Automatic problem-specific hyperparameter optimization and model selection for supervised machine learning

MASTER THESIS

by

Róger Bermúdez-Chacón

supervised by:

Prof. Dr. Gaston H. Gonnet

Dr. Kevin Smith

Oct. 2013 - Apr. 2014

ETH Zurich

Zurich, 2014

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Acknowledgements

Thanks to statistics people and lmsc Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Zurich, April 15th, 2014 R.B.C.

Abstract

Despite the wide range of aplicability of machine learning techniques, the tools and methods available nowadays still require of expert knowledge to guide the decisions about the right machine learning approach for a given problem. Such expert knowledge is rarely available and holds a degree of subjectivity that makes it suboptimal for suggesting the machine learning strategy that best adapts to a specific problem.

Since different machine learning techniques are suitable for different problems, choosing the right technique and fine-tuning its particular settings are crucial tasks that will directly impact the quality of the predictions obtained. However, deciding what machine learning setup is more appropriate for processing specific data is not an easy task, as the number of choices is usually very large and might even be infinite.

This work attempts to automate the choice of the most suitable machine learning algorithm and its respective settings, in order to select the machine learning approach best suited to perform predictions on a specific set of data, in a flexible and customizable way that enables the user to specify their needs in terms of predictive power, sensitivity, specificity, consistency of the predictions, and speed, among other criteria.

The results obtained show that using the machine learning technique and configuration suggested by this automated approach yields predictions of a much higher quality than testing each technique under its default settings. It is also shown that it is possible to guide the search for optimal configurations of supervised machine learning algorithms by identifying ranges of values for each possible setting that produce good results for most problems, and transferring such knowledge to new problems, to quickly find optimal configurations of the algorithm under given data.

Keywords: Model selection, hyperparameter optimization, Supervised Machine Learning

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

Automatic data processing and analysis is nowadays essential in many disciplines in science and beyond. The unmatched availability of computational resources experienced in present times has stimulated the stablishment of new disciplines that strongly depend on analysis of massive amounts of data, for which humans are no longer suitable, which makes automatic processing by computational methods paramount.

A clear example of this is found in biology. Methods such as *genome-wide association studies*, *high-content screening*, or *gene expression profiling* are relatively new, and rely on the analysis of huge amounts of data that would not have been possible in the past.

Machine learning is a very popular approach to automatic data processing. Machine learning is a branch of artificial intelligence that allows to describe and detect patterns in groups of objects that can be later used for recognition of other objects with similar characteristics.

Machine learning methods that infer such patterns by studying a set of labeled (training) data are known as ***supervised* machine learning algorithms**, as opposite to *unsupervised* machine learning methods, which are not exposed to any training information. Supervised machine learning techniques are the most widely used.

However, since many such methods exist, it is usually not obvious which one to use to address a specific problem, i.e., which method recognizes patterns that serve as a good *model* of the data. Furthermore, each different method usually exposes configuration options that modify its behavior and may enhance or harm its predictive capacity.

Supervised machine learning approaches

Supervised machine learning (SML) methods address two families of problems: classification and regression. In the **classification** problem, objects belong to a class from a given set, and the method should infer rules to assign new objects to their corresponding class. The **regression** problem locates objects in a certain space, and should correctly predict the location of new objects.

Model selection

Different SML methods use different strategies to learn the patterns that they use for prediction. Some of them make assumptions on the structure of the data, and do not give accurate predictions when such assumptions are violated. The choice of the method to use is bound to the structure of the data that they are to predict.

The model selection problem is the evaluation of a series of candidate models in order to select the best one, according to some optimality criterion. The underlying rule learned by a SML algorithm, and used by it to map objects to values, is considered as a model of the object space. To select a SML model then means to evaluate a group of SML algorithms in order to choose the one that best maps objects to their target values.

Hyperparameter optimization

Specific configuration options for each SML method control its internal behavior, and can affect the learning of patterns to use for prediction. Finding the right combination of configuration options (*hyperparameters*) might be as critical to the prediction as selecting the right machine learning method.

An exhaustive evaluation of all possible hyperparameter combinations is not practical, and it is not even possible when any of the hyperparameters adopts values from a continuous range. Numerical optimization approaches can be used to systematically suggest new candidate values expected to improve the prediction of the SML algorithm.

CHAPTER 2
Problem description

This work aims to propose a solution for a two-fold problem: find the SML algorithm that works best for a given set of data, and find which particular choice of configuration settings for that algorithm, if required, yields the best results.

Although both sides of the problem correspond conceptually to different ideas (general approach the former, fine-tunning the latter), they are intrinsically linked and should not be treated as separate problems. This is especially true for the present work, since a poor choice of settings for an otherwise well-suited SML algorithm may be easily outperformed by another, less appropriated SML method, applied on the same data.

The formal description of the problem presented here is given in terms of the classification problem. This description extends naturally to the regression problem, but the specific terminology for regression has been excluded for the sake of brevity.

2.1 Formal definition

The following terms and notation will be used for the rest of this document:

- An **instance** is a representation of an object, encoded as a pair $(\mathbf{x}, y)$, with $\mathbf{x}$ a vector of values for attributes (also called *features*) related to the object, and y a mapping of the object to its corresponding class.
- A **dataset** $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$ is a set of instances on which SML is to be applied.
- An **algorithm** A is a specific implementation of any function that uses known instances $(\mathbf{x}, y)$ (also known as *examples*) to learn patterns or rules to associate the feature values $\mathbf{x}$ to their class y, and uses this information to predict the class y' of unseen instances from their feature values $\mathbf{x}'$.

$$A\colon (\{(\mathbf{x}, y)\}, \{\mathbf{x}'\}) \mapsto \{y'\} \tag{2.1}$$

- $\mathcal{A} = \{A^1, \ldots, A^k\}$ represents the set of all algorithms to evaluate.
- An algorithm A accepts a set of **hyperparameters** $\mathcal{H}_A$ that modify its behavior.
- λ_a is the value of a **hyperparameter** $a \in \mathcal{H}_A$ for some algorithm A.
- $\boldsymbol{\lambda}^i = (\lambda_a^i, \lambda_b^i, \lambda_c^i, \ldots)$ denotes a set of hyperparameter values (or **configuration**) for the i-th algorithm in $\mathcal{A}$.
- Λ^i represents the set of all possible values that $\boldsymbol{\lambda}^i$ can take (i.e. the **hyperparameter space** of the i-th algorithm).
- An algorithm A under a specific configuration $\boldsymbol{\lambda}$ is called a **model**, and represented as $A_{\boldsymbol{\lambda}}$.
- The **scoring** function S measures how good a model $A_{\boldsymbol{\lambda}}$ is at predicting the class of unseen instances drawn from the dataset $\mathcal{D}$.

  $$S\colon A^i \in \mathcal{A}, \boldsymbol{\lambda}^i \in \Lambda^i, \mathcal{D} \mapsto \mathbb{R} \tag{2.2}$$

  The scoring function should be designed in such a way that the better the agreement between class prediction and true class assignment is, the higher the score.

It is assumed that the feature values of the examples $\mathbf{x}$ used for training an algorithm and the feature values to predict labeling $\mathbf{x}'$ are drawn from the same underlying distribution, and that instances with similar feature values tend to belong to the same class.

Under the described context, the model selection and hyperparameter optimization problem can be defined as:

$$A^*_{\boldsymbol{\lambda}^*} = \underset{A^i \in \mathcal{A}, \boldsymbol{\lambda}^i \in \Lambda^i}{\arg\max}\ S(A^i_{\boldsymbol{\lambda}^i}, \mathcal{D}) \tag{2.3}$$

The above equation can be simply stated as "find the algorithm and its parameter values that obtain the best score at predicting labels on the given dataset". It is worth noticing that equation 2.3 is the general form of the optimization process, and as such, only defines the structure and general behavior of the different components of the optimization. In practice, additional details such as the implementation of the scoring function, and the actual exploration of the (possibly infinite) hyperparameter spaces, must be considered.

Furthermore, the assumption that a single model $A^*_{\boldsymbol{\lambda}^*}$ will be *significantly* better than the rest is not guaranteed, and hence returning multiple models as the result of the optimization process should also be considered under certain circumstances.

2.2 Related work

Few initiatives to address the model selection and hyperparameter optimization of SML algorithms have been proposed.

For hyperparameter optimization, the *de facto* method, known as *grid search*, is the exhaustive evaluation of a discretization of the hyperparameter space on a regularly-spaced grid. Such method is affected by the *curse of dimensionality* when a large number of hyperparameters needs to be analyzed, and the step size for discretization is often decided arbitrarily.

Alternative methods to avoid these shortcomings have been proposed. The simplest of all consists on randomly retrieving and evaluating values for the hyperparameters (*random search*), and is studied in the context of SML optimization in [Bergstra and Bengio, 2012]. This method is obviously very slow as it relies only on chance to find good configurations. It does, however, allow for a uniform exploration of the hyperparameter space.

A method based on hierarchical density estimation, and another method based on hierarchical Gaussian processes, are proposed in [Bergstra et al., 2011]. These methods are limited to the optimization of hyperparameter values from a single SML algorithm, and do not consider a systematic selection of a model. Furthermore, the work lacks a statistical analysis of the optimized hyperparameters, and disregards the generalization of the selected configuration to unseen data.

The approach implemented in Auto-WEKA ([Thornton et al., 2013]) does consider model selection and hyperparameter optimization simultaneosly. Their work models the hyperparameter space in a hierarchical way. Auto-WEKA considers generalization as the only criterion for both optimization and model selection.

Statistical frameworks for model selection on SML are described in [Pizarro et al., 2002] and [Demšar, 2006], and some of their ideas have been implemented in the present work.

CHAPTER 3
Methodology

The specific implemented approach to find solutions for equation 2.3 is explained in detail in this chapter. Important components of the solution described here include the modeling of the hyperparameter space and its sampling, the choice of the numerical optimization method, and the definition of the target function to optimize, among others.

The general strategy followed here is summarized in figure 3.1:

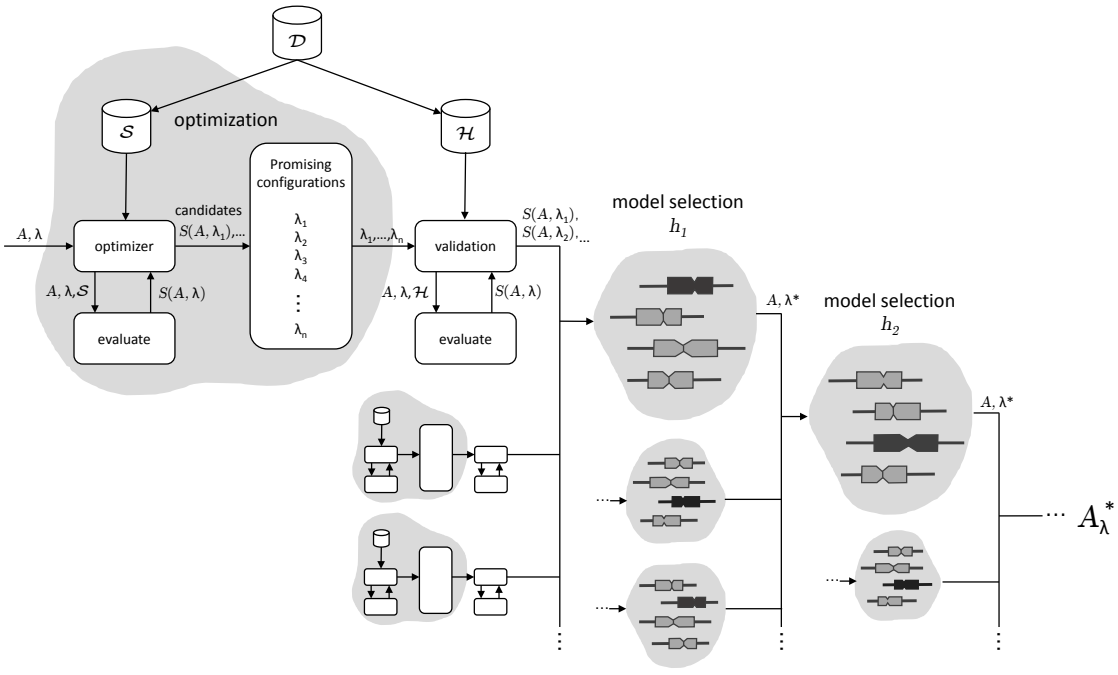


Figure 3.1: Interaction between different components of the implemented hyperparameter optimization and model selection approach.

The entire process is subdivided in two stages. The instances contained in the dataset $\mathcal{D}$ are divided into two subsets $\mathcal{S}$ and $\mathcal{H}$. The **hyperparameter optimization stage** evaluates a large number of models (SML algorithm under a given configuration) on the subset $\mathcal{S}$, in order to

find the ones that exhibit the best predictive performance. The **model selection stage** analyzes the selected models obtained from the hyperparameter optimization stage (candidates) under the subset $\mathcal{H}$ (data unseen to the optimization process), to assess generalization, and uses that information, along with other desirable properties of the models, to systematically rule out suboptimal models and finally return the best.

3.1 Hyperparameter space modeling

Most SML algorithms contain configuration options that control different aspects of their internal behavior, which can be further adjusted to fit specific needs. For example, algorithms that make use of the distance between examples in feature space might accept a distance metric to be specified, or algorithms that internally create trees might allow the user to specify a branching factor (maximum number of branches per node), and so on. Since SML algorithms accept a set of example instances for training and unseen data for predicton as *parameters* (equation 2.1), and since the configuration options control the behavior of the algorithm, they are regarded to as *hyperparameters* of the SML algorithm. The choice of hyperparameter values for a SML algorithm can heavily impact its predictive power.

Different types of hyperparameters exist. Nominal hyperparameters take values from a fixed list of categories that usually correspond to a decision of *how* the SML algorithm performs the predictions. Such **categorical hyperparameters** include, for instance, which kernel to use to compute support vectors in a support vector machine classifier, or whether to consider the distance between neighboring points or not on the prediction of a k-nearest neighbors classifier. Once all these decisions have been made, **numerical hyperparameters** control the values to use for the specific implementation of internal formulas needed for prediction, such as the number of neighbors for k-neighbors classification, or the cost parameter used by a support vector machine as a trade-off between allowing training errors and fitting the support vectors to the training set rigidly.

Other considerations such as restricting the value of a numerical hyperparameter to a fixed range (addressed here by assigning validation rules for each numerical hyperparameter), or encoding dependencies between categorical and numerical or other categorical hyperparameters (implemented here as explained in detail in subsection 3.1.2) must be taken into account when modeling the strategy to explore the hyperparameters of a SML algorithm.

3.1.1 Hyperparameter distributions

Since one of the main objectives of this work is to learn the behavior of SML algorithms under different settings, a systematic way of obtaining valid candidate configurations for performance evaluation is required. In order to achieve this, each hyperparameter is regarded as a random variable and a distribution for its specific type is assigned to it.

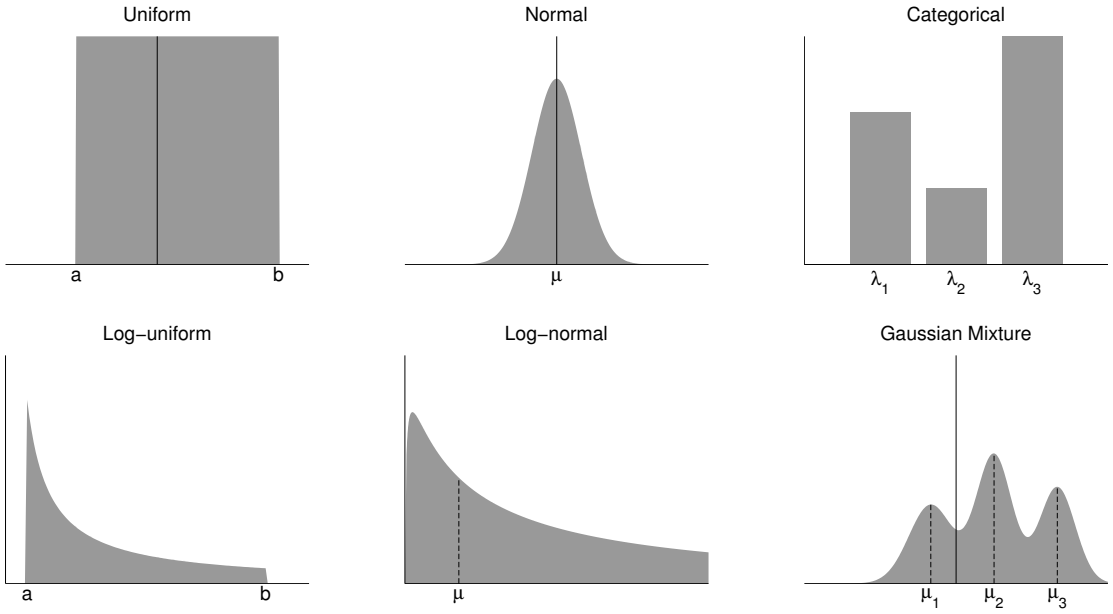


Figure 3.2: Distributions used to model hyperparameters

A one-dimensional distribution is used to sample values for each hyperparameter independently. Section 3.1.2 describes in detail how dependencies between hyperparameters are handled. Drawing values for a hyperparameter from a distribution (other than a Uniform one) will favor some regions of the hyperparameter space over others, which can be useful when there is a good knowledge of what values for the hyperparameter produce good predictions. It is possible that the choice of specific values for the *categorical* hyperparameters affects what values for a numerical hyperparameter yield good results. Because of this, the same numerical hyperparameter is modeled with different distributions for different combinations of the categorical hyperparameters.

The choice of the distribution depends on the nature of each hyperparameter and the knowledge of the range and expected values that the hyperparameter might take. The implemented types of prior distributions, depicted in figure 3.2, are:

- **Categorical** distributions assigned to hyperparameters that can take nominal values from a list of k fixed categories, or to discrete numerical hyperparameters.

$$p(x) = p_1^{[x=1]} \cdot \ldots \cdot p_k^{[x=k]}, \qquad x \in \{1, \ldots, k\} \tag{3.1}$$

  ($[x = i]$ is the *Iverson bracket*: $[P] = 1$ if statement P is true, 0 otherwise)

- **Uniform** distributions assigned to continuous numerical parameters within a bounded range $[a, b]$.

$$p(x) = \frac{1}{b-a}, \qquad a \leq x \leq b \tag{3.2}$$

- **Normal** distributions assigned to continuous unbounded numerical parameters, when a specific value μ and mean variation σ is expected.

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} = \mathcal{N}(\mu, \sigma) \tag{3.3}$$

- **Log-uniform** distributions for hyperparameters that might span over different orders of magnitude.

$$p(x) = \frac{1}{x(\ln b - \ln a)} \tag{3.4}$$

- **Log-normal** distributions assigned to hyperparameters that can not have negative values, and for which a specific value and variability is expected.

$$p(x) = \frac{1}{x\sqrt{2\pi}\sigma} e^{\frac{-(\ln x-\mu)^2}{2\sigma^2}} \tag{3.5}$$

- **Gaussian Mixture Models** (GMMs) are used for continuous hyperparameters that are known to be multimodal.

$$p(x) = \sum_{i=1}^{N} \pi_i \mathcal{N}(\mu_i, \sigma_i), \qquad \sum_{i=1}^{N} \pi_i = 1 \tag{3.6}$$

  GMMs have many parameters and thus should be used when there is solid knowledge of the shape of the hyperparameter distribution. GMMs are learned by analyzing the distribution of each hyperparameter on a large number of datasets and infer promising and harmful regions of the hyperparameter space, as explained in detail in chapter 4.

The values for the category weights, means, variances, and bounds, depend on how each specific hyperparameter is used by the SML algorithm. By default, handcrafted distributions have been designed led by interpretation of the documentation of each SML algorithm, but can be later modified. For instance, a hyperparameter that, according to the documentation, is used by the implementation of the SML algorithm as a ratio, will be assigned a Uniform(0,1) distribution by default, but if the documentation suggests that values around 0.5 often yield good results, a gaussian distribution centered at 0.5 is assigned instead. The parameters for all distributions for all hyperparameters are stored as plain text in a configuration file, and can be manually edited if desired.

3.1.2 Hyperparameter hierarchy

For most SML algorithm implementations, choosing specific values for some hyperparameters may change which other hyperparameters are also used or ignored, and how they are used by the algorithm. For instance, the SVM family of algorithms can use linear, polynomial, sigmoid, or RBF kernels, and if a polynomial kernel is chosen, it is possible to specify its polynomial

degree; such option has no effect when a linear kernel is used. Likewise, it is possible to specify the intependent term for the polynomial and sigmoid functions used as kernels, and such value may have a different impact and different extrema depending on which kernel they operate on. The popular SVM implementation libSVM accepts a single hyperparameter `coef0` for this setting. Disregarding the duality of this hyperparameter and its dependence on the context (selected kernel in this case) when modeling and optimizing it might lead to unwanted results.

What the above example implies is that it is mandatory to consider each hyperparameter for optimization within a context given by the values of other hyperparameters (the **hyperparameter context**). A hierarchical design is chosen, based on two observations:

1. Hyperparameter contexts can be nested. The use and semantics of a hyperparameter may depend on the value of other hyperparameters, which may in turn depend on others.

2. The hyperparameter context is always defined by a set of *categorical*, rather than numerical, hyperparameters. Since the decision of whether or not to use a hyperparameter is a discrete (boolean) value, it does not make sense to consider continuous numerical variables when defining a hyperparameter context.

A specific situation where observation 2 may be violated is when a numerical hyperparameter depends on a *rule* applied on another numerical hyperparameter, rather than on its actual value. For example, if hyperparameter b is only used by the SML algorithm when hyperparameter a adopts values within a given interval, and invalid otherwise. A *virtual* hyperparameter *b_valid* with such rule (which would be a categorical hyperparameter with categories {*True*, *False*}) can be added to the hyperparameter hierarchy. Hyperparameter b would exist in the hyperparameter context given by *b_valid=True* and would not exist in the hyperparameter context given by *b_valid=False*. Virtual hyperparameters are used only to guide the hierarchical hyperparameter sampling and validation, and should be removed from the configuration before passing it to the SML algorithm.

The two observations indicated above allow to encode the entire hyperparameter space for a SML algorithm into a tree structure that models the hierarchical dependences between sets of hyperparameters. The top nodes of the tree refer to categorical hyperparameters that do not depend on the values of others, and each category creates a new hyperparameter context (branch) under which its depending categorical hyperparameters will be represented.

The leaves of the tree correspond to the numerical hyperparameters that are valid for the deepest branching to which they belong. The branching of categorical hyperparameters corresponds to a series of decisions on what values the categorical hyperparameters take, and is refered to as the **signature** of the numerical hyperparameter context. Figure 3.3 provides an example of a hyperparameter hierarchy.

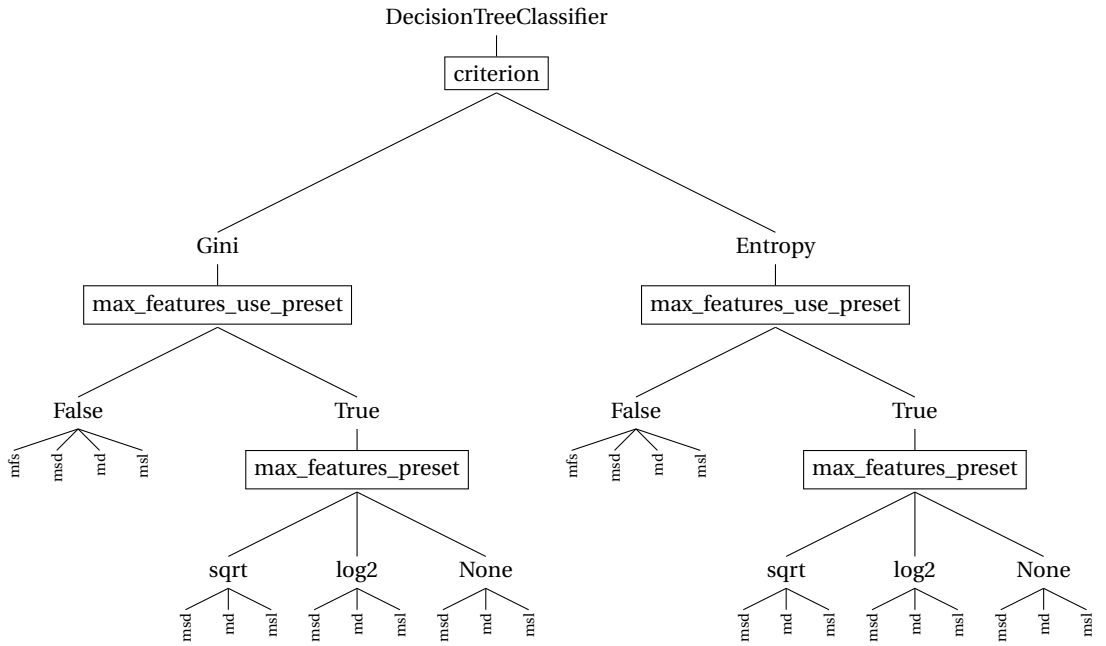


Figure 3.3: Hyperparameter hierarchy for a decision tree classifier. Numerical hyperparameters correspond to the leaves. Categorical hyperparameters (enclosed in a box) create branches for each category that they contain.

The whole model selection process can be viewed as an extension of the hyperparameter hierarchy, where the set of all SML algorithms $\mathcal{A}$ to analyze corresponds to a categorical hyperparameter of the model selection. Each top-level branch contains the hierarchy (subtree) for all the different configurations for a specific type of algorithm (the **algorithm family**).

Note: For simplicity, the solution has been designed under the assumption that numerical hyperparameters are independent of each other. Modeling dependences between numerical hyperparameters is outside the scope of this work, as it would require to model all the hyperparameters belonging to the same signature in the hierarchy as a single multidimensional distribution.

3.1.3 Sampling the hyperparameter space

As stated in subsection 3.1.1, a probability distribution models the parameter space of each hyperparameter, depending on their type and any prior knowledge available. These distributions are used by an optimization method to sample hyperparameter values and evaluate them.

Recall that a SML algorithm accepts a configuration $\boldsymbol{\lambda} = (\lambda_a, \lambda_b, \lambda_c, \ldots)$, i.e. a set of values or realizations of the hyperparameters a, b, c, ... Sampling the hyperparameter space means obtaining such realizations by sampling the distributions describing each hyperparameter.

Since all the information about the categorical dependencies is encoded into the tree structure,

and numerical hyperparameters from the same context are assumed independent from each other, each numerical hyperparameter is sampled independently.

Generating valid samples of the hyperparameter space of a SML algorithm reduces then to getting a valid signature (by recursively sampling the distributions of the categorical hyperparameters in the tree) and sampling each numerical hyperparameter in the context given by the signature independiently. The SML algorithm under the configuration resulting from joining all these values together is the *model* to be evaluated over the data.

3.2 Model performance measurement

Once the strategy for generating models for evaluation has been established, the definition of a model performance metric is required. To measure the performance of a model means to quantify how well the prediction of it on unseen data is, and hence the quality of the labeling obtained on the prediction step of the model is the criterion to be measured.

Because the performance measurements between different algorithms must be comparable, a method that is agnostic to the algorithm should be defined. The labeling returned by the prediction step of the model can be either a single class for each instance in the prediction set, or, for some classification algorithms, a vector describing the probabilities for each instance to be assigned to a specific class.

Predictions consisting of specific classes are transformed into probability vectors with a value of 1 for the position corresponding to the assigned class, and 0 elsewhere. This step provides a consistent representation for the output of all classification algorithms. For regression, the single (possibly multidimensional) predicted value is used without further transformation.

Many metrics comparing the expected and the predicted classes exist, and different metrics allow for evaluation of different properties of the prediction. Table 3.1 summarizes some of the most widely used.

Different metrics may adopt values from different ranges, and the interpretation of the actual values might also differ from metric to metric. As an example, the accuracy of a prediction is measured in the range between 0 and 1, and a *greater* value means a better prediction. The mean squared error is measured as a positive value, and a *lower* value means a better prediction in this case. In order to homogenize the metrics, upper and lower bounds are defined for each metric, and the relative position of the original value with respect to the bounded interval is used instead of the original value.

The upper bound is straightforward to specify, as all metrics have an ideal value that would be achieved if the prediction agrees exactly with the true classes. To define the lower bound, the values for each of the metrics are calculated by using a virtual prediction that assigns the relative frequencies of each class in the dataset as the probabilities of any instance belonging to that class (this is equivalent to weighted-randomly assigning classes to the instances). The

Metric	Formula	Description
Accuracy	$\frac{\text{fp}+\text{tn}}{\text{fp}+\text{fn}+\text{tp}+\text{tn}}$	Ratio of correctly labeled instances
F_β **score**	$\frac{(1+\beta^2)\text{tp}}{1+\beta^2\text{tp}+\beta^2\text{fn}+\text{fp}}$	Weighted average of precision and recall. β is the relative importance of the recall with respect to the precision
Brier score	$\frac{1}{N}\sum_{t=1}^{N}\sum_{i=1}^{R}(p_{ti}-o_{ti})^2$	Squared difference of probabilities returned by the prediction p_{ti} and true probabilities of the labeling o_{ti}.
Matthews correlation coefficient	$\frac{\text{tp}\,\text{tn}-\text{fp}\,\text{fn}}{\sqrt{(\text{tp}+\text{fp})(\text{tp}+\text{fn})(\text{tn}+\text{fp})(\text{tn}+\text{fn})}}$	Balanced measure of true and false positives and negatives, suitable for data having classes with very different frequencies.
Coefficient of determination (R^2)	$1-\frac{\sum_i(f_i-\bar{y})^2}{\sum_i(y_i-\bar{y})^2}$	Measure of fitness of the predictions to the expected classes.
Area under ROC curve	$\int_{-\infty}^{\infty} TPR(y)P_0(y)dy$	Probability to obtain a better score with a randomly chosen correctly-classified instance than with a randomly chosen misclassified one.
Mean absolute error	$\frac{1}{n}\sum_{i=1}^{n}\lvert f_i-y_i\rvert$	Absolute difference between predictions f_i and true values y_i
Mean squared error	$\frac{1}{n}\sum_{i=1}^{n}(f_i-y_i)^2$	Squared difference between predictions f_i and true values y_i

Table 3.1: Components of the Performance Index. tp, tn, fp, and fn correspond to the *true positive*, *true negative*, *false positive*, and *false negative* count, respectively. P_0 is the probability of an instance *not* belonging to a class.

metrics obtained by this prediction are used as a **baseline** that defines the lower bound of the interval for homogenization. It is possible that some homogenized metrics lie below the baseline, when the quality of the classification is worse than the baseline.

The selected metrics are combined into a single **performance index** to be used as the target function that the optimization method will aim to maximize. The performance index S (also

referred to as the ***score***) is defined on a set of metrics $\mathcal{M} = \{m_1, m_2, \ldots, m_z\}$ as:

$$S = \sum_{i=1}^{z} w_i m_i(\mathbf{y}, \hat{\mathbf{y}}), \qquad \sum_{i=1}^{z} w_i = 1 \tag{3.7}$$

The weights w_i control the relative importance of each metric and can be tuned to suit specific needs. For instance, if it is known that none of the SML algorithms to evaluate returns probability estimates of the labeling, the Brier scores will coincide with the reported accuracy, and is therefore safe to disable it by setting its weight to zero. All metrics are weighted uniformly by default.

By default, the performance index used here combines the accuracy, the F_β score, the Brier score, and the Matthews correlation coefficient. These metrics have some properties that make them suitable for a wide range of situations. The accuracy offers a very intuitive evaluation of the quality of the prediction, the F_β score allows for tuning of the relative importance of the recall with respect to the precision, which might be useful depending on the needs of the user, the Brier score takes the probability estimates into account and therefore deals with ambiguity in the classification in a more fair way, the Matthews correlation coefficient is a balanced metric suitable for classes with very different relative frequency.

While encoding metrics into a one-dimensional performance index may hide important details of the individual metrics, and yields values that are not straightforward to interpret, it is much more convenient to use it to represent the overall performance than dealing with a multidimensional score directly. The performance index described is thus the chosen approach for defining the optimization target function.

3.3 Model validation

Using a model validation protocol is important because it reduces the risk of overfitting a model to the particularities of the training set, and hence is a first step to promote generalization of the model to unseen data. The performance index calculated by equation 3.7 maps each model to a single numerical value that represents its ability to correctly predict classes on a very specific set of training and testing instances. In order to assess the predictive quality of the model in a more general setup, repeated rounds of *stratified k-fold cross-validation* are carried out.

Each *repetition* corresponds to a different random shuffling of the data. The k-fold cross-validation splits the data into k equally-sized portions (or folds), and uses one fold at a time for prediction (and the remaining $k-1$ folds for training the model). *Stratified* cross-validation means here that the folds are enforced to all have the same proportion of examples for each class.

In most applications, a single repetition of 10-fold cross-validation is used, the scores ob-

tained for each fold are averaged out and used as a single score. This means that, for each fold, the model is trained with 90% of the data and tested against the remaining 10%. [Kohavi et al., 1995] justifies the choice of $k = 10$ as the smallest number of folds with "acceptable" bias. The actual number of repetitions and folds used in this work is exposed as a parameter that can be customized at will.

For the optimization stage, the standard 10-fold cross-validation is used by default. For the model selection stage, since a more limited number of candidate configurations will be evaluated, and because a statistical analysis is to be performed on the distribution of results, a larger number of evaluations for each model can and should be obtained. Normality of the distribution of scores is assumed. Performing 3 repetitions of 10-fold cross-validation is suggested, to comply with the rule of thumb of using a sample size of at least 30 samples to estimate a normal distribution.

3.4 Optimization

The optimization stage retrieves configurations, evaluates their performance, and uses the performance measurements to guide the retrieval of new configurations towards regions of the hyperparameter space expected to show predictive improvement.

Since the aim of this work is to provide the framework for hyperparameter optimization and model selection, rather than to study different optimization techniques, only a couple of optimization methods have been implemented. More sophisticated optimization methods such as simulated annealing and genetic algorithms could also be used. The framework described here assumes only that the optimization method returns candidate configurations, and can be fed with their evaluation results in order to train itself if needed.

The valid optimization methods work on the numerical dimensions of the hyperparameter space only. This is particularly convenient because most current optimization methods are purely numerical, and the current design does not impose any further restrictions or special handling of the non-numerical dimensions, which removes the need for specially designed optimization methods.

The optimization stage is restricted by a fixed time budget, shared across all the SML algorithms.

3.4.1 Random search

The easiest way to generate candidate configurations is to simply draw values at random for each hyperparameter.

The considerations stated in subsection 3.1.3 must be respected, namely the hyperparameter sampling must start from the root of the hyperparameter hierarchy, obtain a realization

of the top-level hyperparameter (randomly in this case), and use its value to decide what other hyperparameters to sample. The categorical and numerical values retrieved are the components of the configuration.

Random search does not need to keep track of the evaluated model history or the retrieved scores, since every draw of hyperparameter values is independent of all the previous ones. As a consequence, random search is a very slow and expensive optimization method. Random search is, however, very useful when a more thorough exploration of the hyperparameter space is required, as it gives any possible configuration the same chance to be drawn.

A very important application of random search is presented in chapter 4, where it is used as a meta-optimizer for inferring hyperparameter distributions to train a parametric density estimation optimizer.

3.4.2 Shrinking hypercube

Unlike the random search, the shrinking hypercube method described in [Gonnet et al., 2010] does make use of the scores returned by previous optimization rounds to guide the search to a local maximum.

The shrinking hypercube algorithm works by defining a region of the multidimensional parameter space for exploration, delimited by a hypercube centered about the configuration that produces the best result of all the tested configurations. The hypercube will shrink when no improvement is found, to localize the search, and expanded when a new best is found, to stimulate exploration.

The pseudocode of the shrinking hypercube approach is presented in algorithm 1

Algorithm 1 Shrinking hypercube optimization

```
Sample and evaluate a random point from the numerical hyperparameter space.
Define a hypercube of arbitrary length centered about the sampled point.
while the size of the hypercube is larger than a threshold do
    Sample and evaluate another point randomly from within the hypercube.
    if the evaluation of the new point is better than the previous point then
        Duplicate the length of the hypercube on all dimensions.
        Center the hypercube about the new point.
    else
        Reduce the length of the hypercube on all dimensions by a small factor.
        Keep the hypercube centered about the current best point.
    end if
end while
```

Duplicating the length of the hypercube on each dimension favors exploration of the hyperparameter space since new configurations arbitrarily far from the current best can be reached. Shrinking the hypercube slowly when no improvement is achieved helps localize the search

for a local maximum around the current best configuration while not significantly restricting the exploration.

The implementation used in this work considers each numerical hyperparameter space (each leaf in the hyperparameter tree) as a single, independent hypercube, and finds local maxima for each one independently. When the hypercubes are shrunk below a threshold, they are reset to a different starting point and default hypercube side length. Multiple hypercubes can also be used in parallel with different starting points and side lengths for a better coverage of the hyperparameter space.

3.4.3 Parametric density optimization

A simple optimization approach that makes use of clues about generally high and low-scoring regions in the hyperparameter space is presented in detail in chapter 4.

The distributions for all hyperparameters are estimated automatically, by running the optimization process on a large set of standard datasets using random search to explore the hyperparameter space. The results of all the datasets are used to fit gaussian mixtures that encode the high and low-scoring regions for each hyperparameter. The gaussian mixtures returned by this process will replace the default distributions that represent each hyperparameter.

3.5 Model evaluation and selection

The optimization stage explores the hyperparameter space and finds candidate configurations that return a high performance index when evaluated on the *optimization data set*. The selected model will be chosen from one of such candidates according to their predictive performances on a *hold out dataset*, and other desirable criteria.

A very large number of candidate configurations might be proposed by the optimization process, and hence a strategy to efficiently discard or promote configurations must be designed. The tree structure chosen to represent the hyperparameters of a SML algorithm can be exploited to this end.

The chosen strategy analyzes the local optima found by the optimization process on the leaves of the hyperparameter tree (which only contain numerical hyperparameters), selects the ones that are significantly better, evaluates them and ranks them according to other desirable properties to narrow down the model selection even further. When a handful of very good models are selected at the numerical level, they are used to represent their branching in the tree at the deepest categorical level, and compete with representative models of the other categories in the exact same way. The selected models among all categories are promoted upwards to represent the category one level higher in the hyperparameter tree. The process is repeated until the root branching of the tree is reached, i.e., when models from different SML

algorithms are compared. A schematic of the process is shown in figure 3.4.

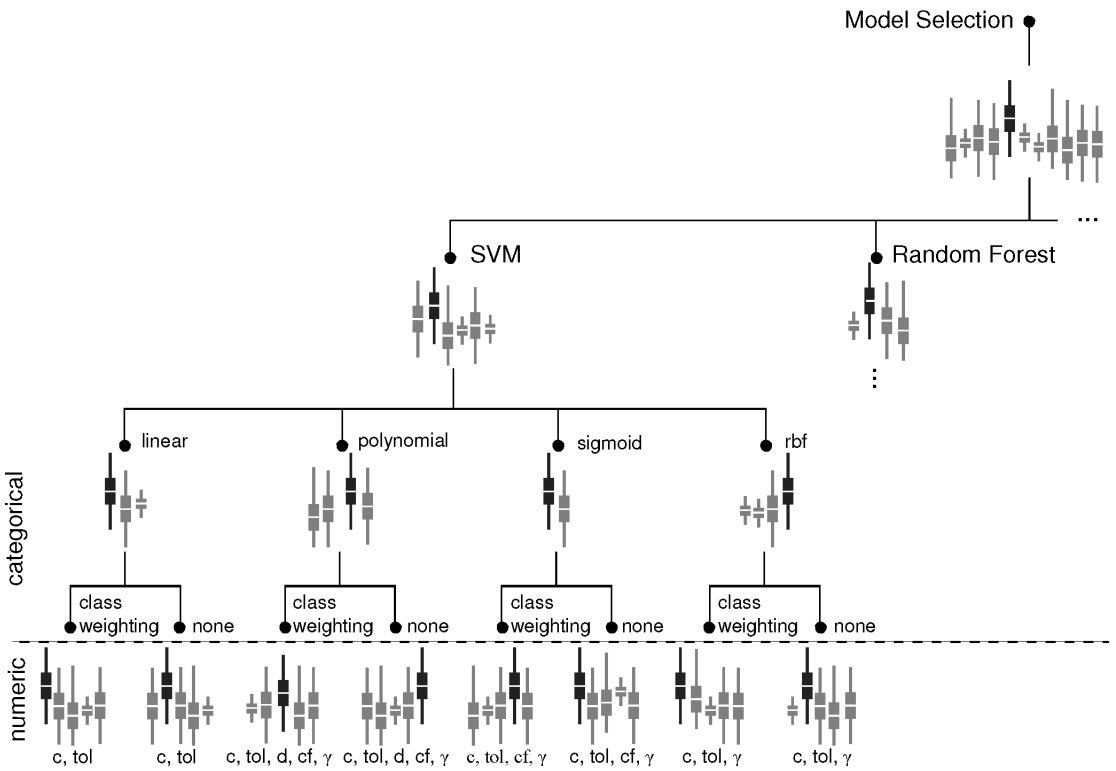


Figure 3.4: Hierarchical model selection. Each box plot represents the distribution of scores for a specific model. Distributions shown in black are promoted upwards to subsequently compete against models from the same local hierarchy.

3.5.1 Candidate model filtering

The optimization stage will mostly retrieve models with high performance indices. Most optimization techniques try to find local optima, and as a consequence, they might end up testing a large number of configurations close to each optimum. This means that the distribution of hyperparameter values sampled and evaluated by the optimization stage will tend to have more density around the different local maxima found. A clustering approach is used here to remove redundant configurations and only retrieve high-scoring configurations from different local maxima. This avoids overrepresentation of configurations from a single hill in the score landscape and returns a small number of configurations that represent more homogeneously the regions that yield good scores.

Clustering the evaluated configurations with the k-**means** algorithm is used here to retrieve k clusters of configurations, and choose the configuration with the highest score from each cluster as one of the candidate configurations that will represent a specific numerical hyperparameter space. The number of means has been arbitrarily chosen to $k = 10$ by default but can be modified via a parameter setting if needed.

3.5.2 Statistical analysis

Given a set of models, the distribution of their scores on different versions of the hold out dataset is studied to determine which model offers enough evidence to be considered the best one.

The hierarchical structure of the hyperparameter space groups models that share the same signature together, at different levels. Statistical analysis is performed at all levels of the hyperparameter space, starting from the configurations that differ only on their numerical hyperparameters, and promoting good models to be compared to others at the immediately broader level of aggregation (the category immediately above in the tree) repeatedly up to the highest level in the hierarchy.

Each model is used for prediction on the hold out dataset several times by using m repetitions of k-fold cross-validation ($m = 3$, $k = 10$ by default). A multiple comparison statistical test (MCT) is applied on the distributions of $m \cdot k$ scores obtained for each model.

Parametric MCTs make use of direct measurements of the data (such as the mean and variance), whereas non-parametric MCTs rely on comparisons of other properties of the data (e.g. the ranking of the groups according to some criteria) which makes them more general and applicable to a wider range of distributions, at the expense of statistical power [Sheskin, 2003]. Parametric comparison tests are more statistically powerful and are thus preferred over non-parametric alternatives. Parametric tests, however, are only valid under certain specific conditions, and when such conditions are not met, non-parametric comparison tests must be used instead.

The MCTs determine, up to a certain probability α, whether there are significant differences between a group of distributions or not, but do not tell the actual distributions that show differences. A *post-hoc* test is required in order to find which distributions differ. Both parametric and non-parametric post-hoc tests exist.

The aim of the statistical analysis in the context of this work is to compare the distributions of performance indices for all candidate models with respect to the highest-scoring one, and discard all the candidate models with strong statistical evidence that they perform worse. The models kept are **statistically indistinguishable** from the best one. The overall procedure is based on [Pizarro et al., 2002] and [Demšar, 2006] for significance testing.

The chosen *parametric* MCT is the **one-way ANOVA test** [Fisher, 1925], which is a generalization of the t-test that corrects for accumulated errors when performing comparisons between more than two distributions, by comparing a statistic against a F-distribution that considers the specific number of distributions to compare as a parameter.

The one-way ANOVA test can be used when:

1. The observations for all the distributions are independent.

2. The distributions are approximately normal. This is tested by applying the **Kolmogorov-Smirnov test** (K-S test, validates that a sample comes from a given distribution) on the sample (after standardization) against a standard normal distribution $\mathcal{N}(0,1)$

3. The distributions have homogeneous variances (homoscedasticity). This is tested by applying the **Bartlett's test** [Bartlett, 1937] on the different distributions of scores.

As stated above, a post-hoc test is required when the ANOVA test determines that there are significant differences between groups of scores. The **Tukey's honest significance test** compares simultaneously all the groups of scores and informs what groups are significantly different. The Tukey's test is then used to find the sets of performance indices that are not significantly different from the the highest-scoring one. Here it is assumed safe to discard models that do not pass the Tukey's test, and keep the rest for further analysis.

When the conditions to apply the parametric MCT are not met, the **Kruskal-Wallis test** is applied instead to decide if significant differences exist, and the **Nemenyi test** is used as the post-hoc test to find the actual significantly different groups in the same way as for the parametric case.

All the statistical tests make use of a critical value to reject hypotheses with a level of certainty α. The choice of α for model selection affects what models are deemed statistically indistinguishable from the best. A larger α will generally reject more models than a smaller one. Since most of the models to compare may be very similar at the family level, it is left up to the user to decide what α values to use, to manually control the number of significantly indistinguishable models to select.

3.5.3 Ranking criteria

The statistical analysis reduces the number of candidate models to those that statistically perform as good as the best, according to their performance indices. Other criteria are evaluated on the selected models in order to compare them and decide which one is the best from a group of models statistically indistinguishable from the best.

The criteria used here are described in table 3.2. Each criterion is evaluated for each candidate model, and used for ranking them. Fractional rankings (to account for ties) are weighted by a user-defined relative importance value and combined into a compound ranking. The top model according to this compound ranking is selected.

The statistical test applied to the model scores compares the mean and spread of the scores obtained by a single model against all other candidate models. The test is summarized in algorithm 2.

It is worth noticing that the proportion of data to be used for optimization and for model selection is defined by the user (a default of 50% of the data for optimization and 50% for

Algorithm 2 Statistical analysis and multiple comparison procedure

Input $\mathcal{A} = \{A^1, \ldots, A^k\}$ a list of algorithms
$\{\mathbf{s}_{A^1}, \ldots, \mathbf{s}_{A^k}\}$ corresponding sets of c.v. test score records
α the desired significance level

Output $\mathcal{A}^r = (A^*_{\boldsymbol{\lambda}^*}, A^{r_2}_{\boldsymbol{\lambda}^*}, \ldots, A^{r_k}_{\boldsymbol{\lambda}^*})$ a ranked list of optimized algorithms from $\mathcal{A}$
$\bar{\mathbf{s}} = (\bar{s}^*, \bar{s}^{r_2}, \ldots, \bar{s}^{r_k})$ sorted c.v. performance scores
$\mathbf{g} = (g^*, g^{r_2}, \ldots, g^{r_k})$ sorted generalization scores
$\mathbf{p} = (\cdot, p^{r_2}, \ldots, p^{r_k})$ significance (p-values) between $A^*_{\boldsymbol{\lambda}^*}$ and others
$\mathbf{t} = (t^*, t^{r_2}, \ldots, t^{r_k})$ sorted model simplicity estimates (time)
$\boldsymbol{\rho} = (\rho^*, \rho^{r_2}, \ldots, \rho^{r_k})$ sorted overfitting risk estimates

function MULTICOMP($\{A^1, \ldots, A^k\}, \{s_{A^1}, \ldots, s_{A^k}\}, \alpha$)
  for every $A^i \in \mathcal{A}$ **do**
    $\mathbf{s}'_{A^i} \leftarrow$ **cluster**$(s_{A^i_1}, s_{A^i_2}, \ldots, s_{A^i_m})$ ▷ Remove very similar configurations.
    $\{\bar{s}_{A^i_1}, \ldots, \bar{s}_{A^i_m}\} \leftarrow$ compute **means** for scores in $\mathbf{s}_{A^i}$
  end for
  $\mathcal{S} \leftarrow \{\mathbf{s}'_{A^i}, \ldots, \mathbf{s}'_{A^k}\}$ ▷ Collect all score sets into $\mathcal{S}$.
  $A^{(*)}_{\boldsymbol{\lambda}} \leftarrow A_{\boldsymbol{\lambda}} \mid \bar{s}_{A^i_{\boldsymbol{\lambda}}} = \max(\bar{s}_{A^1_1}, \ldots, \bar{s}_{A^1_m}, \ldots, \bar{s}_{A^k_1}, \ldots, \bar{s}_{A^k_m})$ ▷ $A_{\boldsymbol{\lambda}}$ with highest mean.
  $D \leftarrow$ **normality**$(\mathcal{S}, \alpha)$
  $T \leftarrow$ **homoscedasticity**$(\mathcal{S}, \alpha)$
  if $Tq \geq \chi^2_{1-\alpha,k-1}$ and $D \geq D^*_{1-\alpha,n}$ use *Analysis of variance* (ANOVA) **then**
    $F_{\text{value}} \leftarrow$ **ANOVA**$(\mathcal{S})$
    if $F_{\text{value}} \geq F_{\alpha,k-1,N-k}$, significant differences exist **then**
      for every $A^i \in \mathcal{A} \setminus A^{(*)}_{\boldsymbol{\lambda}}$ **do**
        $p =$ **Tukey**$(A^i_{\boldsymbol{\lambda}}, A^{(*)}_{\boldsymbol{\lambda}})$
      end for
    end if
  else use *Kruskal-Wallis H-test*
    $K \leftarrow$ **Kruskal-Wallis**$(\mathcal{S})$
    if $K \geq \chi^2_{1-\alpha,k-1}$, significant differences exist **then**
      for every $A^i \in \mathcal{A} \setminus A^{(*)}_{\boldsymbol{\lambda}}$ **do**
        $p =$ **Nemenyi**$(A^i_{\boldsymbol{\lambda}}, A^{(*)}_{\boldsymbol{\lambda}})$ ▷ Probability $\bar{s}_{A^i_{\boldsymbol{\lambda}}}$ and $\bar{s}_{A^{(*)}_{\boldsymbol{\lambda}}}$ from same distribution.
      end for
    end if
  end if
  $(r_*, r_2, \ldots, r_k) \leftarrow$ **rank** combinations of A and $\boldsymbol{\lambda}$ by $\bar{s}, t, \rho$
end function

Criterion	Description
Generalization	Average score of the model on the hold out dataset. Models with greater values are preferred.
Speed	Average measured runtime on the hold out dataset. Models with lower values are preferred.
Stability	Measure of variability of the scores for a model (standard deviation). Models with lower values are preferred.
Simplicity	Measures the complexity of the model. Models with less terms or lower dimensionality are preferred.
Interpretability	Measures how easily the model can be understood by a human. Higher values are preferred.

Table 3.2: Alternative ranking criteria for statistically indistinguishable from the best. The values for Simplicity and Interpretability are subjective: each SML family and category are assigned predetermined values which can be modified by the user.

model selection is suggested). When dealing with small datasets, the choice of this ratio will affect the reliability of both optimization and model selection results. Alternative approaches such as bootstrapping, or overlapping of the optimization set and the model selection set may be helpful to some extent, but should be used cautiously.

Making use of the tree structure for model selection not only provides a very efficient way to discriminate between model performances, but also has the advantage that it compares models with similar characteristics, and selects a small but representative subset of the models to compete against other sets of models.

3.5.4 The model selection algorithm

The process described above can be summarized as shown in algorithm 3

Algorithm 3 Model selection algorithm

```
function SELECT_MODELS(hyperparameter_tree, top_n, α, k)
    candidates ← ∅
    if hyperparameter_tree is numerical then
        candidates ← GET_k-MEANS(hyperparameter_tree, k)
    else
        for category in hyperparameter_tree do
            candidates ← candidates ∪ SELECT_MODELS(category, top_n, α, k)
        end for
    end if
    candidates ← FIND_SCOREWISE_EQUIVALENTS(candidates, α)
    candidates ← GET_TOP_n(candidates, ranking_criteria, top_n)
    return candidates
end function
```

The function obtains the best models at each level of the hyperparameter hierarchy, by recur-

sively filtering and selecting the best models from the lower branches of the hierarchy and passing them to the level immediately above. At the top level, the ranked list of best models will be reported as the result of the model selection stage.

3.6 Implementation details

The actual implementation of all the components described in this chapter was developed as a program that reads datasets from different text formats (only `.arff` files are supported at the moment), and applies a set of different SML algoritms on the read data.

The program is implemented in python 2.7.5, and it makes use of the SML algorithms implemented in the scikit-learn library [Pedregosa et al., 2011]. Most of the numerical handling and statistical tests uses the implementations available in the numpy and scipy libraries [Jones et al., 2001], and the matplotlib library [Hunter, 2007] is used for visualization of the results.

Parallelization of the execution is supported, and has been tested on the Amazon EC2 cloud computing platform, and LSF clusters such as the Brutus Cluster.

CHAPTER 4
Hyperparameter distribution learning

Applying a SML algorithm on different sets of data usually requires that the hyperparameters adopt different values to achieve the best performance. However, there are certain ranges of values that are harmful in all but some pathological cases, and likewise, some regions of the hyperparameter space produce configurations that consistently perform well when applied on different types of data, and hence are worth trying when searching for good models.

This suggest that it might be helpful to guide the hyperparameter optimization by making use of some description of the hyperparameter space that gives hints about what values a given hyperparameter should and should not take. Starting the sampling of hyperparameter values from a hyperparameter-specific **general prior distribution** that has high density around regions of the hyperparameter space known to perform well, and low or no density around regions known to harm the prediction of the SML algorithm, will help the optimization process to discover more quickly where the local optima might be, and what regions of the hyperparameter space it is safe to avoid.

The implemented approach evaluates a large number of configurations on a group of datasets with different properties, and uses the performance indices obtained to learn the general prior distribution for each numerical hyperparameter individually. Using a large number of datasets reduces the risk of overfitting the general prior learned to particularities in the evaluation of a single dataset.

Parametric vs non-parametric modeling of the general prior distribution

Due to the large number of configurations and dimensions to test, a compact representation of the general prior distributions is preferred. Estimation of parametric distributions is convenient because it allows for this compactness and because usually closed-form solutions for maximum likelihood fitting of their parameters exist.

Non-parametric techniques to estimate the general prior distribution, like Kernel Density Estimation [Rosenblatt et al., 1956], [Parzen et al., 1962] keep too much information in memory, are cumbersome to encode and store for later use, and depend on heuristics and rules

of thumb to decide parameter values such as the kernel bandwidth, and are therefore not suitable for the automatic approach presented in this work.

4.1 Learning the general prior distribution

The chosen representation for the general prior distribution is a Gaussian Mixture Model, which is convenient for encoding multimodal distributions and to which it is not very expensive to fit data. Since the true probability of a hyperparameter value being the optimal is not known, the performance index of models containing such value is used as a surrogate for the estimation of this probability, as it measures the quality of the prediction. The implementation of the learning process is a Montecarlo method that accepts or rejects values of the hyperparameters according to their performance indices, summarized in algorithm 4.

Algorithm 4 General prior distribution learning

```
function LEARN_GENERAL_PRIOR(training_datasets, threshold)
    general_prior ← initial GMM
    for dataset D in training_datasets do
        while convergence criterion not met do
            Sample a value λ for the hyperparameter from a uniform distribution
            score ← performance_index(λ, D)
            if random(0, 1) ≤ score then
                COMBINE(general_prior, λ)
                SIMPLIFY(general_prior, threshold)
            end if
        end while
    end for
    return general_prior
end function
```

The algorithm evaluates at each iteration if a more complex GMMs (more Gaussian components) fits the data significantly better than the current GMM. In order to find a trade-off between the GMM complexity and predictive power, the proposed idea is to measure the *fidelity* of the mixture model (i.e. a quantitative estimate of its ability to describe the associated data, as described in [Declercq and Piater, 2007, Declercq and Piater, 2008] for the *uncertain Gaussian model*), and to define a fidelity threshold below which a mixture model is considered overly simplistic. Mixture models will be updated upon arrival of new observations and simplified if the above condition holds.

Learned model update

Everytime a hyperparameter value is sampled and evaluated, its evaluation is added as a new component to the mixture model, by trivially combining the existing distributions with the new one.

If at any given point in time t the mixture model is expressed as a set of N weighted Gaussian distributions:

$$p^{(t)}(x) = \frac{\sum_{i=1}^{N} \pi_i^{(t)} g(x; \mu_i^{(t)}, \sigma_i^{(t)})}{\sum_{i=1}^{N} \pi_i^{(t)}} \tag{4.1}$$

Then the new mixture model for $t+1$ is given by:

$$p^{(t+1)}(x) = \frac{\left(\sum_{i=1}^{N} \pi_i^{(t)} g(x; \mu_i^{(t)}, \sigma_i^{(t)})\right) + \pi^{(t+1)} g(x; \mu^{(t+1)}, \sigma^{(t+1)})}{\left(\sum_{i=1}^{N} \pi_i^{(t)}\right) + \pi^{(t+1)}} \tag{4.2}$$

Learned model simplification

Adding new components to the GMM everytime a new observation arrives hinders all the advantages of using parametric estimation. If possible, merging similar components of the GMM into one should be done, in order to simplify the mixture model.

As proposed in [Declercq and Piater, 2008], two Gaussian distributions (two components of the GMM in this case) can be merged without significant loss of predictive power if the merged distribution has a *fidelity* λ close to 1, otherwise, both components must be kept to properly describe the distribution of the data.

The distance between the distribution to test, and the data to which it should fit, is given by:

$$D = \frac{1}{|I|} \int_I \left| \hat{F}(x) - F_n(x) \right| dx \tag{4.3}$$

Declercq and Piater assume such distances as Gaussian distributed, and define the *fidelity* λ as:

$$\lambda = e^{\frac{-D^2}{T_D^2}} \tag{4.4}$$

With T_D^2 a parameter that controls how much deviation from the observations is allowed and hence how relaxed the computation is about considering the merged Gaussian as an acceptable representation of the two merged components.

Two Gaussian components G_i and G_j can be merged into one by applying the following procedure:

$$\pi = \pi_i + \pi_j \tag{4.5}$$

$$\mu = \frac{1}{\pi}\left(\pi_i \mu_i + \pi_j \mu_j\right) \tag{4.6}$$

$$\sigma = \frac{\pi_i}{\pi}\left(\sigma_i + (\mu_i - \mu)^2\right) + \frac{\pi_j}{\pi}\left(\sigma_j + (\mu_j - \mu)^2\right) \tag{4.7}$$

$$G(x) = \mathcal{N}(\mu, \sigma) \tag{4.8}$$

4.2 Using the general prior for hyperparameter optimization

The distribution returned by algorithm 4 is a rough description of the quality of the hyperparameter in different regions of the hyperparameter space. It can be used as a prior distribution for sampling candidate hyperparameter values to evaluate on a specific dataset.

Since the general prior contains hints of where, in general, high and low-scoring regions of the hyperparameter space are, but is not guaranteed to fit the true distribution of scores for the specific data set, further exploration of other regions of the hyperparameter space not favored by the general prior might be necessary.

The ideal case would be that the general prior already fits the true underlying distribution of scores for the specific dataset, in which case it would suffice to sample values from the general prior, and assume that high-scoring values will be retrieved often. As this is seldom the case, the implementation continuously assesses the goodness of fit of the evaluations to the general prior, and starts to draw samples from an alternative uniform distribution when the fitting to the general prior drops below a threshold.

CHAPTER 5
Evaluation of results

The hyperparameter optimization and model selection process was run on a number of datasets that include 29 standard datasets commonly used in the machine learning community, and 8 datasets taken from biological experiments. The datasets were also analyzed under the default settings for all SML algorithms.

The SML algorithms applied on the datasets were: PassiveAggressive, RadiusNeighbors, GaussianNB, ExtraTreeEnsemble, SVM, LinearDiscriminant, KNN, RandomForest, StochasicGradientDescent, LogisticRegression, NearestCentroid, LinearSVM, NuSVM, DecisionTree, Ridge, QuadraticDiscriminant, and GradientBoosting.

The optimization stage for each of the datasets was run during 8 hours as distributed tasks on the Brutus Cluster.

A comparison between the performance indices obtained by using default and optimized hyperparameters is show in table 5.1.

In general, the performance indices of the optimized models are significantly higher than the performance indices under the default configurations. A few datasets did not show improvement over the default settings. In these cases, the best classification performance was obtained by using the LinearDiscriminant classifier, which does not expose any hyperparameters to configure.

The table shows that the SML algorithm that performs best and improvement over the default settings stictly depends on the dataset.

5.1 Results on standard Machine Learning datasets

Standard datasets are commonly used for testing and evaluating machine learning approaches. Some of the datasets used here are inherently difficult to classify, and hence most SML algorithms consistently obtain a very low performance index; other datasets are simple to classify and hence good performance indices but little improvement over the default values is

	default		Optimized		
Dataset	Worst	Best	Score	Boost	Family
balance-scale	0.0961	0.7452	0.8385	12.52%	LinearSVM
blood-transfusion	0.0553	0.2760	0.3307	19.80%	NearestCentroid
cmc	-0.0607	0.1995	0.2458	23.23%	KNN
dermatology	0.3414	0.9292	0.9378	0.93%	LinearSVM
diabetes	-0.0582	0.4449	0.4454	0.11%	Ridge
haberman	-0.0599	0.1523	0.2746	80.37%	LinearSVM
heart-statlog	-0.0300	0.6894	0.7036	2.05%	Ridge
ionosphere	0.5490	0.7839	0.8373	6.82%	GradientBoosting
iris	0.4250	0.9696	0.9841	1.50%	KNN
kdd_synthetic_control	0.0210	0.9987	1.0000	0.13%	KNN
letter	0.0920	0.8851	0.8851	0.00%	KNN
liver-disorders	-0.1600	0.3198	0.3333	4.25%	Ridge
mfeat-factors	-0.0002	0.9700	0.9700	0.00%	LinearDiscriminant
mfeat-fourier	0.0700	0.8143	0.8154	0.14%	KNN
mfeat-karhunen	0.4696	0.9431	0.9544	1.20%	KNN
mfeat-morphological	-0.0230	0.6719	0.6719	0.00%	LinearDiscriminant
mfeat-pixel	0.6710	0.9522	0.9570	0.51%	KNN
mfeat-zernike	-0.0000	0.7994	0.7994	0.00%	LinearDiscriminant
optdigits	0.1663	0.9743	0.9803	0.61%	QuadraticDiscriminant
page-blocks	-1.8119	0.7864	0.7992	1.64%	GradientBoosting
pendigits	0.0063	0.9825	0.9877	0.53%	KNN
segment-test	0.0180	0.9073	0.9152	0.87%	GradientBoosting
sonar	-0.0775	0.5563	0.5890	5.88%	LinearSVM
spambase	0.0444	0.7786	0.8613	10.63%	GradientBoosting
tae	-0.0165	0.3070	0.3385	10.28%	LinearSVM
vehicle	-0.0484	0.6952	0.6978	0.37%	QuadraticDiscriminant
vowel	0.1243	0.6708	0.8250	22.98%	KNN
waveform-5000	0.4915	0.7971	0.8025	0.68%	Ridge
wine	-0.1486	0.9544	0.9544	0.01%	Ridge

Table 5.1: Comparison of optimized and default performance indices for all datasets.

expected.

Figure 5.1 shows an example of the distribution of performance indices for different models, and their shift with respect to the performance indices obtained by the default models, for a specific dataset. Each row in the plot corresponds to a different algorithm under the default (thin boxplots) and optimized settings. Distributions that are not significantly different from the best, up to a level α shown on the title of the plot, are highlighted. The p-values reported by the statistical test are displayed on the right side of the plot. Models with p-values above the α level are considered statistically indistinguishable.

The baseline is shown as a vertical dotted line. It is possible for some models to score lower than the baseline, as explained in section 3.2.

In cases where algorithms do not work at all on a given dataset, their performance indices are reported as a single point over the baseline.

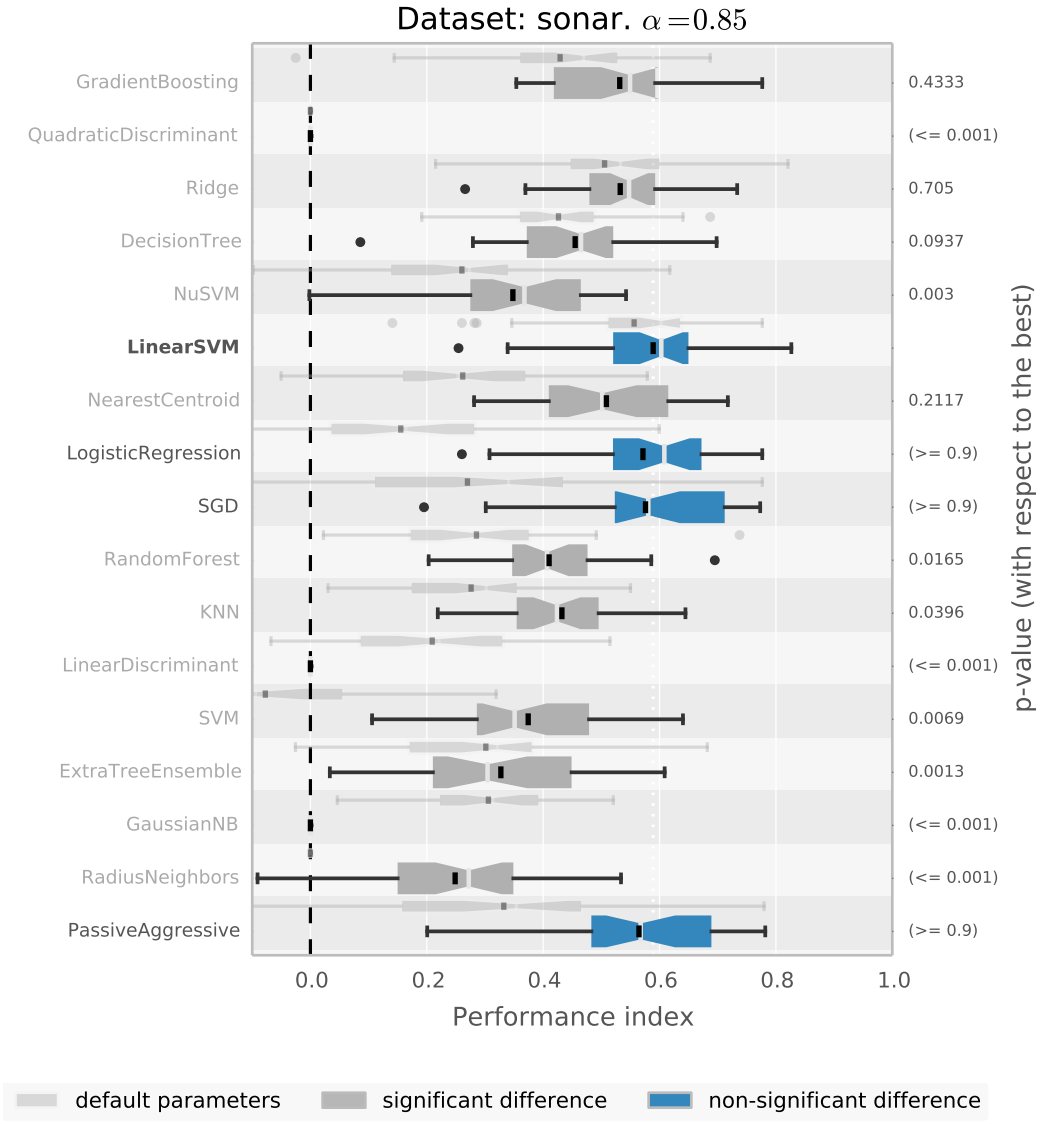


Figure 5.1: Optimized vs default values for a standard machine learning dataset

5.2 Results on biological data

5.3 General prior learning

The 29 standard machine learning datasets were used to learn general priors for all numerical hyperparameters on all SML algorithms, as explained in 4.1.

Gammas from poly approx uniform, gammas from sigmoid: skewed

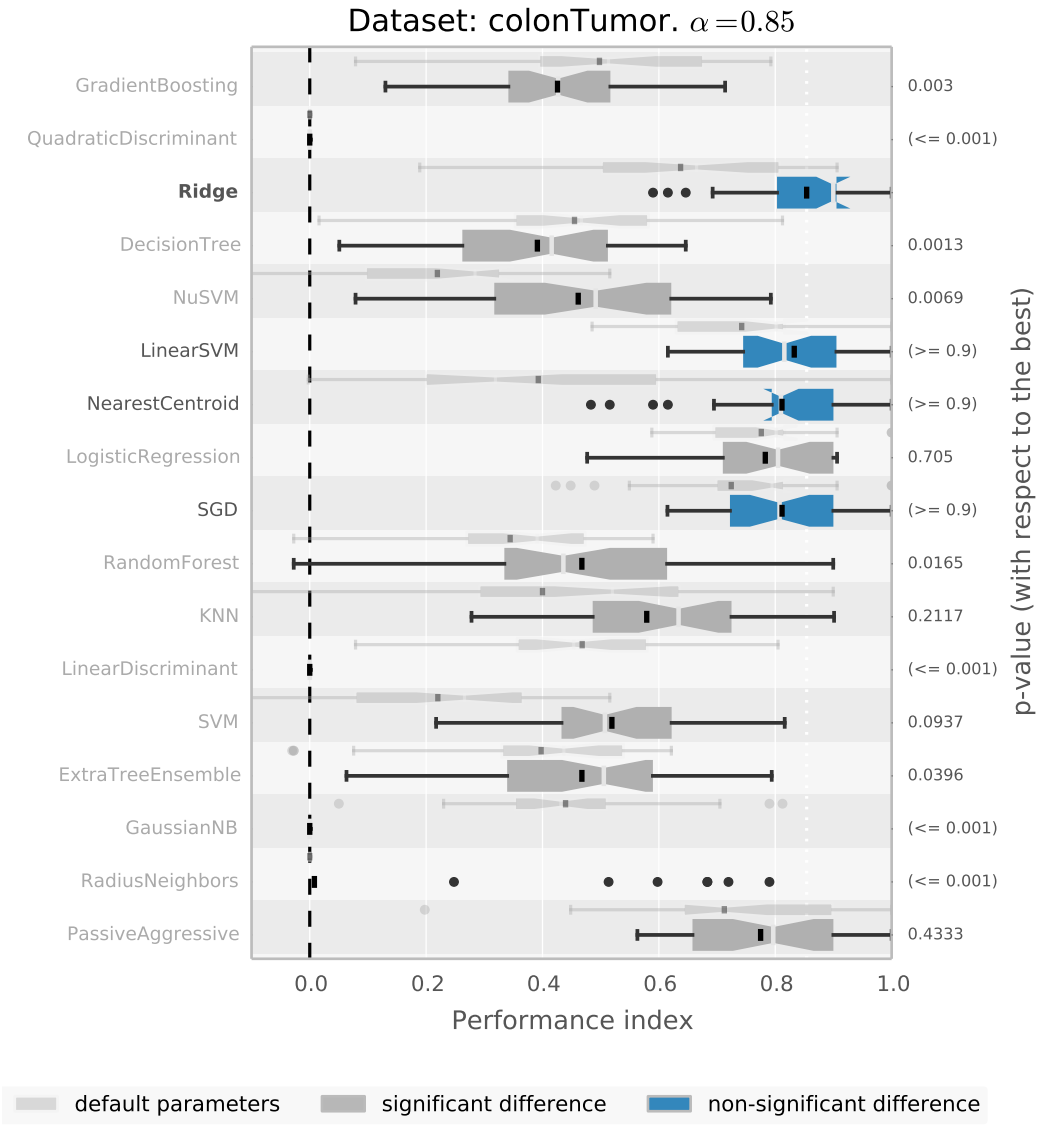


Figure 5.2: Optimized vs default values for a biological dataset

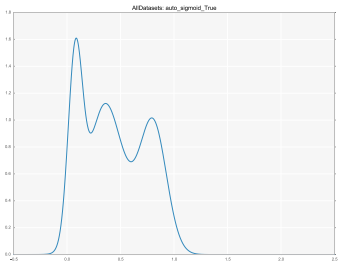


Figure 5.3: gamma for signature auto_sigmoid_True

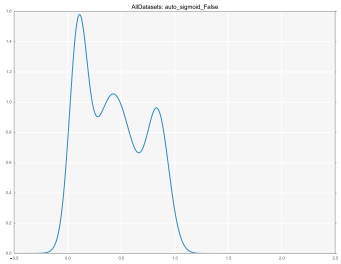


Figure 5.4: learner distribution 2

Figure 5.5: gamma for signature auto_sigmoid_False

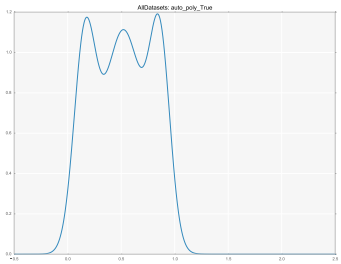


Figure 5.6: gamma for signature auto_poly_True

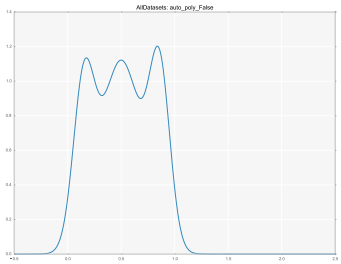


Figure 5.7: gamma for signature auto_poly_False

CHAPTER 6
Conclusion

This work shows that an automatic approach to hyperparameter optimization and model selection is very valuable, especially when little knowledge of the internal function and configuration options of the SML algorithms is available. The optimized configurations retrieved by the implementation of this method significantly boosted the predictive power of most SML algorithms over their defaults, while considering other preferable criteria such as the runtime, stability of the scores, and predictive generalization to unseen instances.

Furthermore, the mostly automated procedure described here is very convenient for non-expert users, as using this implementation requires training in machine learning that does not go beyond the general concepts. The current implementation is already a valuable tool that brings SML algorithms closer to the untrained user, and narrows an existing gap between the purely-technical machine learning community and the vast range of other disciplines that might benefit from SML techniques.

An important consideration regarding the optimization process is that, since the optimization happens simultaneously on a large group of numerical hyperparameters, finding good combinations of hyperparameters may be a slow process. The choice of a optimization method that makes use of the optimization history and other hints to guide the sampling of candidate configurations is critical when the time budget is a concern.

6.1 Future work

Many possibilities to extend this work are considered, and planned. The most important ones are described next.

Optimization methods

Implementing better optimization methods to is the obvious next step. Optimization methods such as simulated annealing or genetic algorithms may speed up the hyperparameter optimization or enforce a guided exploration of the hyperparameter space. Other methods may

be implemented in the future.

Data preprocessing

In most real-world applications, the raw data obtained from experiments is not immediately suitable for training SML algorithms. Data with high dimensionality (many features) usually requires heavier computation to process, and the large number of dimensions can make a very informative signal (i.e., a dimension or group of dimensions that correlate well with the labeling) difficult to detect among all the rest of non-informative dimensions.

Feature selection and dimensionality reduction techniques should be included as a preprocessing step. The decision on which dimensionality reduction to use can also be regarded as a categorical hyperparameter of the model selection process, and included as part of the final suggestion of a best model.

Dataset classification

Having a framework that automatically finds the best model for specific data might be used on a larger scale to infer patterns between dataset-wide features and the SML algorithms that best predict on them. Approaches such as the one presented in [Tatti, 2007], where a distance metric between two datasets is suggested, can be further investigated. Finding dataset-wide features that follow patterns according to the SML algorithms best suited for them might also be used as a first clue of what SML algorithms could work well on a specific dataset.

Custom penalization of misclassifications

Some misclassifications may be more tolerable than others, for data of a certain nature. Penalization of misclassifications can be modified by using a custom confusion matrix that weights misclassifications accordingly. All the metrics that evaluate the quality of the prediction in terms of the true or false positives and negatives should use those weights in their internal computations.

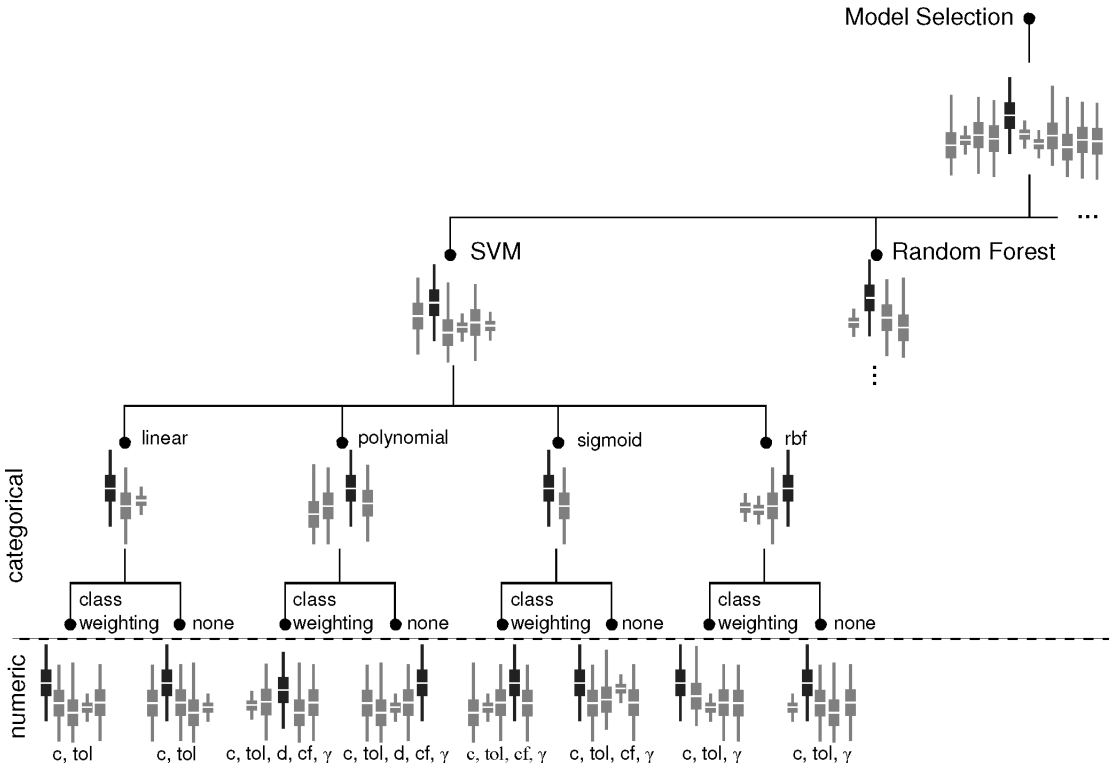
Support for various dataset formats

At the moment only `.arff` files are supported.

Usability improvements

While a graphical user interface, as well as a command line interface, is provided for interaction with the optimization, model selection, and the final suggestion of a model, further improvements on such interfaces must be made.

APPENDIX A
Hyperparameter hierarchy example



Bibliography

[Bartlett, 1937] Bartlett, M. S. (1937). Properties of sufficiency and statistical tests. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 160(901):268–282.

[Bergstra et al., 2011] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B., et al. (2011). Algorithms for hyper-parameter optimization. In *NIPS*, volume 24, pages 2546–2554.

[Bergstra and Bengio, 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305.

[Declercq and Piater, 2007] Declercq, A. and Piater, J. H. (2007). On-line simultaneous learning and tracking of visual feature graphs. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–6. IEEE.

[Declercq and Piater, 2008] Declercq, A. and Piater, J. H. (2008). Online learning of gaussian mixture models-a two-level approach. In *VISAPP (1)*, pages 605–611.

[Demšar, 2006] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30.

[Fisher, 1925] Fisher, R. A. (1925). *Statistical methods for research workers*. Genesis Publishing Pvt Ltd.

[Gonnet et al., 2010] Gonnet, G. H., Scholl, R., and Stevenson, D. E. (2010). *Scientific Computation*, volume 63.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

[Jones et al., 2001] Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy: Open source scientific tools for Python.

[Kohavi et al., 1995] Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14, pages 1137–1145.

[Parzen et al., 1962] Parzen, E. et al. (1962). On estimation of a probability density function and mode. *Annals of mathematical statistics*, 33(3):1065–1076.

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

[Pizarro et al., 2002] Pizarro, J., Guerrero, E., and Galindo, P. L. (2002). Multiple comparison procedures applied to model selection. *Neurocomputing*, 48(1):155–173.

[Rosenblatt et al., 1956] Rosenblatt, M. et al. (1956). Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837.

[Sheskin, 2003] Sheskin, D. J. (2003). *Handbook of parametric and nonparametric statistical procedures*. crc Press.

[Tatti, 2007] Tatti, N. (2007). Distances between data sets based on summary statistics. *The Journal of Machine Learning Research*, 8:131–154.

[Thornton et al., 2013] Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

This sheet must be signed and enclosed with every piece of written work submitted at ETH.

I hereby declare that the written work I have submitted entitled

Automatic problem-specific hyperparameter optimization and model selection for supervised machine learning

is original work which I alone have authored and which is written in my own words.*

Author(s)

Last name	First name
Bermúdez-Chacón	Róger

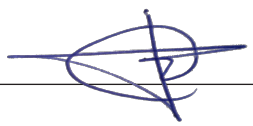
Supervising lecturer

Last name	First name
Gonnet	Gaston

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Zurich, 15.04.2014
Place and date


Signature

*Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Print form