Università degli Studi di Pisa
Dipartimento di Informatica

Dottorato di Ricerca in Informatica

Ph.D. Thesis

# Hypothesis Testing with Classifier Systems

Flavio Baronti

Supervisor
Antonina Starita

May, 2007

# Abstract

This thesis presents a new ML algorithm, HCS, taking inspiration from Learning Classifier Systems, Decision Trees and Statistical Hypothesis Testing, aimed at providing clearly understandable models of medical datasets. Analysis of medical datasets has some specific requirements not always fulfilled by standard Machine Learning methods. In particular, heterogeneous and missing data must be tolerated, the results should be easily interpretable. Moreover, often the combination of two or more attributes leads to non-linear effects not detectable for each attribute on its own. Although it has been designed specifically for medical datasets, HCS can be applied to a broad range of data types, making it suitable for many domains. We describe the details of the algorithm, and test its effectiveness on five real-world datasets.

# Acknowledgements

I would like to thank my supervisor, Antonina Starita, for giving me the intellectual (and economical!) freedom to follow and explore my often diverging ideas, adding guidance and concreteness, and never putting constraints.

I'm very grateful to my reviewers, Stewart W. Wilson and John H. Holmes, for extremely helpful comments and constructive suggestions.

Thanks to Alessandro for countless interesting discussions and exchanges of ideas.

As silly as it sounds, I am thankful to Google. It always gave me the answer I was looking for — once I found the right way to ask the question!

Last, I thank my family for supporting me, asking me "is it done yet?" at just the wrong times (that is... always!), and bearing with my consequent losses of temper. Grazie.

# Contents

# Symbols

| | |
|---|---|
| $A_i$ | One of the sets composing the space of the dataset: $\mathcal{D} = A_1 \times A_2 \times \ldots \times A_s$ |
| $\mathbb{B}$ | The set of boolean values: $\mathbb{B} = \{\text{True}, \text{False}\}$ |
| $\mathcal{D}$ | The space the dataset comes from: $D \subseteq \mathcal{D} = A_1 \times A_2 \times \ldots \times A_s$ |
| $\mathcal{C}$ | The set of all possible classifiers, $C : \mathcal{D} \to \mathbb{B}$ |
| $c$ | A single classifier, $c \in \mathcal{C}$ |
| $D$ | The dataset |
| $d$ | An instance of the dataset: $d \in D$ |
| $\mathcal{F}$ | Indicates the fitness function. Generally, $\mathcal{F} : D \times \mathcal{H} \times \mathcal{C} \to \mathbb{R}$; however, often $\mathcal{H}$ and $D$ are implicit and dropped for readability. When $y : D \to \mathbb{B}$, the fitness can also be defined as $\mathcal{F}(q, t) : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$. In this case, $q = \#\{d \in D/_c \mid y(d) = \text{True}\}$, and $t = \#\{d \in D/_c\}$. |
| $F(x)$ | The cumulative probability of a probability distribution: $F(x) = \mathbb{P}(X \leq x) = \sum_{v \leq x} f(v)$ |
| $f(x)$ | The probability mass function of a probability distribution: $f(x) = \mathbb{P}(X = x)$ |
| $\mathcal{H}$ | The set of all models taken into account by the algorithm |
| $H$ | A function creating a model from a dataset: $H : 2^{\mathcal{D}} \to \mathcal{H}$ |
| $N$ | The maximum number of classifiers used during genetic search ("population size") |
| $Q$ | The number of positive instances in the dataset. Meaningful only if $y : D \to \mathbb{B}$, and in that case $Q = \#\{d \in D \mid y(d) = \text{True}\}$ |
| $\rho$ | The proportion of positive targets in the dataset: $\rho = Q/T$ |
| $s$ | The number of attributes in each instance |

| | |
|---|---|
| $S_D(q,t)$ | A subset of $D$ containing $q$ positive instances and $t$ total instances: $S(q,t) \in \{R \subseteq D \mid t = \#(R) \wedge q = \#\{d \in R \mid y(d) = \text{True}\}$ |
| $T$ | The number of instances in the dataset: $T = \#(D)$ |
| $y$ | The target function, associating each instance of the dataset to its target. $y : D \rightarrow Y$ |
| $\#$ | Size of operator: returns the size of a set |
| $C/_d$ | The *match set* of an instance: $C/_d = \{c \in C \mid c(d)\}$ |
| $D/_c$ | The *cover set* of a classifier: $D/_c = \{d \in D \mid c(d)\}$ |

# Chapter 1

# Introduction

This thesis will present a new Machine Learning algorithm, called HCS, designed to combine the hypothesis formulation capabilities of Machine Learning, and the hypothesis validation proofs of Statistics. The purpose of HCS is providing a new tool for medical research, which can aid the process of understanding medical phenomena by providing comprehensible explanations of the collected data. Fulfilling the requirements of medical datasets makes HCS a very robust algorithm, able to analyze data in most other domains.

Modern sciences are based on the experimental method, as described by Galileo Galilei 400 years ago. The method can be decomposed in a sequence of steps: we will concentrate on the first, which are observation of the phenomenon, formulation of a hypothesis, validation of the results through experiments. Scientists need first to acquire sufficient data regarding the phenomenon they want to analyze. Then the induction step comes: here the intuition of scientists is fundamental, because they are required to provide the formulation of a general hypothesis from the collection of particular situations. Once the hypothesis is set down, its predictions have to be validated and verified through more experiments.

For Galileo, mathematics was the tool to use in the whole process: "It [the book of Nature] is written in mathematical language". The science of Medicine is no exception to this rule; the uncertainty inherent in its results however required Statistics, rather than Mathematics, to provide the models to describe and deal with phenomena — along with the tools for model validation.

The widespread use of computers greatly simplified the tasks of acquiring and managing data, in all fields as well as in medical research. Keeping track of patients, collecting results of exams, recovering old records: all these activities have become more easy and straightforward. Practical limits of dealing with large amounts of data are still present — but the meaning of "large" has radically changed, with a shift of many orders of magnitude. The availability of internet connections facilitates the creation of multi-center studies, further increasing the amount of information available for research. At the same time, medicine itself has advanced, both by replacing previous exams with more precise version, and by introducing new tests, which again lead to an increase in information.

On one side, more data means more experimental evidence to test hypotheses on, and more confidence on the validity of conclusions. On the other side however, the dimensionality of the data has increased along with the quantity; out of the mathematical jargon, this means that not only information regarding many patients can be easily stored, but also that more information about each patient is stored. This is also a requirement of modern medicine, more than a consequence of the presence of computers. Going back to the experimental method, the task medical scientists are faced with nowadays is much more complex: when the amount of available data grows, and knowledge on the phenomenon is scarce, human intuition can fail to see relationships and dependencies which would make good hypotheses.

Statistics offers some instruments to aid intuition, and help the scientist formulate hypotheses (one of the most popular models in medicine is logistic regression). These methods however are based upon simple, typically linear models, which can be insufficient to properly capture the complexity of the data. The derivation of statistics from mathematics is double-edged weapon: it provides provably valid models, but is also limited to provide only those models which are provable.

Differently from statisticians, computer scientists are accustomed to write programs which *show* that they work, rather than *prove* it. Deriving from Computer Science, Machine Learning follows this pragmatic approach: it first shows *evidence* that its models produce correct results, and usually only later attempts to provide *proofs*. This approach allows to experiment with complex models, which are less mathematically tractable, but can find a more precise description of the phenomenon being studied. This difference of focus — provability versus effectiveness — is considered by some researchers the core distinction between data analysis through statistics and machine learning [133].

More traditionally, Machine Learning is defined as a discipline aimed at making programs which learn from experience. In our setting, experience comes from the data we want to analyze; the computer program is a machine learning algorithm which terminates with one (or more) hypothesis explaining the data; improving with experience means that the algorithm creates these hypotheses *learning* from the examples provided in the data. It is important to stress the difference between learning and remembering: the ML algorithms are always concerned with trying to produce a general hypothesis, which should perform well also on unseen data, instead of simply remembering all the examples without any level of abstraction.

Machine Learning can then be an important instrument for researchers, providing empirically valid hypotheses which can then be statistically tested and validated. Our algorithm was designed exactly with this aim in mind.

The HCS (Hypothesis testing with Classifier Systems) algorithm borrows ideas from Statistics, Decision Trees [100], and Learning Classifier Systems [51]. HCS belongs to the class of supervised learning algorithms: it needs a training set containing independent variables (also called *attributes*) and dependent variables (also called *target*). A supervised learner assumes that a relationship between the two sets is present; this is called the *target concept*. The learner must produce an approximation of this concept by examining the data. HCS starts from the basic assumption (null hypothesis) that the target variable is actually independent from the other variables; it therefore builds an independent model of the target. The algorithm then attempts to reject the independence hypothesis, by finding subsets of the data which falsify this model.

Subsets of data are defined by classifiers. A classifier is a conjunction of conditions on the values of the attributes; an example of a classifier is $x_2 \geq 10 \wedge x_4 \in \{\mathtt{A}, \mathtt{B}\}$. If such a classifier identifies a subset of $t$ elements, the null hypothesis should guarantee that the distribution of the target of these elements is the same as a completely random extraction of $t$ elements from the whole dataset. If this does not happen, the model postulating independence between attributes and target is not valid, and must be corrected. The amount of accordance between the actual extraction and the general model is evaluated through the $p$-value of a statistical hypothesis test. The lower the $p$-value, the higher the disagreement between our subset and the global model, the more confident is rejection of the independence hypothesis. The search for low $p$-value classifiers is performed through a genetic algorithm, very similar in structure to the XCS [122] classifier system.

Once the region with lowest $p$-value is found, that is a region with maximum amount of disagreement with the global model, the data is split into two subsets: the identified region, and the rest. The algorithm is then recursively called on both subsets, mimicking decision trees' divide-and-conquer approach. Recursion stops when the amount of disagreement is not sufficient to reject the null hypothesis. The independent model of the target is then accepted as valid.

In the following, chap. 2 will introduce the main concepts of Machine Learning, and briefly illustrate the most common methodologies. In chap. 3, we will discuss the various data-analysis tasks possible in medicine, and illustrate the peculiarities and requirements of medical datasets, whic lead us to develop HCS. Chapter 4 is dedicated at detailing the theoretical and practical structure of the HCS algorithm. Experiments upon real-world datasets are carried over in chap. 5, where a few HCS variants are compared between themselves, and with other ML approaches. We will summarize the work done, draw conclusions, and sketch future research directions in chap. 6.

# Chapter 2

# Machine learning

## 2.1 Introduction

Machine learning is a very broad discipline; this is reflected in its textbook definition, which typically is not much more than a tautological "[...] construct computer programs which automatically improve with experience" [86]. The word *learning* is used in its more proper sense of obtaining knowledge, as opposite to simple remembering. This means that a machine learning algorithm is always concerned with trying to build knowledge which will improve its performance in future, unseen situations.

In the particular (but very frequent) setting of *learning from examples*, experience is presented to the algorithm in form of a set of situations, for which also the expected correct behaviour is provided. The algorithm then is required not to simply remember all these situations, but to come out with a *generalization*, which will be applicable also to unseen situations. For humans generalizing is quite natural, although it proves its limitations in many occasions (the word itself can also be used with a slightly negative meaning). Machine learning on this point is not different; in order for any generalization to work, there are a priori assumptions which must hold, for instance about the shape of the solution. These assumptions go under the name of *inductive bias*. Inductive bias can be dangerous: too strong assumptions can prevent the system to find the best answer. On the other side, it is necessary; without any form of inductive bias, there is no rational support for classifying any unseen situations, and the only thing a learning algorithm can do is to remember past situations.

Often, training algorithms offer one or more parameters which can be adjusted, in order to select the desired amount of inductive bias. For instance, in a simple polynomial fitting situation, the degree of the polyomial is such a parameter: a higher degree allows many more function shapes to be represented. As noted before however, more complexity is not always desirable. In particular, it can be shown (see e.g. [32]) that the error committed by a training algorithm can be decomposed into three components: *bias*, *variance*, and *intrinsic*. The last term comes from the inherent noise in the data, and cannot be avoided. The first term is due to the algorithm learning a function which is an approximation of the true underlying relationship. This error can become lower with increasing model complexity. The second term is related to the dependence of the final model upon the particular sample being used as training. Its value *increases* as the model becomes more complex. This is the *bias-variance* tradeoff: finding a good balance between a low-complexity model, susceptible of *underfitting* the training data, and a high-complexity model, able of perfectly representing data, but at risk of *overfitting*. Consider again the polynomial regression. A set of 10 points following a linear relationship can be approximated perfectly either by a straight line, or by a 10-degree polynomial, or by many higher-degree polynomials. In the latter cases however the shape of the polynomials will greatly vary by changing the sample of points from the original relationship: this is a case of overfitting. Presence of noise in the data further exacerbates the problem. Model selection is generally tackled through regularization techniques, often paired or replaced by validation techniques (as described in sec. 2.1.5).

Although machine learning is a relatively young discipline, it has developed a wide range of different algorithms, specific to various situations. A classification of the algorithms along broad characteristics can then be useful, although unavoidably approximative: often algorithms are modified in order to overcome one of their limitation, or hybridized with another algorithm in the attempt to exploit the strength points of both. In the following, we will describe the most important characteristics of learning algorithm: the task, the hypothesis space, and the search algorithm. From the next section instead, we present some machine learning approaches. First we show logistic regression, a rather simple method widely used in medicine. Then we describe decision trees and Bayes networks, which have good descriptive capabilities. Following, we discuss evolutionary algorithms and in particular XCS, which provide high flexibility. Finally we conclude with neural networks and SVMs, appreciated for their high discriminative power.

### 2.1.1 Task

The first broad distinction we can do regards the *task* we are called to deal with. When we have dependent variables which we want to relate to a set of independent variables, we speak of *supervised* learning; the term comes from the idea of a teacher, telling the system what is the right answer for a set of known input situations. In medicine this situation is quite frequent: we could want to decide whether a person has or not a certain disease, how high is the risk to develop oral cancer, or how long she will survive after a surgical intervention. The teacher generally is just a table of data, containing the expected answer along with the independent variables — although in principle it could be anything: a human being actually classifying examples for instance, or even another computer model.

On the other hand, in some situations we are actually looking for a discrimination between the input situations, but lack the teacher. This setting is called *unsupervised* learning; the general aim is to look for regularities in data. One option we have is to group the input in sets of similar-looking situations; in medical domain, for instance we could want to see if all the people who achieved complete remission from leukemia form a homogeneous set, or if we can find some distinct sub-groups within them, which would point towards different remission causes. This task is solved by *clustering* algorithms, like K-Means or SOM [64]. Another possibility is to look for recurrent patterns in the data, that is associations between input variables. As a trivial example, a system could re-discover that whenever a person has influenza, she is very likely to have higher than average body temperature.

A third learning paradigm is called *reinforcement learning*; this is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment. The agent can perform a set of actions onto the environment; with each action, the environment changes state, and the agent is given feedback through a scalar value, the reinforcement signal. This is quite different from supervised learning, since the agent is never told the "right" action to take, but must find it on its own. The presence of the reinforcement signal differentiates the paradigm from straight unsupervised learning. An excellent survey on the topic can be found in [63].

### 2.1.2 Hypothesis space

Mitchell in [85] describes learning as a search over a hypothesis space. When trying to learn a certain concept, a hypothesis is the system's attempt at giving a definition; the space of hypothesis is made of all the definitions the system is able to try. This choice depends both on the problem to be solved, and on the kind of solutions we want to find. For instance, the hypothesis space of logistic regression for a problem with $n$ real inputs is

$$H = \left\{ f_{\alpha,\beta}(\mathbf{x}) = \frac{1}{1 + e^{\alpha + \sum_{i=1}^{n} \beta_i x_i}} \;\middle|\; \begin{array}{l} \alpha \in \mathbb{R} \\ \beta_i \in \mathbb{R} \\ f \in \{\mathbb{R}^n \to [0,1]\} \end{array} \right\} \equiv \mathbb{R}^{n+1}$$

The hypothesis space of decision trees is the set of all rooted binary trees, which have each node labelled with a single boolean condition of the kind $(x_k \gtrless v)$ (if all the input variables are real). The two spaces differ not only in their size (the second is clearly much bigger), but also on the kind of function they can express: the first ultimately defines an arbitrary hyperplane, while the second builds a composition of hyperplanes parallel to the axes. It is clear then that bigger is not necessarily better: a simple concept as $x_1 + x_2 < v$ will be easily discovered by logistic regression, and only roughly approximated by decision trees.

Classifying hypothesis spaces is more complex, since many choices are possible — depending both on the problem we want to solve, and on the structure of the solution we want to obtain. The following distinctions are given in no particular order, since each of them can be combined with each of the others.

One distinction regards the *expressiveness* (we will also call this *expressive power*) of the hypothesis space; in order to keep the discussion simple, this can be imagined as the number of parameters we can adjust to fit the data (there exist more rigorous definitions of the complexity of the hypothesis space, like VC-dimension [119]). It is important to note that a simple model is not necessarily worse than a complex one; both approaches have their points of strength. The first deals with a smaller hypotheses space; finding the best solution here is usually an easy task, and typically this solution does not change with slight variations of the input data (small *variance* of the model). However, such models have a quite strong *bias*: a small hypothesis space is less likely to contain the real optimum solution, so it is impossible for the system to find it. On the other side, variable-size models show great flexibility, and higher ability to find unexpected or novel solutions (they have a smaller *bias*); this comes of course at the price of a greater *variance*, that is a greater sensitivity of the model to the input data. Moreover, the bigger the hypotheses space becomes, the more difficult it is to properly explore it; while it probably contains good solutions, it is more difficult to find them. Complex representations require then more data to properly orient learning towards the best solution [15, 117, 120].

The second distinction we present is on the *language* of the hypotheses, which can be more oriented towards logic (*symbolic* representation) or towards numbers (*sub-symbolic* representation). A symbolic algorithm generates rules which relate the outcome to some characteristics of the input, performing comparisons and logical operations. An example of a symbolic hypothesis in human-readable form could be "If sex is male and either smoke is greater than 5 or exposure to pollutants is greater than 8, then the probability to get oral cancer is 0.9". On the other side, sub-symbolic approaches deal only with numbers and mathematical functions; in the same setting, such an approach could instead derive the formula $p = 0.76 \times \mathit{sex} + 8.12 \times \mathit{smoke} + 3.15 \times \mathit{poll}$.

The symbolic approach has the strong advantage of being very close to human reasoning: this generally makes the results readily understandable by the researcher, which is very important in critical domains like medicine. On the other side, sub-symbolic models are generally more flexible; their mathematical nature can be exploited to explore cleverly the search space, and to demonstrate properties of the found solution.

The last distinction we will note on the model regards *deterministic* and *stochastic* models[1]. The first ones try to learn a direct function $f : \mathbf{X} \to Y$ from the space of inputs $\mathbf{X}$ to the set of outputs $Y$. Stochastic models instead look for a probability distribution, namely $\mathbb{P}(y|\mathbf{x})$ (where $\mathbf{x} \in \mathbf{X}$ and $g \in Y$). Of course, the deterministic correspondent of a stochastic model can always be obtained by simply taking the most probable outcome. The latter model is then clearly more complex, and as usual more complex models need more data to produce stable results; the choice here can then be guided by the real need of a probabilistic estimate of the outputs, rather than a single choice.

There are other characteristics which can guide towards a hypothesis space (and thus towards a ML algorithm), which are orthogonal to the vari-

[1]Note this is not the difference between *discriminative* and *generative* (or informative) learning [103, 91]; the distinction we are doing is inside discriminative models.

ous discriminations cited so far. One is its interpretability. If the researcher is more interested understanding the solution found by the algorithm rather than using it for predicting unseen data, interpretability becomes a chief factor. However, this usually comes at the expense of flexibility and descriptive power: systems which do not need to satisfy an interpretability constraint can afford more complex representations.

Another important feature of ML algorithms is the ability to incorporate prior knowledge into the learning process. Often the researcher has some information on the data which could help learning, discarding trivial or incorrect hypotheses; being able to convey this information to the algorithm is certainly advantage. It is to be noted however that one of the points of strength of ML against traditional techniques is its ability to propose solutions on poorly understood domains.

### 2.1.3 Search algorithm

The last distinctive feature of a ML algorithm is the way it exploits the data to search through the hypothesis space: this is called *search algorithm* (or learning algorithm in some cases).

In the most simple situations it is actually possible to analytically determine the optimum solution given the data; more commonly however this is not possible. A search algorithm is then necessary, which specifies how the hypothesis space is to be explored. Search strategies can be described in general terms, leaving some details to be specified with the particular structure of the search space. These details typically include a relationship of "better than" between hypotheses, and a way to generate neighbour hypotheses from a given one (along with the definition of neighborhood).

The most simple search algorithm is *exhaustive search*. This involves simply trying every hypothesis in the hypothesis space, and choosing the best one. This method of course ensures optimality (and very often it is the only one so reliable); unfortunately, it is unfeasible for all the non-trivial cases, since the space is too big (or even infinite) to be explored completely in a reasonable amount of time.

At the other end of the spectrum lie the *Monte Carlo* methods: these algorithms choose a random subset of the space of hypotheses and explore only that. Choosing the size of the sample is a very easy method to limit the running time of search; of course, these methods generally do not offer guarantees on the quality of the found solution with respect to the true optimum.

In order to reduce the time necessary to find a solution, several *heuristic* methods have been developed. *Best first* search can be thought as a clever way to perform an exhaustive search. A start point is chosen (at random); then all of its *neighbours* (defined following a problem-dependent neighborhood criterion) are ordered according to the "better than" relationship, and recursively visited. With this strategy, most promising paths are followed first, and then progressively worse-looking parts of the space are explored.

In order to cut down the complexity of this search, *beam search* explores only the best $n$ solutions at each step. This however does not guarantee any more that the best solution will sooner or later be found; so often $n$ is taken as 1, which is the case of *hill climbing*. In continuous domains (like in neural networks), this algorithm makes use of *gradient descent* techniques.

While at first glance this seems a much better solution than random or exhaustive search, it has some drawbacks. Often there is no clear definition of neighboring solutions — or when there is one, the set could be too big or infinite to be explored exhaustively. The main drawback however is that this strategy is deceived by *local optima*, that is solutions which are best of their neighborhood, but not the global optimum; in complex representations, this is a serious problem, also because there is no guarantee on the actual distance to the true optimum. This problems tend to be mitigated with other heuristic techniques, like *restarting* (performing several times hill climbing, each time with a different start point).

Finally, *Evolutionary computation* is a relatively new search paradigm which promises to overcome these limitations, but at the price of introducing variability in the final solution (the algorithm makes probabilistic choices, resulting in executions with the same input and starting point are not to be guaranteed to produce the same output). EC and genetic algorithms are better described in following sections.

### 2.1.4 Evaluation

A crucial issue in machine learning is performance *evaluation*. This means defining one or more synthetic measures which summarize the behaviour of the model — in other words, how much the model corresponds to the data.

In a classification task, the most immediate performance measure is accuracy, that is the fraction of instances correctly classified. But this is far from giving us all the information we could need. As an example, in a clini-

| | | predicted | |
|---|---|---|---|
| | | positive | negative |
| actual | positive | $TP$ | $FN$ |
| | negative | $FP$ | $TN$ |

Table 2.1: A confusion matrix.

cal context, if we want to classify ill patients against healthy ones, it is very relevant if a test misclassifies an ill patient as healthy or viceversa, but this is something crude accuracy does not take into account.

In this chapter we will examine some performance measures, especially those most commonly used in medical applications, and we will consider the most widely used methods of estimating them.

### Confusion Matrix

Consider a binary classification problem (with only two classes: positive and negative) on a dataset of $N$ examples. In this case, a confusion matrix is used as a basis for performance evaluation.

The fields of the confusion matrix shown in Table 2.1 contain the numbers of examples of the following four subsets:

**True positives ($TP$)**: True positive answers denoting correct classifications of positive cases.

**True negatives ($TN$)**: True negative answers denoting correct classifications of negative cases.

**False positives ($FP$)**: False positive answers denoting incorrect classifications of negative cases into positive class.

**False negatives ($FN$)**: False negative answers denoting incorrect classifications of positive cases into negative class.

In the fields of the confusion matrix, for the convenience of computation, the absolute numbers may be replaced by the relative frequencies, e.g. $TP$ by $\frac{TP}{N}$. This may be more convenient when relative frequencies are used as probability estimates.

## Standard Performance Evaluation Measures

The classification accuracy is the most popular performance evaluation measure used in predictive knowledge discovery where the goal of learning is prediction or classification. The classification accuracy measures the proportion of correctly classified cases. In binary classification problems using the confusion matrix notation, the accuracy is computed as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{N} \tag{2.1}$$

Sensitivity is a measure frequently used in medical applications. It measures the fraction of actual positives that are correctly classified. In medical terms, maximizing sensitivity means detecting as many ill patients as possible.

$$Sensitivity = \frac{TP}{TP + FN} \tag{2.2}$$

This measure is identical to recall known from information retrieval (recall of positive cases).

Specificity is the parallel of sensitivity for negative cases (it is also called the recall of negative cases):

$$Specificity = \frac{TN}{TN + FP} \tag{2.3}$$

Maximizing specificity is equivalent to minimizing the false alarm rate, where $FalseAlarm = 1 - Specificity = \frac{FP}{TN+FP}$. In medicine, this measure is aimed at minimizing the fraction of healthy patients declared as ill.

Specificity and sensitivity are extremely important in medical applications because they correctly differentiate between the two types of errors (FP and FN), which usually have very different implications. In a screening test for HIV for instance, a false positive result can be easily detected through the further tests the patient will undergo. A false negative will instead delay cures until the disease becomes clinically evident, therefore possibly impairing the effectiveness of medications. Notice that it is usually difficult to design a medical test with both high specificity and sensitivity. The doctor must therefore choose the most appropriate test, based upon his aims.

A further point of interest of the two measures is their theoretical independence of the prior probability of being positive (the *prevalence*). Their estimates will therefore be correct, regardless of possible differences in prevalence between the estimate set and the application.

Once specificity, sensitivity and apriori prevalence have been estimated, the probability of an instance being positive given that the test classifies it as positive can be calculated through the Bayes' theorem. This value is called *positive predictive value*, which along with the *negative predictive value* describes the performance of a classifier over a population of known prevalence. Calling $A$ the "actually positive" event, and $B$ the "classified positive" event, the formula is

$$\begin{aligned} ppv = \mathbb{P}(A|B) &= \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B|A)\mathbb{P}(A) + \mathbb{P}(B|\sim A)\mathbb{P}(\sim A)} \\ &= \frac{sns \times prv}{sns \times prv + (1 - spc) \times (1 - prv)} \end{aligned} \tag{2.4}$$

To exemplify, suppose a medical test has been estimated to have 95% specificity and 99% sensitivity. If the general prevalence of disease is 0.1%, the probability of a patient to be ill given that the test reports him as ill is $\frac{.99\times.001}{.99\times.001+.05\times.999} \simeq 2\%$. Although the sensitivity appears quite high, the test is still insufficient to declare the patient ill with any reasonable degree of certainty. On the other side, if the patient apriori risk is higher (for instance, an hereditary disease already present in his family), the test can become more decisive. If the apriori risk is $prv = 15\%$, we obtain $ppv \simeq 78\%$.

### ROC curves

A further use of sensitivity and specificity values is to plot a receiver operating characteristic (ROC) curve [35]. Some tests produce a continuous-valued output; in order to produce a binary classification, a threshold on this value is applied, which discriminates positive from negative instances. Different thresholds will clearly produce different predictions, and consequently different sensitivity and specificity values. The ROC curve is the plot of sensitivity against 1 - specificity for each possible threshold value.

Fig. 2.1 shows the ROC curves corresponding to three different tests. The first one plots on the 45-degree diagonal through the origin. This curve corresponds to a classifier with no information content, giving predictions equivalent to completely random. Instead the ROC curves for the other tests, have a shoulder closer to the upper left-hand corner of the plot (higher sensitivity and specificity), thus they are better classifiers. The perfect classifier would produce a plot along the left and top borders of the graph.

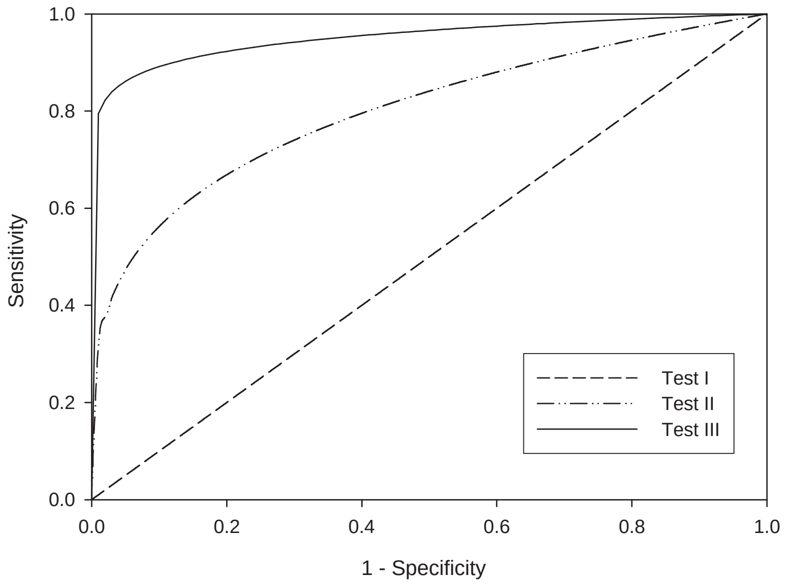


Figure 2.1: Examples of ROC curves.

The ROC curve is of great interest in visualizing the discriminatory accuracy of a test. If the ROC curve of a test is always above another curve, the former test is always more precise in discriminating positive from negative instances, regardless of the global prevalence.

Another measure of interest coming from the ROC curve is the area under the curve (AUC or AUROC). This value is between 0.5 (random classification) and 1.0 (perfect classification), and can be used to summarize the performance of a classifier in a single figure. The value can be interpreted as the probability that a random negative instance will be ranked lower than a random positive instance. Although its use has been advocated [18], it is still not very popular in the Machine Learning community. A clear exposition of ROC curves and their use is given by [39]. Webb and Ting [121] present an interesting discussion on the applicability of ROC curve analysis to changing class distributions.

### Brier score

The Brier score [20] is another well-known measure of performance of a model, which is more precise than accuracy when the algorithm produces risk estimates, rather than simple classification. If positive and negative instances are labelled with 1 and 0 respectively, the Brier score can be expressed as

$$\frac{1}{\#D}\sum_{i\in D}\left(y(d)-f(d)\right)^2 \tag{2.5}$$

where $y(d)$ is the original target value, and $f(d)$ is the probability of $d$ being a positive instance as estimated by the model. Similarly to accuracy, the Brier score ranges from 0 to 1, but here lower values correspond to better models.

### Relationship between performance measures

Accuracy is by far the most common, and most simple, evaluation criterion in Machine Learning. When the goal of a learner is simply to predict a boolean target value, accuracy appears well-suited. However, it has at least two major drawbacks [98]:

- It treats equally misallocation costs. The two types of error (assigning a positive instance to the negative class, and vice-versa) normally have very different implications. In many cases, one kind of error is much more tolerable than the other kind; treating both errors equally is a simplification which is rarely guaranteed to hold.

- It is based upon the dataset prevalence. The amount of positive and negative data in the dataset is not necessarily reflecting the expected real-world prevalence. An algorithm maximizing accuracy is therefore not guaranteed to maximize accuracy also when tested on real-world data, where the prevalence may radically differ.

Analysis of the ROC curve solves these two issues. If the curve of a classifier is always above the curve of another one, the former will perform better than the latter, regardless of the particular prevalence or misallocation costs. Analyzing the curve is however problematic when many classifiers must be compared, or when measuring performance relative to results reported in literature. The area under the ROC curve is then a summary measure which can provide the advantage of independence from particular prevalence or misallocation costs. It is however to be noticed that a greater AUC does not imply that one curve is always below the other.

Accuracy and AUC are measures which focus on the *discrimination* power of a classifier — that is, its ability to differently rank positive and negative instances. The Brier score is instead focused on the correct *calibration* of the forecasts. In a probabilistic environment, where the target cannot be decided with complete certainty, a model with lowest Brier score is one which predicts accurately the probability of each instance to be positive or negative. The Brier score however suffers from the same drawbacks as accuracy. The relationship between Brier Score and AUC can be specified under some assumptions, particularly regarding the correct calibration of the model [61]. In general however this is not possible, since AUC is invariant under order-preserving transformations, while Brier score is not. It can be important therefore to evaluate both performance measures.

### 2.1.5 Validation

As every machine learning researcher knows, the true performance of an algorithm should not be evaluated on the same data it was trained on. Doing so leads to optimistic results, with the amount of "optimism" hardly quantifiable. As an extreme example, a model could simply store all the data it was provided with. It would then show perfect performance on already seen data, but would be unable to make any inference on new data. *Validation* is concerned with predicting the performance of the model on unseen data coming from the same distribution, i.e., its ability to generalize.

To obtain an unbiased estimate of performance, the standard procedure is to employ two datasets (coming from the same distribution): training set and test set. During training, the algorithm can only make use of the data in the training set. Once a model is obtained, its performance is evaluated upon the data contained in the test set.

The split can be completely at random, or *stratified*. In this latter case, the division is performed still at random, but in such a way that both the subsets contain approximately the same distribution of the target as the complete dataset.

This approach is generally feasible only when data is abundant. This is because two competing goals must be reached. First, it is quite obvious that showing less data to the training algorithm can impair its performance, and its ability to generalize. The training set should therefore be as large as possible. On the other side however, data in the test set must be abundant too: a estimate on a small sample will have a very high variance, thus making the estimate less reliable.

When there is not enough data to allow splitting into two subsets without impairing performance, a $k$-fold *cross-validation* procedure can be used. The whole dataset is split into $k$ subsets, of equal size (or very similar, if equal is not possible). The training process is then repeated $k$ times. At each repetition, one of the folds is removed from the data, and the algorithm is trained upon the remaining folds. The obtained model is then tested with the data in the secluded fold. Finally, the performance value for the algorithm is obtained by recombining all the results obtained on each fold, on the execution where it was not included in the training set. Also the $k$-fold cross-validation can be stratified, by forcing every fold to have similar target distribution to the complete dataset.

Estimates based upon a single split (either training/test or cross-validation) will show a certain amount of variance, due to the particular choice of the split. Further, some algorithms are not deterministic, but stochastic — meaning that they can produce different results when applied to the same data. In order to smooth these sources of variability, it is necessary to run multiple tests with different fold splits, and to average the performance across all the tests.

Although cross-validation can be applied also when data is abundant, the simple training/test split is usually preferred [31], for two reasons: it's much faster (a single model is trained, instead of $k$), and it's easier to perform statistical inference on the results.

### Model selection and validation

Validation methods are often applied to give an empirical answer to the bias-variance tradeoff, or more generally to perform model selection. For instance, a typical pruning method in decision trees (see sec. 2.3) chooses the nodes to discard by examining the performance of the tree estimated on a validation set. It is not uncommon for Support Vector Machines to perform several training sessions, with different combination of hyperparameter values, and to choose the one which generalized most by estimating the performance upon a separate data set.

It is important to recognize that the same data cannot be used both for model selection and validation. In order to obtain a truly unbiased estimate of the performance of a model, it is necessary to test it on data which has never been used — neither for training, nor for model selection. To this aim, the validation methods can be nested. In the decision tree example for instance, we could split the whole dataset into three parts. The first one is used for training; the second one is required for choosing the nodes to be pruned. The performance of the model is finally estimated over the third part.

Cross-validation can be nested too, for instance with the following algorithm:

- Split the dataset $D$ into $k_1$ equal-size subsets, $s_1, \ldots, s_{k_1}$.
- For each $i \in k_1$
  - Create $D'_i = D \setminus s_i$
  - Split $D'_i$ into $k_2$ equal-size subsets, $t_{i1}, \ldots, t_{ik_2}$
  - For each $j \in k_2$
    - Train the algorithm on $D''_{ij} = D' \setminus t_{ij}$
    - Obtain model $M_{ij}$
  - Choose the model selection parameters by evaluating each $M_{ij}$ upon $t_{ij}$
  - With these parameters, train the algorithm on $D'_i$ and obtain model $M_i$
- Estimate the overall performance by evaluating each $M_i$ upon $s_i$

Once the performance estimate has been obtained, the final model on all the data can be build by applying a single cross-validation for model selection.

## 2.2 Logistic Regression

One of the most common data analysis methods in medicine is logistic regression (LR) [59]. Logistic regression is usually not considered a Machine Learning algorithm, since it originates from the field of Statistics. However, the procedure perfectly fits inside the supervised learning paradigm — and LR in fact can be seen as a very simple neural network model [106].

Logistic regression is used to describe a categorical output variable from input variables. The output typically has just two possible outcomes (dichotomous variable), which describe the presence or absence of a certain condition (a certain disease, for instance), and are labelled with 0 and 1. The theory is however extensible to more than two outcomes.

We could simply treat this data as real-valued data: it is sufficient to substitute the classes with distinct real values (0 and 1 are the most practical choice). The data could now be analyzed with regression methods — linear regression, for instance. With these methods however our prediction would span over all of $\mathbb{R}$, instead of only $0, 1$.

The goal of logistic regression is to constrain the output in the range $[0, 1]$, while maintaining a linear relationship with the input; moreover, it works on the *probability* to have a certain output given an input, giving rise to useful interpretations of the results.

The model is based on the following equation:

$$\mathbb{P}(G = 1|X = \mathbf{x}) = \frac{1}{1 + e^{\alpha + \boldsymbol{\beta}\mathbf{x}}} \tag{2.6}$$

where $\alpha$ is a scalar, and $\boldsymbol{\beta}$ is a vector; $\boldsymbol{X}$ is the input vector, while $G$ is the outcome. Since we want to build a probabilistic model, the relationship $\mathbb{P}(G = 0|X = \mathbf{x}) + \mathbb{P}(G = 1|X = \mathbf{x}) = 1$ holds. We can then rewrite equation 2.6 as

$$\log \frac{\mathbb{P}(G = 1|X = \mathbf{x})}{\mathbb{P}(G = 0|X = \mathbf{x})} = \alpha + \boldsymbol{\beta}\mathbf{x} \tag{2.7}$$

The quantity on the left of this equation is the *log-odds* of class 1 with respect to class 0: without the log, this ratio describes how much the outcome is more likely to be present (class 1), given that the input is $\mathbf{x}$.

Least mean squares (LMS) fit is not appropriate in this problem: the dependent variable is not continuous, so the desirable properties obtained with LMS do not hold (mainly because the error on the target variable is not normally distributed). The $\alpha$ and $\boldsymbol{\beta}$ parameters are then fit through maximum likelihood. This criterion chooses the model which explains the data with maximum probability — that is, the model which maximizes the quantity $\mathbb{P}(G|X)$. A rather simple derivation shows this model is the one which maximizes the quantity

$$\mathcal{L}(\alpha, \boldsymbol{\beta}) = \sum_{i=1}^{N} \left\{ y_i \boldsymbol{\beta} \mathbf{x}_i - \log\left(1 + e^{\alpha + \boldsymbol{\beta} \mathbf{x}_i}\right) \right\} \tag{2.8}$$

where $\mathbf{x}_i$ is the $i$-th input vector, and $y_i$ is its class (0 or 1). This value can be easily maximized with standard numerical methods (the Newton-Raphson algorithm, for instance).

The usefulness of logistic regression is not only in building a model which can predict unseen data, but also in the interpretation of the found coefficients. An important quantity relative to the input variables is the *odds ratio*; let's see for instance what this means for a single dichotomous input variable, $x_j \in \{0, 1\}$. In this context, the odds ratio is defined as the ratio of the odds for $x_j = 1$ to the odds for $x_j = 0$:

$$OR_j = \frac{\dfrac{\mathbb{P}(G = 1 | x_j = 1)}{\mathbb{P}(G = 0 | x_j = 1)}}{\dfrac{\mathbb{P}(G = 1 | x_j = 0)}{\mathbb{P}(G = 0 | x_j = 0)}} = e^{\beta_j} \tag{2.9}$$

This quantity describes then how much the outcome is more likely when the input condition is present ($x_j = 1$) with respect to when the input condition is not present ($x_j = 0$). This value is then clearly very important in deciding how much an input variable is influent on the final outcome.

If the input variable is continuous instead of dichotomous, the formula is actually quite similar: the odds ratio for an increase of $c$ units on the continuous variable $x_j$ is

$$OR_j = \frac{\dfrac{\mathbb{P}(G = 1 | x_j = v + c)}{\mathbb{P}(G = 0 | x_j = v + c)}}{\dfrac{\mathbb{P}(G = 1 | x_j = v)}{\mathbb{P}(G = 0 | x_j = v)}} = e^{c\beta_j} \tag{2.10}$$

## 2.3 Decision Trees

Decision trees [100, 19] are popular tools for classification and prediction. The models produced by decision trees combine high expressive power, allowing a wide variety of functions to be represented, with immediate readability:

the model is readily understandable by humans. Their main application is therefore in all those situations where the ability to explain the reason for a decision is crucial.

In a decision tree, internal nodes contain tests on the values of the attributes, with one branch and subtree for each possible outcome of the test. Leaves contain instead a classification value. A sample is classified by starting at the root of the tree, and following the branches where conditions on the attributes are fulfilled; when a leaf is reached, it provides the classification of the sample.

While this tree structure is common to all variants of decision trees, there can be differences in many areas:

- Node shape: the algorithm must describe what kind of conditions can be inserted into a node. The usual choice is to have a test on the value of a single attribute in each node. The outcome of the test can be binary, or multi-valued.

- Learning algorithm: each decision tree algorithm must specify the steps to be taken in order to decide which conditions to put at each node, and the overall shape of the tree. Usually, a greedy algorithm is applied, where the tree is recursively built in order to maximize at each step the value of a selection measure.

- Selection measure: most algorithms require a function which compares the "goodness" of the possible alternatives for node tests, in order to choose the best one. Common choices are entropy-related functions, and statistics (typically $\chi^2$) related functions. [84] and [22] provide an empirical comparison of various measures.

The typical decision tree construction algorithm starts with the empty tree, and all the training data. For each attribute, all the possible tests are evaluated; each test will produce a split in the training data. Consider for instance a simple boolean test, in a boolean classification task. If the initial data had $P$ positive instances and $N$ negative instances, two groups will be created after the split: the first group, fulfilling the test condition, with $p_1$ and $n_1$ positive and negative instances; and the second group, with $p_2 = P - p_1$ positive instances, and $n_2 = N - n_1$ negative instances. The selection measure used by the common ID3 and C4.5 algorithms [100] is based upon the entropy function:

$$E(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n} \tag{2.11}$$

The selection measure evaluates the previous split through the *information gain* measure:

$$I(P,N,p_1,n_1) = E(P,N) - E(p_1,n_1) - E(p_2,n_2) \tag{2.12}$$

This value is aimed at obtaining splits with high *purity* — meaning that the two sides should contain as much as possible instances with equal target. The information gain for all possible tests is therefore calculated, and the test with highest value is chosen as the root of the tree. The construction now proceeds recursively. The original training data is split according to the chosen test; the splitting algorithm is then recursively applied on both, obtaining a new test split which goes at the first level of the tree — and so on.

There are several criteria for stopping the recursion; clearly, once a subset contains only instances with equal target, it is pointless to further split it. However, often such a situation is reached too late, and causes the tree to overfit the data. This problem is usually addressed through pruning methods, which cut unnecessary branches of the tree after the building algorithm has terminated. A review of pruning methods can be found in [37]. A more general review of the research on decision tree is reported in [88].

Decision tree induction counts numerous applications in medicine, among which those described in [132, 67, 68].

## 2.4 Bayesian Belief Networks

Bayesian belief networks are powerful tools for modeling causes and effects [93]. They are compact networks of probabilities that capture the probabilistic relationship between variables. Bayesian belief networks are very effective for modeling situations where some information is already known and incoming data is uncertain or partially unavailable (unlike rule-based or "expert" systems, where uncertain or unavailable data results in ineffective or inaccurate reasoning). These networks also offer consistent semantics for representing causes and effects (and likelihoods) via an intuitive graphical representation. Because of all of these capabilities, Bayesian belief networks are being increasingly used in a wide variety of domains where automated reasoning is needed.

A belief network is a directed acyclic graph (DAG) which encodes the causal relationships between particular variables, represented in the DAG as nodes. Nodes are connected by causal links pointing from parent nodes (causes) to child nodes (effects).

As an example, a Bayesian network for diagnosing diseases may have a causal link from the node "cold" to "sneezing". This encodes the causal relationship "sneezing is caused by a cold". A node can have any number of parents; the nodes "hay–fever" and "allergies" may also point to the child node "sneezing".

The advantage of using conditional probability and belief networks over a joint probability distribution is that information is represented in a more understandable and logical manner, making construction and interpretation much simpler.

Nodes with one or more parents require their conditional probability distribution to be provided by way of a conditional probability table, which specifies the conditional probability of the child node being in a particular state, given the states of all its parents: $P(child|parent_1, parent_2, \ldots, parent_N)$. A conditioning state is used to specify the states of all the parents when specifying an entry in the child node's conditional probability table. Nodes without parents only require a prior probability distribution, $P(node)$.

As mentioned earlier, the conditional probability is a summarized form of the joint probability distribution. The relationship between joint and conditional probability is shown in Equation 2.13, where the function $\text{Parents}(X_i)$ represents the set of nodes with causal links directed into the node $X_i$.

$$P(X_1, X_2, \ldots, X_N) = \prod_{i=1}^{N} P(X_i|\text{Parents}(X_i)) \tag{2.13}$$

The output of a Bayesian network employed in a classification task is naturally stochastic, since it corresponds to the probability distribution of the output variables.

Before evidence can be added and beliefs extracted from a Bayesian network, it must first be created. This can be done following different approaches. In many cases, the initial design of the network requires the help of an "expert" to specify the correct causal relationships, whilst the conditional probabilities are determined by an algorithm using training data. A substantially more complex algorithm may even be able to determine the causal relationships between variables and therefore automatically generate an appropriate network structure.

Bayesian networks have been used in medicine for many years and have demonstrated to be well-suited for handling the uncertain knowledge involved in establishing diagnoses of disease [50], in selecting optimal treatment alternatives [78], and predicting treatment outcome in various different areas [118]. Examples include the use of Bayesian networks in clinical epidemiology for the construction of disease models [2] and within bioinformatics for the interpretation of microarray gene expression data [44]. However applications show that they are mostly successful when they can exploit previous knowledge and a large amount of data is available. In other cases other methods can be more efficient.

## 2.5 Evolutionary computation

Genetic Algorithms date back to 1975, when Holland introduced them [51]: after that, the original proposal expanded into a whole research field called *evolutionary computation*. The evolutionary keyword is of primary importance to understand how they work: in fact, much of the inspiration is taken from Darwin's evolution theory.

A genetic algorithm works not with a single hypothesis, but with a population of competing hypotheses (which in the GA jargon are called individuals). Like in nature, better individuals have greater choices to survive, and to propagate their genetic makeup to their children. In nature, "better" is related to the ability to find food, escape predators, and mate; all together, this measure of *fitness* is dependent on the level of adaptation of the individual to the environment it lives in.

In evolutionary computation, since every individual is a hypothesis, its fitness is related to how well the hypothesis explains (*fits*) the data. A genetic algorithm starts typically with random hypothesis; some of them will casually have a better-than-average fitness, and will have greater chances to survive, and evolve towards even better solutions.

But how is evolution achieved? In nature, evolution comes from the combined action of random variations, and selection pressure. GAs again parallel this behaviour, providing two sources of variation: mutation and crossover. When an individual is selected for survival, it has two possibilities to carry its genetic makeup to the next generation: sexual and asexual reproduction. In the first case, the individual mates with another selected one; two new individuals are created, constituted by a random recombination of the genetic makeup of the parents (crossover). In the second case, the individual is "cloned" to the next generation. In both cases there is a small chance of a mutation happening during the process; mutations are the only variation mechanism in asexual reproduction. It is important to note that the single individual cannot change its genotype, or in other words learn: learning comes from the adaptation of the whole population through generations.

As regards selection pressure in GA, a few options exist; the common factor is that higher-fitness individuals have greter chance to reproduce. In *fitness proportionate* selection, each individual has a probability to be selected equal to its fitness, divided by the sum of the fitness value of all the individuals. This mechanism is simple, but has the disadvantage of being dependent on the scale of fitness values. For instance, consider a population with average fitness value of 1; an individual with fitness 2 will have twice the probability to be selected with respect to the average individual. If however the average fitness was 11, an individual with fitness 12 would have selection probability very close to the average — while maintaining the same absolute fitness improvement. *Tournament selection* escapes this problem. In its most general form, it involves two parameters: $t \in \{2, 3, \ldots\}$ and $p \in [0, 1]$. When an individual must be selected, $t$ are chosen at random from the population, and sorted in decreasing fitness order. The first individual is then returned with probability $p$, the second with probability $p(1-p)$, the third with probability $p(1-p)^2$, and so on, until the last which is chosen with probability $(1-p)^{t-1}$. Since the rank matters, rather than the absolute fitness value, this method does not have the drawbacks of the former. It however requires two parameters to be chosen (although the $p$ value is commonly taken as 1).

The problem-solving power of genetic algorithms has been ascribed to two main factors:

- **Ability to escape local minima.** Using a population instead of a single hypothesis, and allowing also lower fitness hypotheses to survive (although with a lower chance), gives genetic algorithm the chance to avoid getting trapped in local minima.

- **Building blocks.** The crossover method could allow to take the good parts of two average-fitness solution and combine them, resulting in a good advance in fitness. This issue is however still quite debated, and certainly depends greatly on the problem, and on the representation of individuals.

### 2.5.1 Implementations

Evolutionary computation, as we just described it, is only a search algorithm. In order to obtain a complete ML algorithm it is necessary to specify the shape of individuals, and the mutation and crossover operators. This leads to varying degrees of expressiveness and almost arbitrary shaped hypotheses.

In the original *Genetic Algorithms* for instance, individuals are structured as sequence of bits, which the learning algorithm interprets in order to build the solution [46]. GAs have been widely used in medicine, often in combination with other machine learning techniques — like neural networks, or Bayes networks. See for some examples [130, 92, 62].

In *Genetic Programming* [66] instead, individuals are mathematical functions or computer programs, which are themselves a solution to the problem. The first is more suited to problems where the structure of the final solution can be decided a priori, before searching it. The latter is on the other side much more flexible, finding itself the structure of the final solution given only the basic building blocks; this comes at the usual price of more variance in the results, since the search space is much bigger. For examples of using GP in medicine, see [96, 47].

## 2.6 Learning Classifier Systems

Learning Classifier Systems (LCS) [52] exploit the ability of the Genetic Algorithm at their heart to search efficiently over complex search spaces. In a Learning Classifier System each individual is basically a production rule, providing a level of readability that is rarely found within subsymbolic approaches without the need for additional post-processing. LCS combine reinforcement learning, evolutionary computing and other heuristics to produce adaptive systems. They maintain and evolve a population of classifiers (rules) through the genetic algorithm. These rules are used to match environmental inputs and choose subsequent actions. Environment's reward to the actions is then used to modify the classifiers in a reinforcement learning process. When used to classify, the set of rules provides a deterministic answer.

### 2.6.1 XCS

XCS is an evolution of Learning Classifier Systems proposed by Wilson [122, 123] which demonstrated to perform very well in comparison to other machine learning techniques [4]. XCS introduces a measure of classifiers' fitness based on their accuracy, which is defined as the reliability of their prediction of the expected payoff; this definition is different from traditional LCS systems, where fitness is based directly on the payoff. A second chief difference is that the GA is applied only on the action set, the subset of classifiers which leads to the choice of the action. The combination of these two characteristics gives the system a strong tendency to develop accurate and general rules to cover problem space and allow the system's "knowledge" to be clearly seen. In the following we provide a brief description of XCS. For full details, see [26].

The core component of XCS is a set of classifiers, that is condition-action-prediction rules, where the **condition** specifies a pattern over the input states provided by the environment, the **action** is the action proposed (e.g. a classification), and the **prediction** is the payoff expected by the system in response to the action. Additionally each classifier has associated an estimate of the **error** made in payoff predictions, and a **fitness** value.

XCS implements a reinforcement learning process: at each step the system is presented a sample from the data set and it examines its set of classifiers to select those matching the input situation. These classifiers form the *match set*. Then for each possible action the system uses the fitness–weighted average prediction of the corresponding classifiers to estimate environmental reward. At this point, the XCS can choose the best action looking for the highest predicted reward. However, during learning, the action is usually selected alternating the previous criterion with random choice, useful to better explore the problem space. The actual reward returned by the environment is then used to update the classifiers in the *action set*, i.e. the subset of the *match set* corresponding to the selected action. A genetic algorithm is also executed on this set to discover new interesting classifiers.

To reduce the number of rules developed, XCS implements various techniques, such as the use of macroclassifiers, the subsumption and the deletion mechanisms. In fact the system uses a population of macroclassifiers, i.e. normal classifiers with a **numerosity** parameter, representing the number of their instances (microclassifiers). This helps in keeping track of the most useful rules and improves computational performance at no cost.

Subsumption is used to help generalization: when the GA creates a new classifier whose condition is logically subsumed by one of its parents (i.e. matching a subset of the inputs matched by the parent's) it is not added to the population, but the parent's numerosity is incremented. A similar check is also occasionally performed among all the classifiers in the current action set.

Finally the deletion mechanism keeps the number of microclassifiers under a fixed bound. The classifier to be removed is chosen with a roulette wheel selection biased towards low–fitness individuals and assuring approximately equal number of classifiers in each action set.

As already stated this process leads to the evolution of more and more general rules. For each classifier we can define a measure of generality following [126], ranging from 0 (most specific) to 1 (most general). A possible termination criterion is to stop evolution when the average generality value of the population gets stable.

### Missing values

Missing values in the dataset are a common issue in classification problems, leading to a degradation of the performance of essentially every classification technique. Many machine learning algorithms do not natively deal with missing data, and require all values to be present. In such cases, two options are usually followed. The simplest choice is deletion: either the instances containing missing data (case-wise deletion), or the attributes (attribute-wise deletion) are removed from the dataset — whatever causes lower loss of information. A more complex option is *imputation*: the missing value is replaced with a statistically meaningful value. This latter option can be implemented with varying degrees of sophistication; see [104] for a more detailed approach to the problem.

The traditional way of dealing with missing data in LCS is to treat the missing value as a wildcard. Tests performed on a missing value will therefore always pass, regardless of the content of the test [56]. With this approach, a sample with missing data will be matched by all classifiers which *could* match it, if it was complete. Other possibilities reported in literature involve some form of imputation [58].

#### Data type integration

Another key aspect of XCS is the ease of integration of different kind of data. In fact, whilst the original formulation of XCS is targeted to binary input, the shift to other data types, such as real or integer ones, has already been proved to be very easy (see respectively [126, 124]).

#### Ruleset reduction

During learning XCS tends to evolve an accurate and complete mapping of condition-action-prediction rules matching the data. Consequently, in particular on very sparse data set, the final number of rules is quite high. Similar problems, which break the knowledge visibility property, were experienced in various studies on "real" data sets [126, 125]. These works suggest to let the system evolve many steps after reaching the maximum performance, and then to extract a small subset of rules which reach the same performance level using the *Compact Ruleset Algorithm* (CRA), first proposed by Wilson [125].

### 2.6.2 EpiCS

The EpiCS system is a Learning Classifier System designed for knowledge discovery in epidemiologic surveillance [55]. Having a common ancestor with XCS (the NEWBOOLE system [16]), the two systems share many characteristics. The primary difference is that XCS is *accuracy*-based, while EpiCS is *strength*-based. Briefly, with the latter system rule fitness is based upon the amount of reward the rule receives from the environment; in the former system, fitness is based upon the precision with which the rule predicts the reward it will receive. They use different heuristics for fitness calculation, and EpiCS implements explicit methods for controlling over- and under-generalization which are implicit in XCS.

The most interesting difference however regards the ability of EpiCS to produce risk estimates, rather than simple classification. EpiCS calculates risk estimates for an instance through the proportion of classifiers in that instance's match set bearing the same class prediction. This idea has the advantage of being easily adaptable to many LCS systems: in fact, it has been employed in the XCS system, creating the EpiXCS algorithm [57]. It has however the drawback to complicate understandability of the system decision, since the risk estimate is demanded to a diverse set of rules for each instance.

### 2.6.3 UCS

UCS [9] is a derivation of XCS tailored for an explicit supervised learning setting. The main difference with XCS regards the definition of fitness. In UCS, fitness of a classifier is directly based upon the estimated precision of its class prediction. The concept of reward is still present, but only the classifiers able to score on average high reward will have a high fitness, and thus be able to survive the evolutionary process. In XCS instead, it is sufficient that the classifier correctly predicts its expected reward — even if this is consistently low — in order to reach high fitness. UCS will then generate a best action map, while XCS produces a complete action map.

## 2.7 Neural Networks

Neural networks currently represent one of the most powerful and general models developed by Machine Learning [49, 10]. Their high flexibility allows them to be successfully applied in several domains, including regression, classification, and pattern recognition problems. Many real-world datasets have been analyzed with Neural Networks, obtaining performances often at the state of the art in the Machine Learning community. An interesting survey on the application of NN to medical analysis, both for diagnosis, prognosis and survival analysis functions, is provided in [71].

Neural networks take inspiration from studies of the structure of neurons in the brain. A networks is made of a set of interconnected neurons. The structure of the connections defines the *topology* of the network; the most common and studied structure is the multi-layer feedforward network. Here the neurons are organized in ordered layers; each neuron receives inputs from the neurons in the previous layer, and sends outputs to the neuron in the following layer. The first layer it the *input layer*, which receives data from the dataset. The last layer is the *output layer*, which provides the results.

Each neuron performs a relatively simple calculation, by summing the values on its inputs, weighted by coefficients on the connections. The sum is then passed through the activation function; its result constitutes the output of the neuron. If the networks employs non-linear activation functions, it has been demonstrated that it constitutes an universal approximator — meaning that it can approximate arbitrarily well any continuous function. Interestingly, if the activation function is logistic, a Neural Network can be regarded as a generalization of the logistic model [33].

NN store the knowledge acquired from the dataset in the connection weights. Adaptation of the weights to the dataset —i.e. learning — is usually performed through gradient descent techniques, which are allowed when the activation function is derivable. The most common of such techniques is backpropagation. Roughly speaking, then backpropagation algorithm works by feeding an instance into the network, calculating the error as difference between the expected and actual output, and adjusting the weights of the network by propagating the error backwards through the neurons.

The high explanatory power of Neural Networks comes with some disadvantages.

- No tolerance of missing data
- Low interpretability
- Non-determinism

Neural networks do not allow missing data: each neuron must have some sort of input. They therefore require imputation techniques to be used, which fill in missing values [6]: such a procedure can however lead to creation of artifacts in the data.

The neural network model is a *black-box* model. It is already very complex to write down the formula calculated by a network — let alone interpret it. Several attempts at solving this problem have been made, with varying degrees of success [3, 34]. It is acknowledged however that a loss of performance is inherent to the simplification of the model into an understandable form.

While the standard back-propagation algorithm is deterministic, it requires a starting point, which is chosen arbitrarily (typically at random). Several executions of the algorithm on the same data are therefore not guaranteed to produce the same result. Neural Networks share this characteristic with more classical stochastic algorithms, like evolutionary algorithms.

Another difficulty in using Neural Networks regards the choice of the network structure, which is originally left to the user's experience. Techniques exist to train structure together with the weights (for instance, Cascade Correlation [38], tiling [80], upstart [41]), or to prune the network after it has been trained by removing unnecessary nodes and connections [101].

Recently, NN methods have been extended to deal with complex data belonging to structured domains, including sequences, trees and classes of graphs (Recurrent and Recursive Neural Networks) [111, 81]. The applications of such models allows to consider more complex types of data, able to facilitate the integration of medical chemistry, biological and clinical data.

## 2.8 Support Vector Machines

Support Vector Machine (SVM) and kernel-based approaches have recently emerged as a major topic within the field of machine learning [120, 23, 110].

Like Neural Networks, SVM, can be used for pattern recognition, classification and regression tasks to learn from data an high-dimensional nonlinear hypothesis.

For the purpose of this survey, we could therefore include this model in the same framework of the NN approach. The basic difference between SVM and other ML models similar to NN is in the inner methodology of learning, rather than in how they are applied. Although is known that there is no general optimization method for learning tasks, interestingly, SVM and kernel approaches are principled learning methods originated from the foundations of the statistical learning theory [120] with the aim to provide good generalization properties to the methods.

The original linear machine built by SVM can be extended including nonlinearity (in the original input space) by implicitly embedding the data into an internal feature space through a *kernel* function. Through the choice of kernel function different hypotheses can be considered, specifying for instance kernels belonging to the classes of polynomials, Gaussian or Radial-basis functions (RBF), or more complex cases such as two-layer Neural Networks type of learning machines. However, it should be noted that the definition of a kernel function corresponds to the definition of a prior similarity measure on data; hence, the choice of the proper kernel is a critical issue in the application of the method.

The SVM algorithm is strongly dependent on some parameters which must be decided by the researcher. One of them, usually called $C$, controls the tradeoff between margin maximization and error minimization. Other important parameters are usually dependent on the choice of the kernel function. The simplest approach involves trying various combinations and choosing the one which leads to more satisfactory performance, although more complex strategies have been discussed [27].

## 2.9 Summary

In this chapter, some established statistical and Machine Learning algorithms for supervised data analysis have been described. The requirements and outputs of each algorithm have been presented. In the next chapter, we will summarize where in the field of medicine these algorithms can be applied, and what are the typical requisites of this domain. We will then see how most algorithms fail to satisfy some requisites; this prompted us to develop the HCS algorithm, which will be described in chap. 4.

# Chapter 3

# Medical data analysis

The characteristics of the data clearly depend on the problem being analyzed. Medical data sets however have some recurring specificities, which are interesting to summarize in order to better understand the typical requirements of medical problems.

- Medical data is *heterogeneous.* Among the various recordings on each patient, there can be real values with different ranges, integer values, ordered or unordered classes. There can be images, variable-length strings; there could even be some non-standardizable natural language text (the physician's conclusions for a certain set of tests, for instance).

  It almost impossible for a single technique to handle all of this variety of data types. On the other side, techniques which require only homogeneous data are of limited usefulness in medical data analysis.

- Medical databases are *incomplete.* Collection of data is generally a byproduct of medical care, rather than an objective in itself; altough completeness is generally required, it is very rarely achieved. There can be technical or economical reasons for which a value is not recorded; or even motivations pertaining the patient's health itself. Certain values for instance could require dangerous tests, which are performed only when considered strictly necessary. Here, the fact that a value is missing is informative on the concept to be examined; these kinds of missing values must then be treated with much care, in order to avoid introduction of bias.

Often however, data in medical records are missing with no specific connection to the value of the data itself, or to the target concept. This is common for instance in retrospective studies, where the data is collected from old patient records, which are often incomplete since they were collected for different objectives. In this situation, missingness does not influence the concept under examination. A good methodology to manage medical data must then be prepared to deal at least with this kind of missing values. For a more precise classification of missing values and their treatment, we refer to [73].

- Medical data is inherently *noisy*. Not only the recorded values can be approximate or uncertain; even the classification can be imprecise or wrong. Noise tolerance is then a primary requirement for analysis.
- Medical problems can show *high dimensionality*. As we noted in the introduction, medicine is trying to consider complex interactions between many factors, in order to reduce prediction error. Moreover, some tests generate large quantity of data alone; think of computer tomography, or microarray analysis.
- Medical data is often *unbalanced*. The class of people who have oral cancer for instance is certainly less numerous than the class of people who do not have it. Learning algorithms then should not suppose that the attributes have a balanced or normal distribution.
- Investigation results must be *interpretable*. Opaque methods, which can not show in human-readable way the reasoning behind their answers, are unlikely to be accepted and used by physicians — even if they demonstrate a very good performance. Understandability of the model is at least as much important as performance itself.

Managing medical data finally presents other issues, still important but less relevant to our goals (ethical and legal, for instance). An enlightening review on the topic can be found in [28].

## 3.1 Personal tasks in medicine

Medical practice is concerned with curing single patients. This involves detection of a disease before clinical evidence appears (screening); finding out what their disease is (diagnosis); predicting how the disease will develop (prognosis); deciding which kind of medical treatment is best for the patient, in order to cure him (treatment selection). It is important to notice that all these steps take into account the patient's status only, and are generally not concerned with the original causes of the illness — unless they are considered important for the prognosis.

### 3.1.1 Screening

Screening is the process of looking for a disease over a population with no clinical indication of illness. In some kinds of disease, early detection can greatly improve the effectiveness of cures: screening can therefore be a very valuable tool in improving the overall population health.

Since screening is performed on large amounts of people, it generally applies simpler tests that medical diagnosis. Such tests can report both false positive and false negative results, with respect to the proper diagnostic test (often called "gold standard"). Estimation of specificity and sensitivity of the test is then necessary, along with the prevalence of disease in the group the patient belongs to, in order to correctly interpret the result [48].

Machine learning in this setting can be helpful to devise new tests, and evaluate their effectiveness. Some reports include [108] and [74].

### 3.1.2 Diagnostic reasoning

A central problem in medicine is the diagnosis of the disease an individual patient is suffering [76]. This in essence amounts to the construction of a hypothesis about the disease, based upon a set of indirect observations from diagnostic tests. Diagnostic tests, however, generally do not serve to unambiguously reveal the condition of a patient: the tests typically have true-positive and true-negative rates unequal to 100%. To avoid misdiagnosis, the uncertainty in the test results obtained for a patient should be taken into consideration.

The above–mentioned issues make the diagnostic problem a very suitable and interesting one for computer–based systems [114]. In fact, the historical development of machine learning and its applications in medical diagnosis shows that from simple and straightforward to use algorithms, systems and methodology have emerged that enable advanced and sophisticated data analysis [65]. In the future, intelligent data analysis will play even a more important role due to the huge amount of information produced and stored by modern technology.

To assist physicians in the complex task of diagnostic reasoning, test-selection methods are required to indicate which tests should be ordered to decrease the uncertainty about the disease present in a specific patient. A test-selection method typically employs an information-theoretic measure for assessing diagnostic uncertainty. Such a measure is defined on a probability distribution over a disease variable and expresses the expected amount of information required to establish the value of this variable with certainty. An example measure often used for this purpose is the Shannon entropy. The measure can be extended to include information about the costs involved in performing a specific test and about the side effects it can have. Since it is computationally hard to look beyond the immediate next diagnostic test, test selection is generally carried out non-myopically, that is, in a sequential manner. The method then suggests a test to be performed and awaits the user's input; after taking the test's result into account, the method suggests a subsequent test, and so on.

### 3.1.3 Prognostic reasoning

In a prognostic process, patient's information is gathered and interpreted to predict the future development of the patient's condition [75]. Due to the predictive nature of this process, prognostic systems are frequently used as tools to plan medical treatments. As knowledge of the future is inherently uncertain, in prognostic reasoning uncertainty is even more predominant than in diagnostic reasoning. Another prominent feature of prognostic reasoning when compared to diagnostic reasoning is the exploitation of knowledge about the evolution of processes over time.

The outcome predicted for a specific patient is generally influenced by the particular sequence of treatment actions to be performed, which in turn may depend on the information that is available about the patient before the treatment is started. The outcome is often also influenced by progress of the underlying disease itself. The outcome of interest may be expressed by a single variable, e.g. modelling life expectancy, but it may be more complex, modelling not just length of life but also various aspects pertaining to quality of life. A subset of variables may then be used to express the outcome.

Approaches to developing prognostic models vary from using traditional probabilistic techniques, originating from the field of statistics, to techniques based on more complex models, originating from the field of artificial intelligence [134, 29].

### 3.1.4 Treatment selection

Another area for which automatic methods are favorable is treatment selection, that is the process of deciding upon the most appropriate treatment alternative for a specific patient. Reasoning about different treatments, however, involves reasoning about the current situation of a patient and the effects to be expected from the treatments. It thus involves diagnostic reasoning and, even more prominently, prognostic reasoning. The reasoning algorithms are therefore often embedded in a decision-support system that offers the necessary constructs from decision theory to select an optimal treatment given the predictions [77, 2].

## 3.2 General tasks in medical research

Medical practice focuses on a single patient, diagnosing his illness and prescribing a cure. Medical research is on the contrary primarily interested in what happens in the "general" case; the single patient has value only if it is contained in a collection of similar cases, where a global hypothesis can be formulated and tested.

The main objective of data analysis in medical research is therefore not to provide exact answers for single patients: at best, this can be considered a byproduct. The real interest is in *understanding* why some phenomenon occurs. This knowledge can then be exploited by physicians for instance to find strategies aimed at preventing diseases, or design better cures. On the contrary, in such a setting a black-box model of the data is not useful, since it cannot give explanations to the user.

Interpretability is then a chief requirement of any model attempting to describe medical data in such tasks [69]. The model should allow to convey information about its decisions in a straightforward manner.

In our opinion, another important requirement of an interpretable model is its *stability*. We define a stable algorithm as one which will produce very similar result when confronted with data extracted from the same distribution. Notice that often stability is defined through accuracy, meaning that the algorithm produces consistently similar accuracy values [17, 36]. This does not imply that the algorithm found a similar explanation of the data: it simply means that it will produce explanations with similar precision. Our concern is instead related to the actual decisions of the classifier, as in [116].

Many researchers in Machine Learning overlook this problem, either ignoring it, or believing that many different classifiers with similar accuracy can give a better understanding of the data [87]. We actually believe the opposite is true. While two or three possible explanations of the data can actually shed more light on the underlying relationships, more and more explanations will only increase confusion in the user, and decrease confidence in the model. Too much information, without a sensible method to discern useful from useless, is equivalent to no information (in the information retrieval context, this is often called *information overload* [8]).

### 3.2.1 Studying risk factors: epidemiology

Diagnosis, prognosis, treatment selection are all activities which involve curing an already developed disease. Epidemiology works on the opposite side, by studying the factors involved into health and illness. Defining the diseases, drawing disease causal chains, and formulation of health strategy are important aspects of epidemiology. Epidemiological information is then used to plan and evaluate strategies to prevent illness, and as a guide to the management of patients in whom disease has already developed. For all these aims, the single patient does not provide enough information. Epidemiology requires a population, where characteristics and disease development can be correlated and relevant causes ascertained and separated from irrelevant factors.

In observational epidemiology, three kinds of study design are employed. The simplest is the *cross-sectional* study: both the exposure status and the disease are measured at a given time. The prevalence of disease in different exposure groups can then be compared, in order to identify possible correlations. This type of study however cannot prove causation. To this aim the more onerous *cohort study* can be employed. Here, a population of initially healthy, exposed subjects is followed through time, along with initially healthy, unexposed subjects. Analyzing the incidence of disease in the two groups over time can then support causality relationships. This kind of study has the drawback of being much onerous, particularly for diseases having a long period between first exposure and manifestation. One possible remedy is to conduct the study retrospectively, by collecting past data. When this is not possible, a *case-control* study can be set up. With this setting, the study originates from a set of diseased subjects (cases). This set must then be matched by a suitable set of controls: subjects not showing the disease, but having similar statistical distribution. Analysis is then performed in order to show relationship between exposure and disease.

In all cases, once data has been collected, a data analysis methodology must be applied. This is where Machine Learning can help, by suggesting possible relationships between risk factors and disease development, which could have been otherwise overlooked with standard data analysis techniques. An interesting example of application of ML to predict breast cancer susceptibility can be found in [72].

### 3.2.2 Discovering functional interactions

As the amount of biological and medical data is more and more growing, methods able to discover functional interactions among them are of the greatest interest. The kind of data to analyze can vary from clinical databases collected in research and health-care centers to genetic data, and can even be a mixture of the two. Moreover, since data are often collected over time, it is possible to analyze the temporal patterns to reveal how the variables interact as a function of time.

The discovery and the study of genetic interactions is central to the understanding of molecular structure and function, cellular metabolism, development of cells and tissues, and response of organisms to their environments. If such interaction patterns can be measured for various kinds of tissues and the corresponding data can be interpreted, potential clinical benefits are obvious and novel tools for diagnostics, identification of candidate drug targets, and predictions of drug effectiveness for many diseases will emerge.

For example, finding interactions between genes based on experimentally obtained expression data in microarrays is currently a significant research topic. Microarrays [105] allow for the study of expression of thousands of genes simultaneously, so to be interpreted they obviously require knowledge discovery tools ranging from clustering techniques to supervised learning methods.

## 3.3 Machine learning in medicine

There are many areas in Medicine where Machine Learning can be applied. Clearly understanding the setting where ML will be applied, along with the result which is expected from the doctors, is a fundamental step in constructing a successful collaboration. The traditional approaches to medical data analysis have the strong advantage of being well-known and thoroughly tested, with a robust corpus of theoretical studies to their support. New algorithms can however complement them, by providing novel ideas, and approaching problems from a different perspective. It is with this objective in mind that we designed HCS: it is thought as a tool to explore data with a non-linear, interpretable model, while keeping in mind the requirement of seamless treatment of different types of data, and ability to cope with missing values.

# Chapter 4

# HCS

Chapter 3 described the requirements of medical data analysis. Our interest will be focused on the use of computational tools for the field of medical research. This means that we have all the typical peculiarities of medical datasets, with the adjoint constraint of clear and immediate understandability of the model inferred from the data.

Most classical machine learning approaches fail to fulfill at least one of the requirements for this particular setting. Numerical methods, like neural networks and SVMs, can represent a very wide variety of functions, and are therefore able to capture many diverse target concepts; they create however black-box models, which require further processing in order to produce understandable knowledge (e.g. see [34]). After the effort for appropriately training the model, a new task starts which seeks to answer the question "Why did the model take such a decision?". Another often disregarded requirement is their need for *complete* data — no missing values are allowed. Medical datasets almost invariably fail this condition. To cope with this shortcoming, typically imputation methods are applied, which "fill" the missing data [6]. Alternatively, missing values are encoded with a special value, different from the others. Both solutions can skew the original data distribution, or anyway add extra noise to the dataset; an algorithm natively able to cope with missing data can therefore prove to be a better alternative.

Among classical methods, decision trees [100] exhibit all the characteristics required by medical data analysis. They are highly interpretable, and treat missing values in a sensible way: there is no need for imputation methods. We were however not entirely satisfied with their approach for a few reasons:

- They originally aim at a perfect division between classes. Such a result is possible in certain areas of medicine (diagnosis is a good candidate), but is not realistic in other situations, like epidemiology, where a risk estimate is more appropriate. Decision trees ultimately produce such a risk estimate, but only after pruning occurred, and only as a fallback of not being able to completely split the classes.

- The tree construction algorithm takes decisions only on a single attribute at each node. Interactions between different attributes come as a by-product of search, and are not immediately evaluated and explored.

The above considerations prompted us to develop a learning algorithm tailored for risk analysis in medical datasets. Such an algorithm must cope well with diverse data types and missing data. It must take into account interactions between various attributes, and produce readily interpretable results. XCS [122] showed most of the above characteristics, but was aimed at accurate predictions rather then probabilistic risk estimates, and applied to the broader field of reinforcement learning, instead of plain supervised learning. A modified version of XCS, EpiXCS [57], was suggested to specialize XCS for risk prediction. While being able to deliver accurate risk score, this algorithm does not directly pursue them during search, retaining XCS accuracy-based learning strategy.

The algorithm we propose is called HCS, which is an acronym for Hypothesis testing with Classifier Systems. It originated as a variation upon XCS, but it has finally evolved into a hybrid between classifier systems and decision trees.

## 4.1 Statistical hypothesis testing

HCS employs statistical hypothesis testing as its main data exploration tool. In this section we will very briefly introduce the subject, pointing to statistics textbooks for a more detailed analysis [128, 70].

According to Popper [97], science is strictly connected with *falsifiability*. Each theory which claims scientific soundness must be falsifiable: it must be stated in such a way to admit the possibility of performing an observation, or an experiment, which can contradict it. If such a situation arises, the theory must be revised and corrected, and an improved theory which accounts for the unexpected observation is formed. Statistical hypothesis testing provides a procedure to follow in order to take such a decision, when the original theory is probabilistic in nature.

Consider a situation where we formulate a probabilistic hypothesis describing the general behaviour of a specific phenomenon. We then acquire a certain number of observations of the phenomenon, and we want to establish whether these observations disprove or do not disprove the initial hypothesis.

The following steps define the statistical hypothesis testing procedure:

- Mathematically formulate the hypothesis, called *null hypothesis* and marked with $H_0$, in such a way that it is possible to calculate the probability to obtain each possible outcome of the observation.

- Establish the *critical region* (or *rejection region*), that is the set of possible outcomes which we regard as evidence against the hypothesis. The probability of the observations to fall in this region, assuming that the null hypothesis is true, is called *significance level* of the test, and is marked with the symbol $\alpha$; it is therefore the probability to wrongly reject the null hypothesis.

- Perform the observation of the phenomenon, and check whether it falls inside the critical region or not.

The null hypothesis $H_0$ is rejected (disproved) if the observation falls inside the rejection region, and accepted (not disproved) otherwise. Notice that acceptance must not be treated as evidence in favour of the null hypothesis; it's just lack of evidence against it.

As a simple example, consider the theory "this coin is well-balanced". Such a theory can be mathematically formulated as "the probability to obtain heads or tails is 1/2". In order to verify the theory, we will perform an experiment consisting of 100 coin tosses, and observe the number of resulting heads. Before starting the experiment, we decide that we will reject the hypothesis if we obtain less than 40, or more than 60 heads. The significance level of this experiment can be calculated through the binomial distribution, and is $\alpha \simeq 0.035$.

The significance value $\alpha$ measures the probability to commit a type I error — that is, the probability to reject $H_0$ in a situation where the hypothesis was correct. Clearly, this value should be kept as low as possible. However, lowering the value generally increases the probability to commit a type II error: failing to reject the hypothesis, when in fact it was wrong. This is intuitively clear: if we require only extreme evidence against the null hypothesis in order to disprove it, we will accept it in many situations where it is actually false. The probability to reject $H_0$ when it is actually false is called *power* of the hypothesis testing procedure.

In the coin-tossing example, $\alpha \simeq 0.035$ means that, if the coin is indeed perfectly balanced, if we repeat the 100-tosses experiment 1000 times, we will reach the wrong conclusion (the coin is biased) 35 times. On the other hand, if the coin is only slightly biased, it will be difficult for us to detect the anomaly. For instance, using a biased coin with 0.6 probability to flip a head, the power of the tests turns out to be $\simeq .46$. Upon repeating the 100-tosses experiment, roughly half of the time we would not be able to detect the bias.

Another quantity strongly connected to significance and power is the *sample size*. Since we deal with probabilistic phenomena, a single observation is generally not sufficient to draw conclusions. A group of observations is then performed, and the probability of the whole group to occur is considered. Changing the size of the group (the sample size) creates a test with different characteristics. Again in the previous example, tossing the coin 200 times instead of 100, and building a rejection region with similar significance level ($\leq 85$ or $\geq 115$, $\alpha \simeq .04$) yields a test with higher power: in the case of a 0.6 biased coin, power is $\simeq .79$. When the sample size is under control by the statistician, a test can be structured to reach both required significance and power.

When sample size is externally decided, it is necessary to choose a tradeoff between significance and power. Statistical tests are usually designed to maximize the power, while maintaining a specified significance level. In many occasions, this aim naturally leads to the definition of a nested family of rejection regions, with varying confidence levels. Calling $\mathcal{R}_\alpha$ the rejection region with $\alpha$ significance level, we have

$$\forall \alpha_1 < \alpha_2 . \mathcal{R}_{\alpha_1} \subseteq \mathcal{R}_{\alpha_2} \tag{4.1}$$

If the tests has this property, it is possible and advisable to avoid the arbitrary decision about the significance level $\alpha$, and report instead the *p-value* of an observation. This value is defined as the probability of the smallest rejection region containing the observed result $X$:

$$p(X) = \min\{\alpha | X \in \mathcal{R}_\alpha\} \tag{4.2}$$

This practically means that, in order to follow the proper definition, the rejection region should include the observed result and all "more extreme" results — that is, the results which would have provided even more clear evidence against the null hypothesis than the current one.

The $p$-value can be informally considered as a measure of the degree to which an observation disproves the null hypothesis. The HCS learning algorithm mainly revolves around this concept to provide guidance for building classifiers.

## 4.2 Definitions

Let $D$ be the *dataset* to be analyzed, with $T = \#(D)$ the size of the set. Each item in the dataset is called *instance*. Each instance $d$ is a tuple belonging to the same cartesian product: $d \in A_1 \times A_2 \times \ldots \times A_n = \mathcal{D}$, where $A_i$ is a (not empty, possibly infinite) set. The values of an instance are called *attributes*. Together with the dataset, we have a function $y : D \to Y$ which specifies the *target* value (or values) for each instance. We will focus on the situation where $Y = \mathbb{B}$, although other possibilities can arise. If the $Y$ set is finite, $y(d)$ can also be called *class* of the instance $d$.

A *classifier* is $c$ a predicate on $\mathcal{D}$, that is $c : \mathcal{D} \to \mathbb{B}$. $\overline{c}$ is the inverse of classifier $c$. The *cover set* of a classifier $c$ over a set of data $D$ is defined as the subset of data where the classifier holds: $D/_c = \{d \in D | c(d)\}$. The *match set* of an instance $d$ over a set of classifiers $C$ is the subset of classifiers which hold on the instance: $C/_d = \{c \in C | c(d)\}$.

## 4.3 HCS: Fundamentals

The underlying idea driving HCS design is to construct a system which can find *interesting* information in a dataset. Formally defining interestingness immediately appears very difficult, if not impossible at all: it is too much a subjective and domain-dependent quantity, to be practically established in all situations (there have been some attempts nevertheless, e.g. [95, 42]). We then turned our research for the *interesting* into a research for the *unexpected*. Starting from an established theory, experiments can be run to obtain a set of data in order to test it. If the data is in accordance with the theory, there is nothing new: nothing of interest. But when an experiment produces unexpected results, then something new has been discovered: the original theory needs to be revised — and this is certainly interesting.

Turning the idea into application, in HCS the initial hypothesis is a model of the data built upon the whole dataset. It is necessary then to establish which kind of data model $\mathcal{H}$ the algorithm will be using, and a function $H$ which can build a model from a dataset $D$. In the approach we will pursue, this model is an uniform risk model, where the risk is estimated through the proportion of classes in the dataset. In a survival analysis setting, the basic hypothesis could be for instance an exponential survival model. The estimated basic hypothesis is called $H_0 = H(D)$. We furthermore pose an independence hypothesis between the attributes and the target, which the algorithm will try to disprove.

The algorithm then searches for areas of the dataset where the basic hypothesis is disproved. The areas will be defined only through the values of the attributes, but the search will directed with knowledge of the target. If the independence hypothesis is true, the resulting areas should contain data with the same target distribution as the basic one. If otherwise the hypothesis is not verified (and this is what we actually expect!), the target distribution in the subset will be different from the general one.

An appropriate fitness function $\mathcal{F}$ evaluates the amount of disagreement between the basic hypothesis and the data contained in the subset. Once the area with highest disagreement has been identified, the dataset is split into two parts — the one belonging to the area, and the remaining. The algorithm is then recursively called on both sub-datasets. Recursion ends when the disagreement between hypothesis and data is low (that is, when the data does not disprove the current hypothesis).

The result of such a procedure is a tree. Each internal node splits the data into two mutually exclusive subsets. Each leaf is a model of the data in the subset identified by the sequence of splits up to the root of the tree.

The algorithm just described is very high-level, avoiding to specify most details. In particular, the following points must be further described in order to fully detail the algorithm:

- The $H$ function, building a hypothesis (or model) from the data. In principle, this could be anything — even another ML algorithm. In HCS we will use a simple constant-risk hypotheses.

- The $\mathcal{C}$ set. This set contains all the possible classifiers, and will thus be $\mathcal{C} \subseteq (\mathcal{D} \rightarrow \mathbb{B})$. It must balance expressiveness (as defined in sec. 2.1.2)

**Algorithm 1** The HCS algorithm

```
Require: $D$ a dataset
  $H_0 \leftarrow H(D)$ {Create the initial hypothesis}
  $\hat{c} \leftarrow \text{argmax}_{\{c \in \mathcal{C}\}} F(D, H_0, c)$ {Find the most disagreeing classifier}
  if $\mathcal{F}(D, H_0, \hat{c}) > \mathcal{F}_{\text{Lim}}$ then {If the disagreement is sufficient}
    $D_m \leftarrow D/_{\hat{c}}$ {Split the dataset; $m$ stands for "matched"}
    $D_u \leftarrow D/_{\overline{\hat{c}}}$ {$u$ stands for "not matched"}
    $H_m \leftarrow \text{HCS}(D_m)$ {Recursive call}
    $H_u \leftarrow \text{HCS}(D_u)$
    return $H_{Tree}(\hat{c}, H_m, H_u)$ {Merge results in a tree}
  else {The data confirms $H_0$}
    return $H_0$
  end if
```

and size, while taking into account readability. It should therefore contain a wide array of possible classifiers, in order to be able to find interesting regions of the data; however, it should not be too big, otherwise searching for the best classifier becomes unfeasible. Moreover, since interpretability is one of our primary goals, the classifiers must be readily understandable.

- The $\mathcal{F}(D, c, H_0)$ function, and its associated $\mathcal{F}_{\text{Lim}}$. This function measures the degree of disagreement between the datasets $D/_c$, $D/_{\overline{c}}$ and a model $H_0$. It must be positive, and increase with the amount of disagreement. The $\mathcal{F}_{\text{Lim}}$ value indicates the limit under which the data is considered to confirm (not disprove) the hypothesis, rather than disprove it.

- When $\mathcal{C}$ cannot be explored exhaustively, a heuristic search algorithm must be defined too, in order to find the maximum of the set — or at least, a good approximation of the maximum.

Notice the current setting is actually open enough to include classical decision trees. In this case, $H_0$ would simply be the proportion of instances belonging to different classes; $\mathcal{C}$ is the set of conditions which rely on a single attribute, and $\mathcal{F}$ would calculate the *information gain* measure of the split provided. The search algorithm over $\mathcal{C}$ depends in principle on the attribute types, that is the $A_i$ sets; however, the C4.5 algorithm defines polynomial deterministic algorithms for all the most usual data types (categorical, ordinal, numerical).

## 4.4 HCS: Implementation

The heart of the HCS algorithm is fitness definition. A proper definition of the fitness function must fulfil two competing goals:

- Assign a high fitness to subsets described by a very different model than the general dataset
- Assign a low fitness to small subsets

The second goal is as important as the first one: it would be very easy to split the dataset so much that every instance takes its own model. Every single model would probably be very different from the global one, being tailored for a specific instance; however, generalization would be extremely poor, and the global resulting model simply useless. The balance between these two issues will be achieved through statistical hypothesis testing.

We will from now on restrict to the case where the learning task is a standard two-classes classification task. The target function will then be $y : D \to \mathbb{B}$.

The dataset contains $T$ instances, out of which $Q = \#\{d \in D | y(d)\}$ have a positive target. We will call $\rho$ the proportion of positive values — that is, $\rho = Q/T$. Any random extraction from the dataset will contain some positive and some negative instances; their proportion will typically be not too far from the global one. Specifically, a random subset extraction will exactly follow a hypergeometric distribution.

Remembering the assumption of independence between attributes and target, selecting instances with respect to their attribute values instead of completely at random should again yield a proportion of positive and negative compatible with the global $\rho$. When this does not happen, we found something unexpected. The more diverse the proportion is from the global one, the more unexpected and interesting is the subset of instances we found. Since the subset is defined by a precise set of conditions on the attributes, the researcher can easily read these conditions and realize what is the distinguishing pattern of the subset.

Statistical hypothesis testing is used in order to establish how much a particular subset of the data is in disagreement with the full dataset. Since we restricted to a Boolean target concept, there are two possibilities to model the result of an extraction from a set of Boolean values: the *hypergeometric* and the *binomial* distribution.

**Hypergeometric distribution** The hypergeometric distribution calculates the probability of obtaining $q$ positive instances in a sequence of $t$ extractions without replacement, from a total set of $T$ instances, out of which $Q$ are positive. The probability mass function is

$$f_t(q) = \frac{\binom{Q}{q}\binom{T-Q}{t-q}}{\binom{T}{t}} \tag{4.3}$$

**Binomial distribution** The binomial distribution calculates the probability of obtaining $q$ positive instances in a sequence of $t$ extractions, where each instance has a fixed probability $\rho = Q/T$ to be positive. The probability mass function is

$$f_t(q) = \binom{q}{t}\rho^q(1-\rho)^{t-q} \tag{4.4}$$

From a practical point of view, the two distributions have a similar shape, but very different values, especially on the extremes (see fig. 4.1 for an example of the difference for particular values of $t$, $Q$ and $T$). This happens because, while with the binomial distribution the probability to get a positive value is constant, with the hypergeometric distribution this probability varies depending on the other extracted values. This however gives no indication on which one could be more appropriate for our purpose.

Theoretically, the hypergeometric distribution is the correct one to apply in this situation. We deal with a finite dataset, and look for the probability to extract a particular combination of positive and negative instances out of that dataset: this is exactly the hypergeometric distribution definition. On the other side, applying the binomial distribution needs to first generalize from the $(Q, T)$ combination of the dataset, to a $\rho = Q/T$ inferred probability.

The reason why we take into account the binomial distribution is that the inference step must be applied sooner or later. The hypergeometric distribution cannot do any prediction on the target of unseen instances; since it deals with extractions from a set, new instances not belonging to the original set cannot be accounted for. The binomial distribution performs

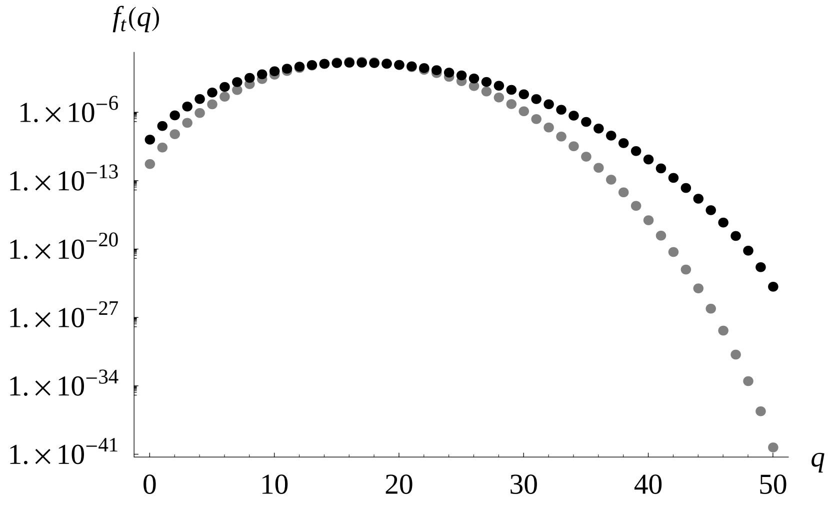


Figure 4.1: Probability mass function (PMF) for binomial (black dots) and hypergeometric (grey dots) distributions, with parameters $T$=150, $Q = 50$, $t = 50$. The $y$-axis is in log scale.

exactly the kind of inductive step we are miss. If we observed $T$ instances, $Q$ of which are positive, we assume that a $\rho = Q/T$ proportion of all future instances will be positive.

From the Machine Learning point of view, the difference between the two distributions is then the timing of the inductive step. A model based on the binomial distribution performs generalization *before* calculating the probability, while with the hypergeometric model generalization will occur *after* probabilities are assessed. The former calculates the $\rho$ value, then assumes that it will apply to the whole dataset, then calculates the probability of a subset. The latter on the contrary follows more closely the information provided by the dataset, and as such should be preferable, but requires to be generalized at the end of learning, when the model is applied to new data.

Since Machine Learning is all about generalization capability of algorithms, this situation makes it unclear whether a hypergeometric distribution would improve or impair understanding of the dataset. Having no clear theoretical indication on the best choice to follow, we choose a pragmatic approach, and tested both distributions on several standard datasets (see chap. 5 for experimental results).

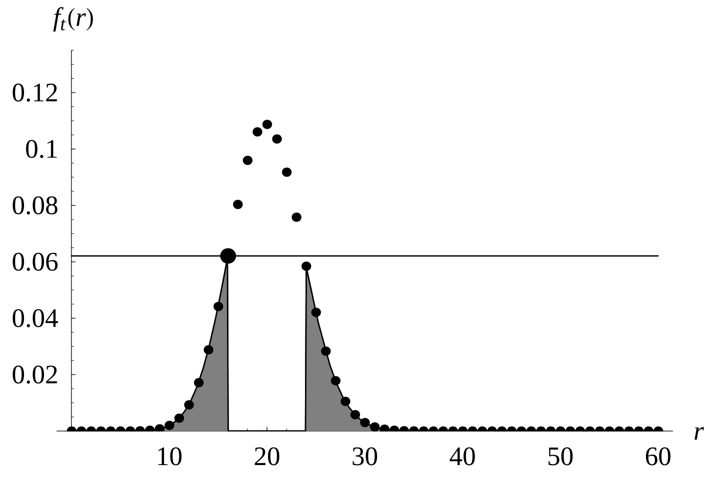


Figure 4.2: Rejection region $R(16, 60)$ in a setting where $\rho = 1/3$. The $x$ axis shows the possible values for the number of positive instances; the $y$ axis contains the corresponding probability value.

Once the probability distribution has been decided, we will call $f_t(q)$ the probability to extract a subset $S(q,t)$ of $t$ instances, $q$ of which are positive. This is exactly the probability mass function (PMF) of the chosen distribution.

To calculate the $p$-value of the $S(q,t)$ subset, we have then to decide a rejection region, containing the sample and every "more extreme" result. Since there is no specific restriction on the direction of search, a two-tailed test is necessary. We decided to define the rejection region of a subset $S(q,t)$ as

$$R(q,t) = \{S(r,t) | r \in \{0,\ldots,t\} \wedge f_t(r) \leq f_t(q)\} \tag{4.5}$$

The rejection region is then the set of all results having probability lower or equal to the examined sample. Other possibilities have been discussed, with particular regard to the "other side" of such a two-tailed distribution: see for instance [45] (our choice is referred there as the "principle of minimum likelihood"). Figure 4.2 provides an example of the rejection region for a binomial distribution. The bigger dot represents the sample subset, with 16 positive instances out of 60 total ($S(16, 60)$). The shaded area highlights all the "more extreme" possibilities, which all together form the rejection region.

It is then straightforward to calculate the $p$-value of a subset:

$$p\text{-value}(S(q,t)) = \sum_{S(r,t)\in R(q,t)} f_t(r) \tag{4.6}$$

The lower the $p$-value is, the more the $S(q,t)$ subset is unexpected under the assumption of the current model. Such subsets are identified by classifiers. In particular, a classifier $c$ identifies two subsets: $D_m = D/_c = \{d \in D | c(d)\}$ and $D_u = D/_{\overline{c}} = \{d \in D | \sim c(d)\}$; in the subscripts, $m$ stands for "matched", while $u$ stands for "not matched"[1]. We can then choose among the two subsets the most unexpected one — that is, the one with minimum $p$-value. Finally, the fitness of a classifier is defined as the $-\log$ of the minimum $p$-value:

$$\mathcal{F}(q,t) = -\log_{10} \min\{p\text{-value}(S(q,t)), p\text{-value}(S(Q-q,T-t))\} \tag{4.7}$$

Notice this quantity is always defined and positive, since $p$-values are in the $]0,1]$ range. Maximizing the fitness will then be the same as minimizing the classifier's $p$-value, i.e. looking for highly unexpected regions.

Algorithm 2 recaps the steps necessary to calculate the fitness of a classifier with the binomial distribution. Figure 4.3 shows how the fitness value changes in an example setting of 60 extractions out of a $\rho = 1/3$ binomial distribution, with varying amount of positive values. Clearly the least interesting subset is the one with 20 positives: this is exactly the expected proportion when $\rho = 1/3$. It would be instead very strange to find a subset with all, or almost all positive: this corresponds to a very high interest value.

It can be interesting to finally notice that our hypergeometric formulation can be viewed as an application of Fisher's exact test [40]. Fisher's test is applied in $2 \times 2$ contingency tables. A $2 \times 2$ contingency table counts the number of positive and negative outcomes of two different random variables; a statistical test is generally applied in order to detect a statistically significant difference. A classical example is the comparison of two medical treatments: a number of tests is carried out with both treatments, recording the number of cured patients versus the number of not cured patients.

The common approach to detect statistically significant differences in this situation is by way of a $\chi^2$ test. This test has the advantage of being easy

[1] $u$ originally stood for *unmatched* — which unfortunately, turned out not to have the meaning we intended!

**Algorithm 2** The binomial fitness algorithm

```
Require: $D$ the dataset, $\rho$ the positive ratio in $D$, $c$ a classifier
  Define $f_t(q) = \binom{t}{q}\rho^q(1-\rho)^{t-q}$
  $t_m \leftarrow \#\{d \in D/_c\}$
  $q_m \leftarrow \#\{d \in D/_c | y(d)\}$
  $v_m \leftarrow \sum\{f_{t_m}(x) | x \in [0,\ldots,t_m] \wedge f_{t_m}(x) \leq f_{t_m}(q_m)\}$
  $t_u \leftarrow \#\{d \in D/_{\overline{c}}\}$
  $q_u \leftarrow \#\{d \in D/_{\overline{c}} | y(d)\}$
  $v_u \leftarrow \sum\{f_{t_u}(x) | x \in [0,\ldots,t_u] \wedge f_{t_u}(x) \leq f_{t_u}(q_u)\}$
  return $-\log\min\{v_m, v_u\}$
```

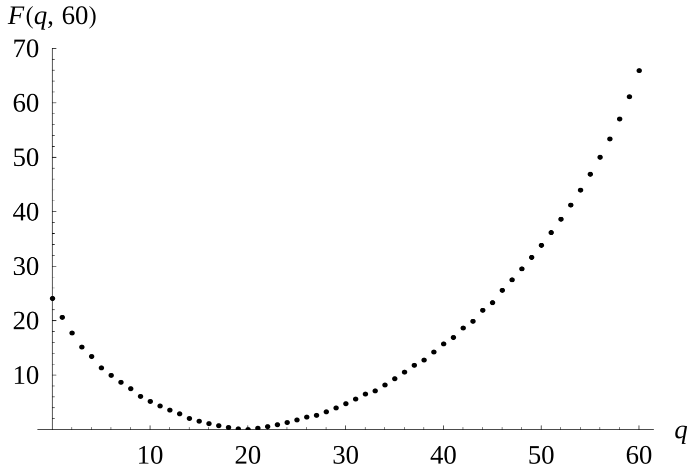


Figure 4.3: Negative log of the $p$-value in a setting where $\rho = 1/3$ and 60 instances. The $x$ axis shows the possible values for the number of positive instances.

to compute; however, its correctness is guaranteed only asymptotically. This means that its results are reliable only with suitably large sample size, and not too skewed marginal frequencies[2]. When such conditions cannot be guaranteed (and this is the HCS case), Fisher's test provides a computationally intensive, but always correct, answer.

Notice how our test, instead of comparing two independent samples, is comparing a set of observations with one if its subsets. This appears to be quite distant from Fisher's test situation; however, it turns out that our calculation is exactly the same as Fisher's, by applying the test in order to evaluate statistical difference of the $D/_c$ and $D/_{\overline{c}}$ subsets.

### Choosing the $\mathcal{F}_{\mathbf{Lim}}$ value

The last (but not less important) step required to specify the HCS algorithm is to define the $\mathcal{F}_{\text{Lim}}$ value. The purpose of this value is to determine when the fitness is high enough to justify the creation of a new model. With an alternative point of view, this value distinguishes subsets confirming (or at least not disproving) the general hypothesis, from those disproving it. In the standard hypothesis testing paradigm, this operation is performed through significance testing: the user decides the confidence level $\alpha$ at which he wants to operate, and considers tests with a $p$-value less than $\alpha$ to disprove the $H_0$ hypothesis. Usual practice sets the $\alpha$ value to 0.05 or 0.01.

In HCS, using such a value would lead to a very high actual error rate. There are three reasons why the standard confidence levels do not apply.

- Hypothesis bias: for the binomial hypothesis, we estimate the binomial parameter $\rho$ from the data, and later test the estimated distribution on the same data. This leads to biased estimates of the significance. This problem can be solved using the hypergeometric distribution.

- Multiple hypothesis testing: in order to find a good classifier, the algorithm performs a very high number of tests — one for each generated classifier. As each one of these tests can wrongly reject the $H_0$ hypothesis with $\alpha$ probability, the probability to commit at least one error quickly rises to very high values[3]. As a further complication, the tests

[2]The exact meaning of *suitably large* and *not too skewed* is not agreed upon by statistics textbooks.

[3]This is sometimes called *data dredging*: testing many hypotheses on the same data, until a significant result is found [109].

are strongly dependent: two classifiers with only slightly different conditions will test very similar subsets, producing a heavily dependent result.

- Sequential hypothesis testing: the tests performed on lower levels of the tree depend on the result of tests performed on higher levels. This creates a dependency which skews the distribution of $p$-values.

For all these reasons, $p$-values lose their usual meaning in the HCS context. We believe it is possible to restore their proper statistical significance, but such an accomplishment is beyond the scope of this thesis. We therefore decided to follow a pragmatic approach to choosing the $\mathcal{F}_{\text{Lim}}$ value, suggesting two possible alternatives.

The first proposal is to focus on the second source of error. It is necessary to apply a correction for multiple hypothesis testing [107]. The most simple and most common of such methods is the *Bonferroni correction* [11], which divides the significance value $\alpha$ by the number of tests performed, which in our case is the total number of classifiers generated during the search. The Bonferroni correction has often been criticized for being too stringent [94]. We must consider however that our attention is focused on the classifier with smallest $p$-value, where the correction is actually accurate enough: the Holm method for instance, often quoted to substitute Bonferroni's, on the smallest $p$-value performs exactly the same calculation [53, 1]. Furthermore, it is extremely complex to derive the statistically correct value for $\mathcal{F}_{\text{Lim}}$ (see App. A). The definition for the $\mathcal{F}_{\text{Lim}}$ value is then as follows:

$$\mathcal{F}_{\text{Lim}} = -\log\left(\frac{\alpha}{\mathit{Tot\ classifiers}}\right) \tag{4.8}$$

Using this formula with typical values for $\alpha$ and the total number of classifiers, the limit results in the 5-7 range.

Another possibility is to completely relinquish the statistical meaning of $p$-value, and apply a standard pruning post-processing. In this case, the training set is split through a $k$-fold cross-validation. For each fold, HCS builds a tree up to a very low fitness level (2, for instance). Then, the fitness values of all the nodes in all the $k$ generated trees are collected and sorted. The limit is set in turn to each of these values, and the cross-validation performance of the pruned trees is assessed, in order to estimate the best threshold. The final tree returned is built on the complete training set, with this threshold value. This method is closely related to the reduced error

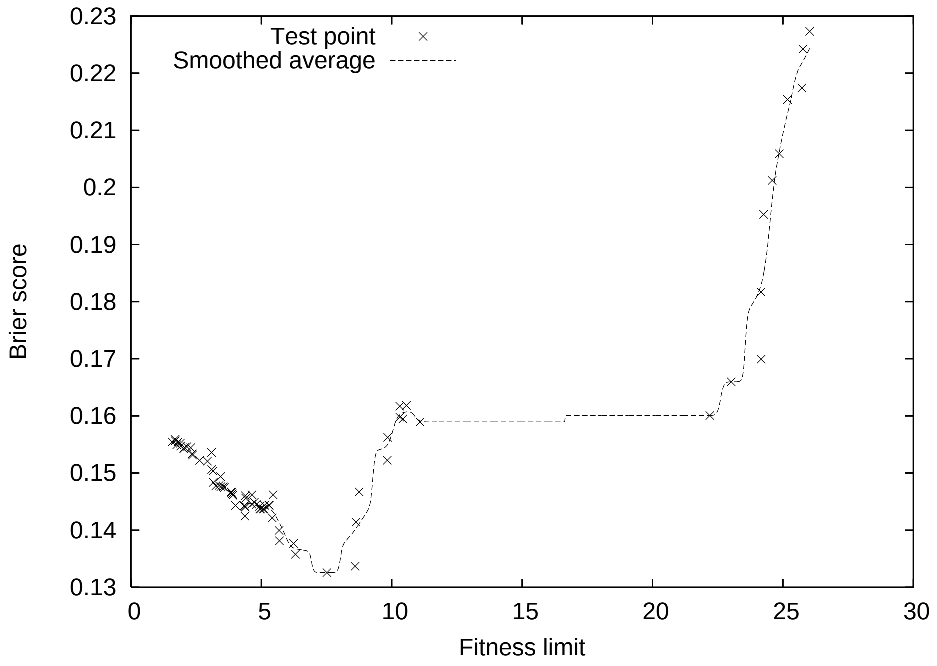


Figure 4.4: Variation of the cross-validation Brier score for various threshold values on a sample execution of the WBC problem. Actual values (crosses) and smoothed average (dashed line).

pruning method proposed by Quinlan [99], but applies $k$-fold cross-validation, instead of a single training/validation split.

The graph in fig. 4.4 is an example of the result of this procedure, for one run on the WBC dataset. The choice of threshold value is a typical example of the bias-variance dilemma in Machine Learning [32]: higher threshold values (on the right of the graph) yield low-power models, which exhibit a high error rate due to their high bias. As the threshold lowers, the power of the model starts to rise, which lowers bias but correspondingly increases variance. To the left of the graph, HCS can build very complex models, but variance has increased so much that these models are easily deceived by the noise present in the data — increasing the generalization error again.

### HCS and sample size

Figure 4.3 showed how the $p$-value varies with changing proportion of negative and positive instances, keeping fixed the sample size. To better characterize our choice of fitness, it can be interesting to illustrate the relationship between the test significance and the sample size, which is not apparent from the formulas. Clearly the sample size is of primary importance: a sample with 10 positive and 20 negative values tells something very different from one with 100 and 200, although the overall proportion is the same. The proposed binomial and hypergeometric tests take this difference into account. In fact, if the null hypothesis postulated a binomial distribution with 0.5 parameter, the $p$-value of first sample would be 0.099, while the second would be $8 \times 10^{-9}$.

To better illustrate this effect, let's consider as null hypothesis the binomial distribution, with $\rho$ parameter. We then extract a sample from another binomial distribution, with $\theta \neq \rho$ parameter. Since we know the two distributions are different, we expect the test to reject the null hypothesis. Rejection is connected with two factors:

- The difference between $\theta$ and $\rho$: the more diverse the two parameters are, the more easy it should be to reject the null hypothesis

- The particular sample obtained from the second distribution: even if $\theta$ is very far from $\rho$, we could have obtained by chance a sample which appears likely to have come from the $\rho$ distribution

In order to show the relationship between rejection and sample size, we will plot for each value $\theta$ the minimum $n$ required in order to obtain, with probability $\beta$, a result with $p$-value less than $\alpha$. Figure 4.5 shows the plot for $\rho = 1/4$, $\alpha = .01$ and $\beta = .99$. In formulas, it plots

$$\min\{n \in \mathbb{N} \mid \mathbb{P}(p\text{-value}(S(\mathcal{B}_{(\rho,n)}, n)) \leq \alpha) \geq \beta\} \tag{4.9}$$

where $\mathcal{B}_{(\rho,n)}$ is a random variable with a $(\rho, n)$ binomial distribution.

The graph clearly shows how, the closer the tested distribution is to the reference one, the more evidence is necessary in order to reject the null hypothesis. On the other extreme, a very different distribution requires little evidence in order to be rejected: for instance, in the extreme case where $\theta = 1$, a sample of size 4 is sufficient to reach a significance level $\leq .01$.

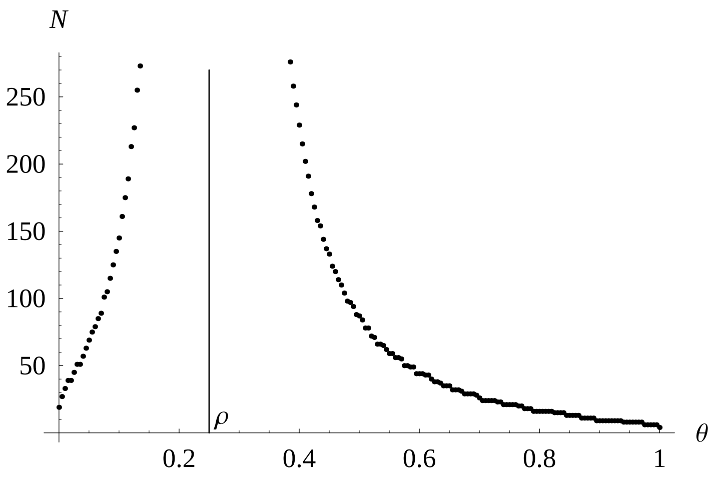


Figure 4.5: The minimum sample size in order to obtain a rejection ($\alpha = .01$) of the binomial distribution ($\rho = 1/4$) null hypothesis, with probability $\beta = .99$ (see text). The graph should be a continuous line, but is sampled for 200 equally-spaced values of $\theta$ for computational feasibility reasons.

## 4.5 HCS: classifiers and search algorithm

Now that a measure of interest of a region has been defined, it is necessary to describe how regions can be defined, and design an algorithm which can discover such regions. We will pose it as a maximization problem: among all the subsets of the training data which can be defined with a conjunction of conditions on the value of the attributes, find the one with highest interest value. This problem was solved with a genetic algorithm [46] (GA), taking inspiration from current Learning Classifier Systems (LCS) research.

### 4.5.1 Genotype definition

The genetic algorithm must identify the classifier with highest fitness. Each individual in the GA population will then be a classifier (*individual* and *classifier* will be used interchangeably from now on). Each classifier will contain exactly one condition for each attribute in the training set; the size of a classifier is therefore constant. A condition is a test on the value of its attribute: examples can be *smoke* $\leq 15$, or *Gender* $=$ Male. Some conditions however can specify a wildcard, meaning that any value will pass them. In this case, the condition can be effectively ignored when reading the classifier. The meaning of the classifier is the logical conjunction of the tests in each condition.

Since we defined $D$ as $D \subseteq \mathcal{D} = A_1 \times A_2 \times \ldots \times A_s$, every classifier will have $s$ conditions. The shape of each condition is independent of the others, and is determined only by is corresponding $A_i$ set. Conditions for the most common data types are as follows ($x$ is the value of an instance for the attribute being tested):

- Categorical data: $A_i = \{v_1, v_2, \ldots, v_n\}$ with no ordering relationship. In this case, the condition can have only two shapes: $x = v_i$ and $x = \#$, meaning respectively that a single value will pass the test, and that every value (including `null`) will pass the test ("wildcard").

- Ordinal data: $A_i = \{v_1, v_2, \ldots, v_n\}$ with $v_i < v_{i+1}$. In this situation, the condition will always be $x \in [v_i, v_j]$, meaning that all the values between $v_i$ and $v_j$ will pass the test. The wildcard, matching also the `null` value, in this case is $x \in [v_1, v_n]$. This situation applies for instance to the integer numbers ($A_i = \mathbb{N}$) [126].

- Real data: $A_i = \mathbb{R}$. The data must first be scaled to the $[0, 1]$ range. A condition is then $x \in [a, b]$, where $a$ and $b$ are two values in $[0, 1]$ with $a < b$[4]. The wildcard, matching also the `null` value, is $x \in [0, 1]$.

With respect to conditions, genotypes in HCS have the same shape as the ones in other LCS systems, like XCS [122]. However, individuals have no additional data other then their genotype; there is thus no action, performance, accuracy, numerosity, etc..

### 4.5.2 Phenotype definition

In genetic algorithms, the phenotype of an individual is defined as its meaning in the original problem space (while the genotype is related to the practical encoding). In our situation, we evaluate classifiers not upon the content of their conditions, but rather on the subset of training data they identify. The phenotype of an individual can therefore be defined as the subset of the training data matched by the individual, that is the *cover set* $D/_c = \{d \in$

[4] Actually, in the implementation $a$ and $b$ can also be swapped around, and the algorithm takes care to perform the test in the proper order. This appears to reduce search bias [113].

$D|c(d)\}$. Matching occurs if a training sample satisfies all the conditions of the individual. A classifier is then interpreted as the logical conjunction of conditions on the attributes.

While the result of a test is obvious if the value is present, some possibilities arise when the value is missing. In our implementation, only the largest ("wildcard") condition is satisfied by missing data. Taking for instance a boolean attribute, only classifiers with a # condition (corresponding to "any boolean value") will match data with the boolean value missing. For instance, 01**11 (where * is a missing value) will be matched by the #1##1# classifier, but not by the ##1## classifier, since the latter specifies a condition upon the third attribute which cannot be verified. The rationale behind this choice is to avoid taking decisions based on unknown values: if a classifier has a wildcard condition upon an attribute, it will take the same decision regardless of the actual (and possibly unknown) value of the attribute. If a classifier requires a certain value, and the current instance has no information regarding that value, we prefer to let the instance be dealt with by another, more general classifier.

Other possible approaches to missing data are analyzed by Holmes [54]. The "classical" ones involve *imputation* — that is, substituting the missing value with a calculated value, based on some probabilistic distribution. A different, LCS-specific way to treat missing values is instead to consider them as wildcards, being matched by any condition of the classifier. With the previous example, the 00**11 instance would be matched by both a 000011 classifier and by a 001111 classifier. We did not pursue this approach in HCS; it could be interesting anyway to compare it with ours in various missing data situations, in order to discover if either is superior to the other.

### 4.5.3 Fitness evaluation

Once the cover sets $D/_c$ and $D/_{\overline{c}}$ have been identified, the evaluation of fitness simply follows the definition given in the previous section, and recapped in Alg. 2.

### 4.5.4 Internal cycle

The genetic algorithm is substantially a generation and selection cycle, here called "internal cycle" to distinguish it from the recursive, external cycle described in Alg. 1. In that context, the genetic algorithm is inside the second instruction, where it performs the search for the most disagreeing classifier.

The internal cycle starts with an empty population, and evolves for a predefined number of *steps*. HCS employs a steady-state population model: at each step, a new offspring is generated and inserted into the population, removing another one if the population reached its maximum size, also defined through a parameter chosen by the user. The $Q$ and $T$ values are evaluated from the dataset at the start of the internal cycle.

### Selection and reproduction

The selection process employs a niching method inspired to XCS in order to maintain diversity in the classifiers population, which in turn greatly improves the GA ability to escape local optima. Notice however the major different between standard LCS: in our algorithm, each individual is competing against the others to earn the *winner* status (Pittsburgh model). On the contrary, in LCS the whole population cooperates towards the final goal of learning (Michigan model).

At each step, a random sample is chosen with replacement from the training set $D$. The current population is scanned to find all the classifiers matching this sample: these classifiers form the *match set*. If the match set is empty, *covering* occurs: a new, random classifier is created, in such a way that its conditions match at least the extracted sample, and immediately added to the population. The match set mechanism is responsible for niches creation and diversity maintenance.

Inside the match set, selection occurs in order to choose which classifier will be allowed to reproduce. The two most important selection methods in evolutionary algorithms are *fitness proportionate* and *tournament* selection. If $\{c_i\}_{i=1\ldots n}$ it the set of classifiers where selection occurs, fitness proportionate assigns each classifier a probability to be chosen equal to $\frac{\mathcal{F}(c_i)}{\sum_j \mathcal{F}(c_j)}$. Tournament selection chooses $\tau$ classifiers from $\{c_i\}$ without replacement, and selects the one with highest fitness among those.

Fitness proportionate selection has often been criticized for leading to premature convergence, and in general choosing individuals with a heavy dependence on the actual values of fitness, rather than their relative position (fitness scaling problem [13]). Tournament selection does not have these disadvantages, and can be applied as long as fitness values can be compared [14] (while the former needs fitness value which can be added).

In this algorithm however, tournament selection might not be the best choice. We apply selection inside match sets: their size greatly varies between classifiers, and can range from 1 to the whole dataset size $T$. A single, fixed value for the tournament size $\tau$ can then be inappropriate in most situations. To solve this problem, Butz suggested to use a tournament size proportional to the size of the match set [25]. We followed his advice; tournament selection still requires however a $\tau \in [0, 1]$ parameter.

Conversely, since our fitness values are strictly defined (rather than totally dependent on the problem, like in most genetic algorithms), fitness proportionate selection operates in a more “controlled” environment — not to mention that it is currently the standard selection method in the LCS community. We decided then to perform experimental tests with both selection operators, in order to assess whether one of them had any significant advantage over the other.

Once selection has been performed, the chosen classifier is passed to the actual discovery mechanism of the genetic algorithm: reproduction. In HCS, reproduction applies the three classical operators. Simple *cloning* just creates a perfect copy of the selected individual. *Mutation* creates instead an imperfect copy, that is a classifier with a single condition slightly modified (the exact specification of variation is dependent on the shape of the condition). With *crossover*, a second individual is selected from the match set; the offspring will contain part of the genes from the first parent, and part from the second parent. Each reproduction operator will be applied (exclusively) with a probability defined at runtime: $\chi$ for crossover, $\mu$ for mutation, and cloning otherwise ($\chi + \mu < 1$).

The created offspring is finally added to the population. If the population size has reached its limit, a classifier is removed from it with an “inverse” tournament selection: a subset $\tau$ of the whole population is chosen, and the classifier with lowest fitness in the subset is selected for removal. Since in this case the set where inverse selection operates upon has a fixed size, we always use tournament selection, with $\tau \in \mathbb{N}$.

The selection and reproduction cycle is repeated until a pre-determined number of steps has been performed. This value is generally decided by the user, in order to balance the search for the best solution with a limited amount of available time. Once the step limit has been reached, searching terminates, returning the classifier with highest fitness value.

### 4.5.5 External cycle

Once the internal cycle is terminated, the classifier with highest fitness in the final population is retrieved and stored, and will form the root of the tree of classifiers returned. The training set is then split into two subsets: the instances matched by the classifier, and the instances not matched by it. The internal cycle is then recursively repeated on both subsets, creating two subtree structures, which are appended to the current root node. The result is clearly a new tree, which is returned to the caller.

The niched algorithm creates at each step many high-fitness classifiers, with possibly disjunct cover sets. It could seem to be possible to exploit this information, and to immediately use the other high-fitness classifiers. The problem with this approach is that, on each of the two new subsets created by the current best classifier, the $H_0$ hypothesis is recomputed, since the proportion of positive instances $Q$ over the total size $T$ has changed. This modification completely changes the fitness surface, making it necessary to start the new computations from scratch. At best, the old population could be used to seed the new ones, in order to speed up convergence; we did not explore this approach.

The external cycle repeats splitting until the classifier fitness is judged too low. This decision is taken comparing the best fitness with the limit value $\mathcal{F}_{\text{Lim}}$. The definition of this value, based upon significance testing, was discussed earlier. If no classifier reached the $\mathcal{F}_{\text{Lim}}$ value, the $H_0$ hypothesis has not been disproved, therefore the model of the whole dataset $D$ is returned — that is, the $\rho = Q/T$ value describing the proportion between positive and total instances in $D$.

The final result of the algorithm is then a tree, with classifiers at each node, and $\rho$ values on the leaves.

## 4.6 HCS: optimizations

### 4.6.1 Optimizing cover set building

The first step necessary to calculate the fitness of a classifier $c$ is to find its *cover set* $D/_c$, that is the subset of all the instances $D$ which the classifier matches. A naïve implementation would scan the whole dataset, and test for each instance whether the classifier covers it or not. The complexity of this operation is $\Theta(sT)$ attribute test operations, to be repeated for every step of the algorithm.

In order to speed up generation of the cover set, we employed a caching mechanism. The fundamental observation is that each classifier is composed of a series of conditions on single attributes; while the number of classifiers is generally very large, the number of possible conditions for each attribute is usually quite limited. Even with a real number attribute, it does not make sense to introduce more cut-off points than the number of different values this attribute takes in $D$. We then created a cache associating conditions found on each attribute to a bitmask representing the instances in $D$ that the condition matched. Since $D$ is fixed, the bitmask for every condition needs to be built only once — namely, the first time the condition is encountered. The bitmask representing a classifier's cover set can then be obtained by the logical AND of the bitmasks of each of the classifier's conditions. The complexity of this method is still $\Theta(sT)$, but now each operation is a bit AND, rather than a condition test: this makes the cached algorithm much faster, also since current computers can perform 32 or 64 such operations in a single clock cycle.

## 4.6.2 Optimizing fitness calculation

Once the cover set has been derived, and the number of positive and negative instances is obtained, the fitness of such a $S(q, t)$ subset must be calculated. Standard statistical methods typically rely on approximations [115], which work under certain conditions (eg. a balanced proportion of positive and negative cases, and a minimum number of cases). Clearly, we cannot anticipate those conditions will be met through every possible classifier being tested; the most appropriate definition of the conditions is also under discussion [21]. Moreover, on some problems classifiers can spot areas with very low $p$-values (on the order of $10^{-100}$), where approximations inevitably commit unpredictable errors[5]. These reason made it necessary to use the exact formula to calculate fitness values.

Indicating as $f_t(p)$ the value of the probability mass function (be it binomial or hypergeometric) for $p$ positive instances out of $t$ extractions, the

[5]For instance, most standard statistical packages return a 0.000 result when the computed $p$-value is below $10^{-3}$.

original formula to be calculated for hypothesis testing is

$$H_b(q,t) = \sum_{\substack{v \in 0,\ldots,t \\ f_t(v) \leq f_t(q)}} f_t(v) \tag{4.10}$$

The first step in optimizing is to recognize binomial and hypergeometric distributions have a first increasing, then decreasing shape. This allows to rewrite the previous formula as

$$\begin{aligned} H_b(q,t) &= \sum_{v=0}^{L_1} f_t(v) + \sum_{v=L_2}^{t} f_t(v) & (4.11) \\ &= \sum_{v=0}^{L_1} f_t(v) + 1 - \sum_{v=0}^{L_2-1} f_t(v) & (4.12) \end{aligned}$$

where $L_1$ and $L_2$ are the two “borderline” values (see Fig. 4.2), that is $L_1 < L_2$ and

$$f_t(L_1) \leq f_t(q) \wedge f_t(L_1+1) > f_t(q) \tag{4.13}$$

$$f_t(L_2) \leq f_t(q) \wedge f_t(L_2-1) > f_t(q) \tag{4.14}$$

One of the two values is always equal to $q$; the other can be efficiently found through local search, and exploiting the symmetry of the binomial and hypergeometric curves. The problem of calculating fitness is then reduced to the problem of calculating the cumulative distribution function (CDF) of the distribution.

## Calculating binomial CDF

The binomial CDF for $q$ positive extractions out of $t$ is defined by the formula

$$F_t(q) = \sum_{v=0}^{q} \binom{t}{v} \rho^v (1-\rho)^{t-v} \tag{4.15}$$

The first step to be taken in order to avoid approximation errors (notably underflow) is to use fractions instead of floating-point values. The formula can then be expanded to

$$\sum_{v=0}^{q} \binom{t}{v} \left(\frac{Q}{T}\right)^v \left(1-\frac{Q}{T}\right)^{t-v} = \sum_{v=0}^{q} \binom{t}{v} \frac{Q^v (T-Q)^{t-v}}{T^t}$$

$$= \frac{1}{T^t} \sum_{v=0}^{q} \binom{t}{v} Q^v (T-Q)^{t-v} \tag{4.16}$$

Notice the formula now is dealing with very big numbers: storing $Q^q$ takes an amount of space linear in $q$. This is however necessary in order not to lose accuracy. The formula could be simplified more (factoring a $t!$ out of the binomial coefficient for instance); anyway, with this format the numbers inside the sum are guaranteed to be integers, rather than fractions. Avoiding divisions and simplifications makes calculations dramatically faster.

Caching was used to avoid recomputing factorials and powers. The former are the same throughout the whole computation, so need to be computed only once. The latter, once the dataset is decided, have only 3 possible bases: $Q$, $T-Q$, and $T$; only 3 caches are then needed (but must be refreshed with each iteration of the recursive algorithm). The space (in bytes) occupied by each cache, including the factorial one, if all the results are stored is approximately

$$\sum_{i=0}^{n} \frac{\log_2 n^i}{8} = \frac{n(n+1)}{16} \log_2 n \tag{4.17}$$

where $n$ is the maximum factorial argument, or the exponent base, which ultimately depend on $T$. Storing everything is then feasible with smaller datasets, where $n < 1000$. In other situations, a least recently used (LRU) policy is probably to be preferred.

Caching is also used to ease recomputing of the sum in eq. 4.16. Each time a CDF value is calculated, the result of the sum (before division) is cached. If successively a value with different $q$ but same $t$ is asked, it is possible to use the cached value as starting point, instead of recalculating all the sum from scratch. Finally, calculating successive addends of the sum can be speeded up with the following relationships:

$$g_t(q) = g_t(q-1)\frac{t-q+1}{q}\frac{Q}{T-Q} \tag{4.18}$$

$$g_t(q) = g_t(q+1)\frac{q+1}{t-q}\frac{T-Q}{Q} \tag{4.19}$$

where $g_t(q) = \binom{t}{q} Q^q (T-Q)^{t-q}$.

**Calculating hypergeometric CDF**

The formula for the CDF of hypergeometric distribution, with at most $q$ positive out of $t$ total extractions, is as follows:

$$F_t(q) = \sum_{v=0}^{q} \frac{\binom{Q}{v}\binom{T-Q}{t-v}}{\binom{T}{t}} = \frac{\sum_{v=0}^{q}\binom{Q}{v}\binom{T-Q}{t-v}}{\binom{T}{t}} \tag{4.20}$$

Similarly to the binomial case, the second formula involves summing only natural numbers, so the calculations are greatly simplified with respect to the sum of fractions in the first formula. The factorials and the results of the already calculated sums are again cached. As for the binomial distribution, it is possible to calculate one addend of the sum from the previous or the following addend, with the following equations:

$$g_t(q) = g_t(q-1)\frac{Q-q+1}{q}\frac{t-q+1}{T-Q-t+q} \tag{4.21}$$

$$g_t(q) = g_t(q+1)\frac{q+1}{Q-q}\frac{T-Q-t-q+1}{t-q} \tag{4.22}$$

where $g_t(q) = \binom{Q}{q}\binom{T-Q}{t-q}$.

## 4.7 An illustrative example

We will now describe the execution of HCS on a simple artificial dataset, in order to provide an example which can help understanding the algorithm. The dataset has 1000 instances with two real attributes, X and Y, both in the [-1.5:1.5] range, and a boolean target value. The classes are balanced, with 500 positive instances, and 500 negative ones. A graphical representation of the dataset is shown in fig. 4.6, where circles represent the positive class, and crosses the negative class.

The dataset is a mixture of 4 bivariate Gaussian distributions, with uncorrelated components. Each point therefore belongs to a $\mathcal{N}(\mu_x, \mu_y, \sigma)$ distribution, where $(\mu_x, \mu_y)$ are the means, and $\sigma$ is the standard deviation (equal for both components). The dataset was generated by picking the 500 negative points from a $\mathcal{N}(0, 0, .5)$ distribution. The 500 positive points were instead split: 200 points from a $\mathcal{N}(.5, .5, .15)$ distribution, 200 points from a $\mathcal{N}(-.5, .5, .15)$ distribution, and finally 100 points from a $\mathcal{N}(-.5, -.5, .1)$

distribution. All the four generating distributions are clearly identifiable by naked eye from the figure.

The first recursive step of the search algorithm produces the population of classifiers represented in fig. 4.7 (this is at the final iteration of the genetic algorithm). Each classifier is represented as a light dashed line, identified by the value of the conditions on X and Y. The thick dashed line is the classifier with highest fitness (62.7 in this case), which is chosen by the algorithm to be used as the root node of the tree. Since this classifier selects 242 instances, 5 of them being positive, its fitness is calculated as

$$\mathcal{F}(5, 242) = -\log_{10} \sum_{f_{242}(i) \leq f_{242}(5)} f_{242}(i) \simeq 62.7 \tag{4.23}$$

Two interesting points deserve to be highlighted:

- This best classifier identifies the lower right area, where no positive Gaussian distribution was set. The algorithm could have easily found in the same area a pure classifier, containing only negative instances; however, this would have reduced the match size, providing less evidence. Hypothesis testing decides the tradeoff between the sample size and composition of the set.

- It is interesting to notice how the niching mechanism is able to maintain many distinct classifiers even at the end of the evolutionary cycle, thus contributing to avoid premature convergence.

Once the best classifier has been found, the dataset is split into two subsets: the data which matches the classifier, and the data which does not. The algorithm is recursively applied on both subsets. On the right side of fig. 4.8 the data matching the classifier is represented. Here, the algorithm does not perform any more tests: since there are 5 positive instances over 242, the maximum reachable fitness (if all the instances could be isolated by a classifier) would be below the minimum threshold.

On the left side, the algorithm starts again looking for classifiers. The one with highest fitness (25.1) is basically excluding the outer area, where only negative instances can be found. Notice how these classifiers were already present in the classifier population at the root of the tree. This suggests that seeding the initial population of a node with the final population of its parent could be beneficial to searching (we did not test this idea).

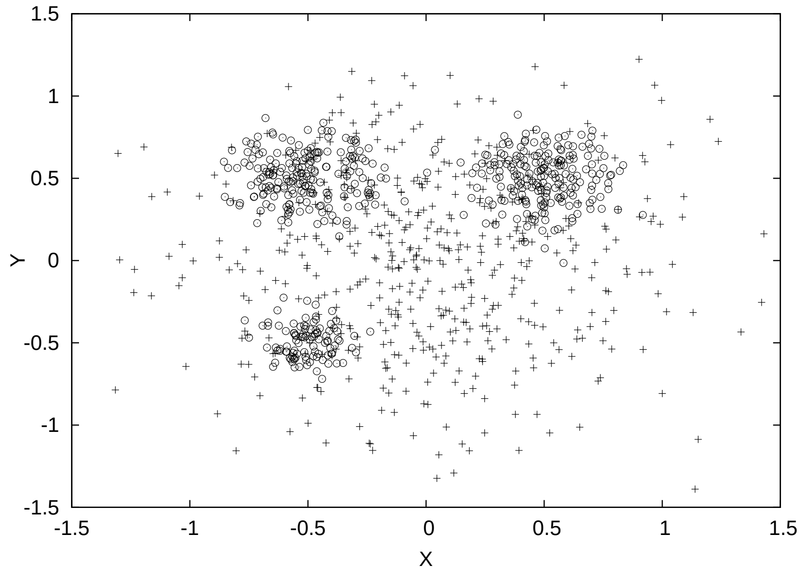


Figure 4.6: Graphical representation of a simple artificial dataset. Crosses are negative instances, circles positive ones.

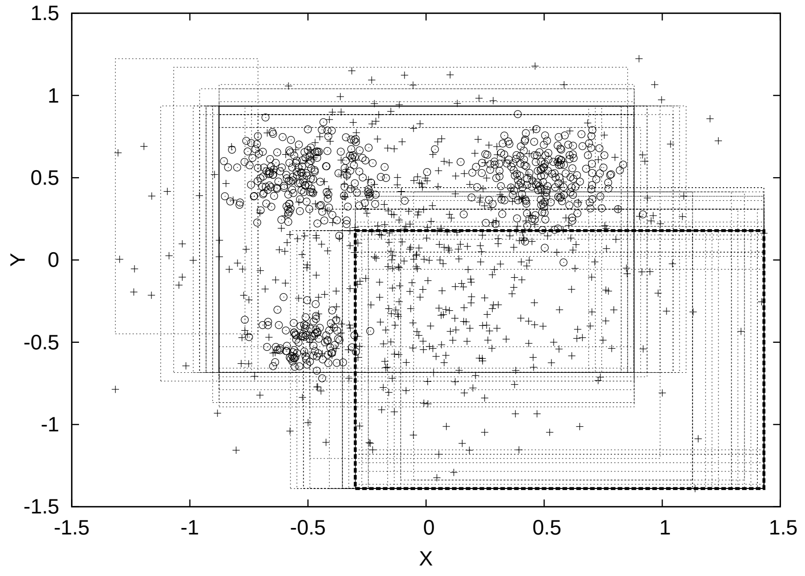


Figure 4.7: Graphical representation of the classifiers population tested at the root of the tree.

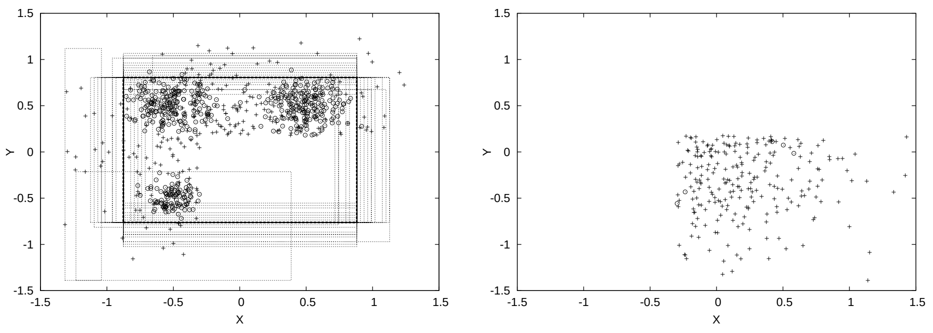


Figure 4.8: The data tested at the first level of the tree, and the classifiers produced by the search algorithm. Right: data matched by the root classifier. Left: data not matched by the root classifier.

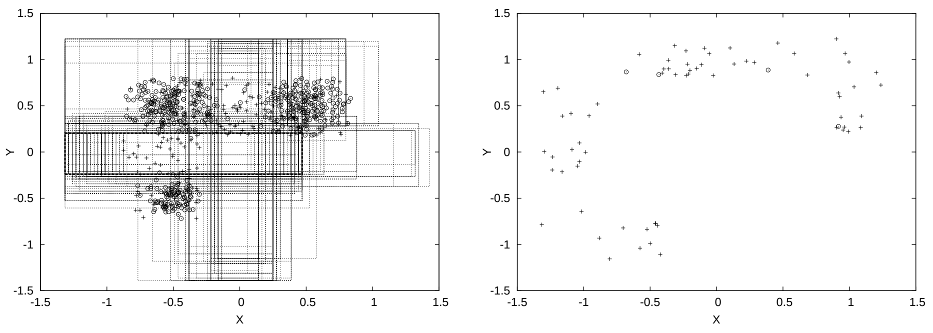


Figure 4.9: The data tested at the second level of the tree, and the classifiers produced by the search algorithm. Right: data matched by the first-level left node. Left: data not matched by the first-level left node.

The algorithm proceeds then to further split the data, obtaining the two subsets reported in fig. 4.9. Again, one of the two subsets is not tested, since the number of positive instances was too low to allow reaching sufficient fitness values. A new significant classifier is found on the other side (fitness: 19.6), and the algorithm proceeds with splitting. The final classifiers tree obtained on this data is shown in fig. 4.10.

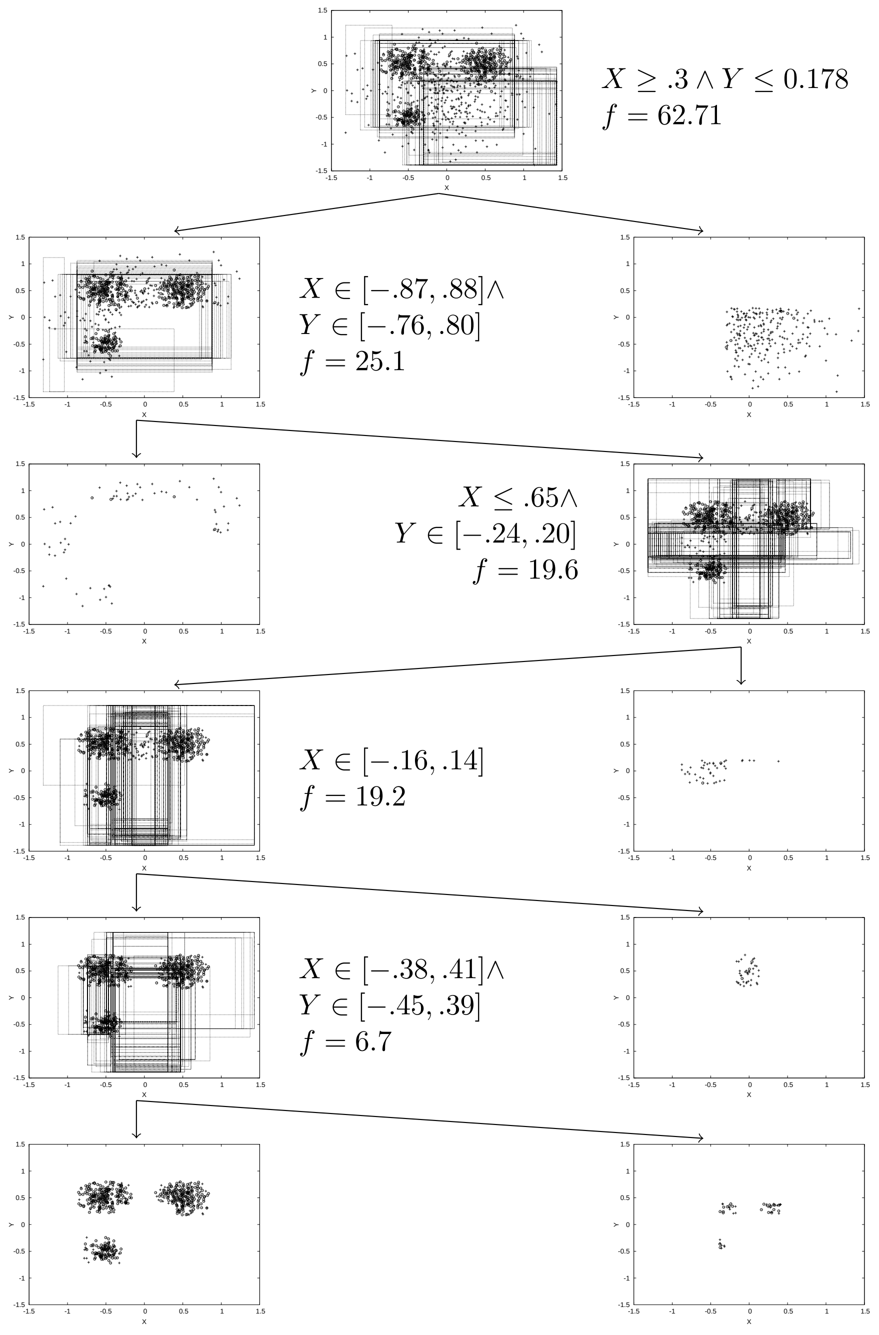


Figure 4.10: Full tree built for example dataset. Right branches contain data matched by the classifier; left branches contain not matched data. Text reports conditions and fitness of the best classifier (chosen for splitting).

# Chapter 5

# Experiments

This chapter presents the results of experimental testing of the HCS. The experiments were performed upon several datasets, and various aspects of the algorithm have been assessed in order to answer three questions.

The first question regards the design choices left open in the previous chapter. We had to decide between two alternatives, both in the fitness calculation, and in the selection process. As regards fitness calculation, the question was raised whether it was more appropriate to model the data through a hypergeometric distribution or a binomial one. The first is more faithful to the setting of extracting instances from a fixed dataset, but needs to be later generalized to a binomial, in order to be able to make predictions on new data. The latter immediately performs the generalization step, at the expense of modelling less precisely the probability distribution of extractions from the dataset — and thus being at risk to take wrong decisions (see sec. 4.4).

In the selection process, the two competing alternatives were tournament selection and fitness proportionate selection (also called roulette wheel selection). While the former has been demonstrated to have some appealing formal properties [14], it's not necessarily the best choice for our situation. Its main drawback is that it requires a tournament size value; such a choice could adversely affect evolution. On the other hand, fitness proportionate selection is generally less well-behaved from a mathematical point of view [13], but it could nevertheless be appropriate for our situation, since the fitness values come from a specific distribution.

The second question we seek to answer through experiments regards the stability of our algorithm. As discussed in chapter 3, we relate the stability of an algorithm to the consistency of its predictions when it is trained on different samples of the same datasets. We will then give a formal definition of stability, and evaluate it through the tests.

Finally, we will compare the results we obtained with the results gathered by other researchers with different algorithms on the same datasets. For ease of comparison, we also executed the algorithms ourselves, and gathered results over the datasets. We already described the theoretical advantages of HCS over other algorithms, but it is necessary to compare its actual performance with the results obtained by other algorithms, in order to evaluate its real usefulness. We chose to test the Naive Bayes and the Logistic Regression algorithms, both for their well-defined statistical structure, and their widespread use in medical research (especially LR). We then tested three well-known Machine Learning algorithm: C4.5, for its similarity with our proposal and its interpretability, Neural Networks and Support Vector Machines, which are considered the state of the art for their proven learning capability, but cannot give an immediate explanation of the model they build.

## 5.1 Datasets description

Experiments with HCS were carried out over five real-world datasets. The first four are well-known and thoroughly studied datasets, coming from the UCI machine learning repository [90]. In particular, they are the *mushroom*, *pima*, *WBC* and *bupa* datasets. The fifth is a dataset concerning risk factors for Head and Neck Squamous Cells Carcinoma (HNSCC), gathered and analyzed in cooperation with prof. R. Barale's research group.

For the tests with Logistic Regression, Neural Networks and Support Vector Machines, missing values were replaced with the mean/mode of the attribute. The other methods could natively cope with missing data, and did not require special treatment: the algorith was provided the original dataset.

### 5.1.1 Mushroom dataset

The mushroom dataset regards distinguishing between edible and poisonous mushrooms; every mushroom is described through a set of quantitative and qualitative features. The data consists of 8192 instances, with 22 categorical (unsorted) attributes, ranging from 2 to 12 different values. 51.8% of the cases represent edible, while the rest poisonous mushrooms. There is a single attribute (*stalk-root*) with 2480 (30%) missing values; all the other attributes are always complete.

This dataset is not really focused on risk prediction: there exist compact sets of rules which achieve 100% accuracy. Nevertheless, it is an important benchmark dataset, studied by most machine learning algorithms. It is moreover a very large dataset — especially for medical standards.

### 5.1.2 Pima Indians diabetes dataset

The Pima Indians diabetes dataset is a medical dataset, gathered to investigate predispositions to diabetes among the females of the Pima Indians population. It comprises 768 instances, with 268 (35%) tested positive for diabetes. Each instance contains 8 numeric attributes, some of which integer and some real valued.

The dataset apparently contains no missing values. However, a more careful examination (even by a non-expert eye) reveals that a few variables present impossible values. Such erroneous data can lead to meaningless rules produced by the algorithm, therefore we decided to remove the wrong values: every number equal to 0 in the plasma glucose concentration, diastolic blood pressure, skin thickness and BMI attributes was deleted. Notice we did not delete attributes, or instances: we just removed the wrong values inside each instance. The resulting dataset has 278 missing values (5 on plasma, 35 on pressure, 227 on skin thickness, 11 on BMI), amounting to 4.5% of the overall data.

This dataset is very interesting from our point of view. It contains in fact most of the characteristics which prompted the development of HCS: medical dataset, missing data, inherently uncertain classification. Results obtained on this data will therefore be especially meaningful to our algorithm.

### 5.1.3 Wisconsin breast cancer (WBC) dataset

The Wisconsin breast cancer dataset [79] is another extremely common benchmark for machine learning algorithms. The data was collected in order to learn to distinguish between benign and malignant tumors, in the context of breast cancer diagnosis. It consists of 699 instances, with 241 (34.5%) malignant and 458 benign cases. Each instance is described by 9 ordinal attributes, ranging in the $\{1, \ldots, 10\}$ set. There are 16 missing values overall.

This dataset is related to a standard diagnostic task; it should then be possible to achieve, in principle, 100% accuracy. In fact, a few simple rules allow to easily reach around 95% accuracy; improving this figure however becomes very difficult, and we are not aware of any ML algorithm claiming perfect classification on this data — possibly due to some misclassified instances.

Our interest in this dataset is due to its medical nature, and the presence of missing values (albeit few). WBC has also been introduced as a benchmark dataset in the LCS community [125].

### 5.1.4 BUPA liver disorders dataset

The BUPA liver disorders dataset is another medical dataset. It was gathered to examine the relationship between drinking habits and liver disorders. It consists of 345 instances, with 145 (42%) showing some kind of liver disorder, and 200 healthy. Each instance has 6 real valued attributes; 5 derive from blood tests, used to detect anomalies caused by drinking; the last attribute is directly related to drinking habits. There are no missing values.

We were interested in this dataset mainly for its medical nature, and for its difficulty: current ML methods reach at best a 72% accuracy on this data. This let us think that the dataset does not allow a clear-cut classification, and would then be suitable for risk analysis. This dataset is also interesting to asses the behaviour of HCS with fewer data points.

### 5.1.5 Oral cancer dataset

The oral cancer dataset, originally presented in [5], was designed to explore the influence of genotype on the chance to develop head and neck squamous cell carcinoma (HNSCC). It is already well-known that this kind of cancer is associated with smoking and alcohol-drinking habits, it is more common among males and its incidence increases with age. The individual risk however could be modified by genetic factors; therefore genotype information, regarding eleven genes involved with carcinogen-metabolizing (`CCND1`, `NQO1`, `EPHX1`, `CYP2A6`, `CYP2D6`, `CYP2E1`, `NAT1`, `NAT2`, `GSTP1`) and DNA repair systems (`OGG1`, `XPD`) was provided by molecular testing.

Nine of these genes have two allelic variants; let's call them $a_1$ and $a_2$. Since the DNA contains two copies of each gene, there exist three possible combinations: $a_1a_1$, $a_2a_2$ (the homozygotes) and $a_1a_2$ (the heterozygote —

order does not matter). The homozygotes where represented with values 0 and 2, while the heterozygote with 1. Due to dominance, for these genes the heterozygote is equivalent to one of the homozygotes; however, for many of the considered genes this dominant effect is not known. So class 1 is either equivalent to class 0, or to class 2. The remaining two genes (`NAT1` and `NAT2`) have 4 allelic variants, which result in 9 combinations; they were sorted by their activity level, and put on an integer scale from 0 to 8.

The full data consists of 355 records, with 124 positive instances (HNSCC patients) and 231 negative (controls). Each record reports the person's gender, age, total smoke and alcohol consumption, gene values, and a boolean target value which specifies whether he had cancer when the database was compiled or not. The data was collected in different periods between 1997 and 2003; this has led to many missing data among the genotypic information of patients. Actually only 122 elements have complete genotypic description; the remaining 233 have missing values ranging from 1 to 9, with the average being 3.58. As an overall figure, of the $11 \times 355 = 3905$ genotype values, just 3070 are present: 21% of the genotype information is missing.

## 5.2 Structure of experiments

The HCS algorithm has two sources of variance. The first one, common to all ML methods, is due to variance in the dataset: executing the same algorithm on different datasets, even drawn from the same underlying distribution, is likely to produce different results. The second source is specific to HCS, and to stochastic algorithms in general. The algorithm used to explore the space of classifiers $\mathcal{C}$ is not deterministic, but proceed stochastically towards the optimum. It is therefore possible that repeated explorations on the same fitness surface produce different results, depending both on different starting points and different decisions during search.

In order to reduce the dependency of results on the particular choice of training set, and on the particular search path chosen during the experiment for non-deterministic algorithms, every test was repeated 10 times, averaging the results. The value of 10 was chosen as a tradeoff between the requirement to reduce variance, and the time constraints on running many experiments. We call *run* each repetition of the same experiment.

As previously recalled, we have two design choices, both with two options: selection method (fitness proportionate / tournament) and fitness function (binomial / hypergeometric). This amounts to four algorithms, to be tested on five datasets. The total is then of 20 experiments; executing 10 runs for each experiment produces a total of 200 runs.

The size of the Mushroom dataset allowed for a straightforward training and test stratified split (25% and 75% of the whole data, respectively). Each run consisted of a single execution of the algorithm. For all the other datasets, a simple split would have generated either not enough data for training, or not enough for a stable evaluation. We therefore decided to apply 10-fold cross-validation. Each run then comprised 10 executions of the algorithm, and the measures of performance were calculated on the reconstructed test set; for every run, a new folding or training/test split was generated[1].

For every experiment, fitness limit training was employed, in order to choose the best fitness cutoff point. The nested cross-validation procedure described in sec. 2.1.5 was applied; in the inner folding, we choose the $\mathcal{F}_{\text{Lim}}$ value which minimized the cross-validated Brier score.

## 5.3 Performance measures

We are interested into two main features of the results obtained through the HCS algorithm, namely correctness and stability.

The standard way to evaluate correctness of the results is by measuring the *classification accuracy*. This measure is the proportion of instances in the (possibly reconstructed) test set, for whom the model produced by the algorithm correctly identifies the class label. As already discussed in chap. 2, other measures are often advocated, usually since they are more robust to imbalanced class distributions [18]. Accuracy however is still the most common in the machine learning community, and we calculated it in order to have a straightforward comparison with other ML algorithms. Since HCS produces probabilities rather than class labels, in order to calculate accuracy we labelled each instance with the class with highest probability. Considering that we restricted to the case where there are only two classes, labelled 0 and 1, and that the risk value is the probability to obtain class 1, this amounts to

[1]The 10-fold cross-validation was not applied to Mushroom because training/test splits, when possible, give more statistically tractable results [31], and because the dataset is too big for HCS and the more complex algorithms (NN, SVM) to complete in a reasonable amount of time. HCS calculations could be speeded up by approximating the fitness, but classifiers matching would nevertheless require a large amount of time.

rounding the risk to the closest integer. Accuracy ranges from 0 to 1, with higher values corresponding to better models.

In order to better evaluate HCS' behaviour, we also recorded the Brier score and AUC values. The AUC metric is however not particularly favourable to HCS. The trees resulting from this algorithm are usually are very compact, with a low number of leaves (3-4 typically); this helps readability, and keeps overfitting low. On the other side, the results inside each group are undifferentiated; this kind of structure does not have a good discrimination power, which is exactly what the AUC intends to measure.

Since these Brier score and AUC figures are not systematically reported in the ML literature, we recalculated the results ourselves (together with accuracy). For all the learning algorithms except SVM, we used the implementation in Weka [127]. Some preliminary tests did not show meaningful differences by modifying the parameters from their default value, therefore all the parameters were left at their default. As for HCS, we report the average result for 10 runs, where the result for each run is obtained through 10-fold crossvalidation (or a training/test split in Mushroom case).

As regards SVM, the results obtained are usually quite sensitive to the choice of initial parameters. We therefore believe that parameter optimization was required, in order to provide meaningful results. We choose to apply the RBF kernel, and optimized the $C$ and $\Gamma$ parameters through a nested 10-fold crossvalidation (the outer folding was necessary to properly estimate performance). We used the mySVM implementation by Stefan Rüping, contained inside the YALE machine learning environment [83].

We are not aware of any method of evaluating stability (as defined in chap. 3) of a risk prediction algorithm. Turney gives a definition in [116], but it is valid only for simple classification algorithms — and it apparently requires knowledge of the probability distribution underlying $D$. We therefore designed our own measure, inspired to Turney's one.

Every experiment is repeated 10 times. With cross-validation, this means that every instance of the data will have 10 evaluations when it belongs to the (reconstructed) test set. With the training/test split used for the mushroom dataset, the number of evaluations of each instance in the test set will be slightly lower (7.5 on average).

Calling $r_i^j$ the risk value assigned to instance $j$ on its $i$-th test evaluation, we first calculate stability for each instance as the standard deviation of

$\{r_1^j, \ldots, r_n^j\}$:

$$s^j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i^j - \overline{r^j})^2} \tag{5.1}$$

where $\overline{r^j} = \frac{1}{n}\sum_{i=1}^{n} r_i^j$. Stability is then defined as the average of the deviations:

$$\overline{s} = \frac{1}{T}\sum_{j} s^j \tag{5.2}$$

Since each $r_i^j$ ranges in the $[0, 1]$ set, $s_j$ can vary from 0 to 0.5, and so does $\overline{s}$ — with 0 being perfect stability, and 0.5 being complete instability.

## 5.4 Results

Table 5.1 summarizes the parameters applied to the genetic algorithm for the various datasets. The $\mu$ and $\chi$ parameters control the probability to perform a mutation or a crossover, respectively. The mutation value is higher than usual because we resort to this operator for fine-tuning of the classifier conditions. The two values are always the same for all datasets. The $\tau$ parameter is the size of the tournament, expressed as a fraction of the match set size; it was set to 0.4, following Butz' suggestion [25]. During "unselection" — that is, while choosing the classifier to remove from the population when a new one must be added — tournament size was fixed at 4. This value is somewhat dependent on the population size; again, it was not tuned for singular datasets. The $S_0$, $Mr$ and $P_\#$ parameters control some aspects of the generation of classifier conditions. $S_0$ is the maximum initial range for real and integer conditions, while $Mr$ is the maximum mutation amount for the borders of the same conditions. $P_\#$ is the probability to put a wildcard value when creating a new condition for a categorical attribute. Again, these values were assigned through past experience and not tuned.

The two most important parameters are the population size, and the number of evaluation steps. These values are not the same on all dataset due to the greater difficulty of the Bupa and HNSCC datasets. For these two, a few preliminary executions suggested that the maximum number of classifiers and the total number of search steps should have been slightly increased, in order to improve the algorithm's performance.

Table 5.1: Genetic algorithm parameters for the 5 datasets.

| | Mushroom | Pima | WBC | Bupa | HNSCC |
|---|---|---|---|---|---|
| Pop. size | 200 | 200 | 200 | 300 | 300 |
| Steps | 30000 | 30000 | 30000 | 45000 | 45000 |
| $\mu$ | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 |
| $\chi$ | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| $\tau$ | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 |
| $S_0$ | — | 0.75 | 0.75 | 0.75 | 0.75 |
| $Mr$ | — | 0.1 | 0.1 | 0.1 | 0.1 |
| $P_{\#}$ | 0.5 | — | — | — | 0.5 |

### 5.4.1 Results: design choices

Tables from 5.2 to 5.6 summarize the results obtained for the 5 datasets with all the 4 variants of the HCS algorithm. TB stands for Tournament+Binomial, TH for Tournament+Hypergeometric, RB for Roulette wheel+Binomial, RH for Roulette wheel+Hypergeometric. At a first glance, the four variants appear to perform very similarly on all the datasets. From a statistical point of view, we followed the procedure suggested in [30], and applied the Friedman test [43] in order to detect significant variations between the results (the mean rank value is reported in the last column of each table). This test reports wheter at least one significant difference is found; a further post-hoc analysis (the Nemenyi test [89]) can then be applied to detect which couples have a significant difference.

The Friedman test suggests that no strong difference between the four design choices can be declared to exist. The only significant result identified is on the Brier score values ($p = .036$), where the Roulette+Hypergeometric combination performs consistently worse than the alternatives. This result is however only partly confirmed by the others, so we are more inclined to attribute it to chance, rather than to an effective difference.

We decided then to perform a simpler test, comparing the the fitness choice and selection method separately. Table 5.7 contains the $p$-value for the Wilcoxon signed ranks test (two-tailed).

For the selection operator, both tournament selection and fitness proportionate selection perform equally well, providing no clear indication of preference. It appears that in HCS the operators maintain a very similar pressure towards more fit solutions; this is probably also to ascribe to the

Table 5.2: Accuracy results. Average and standard deviation values over 10 runs with 10-fold cross-validation, except Mushroom (see text). Mean rank value for Friedman test ($p = .586$).

| | Mushroom | Pima | WBC | Bupa | HNSCC | Rank |
|---|---|---|---|---|---|---|
| TB | $.999 \pm .001$ | $.736 \pm .008$ | $.950 \pm .006$ | $.620 \pm .027$ | $.841 \pm .011$ | 3.1 |
| TH | $.998 \pm .002$ | $.737 \pm .012$ | $.947 \pm .008$ | $.646 \pm .016$ | $.820 \pm .011$ | 2.7 |
| RB | $.998 \pm .002$ | $.728 \pm .013$ | $.948 \pm .006$ | $.621 \pm .026$ | $.837 \pm .015$ | 2.1 |
| RH | $.999 \pm .001$ | $.731 \pm .011$ | $.944 \pm .004$ | $.634 \pm .021$ | $.806 \pm .014$ | 2.1 |

Table 5.3: Brier score results. Average and standard deviation values over 10 runs with 10-fold cross-validation, except Mushroom (see text). Mean rank value for Friedman test ($p = .036$).

| | Mushroom | Pima | WBC | Bupa | HNSCC | Rank |
|---|---|---|---|---|---|---|
| TB | $.000 \pm .001$ | $.186 \pm .003$ | $.040 \pm .006$ | $.243 \pm .012$ | $.134 \pm .007$ | 3.1 |
| TH | $.001 \pm .002$ | $.180 \pm .005$ | $.038 \pm .005$ | $.244 \pm .016$ | $.146 \pm .007$ | 2.7 |
| RB | $.001 \pm .002$ | $.186 \pm .005$ | $.039 \pm .002$ | $.239 \pm .009$ | $.138 \pm .010$ | 3.0 |
| RH | $.001 \pm .001$ | $.187 \pm .004$ | $.043 \pm .003$ | $.256 \pm .014$ | $.152 \pm .011$ | 1.2 |

Table 5.4: AUC results. Average and standard deviation values over 10 runs with 10-fold cross-validation, except Mushroom (see text). Mean rank value for Friedman test ($p = .196$).

| | Mushroom | Pima | WBC | Bupa | HNSCC | Rank |
|---|---|---|---|---|---|---|
| TB | $1.000 \pm .000$ | $.764 \pm .011$ | $.971 \pm .008$ | $.59 \pm .03$ | $.811 \pm .017$ | 2.6 |
| TH | $.999 \pm .002$ | $.775 \pm .012$ | $.973 \pm .005$ | $.63 \pm .02$ | $.821 \pm .010$ | 3.3 |
| RB | $.999 \pm .001$ | $.768 \pm .010$ | $.966 \pm .005$ | $.58 \pm .02$ | $.805 \pm .017$ | 1.6 |
| RH | $.999 \pm .001$ | $.759 \pm .011$ | $.968 \pm .009$ | $.63 \pm .03$ | $.823 \pm .021$ | 2.5 |

Table 5.5: Stability results. Average and standard deviation values over 10 runs with 10-fold cross-validation, except Mushroom (see text). Mean rank value for Friedman test ($p = .658$).

| | Mushroom | Pima | WBC | Bupa | HNSCC | Rank |
|---|---|---|---|---|---|---|
| TB | $.001 \pm .020$ | $.147 \pm .066$ | $.049 \pm .098$ | $.137 \pm .068$ | $.084 \pm .108$ | 2.8 |
| TH | $.004 \pm .033$ | $.129 \pm .064$ | $.049 \pm .094$ | $.188 \pm .074$ | $.120 \pm .107$ | 2.3 |
| RB | $.004 \pm .033$ | $.141 \pm .066$ | $.043 \pm .091$ | $.125 \pm .070$ | $.090 \pm .108$ | 2.9 |
| RH | $.002 \pm .027$ | $.139 \pm .064$ | $.049 \pm .095$ | $.201 \pm .080$ | $.128 \pm .108$ | 2.0 |

Table 5.6: Number of rules. Average and standard deviation values over 10 runs with 10-fold cross-validation, except Mushroom (see text). Mean rank value for Friedman test ($p = .123$).

| | Mushroom | Pima | WBC | Bupa | HNSCC | Rank |
|---|---|---|---|---|---|---|
| TB | $6.4 \pm 0.5$ | $3.8 \pm 1.7$ | $2.5 \pm 0.5$ | $1.6 \pm 0.7$ | $2.1 \pm 0.3$ | 3.6 |
| TH | $7.2 \pm 1.0$ | $3.9 \pm 1.1$ | $3.1 \pm 0.7$ | $3.0 \pm 0.9$ | $3.0 \pm 1.8$ | 2.0 |
| RB | $6.1 \pm 0.9$ | $4.3 \pm 1.2$ | $3.3 \pm 0.9$ | $1.7 \pm 0.9$ | $2.2 \pm 0.4$ | 2.6 |
| RH | $7.1 \pm 0.7$ | $3.7 \pm 0.9$ | $3.5 \pm 1.1$ | $3.2 \pm 1.1$ | $3.7 \pm 1.5$ | 1.8 |

niching mechanism. There could be some difference in the number of evaluations required to obtain the optimum, which in our experiments was fixed at the beginning; however we did not verify this possibility. Although the experiments produce no evidence of improvement, we believe tournament selection to be superior to fitness proportionate – first, for its provable mathematical properties. Second, in situations where very similar high-fitness classifiers are competing, tournament selection still provides an advantage to the best one, while fitness proportionate is basically choosing at random. We then decide to employ tournament selection in HCS.

Also for the fitness function, the results do not point decisively into one or another direction. Anyway, the binomial fitness generaly produces smaller trees, while not impairing performance (except a slight reduction in AUC). A more compact tree is easier to interpret, so this function should be preferred. The binomial fitness was moreover slightly faster to calculate in our implementation. We then choose the binomial distribution for further analysis of the algorithm.

Table 5.7: Split comparison of Binomial vs Hypergeometric, and Tournament vs Roulette wheel. Figures report the $p$-value of Wilcoxon signed ranks test. ○ reports a better result for Binomial / Tournament; ● a better result for Hypergeometric / Roulette wheel.

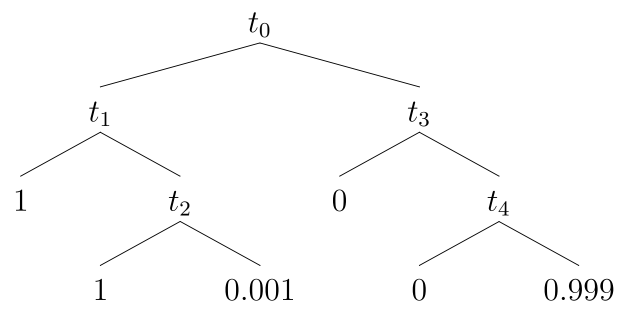
| | Accuracy | Brier | AUC | Stability | Size |
|---|---|---|---|---|---|
| B/H | .787 | .105 | .038● | .348 | .014○ |
| T/R | .016○ | .102 | .068 | .439 | .137 |

### 5.4.2 Results: Mushroom dataset

On the mushroom dataset, the HCS algorithm reached 99.95% accuracy in the test set (averaged over 10 runs). This result is good, but partly unsatisfactory: most algorithms reach perfect classification on this dataset (tab. 5.8), while HCS was able to do so only on 6 runs. There can be two reasons why our method does not always produce a perfect interpretation of the whole dataset. The first is simply that HCS is not designed to do that. The algorithm does not expect to find a perfect classification, and therefore the fitness definition only indirectly prefers classifier leading to 100% accuracy.

The second reason is connected to a limitation in the possible conditions. The attributes in the dataset are categorical, with many possible distinct values. For such attributes, our algorithm can only form conditions which isolate a single class. The most important rule for this dataset, reported by many classifiers (see for instance [34]), is a disjunction of three possible values for the *odor* attribute. This rule would have a very high fitness value in HCS (approximately 250), but cannot be expressed with the current condition format. A partial confirmation to this explanation can come from the results obtained on this dataset with XCS and UCS, which do not reach 100% accuracy as well (they obtain 99.0% and 99.2%, respectively [9]). We did not explore the possibility to employ disjunctive rules for categorical values: although this is technically possible, the algorithm could find some difficulties in exploring such a complex space.

Concerning stability, on this dataset HCS behaves very well. With a stability index of 0.001, we can conclude that the resulting trees consistently assign similar risk values to instances. Fig. 5.1 depicts the tree obtained from a sample run.

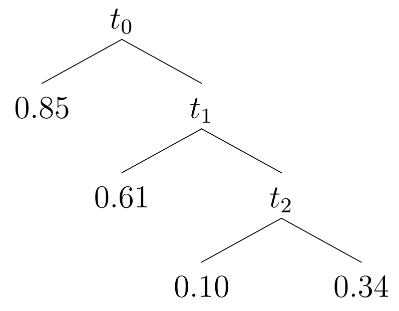


| | |
|---|---|
| $t_0$: [197] | *odor*=none |
| $t_1$: [24] | *gill-size*=broad ∧ *spore-print-color*=green |
| $t_2$: [17] | *gill-size*=narrow ∧ *stalk-surface-below-ring*=scaly |
| $t_3$: [136] | *gill-size*=broad ∧ *stalk-shape*=enlarging ∧ *stalk-surface-above-ring*=smooth |
| $t_4$: [44] | *gill-spacing=crowded* ∧ *stalk-shape*=tapering |

Figure 5.1: Typical result for the mushroom dataset. Conditions are satisfied on the left branch, and unsatisfied on the right branch. Numbers in brackets report rule fitness.

Table 5.8: Performance measures for various algorithms on the Mushroom dataset. Average of 75% test set performance over 10 runs.

| | NB | LR | C4.5 | SVM | NN | HCS |
|---|---|---|---|---|---|---|
| Accuracy | 0.96 | 1.0 | 1.0 | 1.0 | 1.0 | 1.00 |
| Brier score | 0.03 | 0.0 | 0.0 | 0.0 | 0.0 | 0.00 |
| AUC | 0.99 | 1.0 | 1.0 | 1.0 | 1.0 | 1.00 |



| | |
|---|---|
| $t_0$: [26] | $plasma \geq 155 \wedge BMI \geq 24 \wedge pedigree \geq .314$ |
| $t_1$: [19] | $plasma \geq 100 \wedge BMI \geq 27 \wedge pedigree \geq 1.43 \wedge age \geq 29$ |
| $t_2$: [6] | $timesPregnant \leq 11 \wedge plasma \leq 130 \wedge BMI \leq 49 \wedge pedigree \leq 2.28$ |

Figure 5.2: One result for the Pima dataset. Conditions are satisfied on the left branch, and unsatisfied on the right branch. Numbers in brackets report rule fitness.

### 5.4.3 Results: Pima dataset

The Pima dataset should be better suited for analysis through HCS. It does not guarantee that a perfect distinction between classes can be found — and in fact, no ML method has succeeded so far in this task. Reported results for rule-based methods, like C4.5, are generally around 73.0% [112, 60]. Figures become higher with methods able perform linear combinations of the attributes: the best result obtained so far is 77.7% with linear discriminant analysis [82], but a simple logistic regression already reaches 77.2% (our result).

The HCS algorithm obtained 73.8% average accuracy, which is similar to C4.5 performance, but slightly lower than the results obtained with XCS and UCS (around 75.5% [9]). While the accuracy and Brier score figures for HCS are close to the ones obtained from other algorithms, the results for AUC appear much lower. This is however expected, and is a consequence of the low number of different risk values contained in the trees.

Fig 5.2 reports a tree obtained from a sample run.

Table 5.9: Performance measures for various algorithms on the Pima dataset. Average of 10-fold crossvalidated performance over 10 runs.

| | NB | LR | C4.5 | SVM | NN | HCS |
|---|---|---|---|---|---|---|
| Accuracy | 0.76 | 0.77 | 0.75 | 0.77 | 0.75 | 0.74 |
| Brier score | 0.18 | 0.16 | 0.19 | 0.16 | 0.17 | 0.18 |
| AUC | 0.81 | 0.83 | 0.76 | 0.85 | 0.81 | 0.76 |

Table 5.10: Performance measures for various algorithms on the WBC dataset. Average of 10-fold crossvalidated performance over 10 runs.

| | NB | LR | C4.5 | SVM | NN | HCS |
|---|---|---|---|---|---|---|
| Accuracy | 0.97 | 0.96 | 0.95 | 0.97 | 0.96 | 0.95 |
| Brier score | 0.02 | 0.03 | 0.04 | 0.02 | 0.03 | 0.04 |
| AUC | 0.99 | 0.99 | 0.97 | 0.99 | 0.99 | 0.97 |

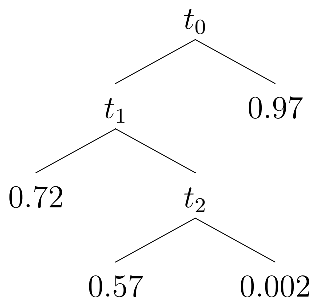
### 5.4.4 Results: WBC dataset

WBC is another medical dataset, and although it is aimed at a diagnostic task, it is still a difficult problem, where a perfect predictor has not been found. Results reported for C4.5 range from 93.4% [7] to 94.7% accuracy [131]. The XCS algorithm has been reported to obtain between 94.1% and 96.4% accuracy [125, 9]. Similarly to the Pima datasets, figures are higher for subsymbolic methods, with SVM and $k$ nearest neighbour reported to reach 97% [7]. Of course, we must remember these methods have the disadvantage of not providing easily interpretable models. Our results confirm the literature reports, and show almost tied values for both Brier score (around 0.03) and AUC (0.97–0.99).

HCS on this dataset reaches on average 94.5% accuracy. This result very similar to C4.5, and slightly worse than the UCS result. However, HCS reaches such an accuracy with an average of 2.9 conditions — while XCS requires approximately 25 rules to reach comparable accuracy levels [125]. Moreover, the results are very stable, meaning that the algorithm consistently produces similar models. Fig 5.3 reports a tree obtained from a sample run.

### 5.4.5 Results: Bupa dataset

The Bupa dataset is even more “difficult” than Pima. Just predicting the most frequent class yields 58% accuracy, and most classifiers don’t improve



$t_0$: [89] $f_1 \in [0,8] \land f_2 \in [0,4] \land f_3 \in [0,6] \land f_4 \in [0,7] \land f_8 \in [0,8]$

$t_1$: [20] $f_3 \in [3,8] \land f_4 \in [0,9] \land f_6 \in [3,10] \land f_7 \in [0,8]$

$t_2$: [6] $f_1 \in [4,10] \land f_4 \in [0,9] \land f_5 \in [0,9] \land f_7 \in [4,10] \land f_8 \in [0,6]$

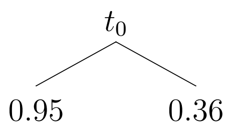
Figure 5.3: One result for the WBC dataset. Conditions are satisfied on the left branch, and unsatisfied on the right branch. Numbers in brackets report rule fitness.

this value too much. C4.5 is reported to reach accuracies varying from 61% [60] to 64% [7] or even close to 66% [9], which is confirmed by our result. XCS and UCS attain 65.4% and 68.7% respectively [9]. HCS on this dataset reaches 62.0% accuracy, which is well below the standard. This is the only dataset where hypergeometric fitness gives a better accuracy than binomial fitness (64.6% with tournament selection). However, the result is not so clear from the Brier score point of view, where the results are mixed. The toughness of the dataset for HCS is reflected by the stability value, and by the tree sizes. Roughly half of the runs ended by reporting only a single condition, and 1/3 with no condition at all (that is, by providing simply the default 58% value). With so small tree size, the average AUC value is worse even than Naive Bayes (while the accuracy and Brier score are better): a tree with no condition will in fact the default value of 0.5 for AUC. Fig 5.4 reports a result with a single condition.

### 5.4.6 Results: HNSCC dataset

The HNSCC dataset is not public, so we have no independent accuracy values to confront with. We therefore will compare HCS results only with performance figures obtained by our own execution of different ML algorithms.

Results with HCS compare very favourably with the other ML algorithms.



$t_0$: [13] $sgpt \geq 20 \wedge sgot \leq 21 \wedge gammagt \leq 22 \wedge drinks \leq 3$

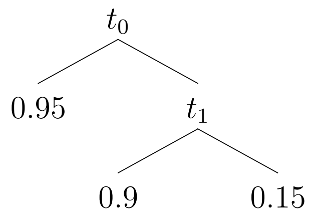
Figure 5.4: One result for the Bupa dataset. Conditions are satisfied on the left branch, and unsatisfied on the right branch. Numbers in brackets report rule fitness.

Table 5.11: Performance measures for various algorithms on the Bupa dataset. Average of 10-fold crossvalidated performance over 10 runs.

| | NB | LR | C4.5 | SVM | NN | HCS |
|---|---|---|---|---|---|---|
| Accuracy | 0.55 | 0.69 | 0.65 | 0.69 | 0.69 | 0.62 |
| Brier score | 0.26 | 0.21 | 0.26 | 0.20 | 0.21 | 0.24 |
| AUC | 0.63 | 0.71 | 0.67 | 0.75 | 0.73 | 0.59 |

Table 5.12: Performance measures for various algorithms on the HNSCC dataset. Average of 10-fold crossvalidated performance over 10 runs.

| | NB | LR | C4.5 | SVM | NN | HCS |
|---|---|---|---|---|---|---|
| Accuracy | 0.69 | 0.71 | 0.70 | 0.76 | 0.79 | 0.84 |
| Brier score | 0.23 | 0.20 | 0.20 | 0.17 | 0.17 | 0.13 |
| AUC | 0.74 | 0.74 | 0.72 | 0.82 | 0.86 | 0.81 |



$t_0$: [28] *gender*=Male $\wedge$ *age*$\in [49, 79]$ $\wedge$ *packyears*$\geq 4$ $\wedge$ *ogg1*$\in \{0, 1\}$ *nat2gen*$\leq 7$ $\wedge$ *gstp1*$\in \{0, 1\}$

$t_1$: [11] *age*$\in [35, 82]$ $\wedge$ *packyears*$\geq 24$ $\wedge$ *ephx1*$\in \{1, 2\}$ $\wedge$ *gstp1*$\in \{0, 1\}$

Figure 5.5: One result for the HNSCC dataset. Conditions are satisfied on the left branch, and unsatisfied on the right branch. Numbers in brackets report rule fitness.

While we cannot exclude a certain amount of publication bias, the only algorithm which appears to be able to reach HCS when applied by an expert is neural networks. It would be interesting to compare our result with the results obtained through rule extraction from neural networks (an interesting algorithm is described in [34]). Once again anyway, the AUC value compares consistently worse than accuracy and Brier score.

We report a sample tree in fig. 5.5; although accuracy was not as high as in simpler datasets, the algorithm still produced very stable results, with the first two conditions of the tree being at most slight variations of the reported ones.

The first classifier extract 77 patients from the dataset, 73 of whom are ill. This is a highly significant area, where the $p$-value of the null hypothesis is roughly $10^{-28}$. The particular genetic combination reported in this rule appears to be favorable to developing cancer: removing *age* and *packyears* from the conditions yielded a subpopulation of 103 patients, 74% of whom were ill — still a high risk value, compared to the baseline of 35%.

The second classifiers, which applies on the people not taken into account by the first one, finds a subset of 22 people, with 18 being ill. Considering that at this point the baseline risk is $(124-73)/(355-77) \approx 0.18$, this subset is still very interesting ($p$-value $\approx 10^{-11}$). Again, a genotype causing a higher risk is identified: on the whole dataset, the second classifier without *age* and *packyears* conditions extracts a subset of 70 patients, 74% of whom were ill.

Since this was a case-control study, the dataset is not a representative sample of the global population. It is then important to underline that the identified risk values are not realistic, and do not represent the actual risk of a random person satisfying their conditions. However, the result it still valid with respect to the identified conditions: the genotypes reported by the classifiers will generally have a worse prognosis than the average — although a different study design is necessary to estimate the exact values.

## 5.5 Discussion

Analysis of the experimental results gives many information on the benefits and limitations of HCS.

The first — and for us most important — information, is that HCS is able to produce results comparable with other current machine learning algorithms. The correction to significance required by multiple hypothesis testing are considered by some to deprive most results of any significance, and generally impair research [102]. With this in mind, applying statistical hypothesis testing on such a broad scale was not guaranteed to produce any significant results at all. Instead, the algorithm is able to identify extremely significant regions, with very low $p$-values even after conservative multiple hypothesis testing corrections like Bonferroni's, also in smaller datasets like HNSCC or WBC.

Results are generally encouraging from the classification accuracy point of view, although this is not HCS primary goal: the fitness calculation only indirectly favours perfect classification, and can (and does, as shown with the Mushroom dataset) prefer larger, impure subsets to smaller, pure subsets. Comparison with very expressive non-linear methods, like neural networks, is generally at a loss for HCS (with the noticeable exception of HNSCC). However, we believe that the impairment of performance is acceptable, when we factor in the immediate interpretability of HCS results. Notice that research on XCS recently proposed classifiers with non-axis parallel borders, which greatly improve expressiveness [24]. Such an improvement anyway comes at the cost of seriously impairing interpretability; we then decided the extra flexibility was not worth the subsequent necessity of complex reasoning to understand the result.

A weak point of HCS is the evaluation of performance through AUC, where the algorithm appears to have some native disadvantage compared to other solutions. We believe this can be connected to the very low number of diverse risk estimates produced by HCS, which produces a structure with low discrimination power. This interpretation can be partly confirmed by C4.5 consistently showing lower AUC values as well. It would be interesting to embed some simple risk estimate algorithm inside the leaves of the final tree (logistic regression comes to mind), in order to "smooth" the prediction and evaluate its impact on AUC and the other measures. Another strategy which can be expected to improve AUC is maximization of this value as an objective of $\mathcal{F}_{\text{Lim}}$ training, instead of minimization of the Brier score.

The comparison with interpretable methods is more favourable to HCS, although not decisive. C4.5 is often on par, and sometimes greatly outperformed. Whether this is due to the vastly higher number of conditions explored, or to the radically different fitness function, remains to be discovered. On the classifier systems side, HCS performance seems comparable to XCS and UCS, with respect to classification accuracy. From the interpretability point of view however, the rules have exactly the same structure, so they are equally readable in both algorithms. HCS however produces much more compact rule sets — approx. 8 times less than XCS with a ruleset reduction algorithm [125]. Some indirect information on UCS suggests that it generates half of the rules produces by XCS, but we could not find direct claim supporting this.

XCS attempts to find rules which split target classes as much as possible, in order to increase accuracy of the classifiers. In the datasets where this is not possible, the result is that many small-size, locally accurate (in the training set) rules are generated, only to obtain the same test performance as a bigger, purposely inaccurate rule. On the other hand, some problems show exactly this kind of highly fragmented, 100% accuracy fitness landscape: the *parity*, *multiplexer* or *intertwined spirals* problems come to mind. On this kind of datasets, HCS is bound to produce sub-optimal results. We believe however that such level of misleading fragmentation would be rare to find in the class of real-world problems HCS is designed to tackle.

# Chapter 6

# Summary and conclusions

This thesis presented a new Machine Learning algorithm designed for exploration and understanding of medical datasets. The algorithm, called HCS, lies at the crossroads of statistics, decision trees and learning classifiers systems. The learning classifier system is used to generate classifiers which identify subsets of the data. Statistical hypothesis testing is applied to distinguish between expected and unexpected classifiers, directing the LCS search process towards the unexpected results. Decision trees finally come into play in order to combine several classifiers together in a single model of the data.

The algorithm has been designed with medical use in mind, and for this objective fulfils several requirements, all coming from the kind of inputs provided and outputs required in the medical research area.

- HCS can deal with data containing several, different types of attributes. Medical data can contain values coming from diverse kinds of tests. In HCS, mixing categorical, ordinal, numerical data is possible without any particular preprocessing. Since the attributes are tested separately, there is no need to find a sensible way to combine them.

- The algorithm natively copes with missing values. Missing data are very common in medical datasets; HCS does not require the dataset to be complete in order to work. Missing data are treated exactly as unknown, and never substituted with any other value — which, as much educated as possible, still remains a guess.

- HCS does not try to reach perfect modelling at all costs. The accuracy value at which the algorithm stops is decided by statistical analysis, which take into account both the precision of the model, and the amount of data supporting it.

- It produces interpretable models. HCS outputs a tree of classifiers: basically a decision tree where each internal node is a conjunction of conditions on the attributes. Each leaf is a model, built upon the data which fulfilled (or did not fulfil) the conditions up to the root.

The main engine of HCS is a genetic algorithm which looks for classifiers, inspired by the XCS learning classifier system. Each classifier is a conjunction of conditions on single attributes of the data. HCS is similar to XCS with respect to the search strategy, but has a very different fitness function. In our algorithm, fitness is calculated through statistical hypothesis testing. In particular, the algorithm starts with a null hypothesis of independence between the attributes and the target. It then uses the genetic search to find classifiers which isolate areas where this hypothesis is rejected, that is areas with extremely low likelihood to exist if the null hypothesis is true, by minimizing the $p$-value of the hypothesis. Once search is over, the classifier with minimum $p$-value is used to split the dataset into two parts, and to recursively restart the algorithm on both subsets. The search ends when the minimum $p$-value found is not lower than the significance value, which is corrected for multiple hypothesis testing, or trained with an internal cross-validation procedure.

HCS was tested upon five real-world dataset, four of which coming from the UCI repository [90]. The tests first involved deciding between roulette wheel and tournament selection, and between binomial or hypergeometric hypothesis modelling. As regards the fitness function, the tests showed a little but consistent advantage of binomial upon hypergeometric testing. As regards selection, no significant difference was detected. We decided then to choose binomial fitness and tournament selection for subsequent development.

Then, the obtained results were compared with those reported in literature by other well-known machine learning methods, and obtained by us upon new execution with standard implementations. The results were comparable, and sometimes superior, to those obtainable with the C4.5 algorithm. From the expressiveness point of view, this is not surprising, since the set of trees explored through HCS fully contains the set of trees explored by C4.5. However, the search process and the fitness function are very different. C4.5 uses a deterministic search, while HCS employs a stochastic algorithm. C4.5

adopts a fitness measure related to information theory, while HCS uses statistical hypothesis testing. It appears that HCS has generally similar results to C4.5, but has an advantage on those datasets (like HNSCC) where an interaction between different factors is necessary to produce visible results. It moreover produces more compact trees, which improves understandability.

With respect to subsymbolic methods, like neural networks or support vector machines, HCS is lacking expressive power, by not being able to test linear combinations of the attributes. This was however necessary, in order to maintain the high readability property of the model. The lack of power anyway did not generally impair performance too much, and on one of the tests (again the HNSCC dataset) it appeared that HCS can produce models of the data which are not so easily discovered by subsymbolic methods.

There are many directions of further development of HCS.

- Fitness calculation is a major bottleneck of the algorithm. Speed would greatly benefit from a closed formula which could quickly approximate the calculation of the log of the $p$-value. If a mathematical approach to this problem is unviable, we could resort to heuristic methods, like genetic programming [66], to identify possible candidates. This problem is especially noticeable with bigger datasets (like Mushroom), where an exact fitness calculation is currently unfeasible.

- Binomial and hypergeometric fitness measures could be tested against classical decision trees fitness measures — most notably, entropy-based and $\chi^2$-based [84]. The treatment of missing data applied in decision trees could also be adopted, and compared with our method.

- The algorithm should become *interactive*. At each external step, once the genetic search is over, the user should be allowed to see the final population — which, thanks to niching, will have many diverse classifiers, covering various aspects of the dataset. He could then choose the most interesting classifier as the basis for the successive iteration, rather than letting the system blindly pick the one with highest fitness. He could moreover tune the classifiers parameters in order to carry higher biological significance. For instance, a real-valued condition could read $x \in [0.01, 0.5]$. The 0.01 value could carry a real significance: if $x$ was related to smoke, it could distinguish smokers from non-smokers. But it could also be an artifact of the dataset, where merging also 0.0 values would just slightly reduce the fitness. Since these decisions affect the subsequent building of the classifier tree by changing the set of data, they should be taken interactively during tree construction.

- Another exploration algorithm should be included, which allowed to find alternative classifiers matching the same instances. It is possible, particularly in datasets with many variables, that approximately the same set of instances can be identified through different conjunctions of conditions. Once an interesting subset has been found, HCS should show all the possible ways to define it, in order to let the user find the most meaningful one for the examined domain.

- Local, informed search could be integrated as a genetic operator. The search algorithm currently relies on mutations in order to fine-tune a good classifier, once its general position has been detected. This is inefficient, and leads to slow convergence. A local search method should be used to speed up tuning of classifier conditions.

- The general HCS algorithm can be applied to a variety of problems, more complex than two-class classification. Higher-complexity models could be used instead of binomial (for instance, a logistic regression model could be fit on the data). More than two classes could be analyzed, either by using specific models, or by iterative methods like pairwise coupling [129]. The system could be adapted for *survival analysis* problems, by applying the appropriate hypothesis tests (the logrank test [12], for instance).

Presenting HCS, we have the ambitious goal to suggest a research line which combines the creativity and computational power of Machine Learning with the soundness of Statistics. We would like HCS to be a step towards this stimulating direction.

# Appendix A

# Testing multiple classifiers

We hereby attempt to calculate the proper multiple hypothesis testing correction for HCS algorithm, which is necessary to find the most appropriate value for $\mathcal{F}_{\text{Lim}}$. We will use the binomial model. The $H_0$ hypothesis will therefore be that the dataset $D$ has been generated from a binomial distribution with $\rho = Q/T$ parameter, and that the values of the attributes are independent of the target. We will require the following two definitions, which are the probability mass function of the binomial and hypergeometric distributions respectively:

$$f_B(q,t) = \binom{t}{q} \rho^q (1-\rho)^{t-q} \tag{A.1}$$

$$f_H(q,t) = \frac{\binom{Q}{q}\binom{T-Q}{t-q}}{\binom{T}{t}} \tag{A.2}$$

We will moreover define the $p$-value of obtaining $q$ positive instances out of $t$ extractions under the binomial distribution:

$$g_B(q,t) = \sum_{\substack{v \in 0,\ldots,t \\ f_B(v,t) \leq f_B(q,t)}} f_B(v,t) \tag{A.3}$$

Under the $H_0$ hypothesis, a classifier can be regarded as a random extraction of instances from the dataset $D$. We will call $P_t$ the random variable describing the distribution of the $p$-value of a classifier $c$ having $t$ size — that is, such that $\#(D/_c) = t$. We will first look for the CDF of $P_t$, which is defined as $\mathbb{P}(P_t \leq x)$, and describes the probability that a random $t$-size classifier will have a $p$-value less than $x$.

The probability that a random $t$-size classifier has a $p$-value less than $x$ is obviously the sum of the probabilities to obtain classifiers with lower $p$-values. Since the dataset $D$ is fixed, the number $q$ of positive instances in a random sample of $t$ size from the dataset $D$ necessarily follows a hypergeometric distribution, with $t$, $Q$, $T$ parameters. Therefore, the probability for a $t$-size classifier to obtain $q$ positive values is exactly $f_H(q,t)$. The CDF of $P_t$ is thus

$$\mathbb{P}(P_t \leq x) = \sum_{\substack{v \in 0,\ldots,t \\ g_B(v,t) \leq x}} f_H(v,t) \tag{A.4}$$

This is a step function, which changes value whenever $x$ reaches a new $p$-value of $g_B(v,t)$. The previous equation can then be written in a slightly simplified format:

$$\mathbb{P}(P_t \leq g_B(q,t)) = \sum_{\substack{v \in 0,\ldots,t \\ f_B(v,t) \leq f_B(q,t)}} f_H(v,t) \tag{A.5}$$

Now that the distribution of the $p$-value for a single classifier has been defined, we can describe the distribution for the minimum $p$-value of many classifiers. We will call $Q_i$ the $p$-value distribution for the $i$th classifier; if this classifier has size $t$, we will have $Q_i \sim P_t$. We will moreover call $M$ the variable describing the minimum of all the $p$-values. We have $M \leq x$ when at least one of the $Q_i$ is $\leq x$, or equivalently when not all of the $Q_i$ are $> x$. Since each $Q_i$ is independent of the others, we have

$$\mathbb{P}(M \leq x) = 1 - \prod_i \mathbb{P}(Q_i > x) = 1 - \prod_i (1 - \mathbb{P}(Q_i \leq x)) \tag{A.6}$$

For instance, $\mathbb{P}(M \leq 0.01) = 0.8$ means that we will get a classifier with $p$-value less than 0.01 with 0.8 probability. We should therefore adjust our confidence level as follows: if we want the $\alpha$ of all the tests to have a certain value, we should take only classifiers with a $p$-value less than $\hat{\alpha}$, defined as

$$\hat{\alpha} = \operatorname*{argmax}_x \mathbb{P}(M \leq x) \leq \alpha \tag{A.7}$$

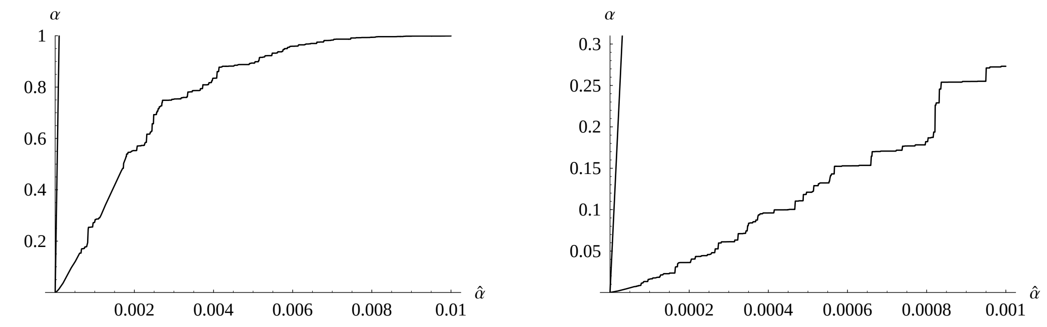


Figure A.1: Plot of $\mathbb{P}(M \leq x)$, with parameters $Q = 60$, $T = 180$, $n = 10000$ (jagged line), and the corresponding Bonferroni correction for 10000 tests (straight line). The graph on the right is a closeup of the graph on the left.

Unfortunately, it was not possible to find a closed formula for $\hat{\alpha}$, both due to the definition of $P_t$, and to the fact that the distribution of the size of the classifiers is a priori unknown. The graph in Fig. A.1 gives a hint of the shape of the true correction, with respect to Bonferroni's correction, for a particular choice of the $Q$, $T$ and $n$ parameters. To draw the plot, we first generate $n$ random numbers, uniformly distributed in the $\{0, \dots, T\}$ set. These values will be the sizes of the tested classifier. We then plot the $\mathbb{P}(M \leq x)$ function, according to the previous set of classifiers. The plot for Bonferroni correction is instead simply $\alpha = n\hat{\alpha}$. To reach the standard $\alpha = 0.01$ global confidence level, the graph reports that the $p$-value of a classifier should be lower than approximately $7 \times 10^{-5}$. The Bonferroni correction instead demands the classifier to have a $p$-value lower than $10^{-6}$.

It is important to notice that our derivation applies to $n$ randomly generated, independent classifiers. This is definitely not the case for the classifiers created through the genetic algorithm: the evolutionary mechanism builds strongly correlated classifiers, and with the specific purpose to minimize their $p$-value. More generally, the objective of a search algorithm is to find the maximum (or the minimum) of a set, by cleverly selecting a small subset of sampling points. If we were certain that the GA returns the best classifier of the whole $\mathcal{C}$ set, we should then take $n = \#(\mathcal{C})$ in the previous formulas. Anyway, calculating $\#(\mathcal{C})$ is a complex task itself (see App. B), and the genetic algorithm does not guarantee the actual minimum is found.

For all these reasons, we could not implement the proper multiple hypothesis test correction, and we fell back on the classical Bonferroni correction, applied to $n$ independent tests. Whether this gives a good approximation of the statistically correct $\mathcal{F}_{\text{Lim}}$ value, will be checked through experiments.

# Appendix B

# Evaluating $\mathcal{C}$ size

The previous chapter raised the problem to calculate the size of the set $\mathcal{C}$ of all possible classifiers.

This could seem in principle an easy task. Each classifier is a composition of $s$ conditions. It should then be sufficient to calculate how many conditions can be built for attribute 1, how many for attribute 2, etc., and then multiply the numbers together.

This simplistic approach however greatly overestimates the total number of different classifiers. Many different sets of conditions can in fact produce the same subset of data. For instance, consider a dataset where only a single instance has the 0 value in its first attribute. Every classifier with the $x_1 = 0$ condition will then produce the same subset of the data, as long as the other conditions do not rule out the pattern. Another, more striking example is the number of possible different combinations of conditions which produce an empty subset.

The true size of the $\mathcal{C}$ set should then be calculated by counting all the *different* subsets obtainable with a conjunction of conditions on the attributes. This approach presents two major problems:

- It requires enumerating all the possible combinations of conditions, and calculate the subset they isolate. This task is too complex to be performed

- Even if the previous step could be accomplished, it would require to store all the identified subsets, and to test new subsets against the previous one in order to see whether we built a new subset or not

We think therefore that a precise calculation of the magnitude of $\mathcal{C}$ is not feasible. We decided then to proceed with an estimate of this value for a specific dataset. We choose the WBC dataset for its simplicity.

The dataset contains 9 integer attributes, ranging from 1 to 10. Each condition is then of the $x \in \{m, \ldots, M\}$ kind, where $m, M \in \{1, \ldots, 10\}$ and $m \leq M$. This gives 55 conditions for each attribute. The last attribute does never have the 9 value, so it can actually show 45 conditions. The number of apparently different classifiers for this dataset is then $55^8 \times 45 \approx 4 \times 10^{15}$.

In order to produce an estimate, we approximated the solutions of both problems outlined earlier. For the first problem, decided to randomly sample the space of conditions. We took many classifier samples, by choosing a random condition for the first attribute with uniform probability among the 55, a random condition for the second one, and so on. We stopped building the classifier when all the conditions were chosen, or if the partial subset of conditions already identified an empty set of instances.

For the second problem, we simplified the estimate by considering all non-empty subsets different from each other. We had then to count the number of samples which produced a non-empty subset, over the total number of samples. The estimate can finally be accelerated by noticing that, when we stop building a classifier because a partial number of conditions already produces an empty subset, all the possible combinations of conditions for the remaining attributes will produce an empty subset too. In such a case we can therefore add the number of possible classifiers to the total.

With this algorithm, a sample of $10^7$ randomly built classifiers estimated that the proportion of non-empty classifiers over the total is $1.7 \times 10^{-7}$. Combining the estimate with the0 total number of apparently different classifiers, we obtain that the number of actually different classifiers for the WBC dataset is $\approx 6 \times 10^8$.

# Bibliography

[1] Mickel Aickin and Helen Gensler. Adjusting for multiple testing when reporting research results: The Bonferroni vs Holm methods. *American Journal of Public Health*, 86(5):726–728, May 1996.

[2] S. Andreassen, C. Riekehr, B. Kristensen, H.C. Schønheyder, and L. Leibovici. Using probabilistic and decision-theoretic methods in treatment and prognosis modeling. *Artif. Intell. Med.*, 15:121–134, 1999.

[3] R. Andrews, J. Diederich, and A. B. Tickle. A survey and critique of techiques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8:373–389, 1995.

[4] A.J. Bagnall and G.C. Cawley. Learning classifier systems for data mining: A comparison of XCS with other classifiers for the Forest Cover data set. In *Proceedings of the IEEE/INNS International Joint Conference on Artificial Neural Networks (IJCNN-2003)*, volume 3, pages 1802–1807. IEEE Press, 2003.

[5] F. Baronti, V. Maggini, A. Micheli, A. Passaro, A. M. Rossi, and A. Starita. A preliminary investigation on connecting genotype to oral cancer development through XCS. In Bruno Apolloni, Maria Marinaro, and Roberto Tagliaferri, editors, *Biological and Artificial Intelligence Environments*, pages 11–20. Springer, NY, 2006.

[6] Gustavo E. A. P. A. Batista and Maria C. Monard. An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17(5):519–533, 2003.

[7] K. P. Bennett and J. A. Blue. A support vector machine approach to decision trees. In *The 1998 IEEE International Joint Conference On Neural Networks Proceedings*, volume 3, pages 2396–2401, 1998.

[8] Hal Berghel. Cyberspace 2000: Dealing with information overload. *Communications of the ACM*, 40(2):19–24, 1997.

[9] Ester Bernadó-Mansilla and Josep M. Garrell-Guiu. Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

[10] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., 1995.

[11] J. Martin Bland and Douglas G. Altman. Statistics notes: Multiple significance tests: the Bonferroni method. *British Medical Journal*, 310:170, Jan 1995.

[12] J. Martin Bland and Douglas G. Altman. The logrank test. *British Medical Journal*, 328(7447):1073, 2004.

[13] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. Technical Report 11, Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, 8092 Zurich, Switzerland, 1995.

[14] Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16, San Francisco, CA, 1995. Morgan Kaufmann.

[15] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[16] Pierre Bonelli, Alexandre Parodi, Sandip Sen, and Stewart Wilson. NEWBOOLE: a fast GBML system. In *Proceedings of the seventh international conference (1990) on Machine learning*, pages 153–159, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[17] O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[18] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

[19] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[20] G. W. Brier. Verification of forecasts expressed in terms of probability. *Montly weather review*, 78(1):1–3, 1950.

[21] L.D. Brown, T. Cai, and A. DasGupta. Confidence intervals for a binomial proportion and asymptotic expansions. *The Annals of Statistics*, 30:160–201, 2002.

[22] Wray Buntine and Tim Niblett. A further comparison of splitting rules for Decision-Tree induction. *Machine Learning*, 8:75–85, 1992.

[23] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121 – 167, 1998.

[24] M. V. Butz, Pier Luca Lanzi, and S. W. Wilson. Hyper-ellipsoidal conditions in XCS: Rotation, linear approximation, and solution structure. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006)*, pages 1457–1464, 2006.

[25] Martin V. Butz, Kumara Sastry, and David E. Goldberg. Strong, stable, and reliable fitness pressure in XCS due to tournament selection. *Genetic Programming and Evolvable Machines*, 6(1):53–77, 2005.

[26] Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. In P. L. Lanzi and et al., editors, *IWLCS 2000*, volume 1996 of *LNAI*, pages 253–272. Springer-Verlag, 2001.

[27] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002.

[28] Krzysztof J. Cios and William G Moore. Uniqueness of medical data mining. *Artificial Intelligence in Medicine*, 26(1–2):1–24, 2002.

[29] Joseph A. Cruz and David S. Wishart. Application of machine learning in cancer prediction and prognosis. *Cancer Informatics*, 2:59–78, 2006.

[30] Janez Demšar. Statistical comparison of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, Jan 2006.

[31] Thomas G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[32] Pedro Domingos. A unified bias-variance decomposition and its applications. In *Proc. 17th International Conf. on Machine Learning*, pages 231–238. Morgan Kaufmann, San Francisco, CA, 2000.

[33] S. Dreiseitl and L. Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, 35:352–359, 2002.

[34] Włodzisłav Duch, Rafał Adamczak, and Krzysztof Grąbczewski. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks*, 11(2):1–31, Mar 2000.

[35] J. P. Egan. *Signal Detection Theory and ROC Analysis.* Academic Press, New York, 1975.

[36] Andre Elisseeff, Theodoros Evgeniou, and Massimiliano Pontil. Stability of randomized learning algorithms. *Journal of Machine Learning Research*, 6:55–79, 2005.

[37] F. Esposito, D. Malerba, G. Semeraro, and J. Kay. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.

[38] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. San Mateo, CA: Morgan Kaufmann, 1990.

[39] T. Fawcett. Roc graphs: Notes and practical considerations for researchers. Technical report HPL-2003-4, HP Laboratories, 2003.

[40] R. A. Fisher. *Statistical Methods for Research Workers.* Oliver and Boyd, Edinburgh, 14th edition, 1970.

[41] M. Frean. The Upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209, 1990.

[42] A. A. Freitas. On rule interestingness measures. *Knowledge-Based Systems*, 12:309–315, 1999.

[43] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.

[44] N.I.R. Friedman, M. Linial, I. Nachman, and D. Peer. Using bayesian network to analyze expression data. *J. Comput. Biol*, 7:601–620, 2000.

[45] Jean D. Gibbons and John W. Pratti. P-Values: Intepretation and methodology. *The American Statistician*, 29(1):20–25, Feb 1975.

[46] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[47] Helen F. Gray, Ross J. Maxwell, Irene Martinez-Perez, Carles Arús, and Sebastián Cerdán. Genetic programming for classification and feature selection: analysis of $^1$h nuclear magnetic resonance spectra from human brain tumour biopsies. *NMR in Biomedicine*, 11(4-5):217–224, 1998.

[48] Trisha Greenhalgh. How to read a paper: Papers that report diagnostic or screening tests. *British Medical Journal*, 315(7107):540–543, Aug 1997.

[49] S. Haykin. *Neural Networks, A Comprehensive Foundation.* Prentice Hall, 2nd edition, 1999.

[50] D.E. Heckerman and B.N. Nathwani. Towards normative expert systems. II. probability-based representations for efficient knowledge acquisition and inference. *Meth. Inform. Med.*, 31:106–116, 1992.

[51] John H. Holland. *Adaptation in natural artificial systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[52] John H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in theoretical biology, 4*. New York: Plenum, 1976.

[53] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[54] J. H Holmes, J. A. Sager, and W. B. Bilker. Methods for covering missing data in XCS. In M Keijzer, editor, *Late Breaking Papers at GECCO 2004*, Seattle, WA, June 2004.

[55] John H. Holmes. Discovering risk of disease with a learning classifier system. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann.

[56] John H. Holmes and Warren B. Bilker. The effect of missing data on learning classifier system learning rate and classification performance. In Lanzi et al., editor, *IWLCS 2002*, volume 2661 of *LNAI*, pages 46–60. Springer-Verlag, 2003.

[57] John H. Holmes and Jennifer A. Sager. Rule discovery in epidemiologic surveillance data using EpiXCS: an evolutionary computation approach. In S. Miksch, J. Hunter, and E. Keravnou, editors, *Artificial Intelligence in Medicine*, volume 3581 of *LNAI*, pages 444–452. Springer-Verlag, 2005.

[58] John H. Holmes, Jennifer A. Sager, and Warren B. Bilker. A comparison of three methods for covering missing data in XCS. In *Seventh International Workshop on Learning Classifier Systems (IWLCS-2004) during the Genetic and Evolutionary Computation Conference (GECCO 2004)*, 2004.

[59] David W. Hosmer and Stanley Lemeshow. *Applied logistic regression.* Wiley, New York, 1989.

[60] Jin Huang, Jingjing Lu, and Charles X. Ling. Comparing naive Bayes, decision trees, and SVM with AUC and accuracy. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM03)*, 2003.

[61] Mitsuru Ikeda, Takeo Ishigaki, and Kazunobu Yamauchi. Relationship between Brier score and area under the binormal ROC curve. *Computer Methods and Programs in Biomedicine*, 67:187–194, 2002.

[62] M. F. Jefferson, N. Pendleton, S. B. Lucas, and Horan. M. A. Comparison of a genetic algorithm neural network with logistic regression for predicting outcome after surgery for patients with nonsmall cell lung carcinoma. *Cancer*, 79(7), 1995.

[63] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[64] T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, Berlin, 3rd edition, 2001.

[65] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artif. Intell. Med.*, 23(1):89–109, 2001.

[66] John R. Koza. *Genetic programming — On the programming of computers by means of natural selection.* MIT Press, Cambridge, MA, 1992.

[67] N. Lavrač, E. Keravnou, and B. Zupan, editors. *Intelligent data analysis in medicine and pharmacology.* Kluwer, 1997.

[68] N. Lavrač, I. Kononenko, E. Keravnou, M. Kukar, and B. Zupan. Intelligent data analysis for medical diagnosis: using machine learning and temporal abstraction. *AI Commun.*, 11(3-4):191–218, 1998.

[69] Nada Lavrač. Selected techniques for data mining in medicine. *Artificial Intelligence in Medicine*, 16:3–23, 1999.

[70] E. L. Lehmann. *Testing Statistical Hypotheses.* Springer texts in statistics. Springer, New York, 2nd edition, 1997.

[71] P. J. G. Lisboa. A review of evidence of health benefit from artificial neural networks in medical intervention. *Neural Networks*, 15(1):11–39, 2002.

[72] J. Listgarten and S. et al. Damaraju. Predictive models for breast cancer susceptibility from multiple single nucleotide polymorphisms. *Clinical Cancer Research*, 10:2725–2737, Apr 2004.

[73] R. J. Little and D. B. Rubin. *Statistical Analysis with Missing Data.* John Wiley and Sons, New York, 1987.

[74] W. Z. Liu, A. P. White, M. T. Hallissey, and J. W. L. Fielding. Machine learning techniques in early screening for gastric and oesophageal cancer. *Artificial Intelligence in Medicine*, 8:327–341, 1996.

[75] Peter J. F. Lucas and Ameen Abu-Hanna. Prognostic methods in medicine. *Artif. Intell. Med.*, 15(2):105–119, February 1999.

[76] P.J.F. Lucas. Analysis of notions of diagnosis. *Artif. Intell.*, 105(1-2):295–343, 1998.

[77] P.J.F. Lucas, H. Boot, and B.G. Taal. Computer-based decision-support in the management of primary gastric non-Hodgkin lymphoma. *Meth. Inform. Med.*, 37:206–219, 1998.

[78] P.J.F. Lucas, N.C. De Bruijn, K. Schurink, and I.M. Hoepelman. A probabilistic and decision-theoretic approach to the management of infectious disease at the ICU. *Artif. Intell. Med.*, 19(3):251–279, 2000.

[79] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1 & 18, Sep 1990.

[80] M. Mezard and J.P. Nadal. Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics*, A 22:2191–2203, 1989.

[81] A. Micheli. *Recursive Processing of Structured Domains in Machine Learning.* PhD thesis, Department of Computer Science, University of Pisa, 2003. TD-13/03.

[82] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[83] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. YALE: Rapid prototyping for complex data mining tasks. In *In Proceedings of the 12th ACM SIGKDD International Conference*

*on Knowledge Discovery and Data Mining (KDD 2006)*. ACM Press, 2006.

[84] John Mingers. An empirical comparison of selection measures for Decision-Tree induction. *Machine Learning*, 3:319–342, 1989.

[85] Tom M. Mitchell. Generalization as search. *Artificial intelligence*, 18(2):203–226, 1982.

[86] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[87] Patrick M. Murphy and Michael J. Pazzani. Exploring the decision forest: An empirical investigation of Occam's razor in decision tree induction. *Journal of Artificial Intelligence Research*, 1:257–275, 1994.

[88] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Journal Data Mining and Knowledge Discovery*, 2(4):345–389, Dec 1998.

[89] P. B. Nemenyi. *Distribution-free multiple comparisons*. Phd thesis, Princeton University, 1963.

[90] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases [`http://www.ics.uci.edu/∼mlearn/MLRepository.html`], 1998.

[91] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In Thomas G. Dietterich, Sue Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2001. MIT Press.

[92] Po Shun Ngan, Man Leung Wong, Wai Lam, Kwong Sak Leung, and Jack C.Y. Cheng. Medical data mining using evolutionary computation. *Artificial Intelligence in Medicine*, 16(1), 1999.

[93] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.

[94] Thomas V. Perneger. What's wrong with Bonferroni adjustments. *British Medical Journal*, 316:1236–1238, Apr 1998.

[95] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases*. AAAI/MIT Press, Cambridge, MA, 1991.

[96] Riccardo Poli, S. Cagnoni, and G. Valli. Genetic design of optimum linear and nonlinear QRS detectors. *IEEE Transactions on Biomedical Engineering*, 42(11):1137–41, 1995.

[97] Karl Popper. *The Logic of Scientific Discovery.* Basic Books, New York, NY, orig.: logik der forschung, 1934 edition, 1959.

[98] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Procedings of the Fifteenth International Conference on Machine Learning*, pages 445–553, 1998.

[99] J. R. Quinlan. Simplifying decision trees. In B. Gaines and J. Boose, editors, *Knowledge Acquisition for Knowledge-Based Systems*, pages 239–252. Academic Press, London, 1988.

[100] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[101] Russel Reed. Pruning algorithms — a survey. *IEEE Transactions on Neural Networks*, 4(5):740–748, Sep 1993.

[102] K. J. Rothman. No adjustments are needed for multiple comparisons. *Epidemiology*, 1(1):43–46, Jan 1990.

[103] Y. D. Rubinstein and T. Hastie. Discriminative vs. informative learning. In *Proceedings of the third international conference on Knowledge Discovery and Data Mining*, pages 49–53. AAAI Press, 1997.

[104] J. L. Schafer and J. W. Graham. Missing data: our view of the state of the art. *Psychological Methods*, 7(2):147–177, 2002.

[105] M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270:467–470, 1995.

[106] M. Schumacher, R. Roßner, and W. Vach. Neural networks and logistic regression: Part I. *Computational Statistics and Data Analysis*, 21:661–82, 1996.

[107] J. P. Shaffer. Multiple hypothesis testing. *Annual Review of Psychology*, 46:561–584, 1995.

[108] W. R. Shankle, Subramani Mani, Michael J. Pazzani, and Padhraic Smyth. Detecting very early stages of dementia from normal aging with machine learning methods. In E. Keravnou, C. Garbay, R. Baud, and Wyatt, editors, *Lecture Notes in Artificial Intelligence: Artificial Intelligence in Medicine, AIME97*, volume 1211 of *LNAI*, pages 73–85. Springer-Verlag, 1997.

[109] G. D. Smith and S. Ebrahim. Data dredging, bias, or confounding. *British Medical Journal*, 325(7378):1437–8, Dec 2002.

[110] A.J. Smola and B. Schölkopf. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.

[111] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[112] B. Ster and A. Dobnikar. Neural networks in medical diagnosis: Comparison with other methods. In A. et al. Bulsari, editor, *Proceedings of EANN'96*, pages 427–430, 1996.

[113] Christopher Stone and Larry Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11(3):299–336, 2003.

[114] P. Szolovits, R.S. Patil, and W.B. Schwartz. Artificial intelligence in medical diagnosis. *Ann. Intern. Med.*, 108(1):80–7, Jan 1988.

[115] Gary Tietjen. Recursive schemes for calculating cumulative binomial and Poisson probabilities. *The American Statistician*, 48(2):136–137, may 1994.

[116] Peter Turney. Technical note: Bias and the quantification of stability. *Journal of Machine Learning*, 20:23–33, 1995.

[117] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[118] L.C. van der Gaag, S. Renooij, C.L.M. Witteman, B.M.P. Aleman, and B.G. Taal. Probabilities for a probabilistic network: a case study in oesophageal cancer. *Artif. Intell. Med.*, 25:123–148, 2002.

[119] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[120] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 1995.

[121] Geoffrey I. Webb and Kai Ming Ting. On the application of roc analysis to predict classification performance under varying class distributions. *Machine Learning*, 58(1):25–32, 2005.

[122] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2), 1995.

[123] Stewart W. Wilson. Generalization in the XCS classifier system. In John R. Koza and et al., editors, *Proceedings of GECCO 1998*, pages 665–674. Morgan Kaufmann, 22-25 1998.

[124] Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In Lanzi et al., editor, *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *LNAI*, pages 209–219. Springer-Verlag, 2000.

[125] Stewart W. Wilson. Compact rulesets from XCSI. In P. L. Lanzi and et al., editors, *IWLCS 2001*, volume 2321, pages 197–210. Springer-Verlag, 2001.

[126] Stewart W. Wilson. Mining oblique data with XCS. In P. L. Lanzi and et al., editors, *IWLCS 2000*, volume 1996 of *LNAI*, pages 158–174. Springer-Verlag, 2001.

[127] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, San Francisco, 2$^{\text{nd}}$ edition, 2005.

[128] T. H. Wonnacott and R. J. Wonnacot. *Introductory Statistics.* Wiley, New York (NY), 1972.

[129] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.

[130] Y. Yu and M. C. Schell. A genetic algorithm for the optimization of prostate implants. *Medical physics*, 23(11):2085–91, 1996.

[131] Frederick Zarndt. A comprehensive case study: An examination of machine learning and connectionist algorithms. Master thesis, Brigham Young University, 1995.

[132] I. Zelic, I. Kononenko, N. Lavrač, and V. Vuga. Induction of decision trees and bayesian classification applied to diagnosis of sport injuries. *Journal of Medical Systems*, 21(6):429–444, 1997.

[133] Zhi-Hua Zhou. Three perspectives of data mining. *Artificial Intelligence*, 143(1):139–146, 2003.

[134] Blaz Zupan, Janez Demšar, Michael W. Kattan, J. Robert Beck, and I. Bratko. Machine learning for survival analysis: a case study on recurrence of prostate cancer. *Artif. Intell. Med.*, 20(1):59–75, September 2000.