y_i$$

In the most general definition, $\mathbf{x}_i$ and y_i can be *any* kind of object (numbers, vectors, matrices, expressions, images, movies, sentences, other objects from the real world, etc.), however the most typical situation is the one in which the $\mathbf{x}_i$ are m-dimensional vectors of objects of any type (including, but not necessaily, numbers), while the y_i are scalar values.

Before having a closer look at the problem of learning, it is useful to fix some terminology:

- D is called dataset;
- $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ are called input data, input vectors, instances or observations;
- $\{y_1, y_2, ..., y_n\}$ are called expected outputs, or target values;
- the sought for function ϕ, i.e. the function that perfectly matches all possible data in the input domain into the corresponding targets, is called target function;
- learning is a process that allows us to obtain a function f that approximates the target function ϕ;
- function f, i.e. the function obtained as a result of the learning process, is called *data model*, or simply *model*;
- finally, we will talk of supervised learning in case the target values $\{y_1, y_2, ..., y_n\}$ are known for each observation, and unsupervised learning otherwise. This manuscript focuses on supervised learning given that, for the used datasets, target values (survival rates of cancer patients) are known for a set of observations (cancer patients).

Last but not least, we will say that model f has a good *generalization ability* if f behaves like the target function ϕ also for data that do not belong to D. Understanding

if a model has a good generalization ability can be a hard task, because, of course, in general the target function ϕ is not known *a priori* and cannot be extrapolated by simply looking at the data. Actually, in some senses, we could even say that ϕ is not even an existing function, but more the *concept*, the *logic*, or the *underlying knowledge*, that allowed a given entity (a person, a device, etc.) to generate the data (Example 2.2 should clarify this). Furthermore, in some cases, several different functions can perfectly match the known data, and in such a situation, deciding which one is the target function is impossible, unless further data are provided. However, the concept of generalization is crucial to ML. The following examples should clarify the issue.

Example 2.1. (A "Toy" Numeric Example). Let us consider the following simple numeric dataset, where x_1 and x_2 are the components of the bi-dimensional input vectors (input variables, or features) and y is the target:

$$D = \begin{array}{|cc|c|}\hline x_1 & x_2 & y \\ \hline 1 & 8 & 9 \\ 3 & 2 & 5 \\ 4 & 1 & 5 \\ 7 & 3 & 10 \\ \hline\end{array}$$

A question arises natural: "What is the target function?". Any attempt to answer this question in a formal way can only lead to the answer "I don't know", given that one may imagine several functions matching the data in D, and no information is given on how to choose among them. However, given the simplicity of this example, one could easily hypothesize that in this case, the target function is the function that sums two numbers, in other words: $\phi(x_1,x_2) = x_1 + x_2$.

Now, let us assume that a ML system is able to find a model like:

$$f(x_1,x_2) = x_1 + x_2$$

It is obvious that now we can apply f to *any* pair of numbers, and not only the ones in D, and the result will be the sum of those numbers; for instance $f(2,6) = 8$. Let us, instead, assume that our ML system finds a model like:

$$\begin{array}{rcl} g(x_1,x_2) & = & \textbf{if } ((x_1 == 1) \,\&\, (x_2 == 8)) \textbf{ then return } 8; \\ & & \textbf{else if } ((x_1 == 3) \,\&\, (x_2 == 2)) \textbf{ then return } 5; \\ & & \textbf{else if } ((x_1 == 4) \,\&\, (x_2 == 1)) \textbf{ then return } 5; \\ & & \textbf{else if } ((x_1 == 7) \,\&\, (x_2 == 3)) \textbf{ then return } 10; \\ & & \textbf{else return } \text{a random value}; \end{array} \tag{2.1}$$

If one looks at these two models f and g, it is not difficult to convince oneself that *both* of them work perfectly on the data in D, but (still in the hypothesis that $\phi(x_1,x_2) = x_1 + x_2$ is the target function) f is a perfect approximation of ϕ

also for data that are not in D, while g has a completely different behavior: for each pair of input values that are not in D, the output of g will be identical/similar to the output of ϕ only in extremely rare and lucky cases. This is a typical situation in which we can say that f has a good generalization ability, while g has not. Furthermore, the reason why g is not able to generalize reasonably clearly seems *overfitting* (a concept that will be discussed more deeply later): g is clearly too "specialized" for the data in D, and thus lacks generality.

In general, we could say that the difference between having a good generalization ability and overfitting (which is the difference between f and g in this example) consists in the difference between learning the knowledge that is hidden in the data (and code that knowledge in the data model) and just mimicking/mocking what is written in the dataset. It is clearly the former behavior that we want our machines to have: data have to used as examples to build a knowledge that goes beyond them (that is, actually, more general). In other words, we want our ML systems to *learn by examples* and not *learn the examples*.

Example 2.2. (A "Real-Life" Example). In this example, we discuss a real-life ML application, consisting in predicting the *toxicity* of a candidate new drug, a step that is an important part of the *drug discovery* process, i.e. the process of descovering and commercializing a new drug. Studies of this application can be found in several bibliographic references, including, for instance, [Archetti et al., 2007a]. Let us consider a dataset of the following shape:

$$D = \left[\begin{array}{cccc|c} x_{11} & x_{12} & \ldots & x_{1m} & y_1 \\ x_{21} & x_{22} & \ldots & x_{2m} & y_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_{n1} & x_{n2} & \ldots & x_{nm} & y_n \end{array}\right]$$

where, for each $i = 1,2,...,n$, vector $\mathbf{x}_i = \{x_{i1}, x_{i2}, ..., x_{im}\}$ represents a molecular compound, that is candidate to become a new drug, represented by means of its *molecular descriptors*, and y_i represents the corresponding value of the toxicity of that molecular compound.

Estimating the toxicity of a molecular compound is generally a very expansive and prone to error process. It is done by feeding a sample of test animals with incremental doses of the compound. The amount of compound that was given when half of the cavies have died is one of the most used measures of toxicity, called Lethal Dose 50% (LD50). Imagine that this is the process that has been applied to create a dataset like D (for each molecular compound $i = 1,2,...,n$, 50% of the test animals have been killed in order to estimate the target value y_i!). The objective of ML is now to learn the hidden relationship (assuming it exists!) between the molecular descriptors and the value of the toxicity, codying it in a data model. In this way, whenever in the future it is needed to estimate the toxicity of a new molecular compound, we can simply apply the model to its molecular descritors, thus not

having to sacrifice any more test animals. At this point, two observations can be done:

1. The toxicity of new molecular compounds is unknown. As such, we have no way of verifying the correctness of the prediction of the model on those new data. The prediction made by the model needs to be *trusted*.

2. Generalization is not just important, it is the only thing that matters. A "model" that is able to correctly quantify the toxicity only for the compounds that are in the dataset is totally useless: we already know the toxicity of those compounds. What matters is that the model is able to generalize and return a reliable estimation for new compunds, that are not in D.

Both these observations are not peculiar of the specific problem discussed in this example, but they can be extended to practically all ML problems, including the application studied in this manuscript. The reader is particularly invited to reflect about the importance of Point 1. Pharmaceutical companies usually invest ingent amounts of money in the drug discovery process. If we want ML models to be *trusted*, we need to find a way to learn so that generalization becomes *likely*, or at least we need to be able to *test* the generalization ability of our models (this issue will be deepened in the continuation, when subjects such as data splitting and crossvalidation will be presented).

The ML process, at least in its most basic formulation, can be visualized as in Figure 2.1. The process of applying a ML system to generate a data model is called

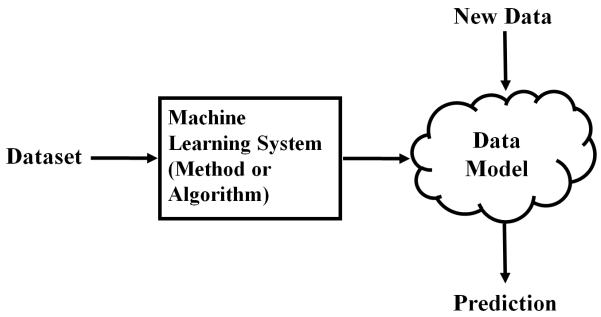


Fig. 2.1 Illustration of a simple ML process.

learning or *training* phase, while the process of applying the model on new data is called *generalization* or *prediction* phase.

The learning problem discussed so far has two significant particular cases:

- *Classification*, in which the target values $y_1, y_2, ..., y_n$ have a discrete and "limited" codomain;

- *Regression*, in which the target values $y_1, y_2, ..., y_n$ have a coninuous, or "vast", codomain[1].

In the case of classification, the target values can be interpreted as classes, or groups, and the learning problem can be interpreted as the task of partitioning data into groups. Simple example of classification problems can be, for instance: partitioning a set of images portraying faces into men and women, partitioning a text into English, French, Portuguese or Italian language, partitioning a set of numbers into small, medium and large, etc.

2.1 Methods to Test Generalization Ability

2.1.1 Data Splitting

Let

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$$

be a supervised dataset. k observations, with $k < n$, are selected and inserted into a new set J. Even though, in general, the choice of the instances to be inserted in J is made at random, let us assume for simplicity that the first k observations in D are inserted in J[2], in other words:

$$J = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_k, y_k)\}$$

Learning is performed using only the data in J. Let g be the model obtained as the result of this learning phase; the set:

$$D - J = \{(\mathbf{x}_{k+1}, y_{k+1}), (\mathbf{x}_{k+2}, y_{k+2}), ..., (\mathbf{x}_n, y_n)\}$$

can be used to test the generalization ability of g. This can be done by evaluating g on the input data $\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, ..., \mathbf{x}_n$ and comparing the calculated outputs with the respective expected outputs $y_{k+1}, y_{k+2}, ..., y_n$. Any error measure can be used to make this comparison. For instance if we assume that we use the absolute error, a measure of the generalization ability of our ML system can be:

$$E = \sum_{i=k+1}^{n} |g(\mathbf{x}_i) - y_i| \tag{2.2}$$

[1] The distinction between classification and regression can be fuzzy in some cases, and this is why the definition was deliberately based on informal terms such as "limited" and "vast".

[2] The reader is invited to convince herself that the same effect as selecting k instances at random from D can be obtained by first shuffling at random the instances in D, and then selecting the first k.

The set J is usually called *training set*, and $D-J$ is called *test set*, while we refer to the whole set of available data D using the term *dataset*. Usually, the training set is built by selecting k random instances from D with uniform probability, and the remaining instances form the test set. No rules exist for quantifying k, but it is frequent to find studies in the literature in which the training set contains (approximately) 70% of the instances in D, while the test set contains the remaining 30%.

Data splitting has the advantage of being very simple, but it has an important drawback: the results may be dependent on the particular set used for training. In other words, if we repeat the experiments with a different splitting the results may be (and in many circumstances actually are significantly) different. Avoiding any kind of logical "choice" when we select the training instances is a first advisable step to counteract this issue (for instance, it is generally not recommended to select the first k instances, in the order in which they appear in D, even in case no apparent logic exists in the order of the observations). This is why it is a general practice to select the training instances randomly. However, this is generally not enough to avoid bias given by the particular used training set. The method explained in the continuation was introduced to extend data splitting and limit this problem.

2.1.2 Crossvalidation

In the standard version of crossvaliadtion, also called *h-folds crossvalidation*, the entire dataset is partitioned into h subsets[3] and data splitting is repeated h times. At each iteration, one of the subsets is used as a test set, while the remaining $h-1$ subsets are used as a training set. At the end of the process, each one of the h partitions is used exactly once as test set. Let $\{E_1, E_2, ..., E_h\}$ the the errors obtained in each one of the iterations. A measure of the generalization ability of the ML system can be given, for instace, by:

$$\overline{E} = \frac{1}{h} \sum_{i=1}^{h} E_i \tag{2.3}$$

which corresponds to the average error obtained in the different iterations.

The advantage of the crossvalidation is that it is less dependent than a single data splitting on the particular choice of the training data. The drawback is that training has to be performed h times, so it can be executed in an amount of time that is approximately h times bigger than a single data splitting. Furthermore, in general, the variance associated to $\overline{E}$ should get smaller as h gets larger. So, to have a small variance between the results of each iteration, the number of iterations should be large, which contributes to slow down even more the process.

Last but not least, the reader is invited to reflect on the fact that at the end of the crossvalidation we have h different models, resulting from the h independent training phases. In case we need one final model, which one should be chosen? The

[3] We remind that, by definition, a partition of a set is a grouping of its elements into non-empty subsets, in such a way that every element is included in exactly one subset.

answer to this question is not unique, and several options can exist, but in general, it is typical to create the final model by combining the h obtained models. For instance, for classification problems, one may want to consider a *voting* among the h models. The final result could be the class that is given as output by the largest number of models. Analogously, for regression problems, the final output could be given by a statistic, like the average or the median, calculated on the outputs of the single models.

The crossvalidation has several variants, the most popular of which is possibly the so called *repeated random sub-sampling validation*, or *Monte Carlo cross-validation* [Dubitzky et al., 2006]. This method creates multiple random splits of the dataset into training and test set. The advantage is that the proportion of the training/test split is not dependent on the number of iterations (i.e., the number of partitions). The disadvantage is that some observations may never be selected in the test set, whereas others may be selected more than once. In other words, the test sets used in the different iterations may overlap. Another variant of the crossvalidation is the so called *leave-p-out cross-validation*, in which p observations are used as the test set and the remaining observations as the training set. This is usually repeated on all possible ways in which the original dataset can be partitioned into a test set of p observations and a training set.

2.1.3 How to Perform a Fair Experimental Study

Data splitting, crossvalidation and their existing variants are generally used to assess the generalization ability of a ML system. The test set is used to simulate new, unseen, data and thus one golden rule exists: it is *strictly prohibited* to use the test set (or any part of it) in any process different from the final testing of the model. In other words, we should behave as if the data belonging to the test set are unknown in the moment in which we train the system. Only when we have a final model, those data can be used for testing its generalization ability.

However, very often, ML algorithms depend on a number of parameters (often called hyper-parameters), and in order to find appropriate values for those parameters, a preliminary experimental tuning phase is often necessary. When this is the case, how should we organize and use our data? Hyper-parameter tuning has to be considered a part of the training process. As such, the previous rule applies also to this phase: it is strictly forbidden to use the test set for optimizing parameters. For instance:

> *training the system with different parameter configurations, and then using the configuration that has returned the best results on testing data is a* highly unfair *and* completely wrong *practice!*

On the other hand, one may argue that using the configuration that has obtained the best results on the test set may be misleading, because that configuration may be overfitting training data. So, what to do? Many possible answers to this question exist, but roughly all of them envisage at least the following steps:

- Partition the dataset D into a *learning set* J and a *test set* $D-J$;

- "Forget" the test set, until a final model is obtained, with the appropriate parameter setting;

- Further partition the learning set into at least one *training set* U and at least one *validation set* $J-U$[4]. Use the training set to generate models using different parameter settings, and use the validation set to select the preferred one.

When the most appropriate parameter setting has been found, it is customary to train again the ML system using the whole learning set, and assess its generalization ability using the test set. However, this last step is optional, and in some cases a model that has been generated using only the training set U is used.

As it is not hard to understand, even though the previous data partitionings (i.e. the splitting of the dataset D into a learning set and a test set, and the further splitting of the learning set into a training and a validation set) are done at random, the results are generally dependent on the particular partitions used. For this reason, the previous schema should be iterated several times, using each time different sets of data for the different tasks. Typical cases are:

- *Nested Data Split, or* $(k*\ell)$*-Fold Monte Carlo Crossvalidation*. The method is characterized by two nested loops. The internal loop optimizes the hyperparameters and finds the most appropriate configuration. In the external loop, that configuration is used to generate and test the final model. Let k be the number of iterations of the external loop, and ℓ the number of iterations of the internal loop. The method is shown in Algorithm 1.

- $(k*\ell)$*-Fold Crossvalidation.* This is a nested variant of the crossvalidation. Analogously to the previous method, it contains an outer loop of k folds and an inner loop of ℓ folds. The whole available dataset is partitioned into k subsets. One by one, a set is selected as test set and the $k-1$ other sets are combined into the corresponding learning set. This is repeated for each of the k sets. At each iteration, each learning set is further sub-divided into ℓ subsets. One by one, a set is selected as validation set and the $\ell-1$ other sets are combined into the corresponding training set. This is repeated for each of the ℓ sets. As previously, the training sets are used to fit model parameters, while the validation sets are used

[4] The reader has to be aware of the fact that there is no common agreement in the literature about this terminology. For instance, it is not infrequent to find references in which the terms test set and validation set are exchanged compared to the terminology used here (in other words, some references call validation set the set $D-J$ and test set the set $J-U$). Analogously, in some references the term learning set is not used. In those cases, often the set J is identified using the term training set, or outer training set, while different terminologies can be used to identify the set U, including the expression inner training set. Given that no precise convention exists, none of these terminologies can be considered "wrong", as well as none of them can be considered as "the correct one". However, the terminology adopted in these pages can be considered clearer than others. For instance, it has the advantage to always identify with the term training set the set of data on which the ML system is executed, with the objective of generating a model.

Algorithm 1: Pseudo-code for the nested data split, or $(k * \ell)$-random folds crossvalidation.

1. **for** $i = 1, 2, ..., k$ **do**
   - 1.1. partition the dataset D into a learning set J_i and a test set $D - J_i$;
   - 1.2. **for** $j = 1, 2, .., \ell$ **do**
     - 1.2.1. partition the learning set J_i into a training set U_{ij} and a validation set $J_i - U_{ij}$;
     - 1.2.2. **for each** parameter setting that needs to be compared **do**
       - train the ML system on U_{ij};
       - collect the results obtained on $J_i - U_{ij}$;

     end

     end
   - 1.3. Select the configuration c_i that returns the best average result, calculated on all the validation sets $J_i - U_{i1}$, $J_i - U_{i2}$, ..., $J_i - U_{i\ell}$;
   - 1.4. Train the ML system on the whole learning set J_i, using configuration c_i. Let f_i be the obtained model, whose generalization ability can be assessed using the test set $D - J_i$;

   end
2. Aggregate models $f_1, f_2, ..., f_k$ (for instance performing a voting for classification problems or an average for regression), in order to obtain the final model f.

to provide an unbiased evaluation of the models on unseen data. The configuration that obtained the best average results on the validation sets is fit on the entire learning set. The performance of these models is then evaluated using the test sets. Analogously to the difference between the traditional crossvalidation and the Monte Carlo crossvalidation, the difference between the $(k * \ell)$-fold crossvalidation and the $(k * \ell)$-fold Monte Carlo crossvalidation is that in the $(k * \ell)$-fold Monte Carlo crossvalidation some observations may never be selected in the test and/or validation sets, whereas others may be selected more than once. On the other hand, contrarily to the $(k * \ell)$-fold crossvalidation, in the $(k * \ell)$-fold Monte Carlo crossvalidation the proportion of the different data splits is not dependent on k and ℓ.

- *k-Fold Crossvalidation with Validation and Test set*. This method can be seen as a $k * 1$-fold crossvalidation. The whole dataset is partitioned into k subsets. One by one, a set is selected as test set. Then, one by one, one of the remaining sets are used as a validation set and the other $k - 2$ sets, joined, are used as a training set. This is repeated for all possible combinations. As for the previous methods, the training set is used for model fitting and the validation set is used for model evaluation for each of the parameter settings. Finally, for the configuration that obtained the best average result on the validation sets, the test set is used to assess the generalization ability. Two variants are possible: either evaluating the model that was trained on the training set or evaluating a new model that was fit on the combination of the training and the validation set.

2.2 Measures of Performance of a Classifier

The discussion of the previous section is general, in the sense that it holds both for classification and regression problems. In this section, we focus on classification. The simplest and most popular measure to quantify the *error* made by a classification model is the *number of correctly classified instances*. To calculate this measure, we simply have to count the number of instances for which the predicted class label is identical to the class label that appears in the supervised dataset. The number of correctly classified instances is a measure that depends on the number of instances. In order to make the measure comparable when used on datasets of different sizes, it is typical to normalize this measure, so as to obatin another measure called *accuracy*, defined as:

$$\text{Accuracy} = \frac{\text{number of correctly classified instances}}{\text{total number of instances}}$$

Using a measure like the number of correctly classified instances, or the accuracy, can be not sufficient to understand the behavior of a classifier. Often, more sophisticated measures of performance are needed for classifiers. For instance, we may need measures that express the quality of a classification *for each class*, and not just one single general number. To convince oneself about the importance of having a measure that expresses a different value for each class, one may imagine, for instance, a binary classification problem, where the objective is to categorize a set of patients into one of the two possible classes: *healthy* or *sick*. It is clear that, in some cases, misclassifying a sick patient can have more serious consequences than misclassifying a healthy patient. Starting by saying that both misclassifications are mistakes, and, as such, both of them have serious consequences, treating a healthy patient as a sick one, among other consequences, may cause the patient to undertake unnecessary treatments or to get uselessly worried. On the other hand, treating a sick patient as a healthy one may cause the patient to not undertake treatments that may be crucial to save her life. In such an application, it is clear that an information like the number of misclassifications made by the system is not sufficient. On what class those missclassifications happened is also a needed information.

Two of the most known and employed measures to quantify the performance of a classifier on the single existing classes are *precision* and *recall*. Given a class C in which data can be partitioned, these measures are defined as follows:

$$\text{Precision}(C) = \frac{\#\text{instances belonging to } C\text{, classified as } C}{\#\text{instances classified as } C} \tag{2.4}$$

$$\text{Recall}(C) = \frac{\#\text{instances belonging to } C\text{, classified as } C}{\#\text{instances belonging to } C} \tag{2.5}$$

As we can see, the measures share the same numerator, consisting in the intersection between the observations labelled as C in the given dataset and the observations that the model has categorized as belonging to class C (in other words, the numer-

ator contains the number of instances that the classifier has categorized correctly as class C). The two measures differ because of the denominator: for precision, the denominator tells us about the work made by the classifier, while for recall, it tells us about the ground truth. Under this perspective, one may have an intuition on the meaning of precision and recall by comparing them to concepts such as correctness and completeness, respectively.

To have a better understanding on how to calculate precision and recall, consider the example shown in Figure 2.2. In this example, three classes are given: C_1, C_2

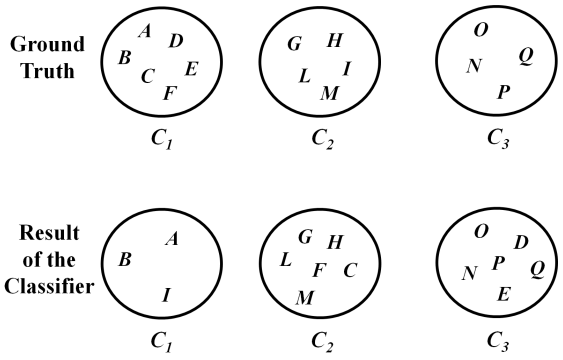


Fig. 2.2 Example used to explain the concepts of precision and recall. The upper line shows how data are really partitioned into three classes C_1, C_2 and C_3 in the given dataset. The lower line shows how a classification model has categorized the same objects in the same classes.

and C_3. The upper line of the figure shows how 15 objects (observations) are partitioned into these three classes in the given dataset. The lower line, instead, shows how those objects have been categorized by a ML model. Precision and recall for the different classes are:

$$\text{Precision}(C_1) = \frac{2}{3} \approx 0.66, \quad \text{Recall}(C_1) = \frac{2}{6} \approx 0.33$$

$$\text{Precision}(C_2) = \frac{4}{6} \approx 0.66, \quad \text{Recall}(C_2) = \frac{4}{5} = 0.8$$

$$\text{Precision}(C_3) = \frac{4}{6} \approx 0.66, \quad \text{Recall}(C_3) = \frac{4}{4} = 1$$

Both precision and recall are numbers included in $[0,1]$, where 1 represents the best value, while 0 is the worst one. An ideal model, i.e. the one that classifies each observation correctly, has both precision and recall equal to 1. Models that have only one among precision and recall is equal to 1 deserve a discussion. One may be tempted to consider such models as good ones, but this can be very misleading. Consider, for instance, the case of C_3 in the previous example. The recall of C_3 is equal to 1 be-

cause the classifier has categorized as C_3 all the instances that are really in C_3, plus others. If we extremize this situation, even a "naive" model that blindly categorizes all existing observations in C_3 can have a recall equal to 1. However, this model has clearly not learned anything. A similar argument also holds for precision: any model that categorizes in a class only a subset of the objects that actually belong to that class is equal to 1. But this argument also holds if many objects belong to the class and only one is categorized in it. For instance, given the objects A,B,C,D,E,F that belong to class C_1, a model that categorizes object A as the only member of class C_1 has a precision equal to 1. However, also in this case, the amount of information that this model has learned is poor. In conclusion, it does not make much sense to study only one among precision and recall, without studying the other. These measures give us two different pieces of information, each of which is incomplete without the other. Only studying them together makes sense.

Other popular measures of performance of a classifier are *true positives* (TP), *true negatives* (TN), *false positives* (FP) and *false negatives* (FN). Given a class C, these measures are defined as follows:

- $\text{TP}(C) = \#$ instances belonging to C, classified as C
- $\text{TN}(C) = \#$ instances that do *not* belong to C, that have *not* been classified as C
- $\text{FP}(C) = \#$ instances that do *not* belong to C, classified as C
- $\text{FN}(C) = \#$ instances belonging to C, that have *not* been classified as C

When the first word is *true*, the measure quantifies a correct behavior of the model: true positives quantify the number of times an object of C has been correctly categorized, while true negatives quantify the number of times that an object has been correctly identified as not belonging to C. Analogously, when the first word is *false*, the measure quantifies errors of the model: false positives count the number of times the model has categorized an object in C erroneously, while false negatives quantify the number of objects that have not been classified in C, but they were supposed to.

Knowing the values of TP, FP and FN, it is possible to immediately obtain the precision and the recall. In fact, directly from the definition of precision and recall, we have that, for each class C:

$$\text{Precision}(C) = \frac{\text{TP}(C)}{\text{TP}(C) + \text{FP}(C)}$$

$$\text{Recall}(C) = \frac{\text{TP}(C)}{\text{TP}(C) + \text{FN}(C)}$$

Another important measure that joins precision and recall into one single number for each class is the *F-measure* (also called *F-score* or *F1-score*), defined as:

$$\text{F-measure}(C) = \frac{2 * \text{Precision}(C) * \text{Recall}(C)}{\text{Precision}(C) + \text{Recall}(C)}$$

The F-measure is often preferred over the accuracy in case of *unbalanced datasets*. In fact, let us consider, for instance, a binary classification problem, i.e. a problem that consists in categorizing observations into one of the two possible classes C_1 and C_2. Let us assume, without loss of generality, that, in our dataset, numeous observations labelled with class C_1 are available, while only a negligible number of observations are labelled with class C_2. It is clear that a "naive" model, that blindly categorizes all observations as belonging to class C_1 has an excellent accuracy. If the ML system is guided by accuracy, as a performance measure to choose among the candidate solutions, it is clear that such a model is likely to be the preferred one, even though it has learned none of the information available in the data. On the other hand, given that such a model has poor precision and recall on class C_2, its F-measure is also poor. In order to have a good F-measure, also in case of unbalanced datasets, the ML system is forced to learn the information that allows us to distinguish between the different classes.

In some cases, it is useful to understand how better or worse a classifier is, compared to a random classifier, where by random classifier, here, it is meant an algorithm that, for each possible instance, always returns a random class, picked up with uniform distribution among all the possible existing alternatives. This can be quantified, for instance, by the *K Statistic* or *K measure*, that some ML packages have implemented, including Weka [Hall et al., 2009]. This measure is defined as:

$$K = \frac{\text{Accuracy } - P(E)}{1 - P(E)}$$

where $P(E)$ is the probability of the random classifier to correctly classify all elements in the considered dataset. It is clear that K gets closer to the ideal value of 1 as much as also the accuracy gets closer to its ideal value of 1. When stratifying the results class by class is not a requirement, the K measure can give some interesting information, that may be integrated with the information given by the accuracy, and/or with statistics calculated over other measures such as precision, recall and F-measure.

We conclude this presentation of measures of performance of classifiers with the discussion of a measure that is very popular, but can be used only for binary classification, and only in case the classifier works with a *threshold* mechanism. Imagine, for instance, the two classes of a binary classification problem to be represented by labels 0 and 1. Given an observation, the ML model could work by generating a number x, that is then transformed into either 0 or 1. The typical case is: if x is smaller than 0.5, the return 0, else return 1. In this case, a threshold equal to 0.5 is used. This is the typical functioning, for instance, of supervised artificial neural networks. In such a situation, the performance of the model can be represented by a plot, called *Receiver Operating Characteristic* (ROC) curve. The plot is created by reporting the values of the true positive rate (TPR) against the false positive rate (FPR) for various different values of the threshold.

TPR and FPR are defined as follows:

$$TPR = \frac{TP}{TP+FN}, \quad FPR = \frac{FP}{FP+TN}$$

TPR is identical to recall, and it is also known as *sensitivity* or *probability of detection*. FPR is also known as *fall-out* or *probability of false alarm*. In some references, it is also possible to find the term *specificity*, where:

$$\text{specificity } = 1 - FPR$$

Let us consider the frequent case in which the output of the model is a number in $[0,1]$. In this case, to combine the FPR and the TPR into a single metric, we first compute the two former measures with a set of different threshold values (like, for instance, $0.00, 0.01, 0.02, ..., 1.00$). Then we plot them on a single graph, with the FPR values on the abscissa and the TPR values on the ordinate. The resulting curve is the ROC curve, and the metric we consider is the area under the curve (AUC), also called AUROC. An example of a ROC curve is reported in Figure 2.3. In this

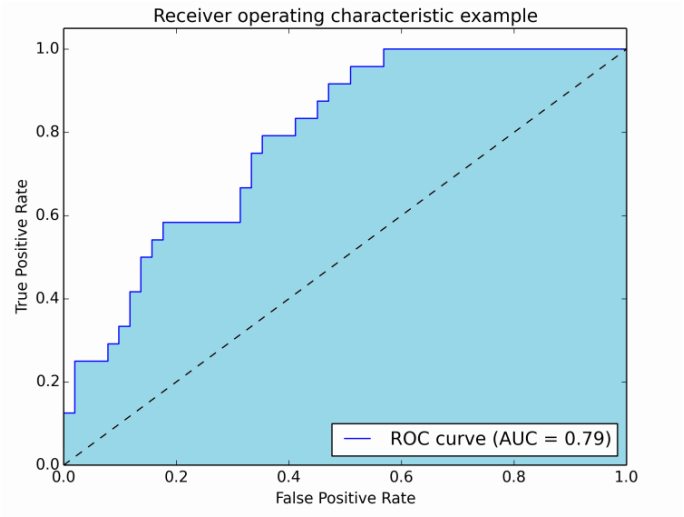


Fig. 2.3 Example of a ROC curve. The AUROC is represented in light blue.

figure, the blue area corresponds to the AUROC. The dashed line is the diagonal; it represents the ROC curve of a random predictor, and it has an AUROC equal to 0.5. The random predictor is commonly used as a baseline to compare with the model. The value of the AUROC is always included in $[0,1]$. The best possible prediction method would yield a point in the upper left corner (coordinate $(0,1)$) of the ROC space, representing 100% specificity (i.e. no false positives). The point $(0,1)$ is also called a perfect classification. A completely random guess would give a point along the diagonal (the so-called line of no-discrimination) from the left bottom to the top right corner.

2.3 Features

A feature is a characteristic of the objects that have to be classified or, more generally, for which a prediction is needed. So, datasets are usually a collection of values (or instances) of features. Given a dataset of the form:

$$D = \left[\begin{array}{cccc|c} x_{11} & x_{12} & \ldots & x_{1m} & y_1 \\ x_{21} & x_{22} & \ldots & x_{2m} & y_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_{n1} & x_{n2} & \ldots & x_{nm} & y_n \end{array}\right]$$

We normally use the following termnilogy:

- For each $j = 1,2,...,m$, column $\{x_{1j},x_{2j},...,x_{nj}\}$ represents a feature, and for each $i = 1,2,...,n$, element x_{ij} is a feature value, or feature instance.
- For each $i = 1,2,...,n$, line $\{x_{i1},x_{i2},...,x_{im}\}$ is a dataset instance, observation or sample, and y_i is the corresponding target value.

In case of classification, features are appropriate or useful if they allow us to make a difference between one class (or more) and the others. This is why an appropriate choice of the features is often crucial in supervised ML. Let us consider, for instance, the following toy dataset, whose objective is to classifiy animals into roosters and dogs:

	# paws	# eyes	having a crest	body fat	blood pressure	target
animal 1	2	2	True	7%	97	Rooster
animal 2	4	2	False	18%	118	Dog
animal 3	4	2	False	22%	126	Dog
animal 4	2	2	True	10%	101	Rooster

This dataset contains four observations, each one representing a different animal. Each animal is represented by five features: *number of paws* and *number of eyes*, which are integer numbers, *having or not having the crest*, a Boolean value, *body fay rate*, which is a percentage, and *blood pressure*, which is a floating point number. Observing this dataset, we can immediately notice that:

- Number of paws and having/not having the crest are *good* features: they clearly allows us to tell dogs from roosters.
- number of eyes is a totally useless feature: its value is the same for both classes, and so the feature is constant in the whole dataset.
- Body fat rate and blood pressure may help to make the classification, but if we use these two features, the classification may be harder than if we simply use one among number of paws and having/not having the crest.

Examples of models that allow us to make a perfect classification for each instance in the dataset are:

if (having crest) **then** Rooster **else** Dog

or:

if (number of paws == 2)
    then Rooster
    else if (number of paws == 4)
        then Dog

Both these models use a restricted number of features, compared to the total number of features that appear in the dataset. Removing several features from the dataset, possibly leaving only number of paws and/or having/not having a crest, may significantly help the work of a classifier. The presence of useless features, or of features which make the classification harder, in fact, increments the search space and makes the model's optimization harder.

2.3.1 Feature Engineering

Feature selection is the process of choosing the features that are useful to make the prediction, disregarding all the others. It is often a very hard and complex task, and in can, in principle, be based on previous knowledge of the problem, or on mathematical relationships between data. *Feature extraction* is the process of combining one or more existing features to create a smaller number of more insightful features. Contrarily to feature selection, in feature extraction features are typically not chosen or disregarded, but only combined. Feature selection and feature extraction can both be used, or only one of them can be used. Reducing the dimensionality of the feature space can be a crucial task to improve the generalization ability of a ML system, so choosing or creating the appropriate features is a fundamental step from which the performance of the whole system can depend. They are usually applied before beginning the learning process, and for this reason, they are usually integrated in a so called *data preprocessing* phase, a phase that usually contains also a step of data cleaning, aimed at removing mistakes, imperfection or noise from the data.

Modern datasets have hundreds to tens of thousands of variables or features. Feature selection and feature extraction have three main objectives:

- improving the prediction performance of the models,
- providing faster and more cost-effective predictors, and
- providing a better understanding of the underlying process that generated the data.

Besides this, there are many other potential benefits of feature selection/extraction: facilitating data visualization and data understanding, reducing the measurement and storage requirements, reducing training and utilization times, etc.

Methods for feature selection can essentially be partitioned into:

- Filters;
- Wrappers;
- Embedded methods.

Wrappers utilize the learning machine of interest as a black box to score subsets of variable according to their predictive power. Filters select subsets of variables independently of the chosen predictor. Embedded methods perform variable selection in the process of training and are usually specific to given learning machines (Genetic Programming is one of these methods).

The most popular kinds of filters (although by far not the only ones known) are:

- Correlation based methods;
- Information Theory based methods;

Both these methods have the objective of ranking the features according to their "usefulnes" in helping prediction, so that only the k top-ranked ones can be used for generating the predictive model. The intuition is that if a feature is independent from the target, it is uninformative for predicting it. Of course, these methods introduce a new parameter k, that can have a crucial influence on the performance of the system, and that can only be set by means of experimental comparisons.

The idea of correlation-based feature selection is simple: calculate the correlation between all features and the target, and then rank the features according to this correlation value. One of the most known measures is the Pearson correlation coefficient. For a particular feature, given the vector of all the feature values $\mathbf{x} = x_1, x_2, ..., x_n$ and the vector of the target values $\mathbf{y} = y_1, y_2, ..., y_n$, the Pearson correlation between X and Y is:

$$Corr = \frac{cov(\mathbf{x}, \mathbf{y})}{\sqrt{var(\mathbf{x})\ var(\mathbf{y})}}$$

where *cov* is the covariance of two vectors and *var* is the variance of one vector, so:

$$Corr = \frac{\sum_{i=1}^{n}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \cdot \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $\bar{x}$ is the average of the elements of vector $\mathbf{x}$. By definition, *Corr* is a value in $[-1, 1]$. Usually, the measure that is used to perform the ranking is $Corr^2$, because a negative correlation can be useful (it is enough to consider the feature with a negative sign in the model). One possible drawback of correlation criteria is that they can only detect linear dependencies between features and target. A simple way of lifting this restriction is to make a non-linear fit of the target with single variables and rank according to the goodness of that fit.

Concerning information theory-based feature selection, the ranking of features is done using mutual information between features and the target:

$$Inf = \sum_{x_i} \sum_{y_i} P(X = x_i, Y = y_i) \cdot log \frac{P(X = x_i, Y = y_i)}{P(X = x_i) \cdot P(Y = y_i)}$$

This measure is appropriate in case the features are discrete variables. The case of continuous variables (and possibly continuous targets) is harder and one can consider discretizing the variables.

Besides correlation and information theory, another possible measures to rank the features is the χ^2 between features and targets, which also aims at quantifying the depencence between features and target.

One common criticism of variable ranking is that it may lead to the selection of a redundant subset. The same performance could possibly be achieved with a smaller subset of complementary variables. Still, one may wonder whether adding presumably redundant variables can result in a performance gain. Actually, it is an experimental fact that, in classification, better class separation may be obtained by adding variables that are presumably redundant. More precisely, perfectly correlated variables are truly redundant in the sense that no additional information is gained by adding them; but very high variable correlation (or anti-correlation) does not mean absence of variable complementarity. Furthermore, experimental evidence tells us that a variable that is completely useless by itself can provide a significant performance improvement when taken with others, and two variables that are useless by themselves can be useful together. These two last observations lead the scientific community to the idea that filters can have important limitations, and they can be overcome by means, for instance, of wrappers or the use of embedded methods.

The ML process reported in Figure 2.1 can be extended including data preprocessing, leading to the more complete scheme represented in Figure 2.4. As we

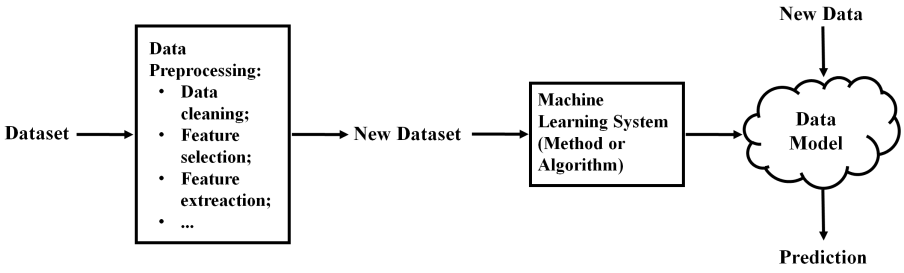


Fig. 2.4 Extension of the schema of Figure 2.1, to include data preprocessing.

can see, the objective of data preprocessing is usually the one of generating a new

dataset, that is generally smaller and possibly more informative, than the original one. This step is often crucial to facilitate the work to the ML system and often allows us to generate better models.

Chapter 3
Materials and Methods

3.1 Material

The data used in the work presented in this manuscript is the NKI breast cancer dataset [van de Vijver et al., 2002], providing gene expression and survival data for 295 consecutive breast carcinoma patients. Of all the observations available in the dataset, only the expression data for the genes included in the "70-gene" signature [van 't Veer et al., 2002] was considered.

Both survival and gene expression data were transformed into binary form. For the survival data, the outcome was defined as the survival status of the patient at time $t_{end} = 10.3$ years. By choosing this particular endpoint the number of dead and alive patients were balanced: out of 148 patients for which the status at t_{end} is known, 74 were dead and 74 were alive. Binary expression data were obtained by replacing all positive logarithmic fold changes in the original dataset with 1 and all negative and missing ones with 0.

The resulting dataset is a matrix $H = [H_{(i,j)}]$ of binary values composed by 148 rows (instances) and 71 columns (features), where each line i represents the gene signature of a patient whose binary target ($0 =$ survived after t_{end} years, $1 =$ dead for breast cancer before t_{end} years) has been placed at position $H_{(i,71)}$. In this way, the last column of matrix H represents all the known target values. Our task is now to generate a mapping F such that $F(H_{(i,1)}, H_{(i,2)}, ..., H_{(i,70)}) = H_{(i,71)}$ for each line i in the dataset. Of course, we also want F to have a good generalization ability, i.e. to be able to assess the target value for *new* patients, that have not been used in the training phase. For this reason, we used a set of machine learning techniques, as discussed in the next section. To compare the predictive power of the computational methods, we performed 50 independent choices of learning and test set, the learning set including 70% of the patients, chosen randomly with uniform distribution, and the test set consisting in the remaining 30%. The various prediction methods were then run on these datasets, so that the choice of training and testing sets in each run was the same for all methods. Parameter setting was obtained by further partitioning the learning set into 50 independent choices of a training set, formed by 70% of the

observations in the learning set chosen randomly with uniform distribution, and a validation set, consisting in the remaining 30% of the observations of the learning set. In other words, $(k*\ell)$-random folds crossvalidation presented in Section 2.1.3, and detailed in Algorithm 1 was employed.

3.2 Methods

The studied machine learning methods are described here, with references to more detailed expositions.

Genetic Programming

Genetic Programming (GP) [Koza, 1992, Poli et al., 2008, Vanneschi, 2004] is an evolutionary approach which extends Genetic Algorithms (GAs) [Holland, 1975, Goldberg, 1989] to the space of programs. Like any other evolutionary algorithm, GP works by defining a goal in the form of a quality criterion (or *fitness*) and then using this criterion to *evolve* a set (also called population) of solution candidates (also called individuals) by mimic the basic principles of Darwin's theory of evolution [Darwin, 1859]. The most common version of GP, and also the one used here, considers individuals as *abstract syntax tree* structures[1] that can be built recursively from a set of function symbols $\mathscr{F} = \{f_1, f_2, \ldots, f_n\}$ (used to label internal tree nodes) and a set of terminal symbols $\mathscr{T} = \{t_1, t_2, \ldots, t_m\}$ (used to label tree leaves). GP breeds these solutions to solve problems by executing an iterative process involving the probabilistic selection of the fittest solutions and their variation by means of a set of genetic operators, usually crossover and mutation.

We used a tree-based GP configuration inspired by boolean problems introduced in [Koza, 1992]: each feature in the dataset was represented as a boolean value and thus our set of terminals $\mathscr{T}$ was composed by 70 boolean variables (i.e. one for each feature of our dataset). Potential solutions (GP individuals) were built using the set of boolean functions $\mathscr{F} = \{AND, OR, NOT\}$. The fitness function is the number of incorrectly classified instances, which turns the problem into a minimization one (lower values are better)[2].

Finally no explicit feature selection strategy was employed, since we want to point out GP's ability to automatically perform an implicit feature selection. The mechanism allowing GP to perform feature selection, already pointed out for instance in [Archetti et al., 2006, Archetti et al., 2007b, Archetti et al., 2007c, Rosskopf et al., 2007], is simple: GP searches over the space of all boolean expres-

[1] Traditionally represented in Lisp notation.

[2] We are aware that, in case of minimization problems, the term "fitness" might be inappropriate, given that a fitness is usually a measure that has to be maximized. Nevertheless, we chose to use this term for simplicity.

sions of 70 variables. This search space includes the expressions that use *all* the 70 variables, but also the ones that use a *smaller* number of variables. In principle there is no reason why an expression using a smaller number of variables could not have a better fitness value than an expression using all the 70 variables. If expressions using smaller number of variables have a better fitness, they survive into the population, given that fitness is the only principle used by GP for selecting genes. If it happens that GP finds expressions using a small number of variables with a better fitness value than the ones using all variables, the former expressions survive into the population, while the latter ones are extinguished.

The parameters used in our GP experiments are reported in Table 4.1, together with the parameters used by the other machine learning methods we studied. GPLab, a public domain GP system implemented in MatLab, was used (for the GPLab software and documentation, see [Silva, 2007]).

Support Vector Machines

Support Vector Machines (SVM) are a set of related supervised learning methods used for classification and regression. They were originally introduced in [Vapnik, 1998]. Their aim is to devise a computationally efficient way of identifying separating hyperplanes in a high dimensional feature space. In particular, the method seeks separating hyperplanes maximizing the margin between sets of data. This should ensure a good generalization ability of the method, under the hypothesis of consistent target function between training and testing data. To calculate the margin between data belonging to two different classes, two parallel hyperplanes are constructed, one on each side of the separating hyperplane, which are "pushed up against" the two data sets. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the neighboring data points of both classes, since in general the larger the margin the lower the generalization error of the classifier. The parameters of the maximum-margin hyperplane are derived by solving large quadratic programming (QP) optimization problems. There exist several specialized algorithms for quickly solving these problems that arise from SVMs, mostly reliant on heuristics for breaking the problem down into smaller, more manageable chunks. In this work we used the implementation of John Platt's [Platt, 1998] sequential minimal optimization (SMO) algorithm for training the support vector classifier. SMO works by breaking the large QP problem into a series of smaller 2-dimensional sub-problems that may be solved analytically, eliminating the need for numerical optimization algorithms such as conjugate gradient methods. The implementation we used is the one contained in the Weka public domain software [**?**]. This implementation globally replaces all missing values and transforms nominal attributes into binary ones. It also normalizes all attributes by default (in that case the coefficients in the output are based on the normalized data, not the original data and this is important for interpreting the classifier).

The main parameter values used in this work are reported in Table 4.1 [Platt, 1998]. Being aware that in several application domains, SVM have been shown to outperform competing techniques by using nonlinear kernels, which implicitly map the instances to very high (even infinite) dimensional spaces, we used polynomials kernels with degrees 1, 2, and 3.

Multilayered Perceptron

Multilayered Perceptron is a feed-forward artificial neural network model [S. Haykin, 1999]. It is a modification of the standard linear perceptron in that it uses three or more layers of neurons (nodes) with nonlinear activation functions, and is more powerful than simple perceptron in that it can distinguish data that are not linearly separable, or separable by a hyperplane. It consists of an input and an output layer with one or more hidden layers of nonlinearly-activating nodes. Each node in one layer connects with a certain weight to every other node in the following layer. The implementation we have adopted is the one included in the Weka software distribution [**?**]. We used the Back-propagation learning algorithm [S. Haykin, 1999] and the values used for all the parameters are reported in Table 4.1. The used parameter values are reported in Table 4.1.

Random Forests

Random Forests denotes an improved Classification and Regression Trees method [Breiman et al., 1984]. It works by creating a large number of classification trees or regression trees. Every tree is built using a deterministic algorithm and the trees are different owing to two factors. First, at each node, a best split is chosen from a random subset of the predictors rather than from all of them. Secondly, every tree is built using a bootstrap sample of the observations. The out-of-bag data, approximately one-third of the observations, are then used to estimate the prediction accuracy. Unlike other tree algorithms, no pruning or trimming of the fully grown tree is involved. In this work we use the Breiman model presented in [Breiman, 2001] and implemented in the Weka software [Hall et al., 2009]. As it can be seen from Table 4.1, this method, compared to the other ones, has the advantage of a smaller amount of parameter setting required. In order to allow a fair comparison with GP, we have considered random forests composed by 2500 trees (given that the GP population is composed by 500 trees and it runs for 5 generations, 2500 trees are globally inspected by GP too) and such that each node corresponds to exactly one feature (as it is for GP). All the other parameters are reported in Table 4.1.

Chapter 4
Experimental Study

4.1 Predictive Power of Machine Learning Methods

Table 4.2 summarizes the results returned by each machine learning method on 50 independent runs. The first line indicates the different methods, the second line shows the best (i.e. lowest) value of the incorrectly classified instances obtained on the test set over the 50 runs, and the third line reports the mean performance of each group of 50 runs on their test sets, together with the corresponding standard error of mean (SEM). As Table 4.2 clearly shows, the best solutions were found by GP and Multilayered Perceptrons and the best average result was found by GP. Moreover, statistical analysis indicates that GP consistently outperforms the other methods except SVM using polynomial kernel with degree 2. In fact, as it can be seen in Table 4.3, the difference between the various average results is statistically significant (P-value 3.05×10^{-5} for ANOVA test on the 4 samples of solutions found by each method). Finally, pairwise 2-tailed Student t-tests comparing GP with each other method demonstrate its general better performance. These statistical tests were performed since there was no evidence of deviation from normality or unequal variances.

The solutions found by GP typically use a rather small number of features (i.e. terminals). In fact, the solutions of the 50 GP runs are functions of a number of terminal that ranges from 1 to 23, with a median value of 4, and first and third quartiles of 2 and 7 respectively. Few of these features tend to recur in several solution as it can be seen in Table 4.6, where the gene symbol, the gene name of each feature, together with the number of solutions where the feature occurs are shown.

4.2 Comparison with the Scoring Method

The authors of Ref. [van de Vijver et al., 2002] used the seventy-gene signature by computing the correlation coefficient between the expression profile of the patient

(limited to the 70 genes of the signature) and a previously computed typical expression profile of a good prognosis patient. To compare the performance of the various machine learning algorithms with this scoring system, the following process was implemented:

- prognostic score s of the patients (excluding the ones used to train the signature in [van 't Veer et al., 2002]) was obtained from the Supplementary Material of [van de Vijver et al., 2002], and classified as good prognosis the patients with $s > 0.4$ and as bad prognosis the ones with s$\leq$ 0.4. This is the cutoff used in [van de Vijver et al., 2002].
- 50 random lists of 44 patients were generated from this set, to match the statistic used for machine learning techniques, and computed for each list the number of false predictions given by the scoring method.

The mean number of false predictions was 16.24, with a SEM of 0.37. Therefore the scoring method appears to be superior to all machine learning algorithm other than GP, and slightly superior to GP. The difference between the performances of GP and the scoring method are not statistically significant ($P = 0.49$, 2-tailed Student t-test).

4.3 The Role of Feature Selection

To determine to what extent feature selection is responsible for the good performance of GP, we identified the 10 features most often selected by GP among the 70 initial features and ran again both GP and SVM with quadratic kernel using only these features. Remarkably, the performance of *both* methods significantly improved: for GP, the number of incorrectly identified features decreased from 16.40 (SEM 0.30) to 12.86 (0.40); for the SVM it went from 16.76 (0.18) to 14.96 (0.41). Using this preliminary round of feature selection the performance of GP becomes significantly better than both SVM and the original scoring method.

These results suggest, on one hand, that the feature selection performed by GP has intrinsic value, not necessarily tied to the use of syntax trees, since the SVM can take advantage of the feature selection performed by GP to improve its performance. Second, that a recursive use of GP, in which a first run is used to select the best features to be used in a second run, might be a promising way of optimizing the method.

4.4 Performance on Unbalanced Datasets

To check whether the performance of the GP is tied to the choice of a balanced dataset, the analysis was repeated using different time cutoffs (5 and 7.5 years) and the performance of GP was compared with the SVM using polynomial kernel with

degree 2, which was the best performing method after GP in the balanced dataset. At 7.5 years there is again no significant difference between the performance of the two methods. However, at 5 years GP performs significantly better than the SVM ($P = 6.46 \times 10^{-6}$ from two-sided t-test). We conclude that the balancing of the dataset is not crucial to obtain a good performance from GP.

4.5 Performance on an Independent Dataset

An important feature of any predictor based on gene expression data is its robustness with respect to the choice of dataset, since gene expression data from cancer patients come from studies using different protocols and/or microarray platforms. Thus, the best predictors found by GP in each of the 50 runs were applied to an independent breast cancer dataset [Miller et al., 2005], obtained on a different microarray platform. Due to the difference in gene content between platforms, only 17 of the 50 best solutions found by GP could be applied to the new dataset. All of them showed statistically significant predictive power (P-values between 7.6×10^{-3} and 2.9×10^{-4} from Fisher exact test). Since this result was obtained with no further training, it shows the robustness of the solutions obtained by GP with respect to the choice of dataset and microarray platform.

4.6 Assessment of Sensitivity

When using gene signatures to predict the survival of a cohort of breast cancer patients, one of the main goal in clinical applications is to minimize the number of false negative predictions. Table 4.4 summarizes the false negative predictions returned by each machine learning method on the 50 runs. The first line indicates the different methods, while the second and the third lines show the best (i.e. lowest) and mean performances (together with the corresponding SEM) values of incorrectly classified instances.

The best solutions were found by GP, and statistical analysis indicates that GP consistently outperforms the other five methods as it can be seen in Table 4.5. The difference between the various average results is statistically significant (P-value 2.75×10^{-9} for ANOVA test on the 4 samples of solutions found by each method). Finally, pairwise 2-tailed Student t-tests comparing GP with each other method demonstrate its better performance.

The original scoring method of [van 't Veer et al., 2002, van de Vijver et al., 2002], and in particular the suggested cutoff of 0.4, was chosen in such a way as to minimize the number of false negatives. Therefore it is not surprising that in this respect the scoring method is far superior to all machine learning methods, including GP. Indeed the average number of false negatives given by the scoring method is 1.78, to be compared to the numbers reported in Table 4.5.

4.7 Maximizing Sensitivity in GP

It is well know that the fitness function driving the evolutionary dynamics in a GP framework can be modified in order to let emerge solutions with different characteristics. The results presented and discussed in the previous section were obtained with the goal of minimizing all incorrectly classified instances, summing both false negative and false positive predictions obtained by the solutions. However, when using gene signatures to predict the survival of a cohort of breast cancer patients, minimizing the number of false negative predictions is recognized as one of the most important goals.

For all these reasons, we modified the GP fitness function so that false negatives (positives) are penalized more than errors of the other type, hoping to tune the algorithm towards better sensitivity (sensibility). In particular, solutions with greater sensitivity can emerge if larger weights are assigned to false negatives compared to false positives. In general, we can transform the fitness function in a weighted average of the form:

$$Fitness = 0.9 \times FalseNegative + 0.1 \times FalsePositive$$

With respect to this new formulation, the fitness function of the GP algorithm whose results were presented in the previous section can be expressed as $0.9 \times FalseNegative + 0.1 \times FalsePositive$. The results of 50 runs of this new version of the GP technique showed an average of 16.04 (with SEM $= 0.44$) of total incorrectly classified instances. Compared with the performances of the previous GP algorithm, no statistically significant difference can be highlighted (Student t-test $P = 0.50$). When looking only at the number of false negative incorrectly classified instances, the average performance of 4.32 (SEM $= 0.346$) is better than the one of standard GP reported in Table 4.5 (Student t-test $P = 6.62 \times 10^{-16}$), even if still worse than that of the original scoring method.

Figures and Tables

```
(or (and (or ORC6L RFC4) (or UCHL5 PRC1))
    (and (or PRC1 AI554061) (or ESM1 AW014921)))
```

Fig. 4.1 Tree representation and the traditional Lisp representation of the model with the best fitness found by GP over the studied 50 independent runs.

GP Parameters	
population size	500 individuals
population initialization	ramped half and half [Koza, 1992]
selection method	tournament (tournament size = 10)
crossover rate	0.9
mutation rate	0.1
maximum number of generations	5
algorithm	generational tree based GP with no elitism
SVM Parameters	
complexity parameter	0.1
size of the kernel cache	10^7
epsilon value for the round-off error	10^{-12}
exponent for the polynomial kernel	$1.0, 2.0, 3.0$
tolerance parameter	0.001
Multilayered Perceptron Parameters	
learning algorithm	Back-propagation
learning rate	0.03
activation function for all the neurons in the net	sigmoid
momentum	0.2 progressively decreasing until 0.0001
hidden layers	(number of attributes + number of classes) / 2
number of epochs of training	500
Random Forest Parameters	
number of trees	2500
number of attributes per node	1

Table 4.1 Parameters used in the experiments.

	GP	**SVM-K1**	**SVM-K2**	**SVM-K3**	**MP**	**RF**
best	10	13	14	15	10	12
average (SEM)	16.40 (0.30)	18.32 (0.37)	16.76 (0.18)	17.62 (0.17)	18.08 (0.39)	17.60 (0.35)

Table 4.2 Experimental comparison between the number of incorrectly classified instances found on the test sets by the different machine learning methods. Each method was independently run 50 times using each time a different training/test partition of the validation dataset (see text for details). The first line indicates the method: Genetic Programming (GP), Support Vector Machine with exponent for the polynomial kernel 1.0 (SVM-K1), 2.0 (SVM-K2), and 3.0 (SVM-K3), Multilayer Perceptrons (MP), and Random Forest (RF). The second line shows the best value of the incorrectly classified instances obtained on the test set over the 50 runs, and the third line reports the average performances of each group of 50 runs on their test sets (standard error of mean is shown in parentheses).

ANOVA				
$P = 3.05 \times 10^{-5}$				
GP vs. SVM-K1	GP vs. SVM-K2	GP vs. SVM-K3	GP vs. MP	GP vs. RF
$P = 0.0001$	$P = 0.3107$	$P = 0.0008$	$P = 0.0009$	$P = 0.0103$

Table 4.3 Statistical significance of the difference in performance between the methods. First line shows ANOVA test on the 6 samples of solutions found by each method, while second line depicts pairwise 2-tailed Student t-tests comparing GP with each other method.

	GP	SVM-K1	SVM-K2	SVM-K3	MP	RF
best	2	6	6	6	5	6
average (SEM)	9.82 (0.44)	13.26 (0.51)	12.60 (0.35)	14.08 (0.39)	12.88 (0.51)	13.38 (0.49)

Table 4.4 Experimental comparison between the number of false negatives found on the test sets by the different machine learning methods. Each method was independently run 50 times using each time a different training/test partition of the validation dataset (see text for details). The first line indicates the method: Genetic Programming (GP), Support Vector Machine (SVM), Multilayer Perceptrons (MP), and Random Forest (RF). The second line shows the best value of the incorrectly classified instances obtained on the test set over the 50 runs, and the third line reports the average performances of each group of 50 runs on their test sets (standard error of mean is shown in parentheses).

ANOVA				
$P = 2.75 \times 10^{-9}$				
GP vs. SVM-K1	GP vs. SVM-K2	GP vs. SVM-K3	GP vs. MP	GP vs. RF
$P = 2.74 \times 10^{-6}$	$P = 3.32 \times 10^{-6}$	$P = 1.27 \times 10^{-10}$	$P = 8.53 \times 10^{-6}$	$P = 4.65 \times 10^{-7}$

Table 4.5 False negative prediction: statistical significance of the difference in performance between the methods. First line shows ANOVA test on the 6 samples of solutions found by each method, while second line depicts pairwise 2-tailed Student t-tests comparing GP with each other method.

Accession ID	Gene name	Gene description	Solutions
NM_003981	PRC1	protein regulator of cytokinesis 1	48
NM_002916	RFC4	replication factor C (activator 1) 4, 37kDa	23
AI992158	-	-	16
AI554061	-	-	10
NM_006101	NDC80	NDC80 homolog, kinetochore complex component (S. cerevisiae)	9
NM_015984	UCHL5	ubiquitin carboxyl-terminal hydrolase L5	7
NM_020188	C16orf61	chromosome 16 open reading frame 61	6
NM_016448	DTL	denticleless homolog (Drosophila)	6
NM_014791	MELK	maternal embryonic leucine zipper kinase	6
NM_004702	-	-	6

Table 4.6 The 10 most recurring features in the solutions found by GP. The four columns show: accession ID, gene name, gene description, and number of solutions where that feature occurs.

Chapter 5
Conclusions and Future Work

The investigation presented in this document was aimed at refining the set of criteria that could lead to better risk stratification in breast cancer. To reach this goal, the starting point was the use of the well known "70-genes signature" and the application of several machine learning schemes, in order to perform a comparison between them. Some simplifying assumptions were made, preprocessing the data accordingly and several evaluation experiments were ran. The presented results showed that while all the studied machine learning algorithms do have predictive power in classifying breast cancer patients into risk classes, GP clearly outperforms all other methods. The fact that all methods other than GP had very similar performance suggests that GP is indeed the most promising method. The improvement in performance shown by GP compared to the original scoring method was rather small and not statistically significant. As expected, the scoring method was superior to all machine learning algorithms in minimizing false negatives. In a second phase, GP was enriched by changing its fitness function into a weighted average between false negatives and false positives. It was shown that, when larger weight is given to false negatives, it is possible to tune the GP algorithm towards greater sensitivity. While the sensitivity of GP is still less than the original scoring method, the possibility of tuning the fitness function is another intrinsic advantage of this technique with respect to the other machine learning ones considered here. Nevertheless these results warrant further investigation into the use of GP in this context for at least three reasons:

- The implementation of GP was purposely not optimized, and we can expect substantial improvements in performance from further work aimed at tuning the various GP settings.
- Maybe more importantly, GP can potentially offer biological insight and generate hypotheses for experimental work (see also [Yu et al., 2007]). Indeed an important result of the presented analysis is that the trees produced by GP tend to contain a limited number of features, and therefore are easily interpretable in biological terms. For example, the best-performing tree is shown in Figure 4.1 and includes 7 genes (features).

- Finally within the context of GP there is a natural way to tune the algorithm towards better sensitivity (specificity), simply by defining a fitness function in which false negatives (positives) are penalized more than errors of the other type.

Future work along these lines should therefore focus on both improving the performance of GP and interpreting the results from the biological point of view. An obvious first step towards optimization would be to abandon the binarization of the data (which here was used to produce trees that are easier to interpret) and build a GP based on continuous expression values. The biological interpretation might benefit from a statistical and functional analysis of the most recurring subtrees in optimal GP solutions. As a partial conclusion, it is possible to assert that GP outperforms other machine learning methods as a tool to extract predictions from an established breast cancer gene signature. Given the possibility of generating biological insight and hypotheses that is intrinsic to the method, it deserves deeper investigation along the lines described above. Finally, it would be an appropriate task to test the GP approach on other features/gene sets, that account for other cancers or other diseases, always with the objective of providing clinicians with more precise and individualized diagnosis criteria.

Another important research avenue to explore in the future concerns the possibility of increasing our datasets by means of Radiomics [Lambin et al., 2012]. Radiomics is an emerging field of medical studies, aimed at extracting large amounts of highly informative features from medical images, thus converting images into mineable data, and analysing those data for decision support. The hypothesis of Radiomics is that the distinctive imaging features between disease forms may be crucial for predicting prognosis and therapeutic response for various conditions, thus providing valuable information for personalised therapy. The Radiomics workflow can be organized into distinct phases, each with its own challenges:

- identification of a patient cohort;
- optimization of acquisition protocols;
- tumor and organ segmentation;
- feature extraction and feature selection; and
- model development and validation.

An important objective for future research is to improve the state of the art in two of these phases: tumor and organ segmentation and model development and validation, with particular reference to breast cancer, that is the type of disease that is discussed in this document. These phases may be approached using existing and novel machine learning and deep learning technologies. These technologies need to be studied and compared, to discover the most appropriate algorithm, able to outperform the state of the art in each specific case. Concerning machine learning, also in sight of the results presented in this document, focus should be given to two recently defined and extremely promising bio-inspired algorithms, which are new developments of GP: Geometric Semantic GP (GSGP) [Vanneschi, 2017], mostly used for regression problems, and Multidimensional Multiclass GP (M3GP) [Muñoz et al., 2015], mostly used for (binary or multiclass) classification problems. These methods will be compared to the state of the

art methods in Radiomics, including Random Forests, Support Vector Machines, Bayesian Networks and Linear, Least Square and Logistic regression. In the last few years, GSGP has developed enormously, becoming one of the most popular hot topics in the GP community. Thanks to an efficient and innovative implementation of GSGP [Vanneschi, 2017], it was possible to apply GSGP to a vast set of applications from different domains, including prediction of pharmacokinetic parameters in drug discovery, positioning of computer tomography slices, prediction of the unified Parkinson's disease rating scale assessment, prediction of anticoagulation level in pharmacogenetics, and also rather different application domains like energy forecasting, prediction of high performance concrete strength, and prediction of vessels' trajectories at sea for improving maritime safety and security.

Concerning deep learning, Convolutional Neural Networks (CNNs), which represent the state of the art in computer vision and many other problem domains, represent an interesyting starting point for comparison. Future work should include the delopment of new methods for integrating GSGP with CNNs, or to introduce into CNNs the concept of semantics, that is characteristic of GSGP. The idea is that sharing the same properties as GSGP, and extending them to deep learning, these novel systems will induce a unimodal error surface (i.e. an error surface characterized by the absence of locally optimal solutions) for any supervised learning problem. This fact should bestow on these novel systems a competitive advantage in terms of evolvability. At the same time, as for GSGP, these systems should be able to limit overfitting, thus being able to generate accurate and robust predictive models.

Besides an extensive performance comparison of machine learning and deep learning methods, in the future importance should be given to an attentive evaluation of the relative pros and cons. Generally speaking, we expect the machine learning methods to require more effort in the pre-processing phase (for feature extraction and selection), as opposite to the deep learning methods, that incorporate feature extraction and selection directly in some internal learning layers. On the other hand, we expect deep learning methods to have a better performance in problems characterized by a vast amount of data, while they may be outperformed by the machine learning methods in case of smaller amounts of data, a not so infrequent event in the medical field. The amount of available data is indeed a crucial theme for any medical application, and for oncology in particular. The studied applications are generally characterized by a vast amount of data, but supervising those data is generally a very hard and time consuming task. For this reason, currently only a small part of the available data are supervised. This leads to the potential for the existence of very large datasets, in which unsupervised observations are much more numerous than the supervised ones. For this reason, a significant part of the future studies should be dedicated to advancements in the area of semi-supervised learning. Particularly promising seems the idea of extending the properties that characterize the most recent versions of GP (and that determined their recent success) to semi-supervised learning. A significant first step has been taken recently for the case of the M3GP algorithm.

References

[Alon et al., 1999] Alon, U., Barkai, N., Notterman, D., Gish, K., Ybarra, S., Mack, D., and Levine, A. J. (1999). Broad patterns of gene expression revealed by clustering analysis of tumour and normal colon tissues probed by oligonucleotide arrays. In *Proc. Nat. Acad. Sci.*, pages 6745–6750. USA 96.

[Archetti et al., 2006] Archetti, F., Lanzeni, S., Messina, E., and Vanneschi, L. (2006). Genetic programming for human oral bioavailability of drugs. In M. Cattolico *et al.*, editor, *Proceedings of the 8th annual conference on Genetic and Evolutionary Computation*, pages 255 – 262, Seattle, Washington, USA.

[Archetti et al., 2007a] Archetti, F., Lanzeni, S., Messina, E., and Vanneschi, L. (2007a). Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, 8(4):413–432.

[Archetti et al., 2007b] Archetti, F., Messina, E., Lanzeni, S., and Vanneschi, L. (2007b). Genetic programming and other machine learning approaches to predict median oral lethal dose (LD50) and plasma protein binding levels (%PPB) of drugs. In E. Marchiori *et al.*, editor, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Proceedings of the Fifth European Conference, EvoBIO 2007*, Lecture Notes in Computer Science, LNCS 4447, pages 11–23. Springer, Berlin, Heidelberg, New York.

[Archetti et al., 2007c] Archetti, F., Messina, E., Lanzeni, S., and Vanneschi, L. (2007c). Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, 8(4):17–26.

[Bojarczuk et al., 2001] Bojarczuk, C., Lopes, H., and Freitas, A. (2001). Data mining with constrained-syntax genetic programming: applications to medical data sets. *Proceedings Intelligent Data Analysis in Medicine and Pharmacology*, 1.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Belmont, California, Wadsworth International Group.

[Chu and Wang, 2005] Chu, F. and Wang, L. (2005). Applications of support vector machines to cancer classification with microarray data. *Int J Neural Syst*, 15(6):475–484.

[Darwin, 1859] Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection*. John Murray.

[Deb and Reddy, 2003] Deb, K. and Reddy, A. R. (2003). Reliable classification of two-class cancer data using evolutionary algorithms. *Biosystems*, 72(1-2):111–129.

[Deutsch, 2003] Deutsch, J. M. (2003). Evolutionary algorithms for finding optimal gene sets in microarray prediction. *Bioinformatics*, 19(1):45–52.

[Dubitzky et al., 2006] Dubitzky, W., Granzow, M., and Berrar, D. P. (2006). *Fundamentals of Data Mining in Genomics and Proteomics*. Springer-Verlag, Berlin, Heidelberg.

[Friedman et al., 2000] Friedman, N., Linial, M., Nachmann, I., and Peer, D. (2000). Using bayesian networks to analyze expression data. *J. Computational Biology*, 7:601–620.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

[Guyon et al., 2002] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422.

[Hall et al., 2009] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18.

[Hernandez et al., 2007] Hernandez, J. C. H., Duval, B., and Hao, J. (2007). A genetic embedded approach for gene selection and classification of microarray data. *Lecture Notes in Computer Science*, 4447:90–101.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan.

[Hong and Cho, 2006] Hong, J. and Cho, S. (2006). The classification of cancer based on dna microarray data that uses diverse ensemble genetic programming. *Artif. Intell. Med*, 36:43–58.

[Hsu et al., 2003] Hsu, A., Tang, S., and Halgamuge, S. (2003). An unsupervised hierarchical dynamic self-organizing approach to cancer class discovery and marker gene identification in microarray data. *Bioinformatics*, 19(16):2131–40.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming*. The MIT Press, Cambridge, Massachusetts.

[Lambin et al., 2012] Lambin, P., Rios-Velazquez, E., Leijenaar, R., Carvalho, S., van Stiphout, R. G., Granton, P., Zegers, C. M., Gillies, R., Boellard, R., Dekker, A., and Aerts, H. J. (2012). Radiomics: Extracting more information from medical images using advanced feature analysis. *European Journal of Cancer*, 48(4):441 – 446.

[Langdon and Buxton, 2004] Langdon, W. B. and Buxton, B. F. (2004). Genetic programming for mining dna chip data from cancer patients. *Genetic Programming and Evolvable Machines*, 5(3):251–257.

[Liu et al., 2005] Liu, J., Cutler, G., Li, W., Pan, Z., Peng, S., Hoey, T., Chen, L., and Ling, X.-B. (2005). Multiclass cancer classification and biomarker discovery using ga-based algorithms. *Bioinformatics*, 21:2691–2697.

[Lu and Han, 2003] Lu, Y. and Han, J. (2003). Cancer classification using gene expression data. *Inf. Syst.*, 28(4):243–268.

[Michie et al., 1994] Michie, D., Spiegelhalter, D., and Taylor, C. (1994). *Machine learning, neural and statistical classification*. Prentice Hall.

[Miller et al., 2005] Miller, L. D., Smeds, J., George, J., Vega, V. B., Vergara, L., Ploner, A., Pawitan, Y., Hall, P., Klaar, S., Liu, E. T., and Bergh, J. (2005). An expression signature for p53 status in human breast cancer predicts mutation status, transcriptional effects, and patient survival. *Proc Natl Acad Sci U S A*, 102(38):13550–13555.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[Moore et al., 2001] Moore, J., Parker, J., and Hahn, L. (2001). Symbolic discriminant analysis for mining gene expression patterns. *Lecture Notes in Artificial Intelligence*, 2167:372–381.

[Muñoz et al., 2015] Muñoz, L., Trujillo, L., and Silva, S. (2015). M3gp multiclass classification with gp. In *Genetic Programming - 18th European Conference, EuroGP 2015, Proceedings*, volume 9025 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 78–91. Springer-Verlag. 18th European Conference on Genetic Programming, EuroGP 2015 ; Conference date: 08-04-2015 Through 10-04-2015.

[Nevins and Potti, 2007] Nevins, J. R. and Potti, A. (2007). Mining gene expression profiles: expression signatures as cancer phenotypes. *Nat Rev Genet*, 8(8):601–609.

[Paul and Iba, 2005] Paul, T. K. and Iba, H. (2005). Gene selection for classification of cancers using probabilistic model building genetic algorithm. *Biosystems*, 82(3):208–225.

[Platt, 1998] Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods – Support Vector Learning*.

[Poli et al., 2008] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

[Rosskopf et al., 2007] Rosskopf, M., Schmidt, H., Feldkamp, U., and Banzhaf, W. (2007). Genetic programming based dna microarray analysis for classification of tumour tissues. Technical Report Technical Report 2007-03, Memorial University of Newfoundland.

[S. Haykin, 1999] S. Haykin (1999). *Neural Networks: a comprehensive foundation*. Prentice Hall, London.

[Shalev-Shwartz and Ben-David, 2014] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA.

[Silva, 2007] Silva, S. (2007). GPLAB – a genetic programming toolbox for MATLAB, version 3.0. `http://gplab.sourceforge.net`.

[van de Vijver et al., 2002] van de Vijver, M. J., He, Y. D., van't Veer, L. J., Dai, H., Hart, A. A. M., Voskuil, D. W., Schreiber, G. J., Peterse, J. L., Roberts, C., Marton, M. J., Parrish, M.,

Atsma, D., Witteveen, A., Glas, A., Delahaye, L., van der Velde, T., Bartelink, H., Rodenhuis, S., Rutgers, E. T., Friend, S. H., and Bernards, R. (2002). A gene-expression signature as a predictor of survival in breast cancer. *N Engl J Med*, 347(25):1999–2009.

[van 't Veer et al., 2002] van 't Veer, L. J., Dai, H., van de Vijver, M. J., He, Y. D., Hart, A. A. M., Mao, M., Peterse, H. L., van der Kooy, K., Marton, M. J., Witteveen, A. T., Schreiber, G. J., Kerkhoven, R. M., Roberts, C., Linsley, P. S., Bernards, R., and Friend, S. H. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415(6871):530–536.

[Vanneschi, 2004] Vanneschi, L. (2004). *Theory and Practice for Efficient Genetic Programming*. PhD thesis, Faculty of Sciences, University of Lausanne, Switzerland.

[Vanneschi, 2017] Vanneschi, L. (2017). *An Introduction to Geometric Semantic Genetic Programming*, pages 3–42. Springer International Publishing, Cham.

[Vapnik, 1998] Vapnik, V. (1998). *Statistical Learning Theory*. Wiley, New York, NY.

[Yu et al., 2007] Yu, J., Yu, J., Almal, A. A., Dhanasekaran, S. M., Ghosh, D., Worzel, W. P., and Chinnaiyan, A. M. (2007). Feature selection and molecular classification of cancer using genetic programming. *Neoplasia*, 9(4):292–303.