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

**DRAFT**

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

This document is a DRAFT. Dedication, acknowledgements, abstract, table of contents, list of figures, and list of tables will be displayed in the final version.

# CHAPTER 1
# Introduction

describe why automatic data analysis is needed, who uses it, ...

Define SML

Supervised Machine Learning approaches are ubiquitous in a vast range of disciplines nowadays.

why different ML approaches exist

introduce the term object

experimental design

## 1.1 Model selection

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

## 1.2 Hyperparameter optimization

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetuer mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate conval-

lis est. Integer aliquet. Pellentesque aliquet sodales urna.

## 1.3 Supervised machine learning approaches

Nominal values are used for classification problems, in which each instance belongs to a class from a given set. Regression problems relate an instance to a vector of numerical values. For simplicity, $y$ represents here such values, regardless of their cardinality (scalars or vectors) or type (nominal class or numerical value).

# 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.
- $\boldsymbol{\lambda}^i = (\lambda^i_a, \lambda^i_b, \lambda^i_c, \ldots)$ denotes a set of hyperparameter values (or **configuration**) for the $i$-th algorithm in $\mathcal{A}$.
- $\Lambda^i$ 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

A large number of model selection methods exist, most of which rely on statistical hypothesis testing, where usually the *model* refers to a probability distribution that describes a set of data.

Autoweka

Surprisingly, despite its importance, few initiatives to address the problem in the context of machine learning have emerged.

Brief literature review. Bergstra's work [Bergstra et al., 2011] lacks the statistical analysis, and package is cumbersome to use for a non-expert user

# CHAPTER 3
# Methodology

In order to find solutions for equation 2.3, a series of steps and decisions must be made. Such decisions 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:

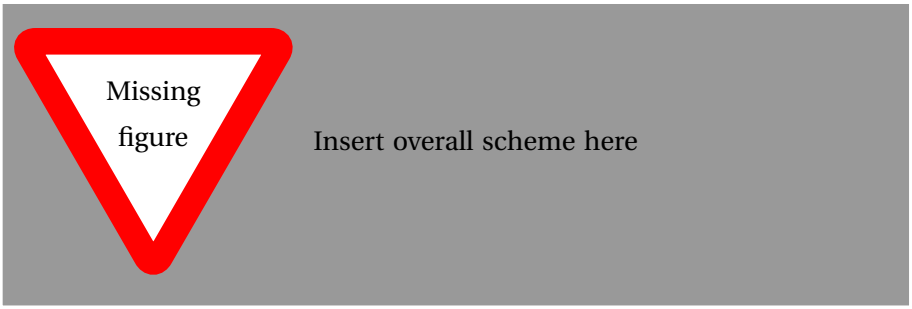



Figure 3.1: Interaction between different components of the implemented hyperparameter optimization and model selection approach.

## 3.1 Data preprocessing

The training step for any SML algorithm performs analysis on a set of example instances of the form $(\mathbf{x}, y)$ to detect patterns or rules that enable it to predict the class of unseen instances. This necessarily implies that a good prediction would strongly depend on the numerical values in the feature vector $\mathbf{x}$ and their relation with their class $y$.

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. Non-

homogeneous scaling of the features may cause for certain algorithms to erroneously assume some features more relevant than others. Some features may only significantly contribute to a good signal for discrimination when combined with others.

To address this problem, many *feature selection* and *dimensionality reduction* techniques exist...

Name feature selection techniques? (None has been included so far.)

## 3.2 Hyperparameter space modeling

Most SML algorithms ship with default hyperparameter values 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. The choice of hyperparameter values can heavily impact the predictive power of a SML algorithm.

Different aspects of the hyperparameter modeling must be taken into account. Individual hyperparameters can be either nominal or ordinal, ordinal hyperparameters can be discrete or continuous, or can be valid only within a specific range. Furthermore, some hyperparameters are only used in combination with specific values of others and thus exhibit conditional dependencies.

### 3.2.1 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 getting 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.

A prior one-dimensional distribution is used to sample values for each hyperparameter independently. Section 3.2.2 describes in detail how dependencies between hyperparameters are handled. The general approach is to create separate distributions for hyperparameters that depend on other categorical hyperparameters, according to the particular values that the category might take.

The choice of the prior 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 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 $\mu$ and mean variation $\sigma$ 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** (GMM) are used for continuous hyperparameters that are known to be multimodal.

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

  GMM have many parameters and thus should be used when there is a very solid knowledge of the shape of the hyperparameter distribution. GMM priors 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. Initially, handcrafted values drawn from the documentation of each SML algorithm are suggested, but can be later modified. For instance, a hyperparameter that, according to the documentation, represents a probability, will use a handcrafted Uniform(0,1) prior by default that makes no further assumptions, but the bounds can be changed manually by the user if they determine that values close to one or both bounds do not produce good results.

### 3.2.2 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, a general Support Vector Machine implementation can use linear, polynomial, sigmoid, or RBF kernels, and if a polynomial kernel is chosen, it is possible to specify the polynomial degree; such option has no effect when a linear kernel is used. Under the same example, the coefficient controlling the independent term in a polynomial kernel may have a different impact and different extrema than the same hyperparameter for a sigmoid kernel.

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**). The present design is based on two observations:

1. Hyperparameter contexts can be nested. When some hyperparameters adopt certain values, the meaning and use of some others change, and among those that are affected, the same principle might apply, up to arbitrarily many levels.
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.

   In the rare cases that numerical hyperparameters might decide the fate of others, artificial boolean categorical hyperparameters can be added trivially, used for the structured sampling, and discarded before running the actual algorithm. These artificial hyperparameter would hold the truth value of rules applied on the numerical hyperparameter defining the 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, an artificial hyperparameter *b_valid* with such rule (which would be a categorical hyperparameter with categories {*True*, *False*}) can be added to the hyperparameter hierarchy, and 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*. The only consideration to take into account is not to pass the artificial hyperparameters 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. An example is provided in appendix A

### 3.2.3 Sampling the hyperparameter space

As stated in subsection 3.2.1, a probability distribution is assigned to 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 $a$, $b$, $c$, .... Getting a sample of the hyperparameter space means getting such realizations by sampling the distributions describing each hyperparameter.

Since all the information about the dependencies is encoded into the tree structure, numerical hyperparameters from the same context are conditionally independent from each other given their signature. This fact makes it valid to sample each distribution describing the numerical hyperparameters under the same signature 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.

## 3.3 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 the ones used in this work.

| Metric | Formula | Description |
|---|---|---|
| **Accuracy** | $\frac{\mathrm{fp}+\mathrm{tn}}{\mathrm{fp}+\mathrm{fn}+\mathrm{tp}+\mathrm{tn}}$ | Ratio of correctly labeled instances |
| $F_\beta$ **score** | $\frac{(1+\beta^2)\mathrm{tp}}{1+\beta^2\mathrm{tp}+\beta^2\mathrm{fn}+\mathrm{fp}}$ | Weighted average of precision and recall. $\beta$ 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** | $\frac{\mathrm{tp}\,\mathrm{tn}-\mathrm{fp}\,\mathrm{fn}}{\sqrt{(\mathrm{tp}+\mathrm{fp})(\mathrm{tp}+\mathrm{fn})(\mathrm{tn}+\mathrm{fn})(\mathrm{tn}+\mathrm{fn})}}$ | Balanced measure of true and false positives and negatives, suitable for data having classes with very different frequencies. |

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

Also include 'roc_auc', 'pr_auc', 'mean_abs_err', 'mean_sq_err', 'r2 mention custom confusion matrix

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.

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, because it is easier to optimize and yields good results in practice, and is thus the chosen approach for defining the optimization target function.

## 3.4 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 considers the generalization of the model to some extent. The performance index calculated by equation 3.7 maps each model to a single numerical value that represents its quality 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 obtained 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 model selection, since a more limited number of candidate configurations will be evaluated, and because the candidates are already expected to perform well, 15 repetitions of 2-fold cross-validation is suggested.

## 3.5 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 very simple optimization methods have been implemented. More sophisticated optimization methods such as simulated annealing and genetic algorithms are planned as part of a

follow-up project.

The optimization methods implemented work on the numerical dimensions of the hyperparameter space only.

### 3.5.1 Random search

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

The considerations stated in subsection 3.2.3 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.

### 3.5.2 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 pseudocode of the shrinking hypercube approach is presented in algorithm 1

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.

**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
```

### 3.5.3 Parametric density optimizer

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.

## 3.6 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 *held 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 3.2.

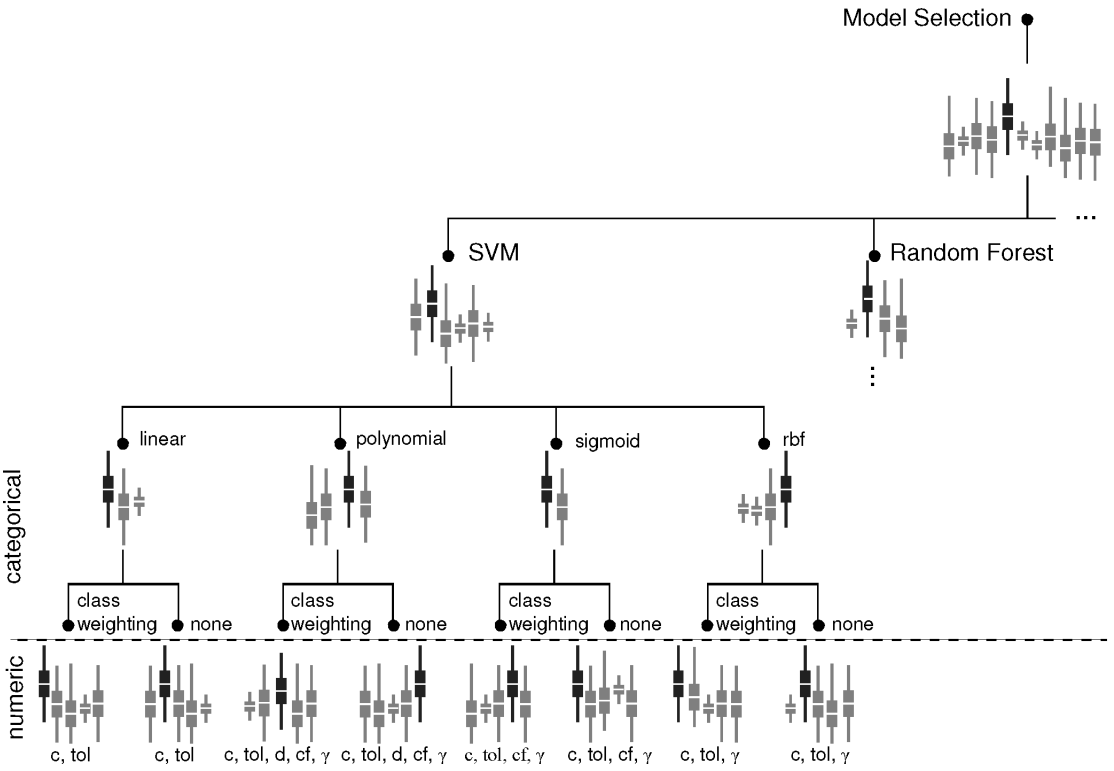



Figure 3.2: Hierarchical model selection. Each box plot represents the distribution of scores for a specific model. Distributions shown in black are promoted upwards to subsequently compete against models from the same local hierarchy.

### 3.6.1 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 exploit this fact and 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.

### 3.6.2 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.

Each model is used for prediction on the hold out dataset several times by using $m$ repetitions of $k$-fold cross-validation ($m = 15$, $k = 2$ 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 $\alpha$, 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 **score-wise equivalent** to 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.

### 3.6.3 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 score-wise equivalent models.

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 $n$ models according to this compound ranking are selected.

| Criterion | Description |
|---|---|
| Generalization | Average score of the model on the hold out dataset. Models with greater values are preferred. |
| Speed | Average measured runtime of applying the model 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 | Predefined value associated to the SML algorithm. Models with lower values are preferred. |
| Interpretability | Predefined value associated to the SML algorithm. Models with lower values are preferred. |

Table 3.2: Alternative ranking criteria for score-wise equivalent models

for each numerical hyperparameter space: find representative local maxima from optimization process use m repetitions of k fold cross-validation apply statistical test of means discard models significantly different to the best evaluate other qualities of the selected models rank the models by such criteria and keep the top n compare all selected models to the models from the different categories on the same subtree

find name for this

find a name for this

==========

getselectedmodels (node, n) if node is numerical hyperparameter space: candidate models = select representatives by kmeans else (is category) candidate models = [] for each category at this level: candidatemodels.append(getselectedmodels(category)) selected models = statistically test(candidatemodels) selected models = top n rank trimmed return trimmed

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 **??**

param: list of score groups, alpha if score groups are normal with similar variance: use anova to decide if there are significant differences if so, apply tuckey test to see

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 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 and the model selection set may be helpful to some extent, but should be used cautiously

find a way to say this properly

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.

### 3.6.4 The model selection algorithm

The process described above can be summarized as shown in algorithm 2

**Algorithm 2** 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 recursively 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.

# CHAPTER 4
# Hyperparameter distribution learning

[Briefly justify]

### 4.0.5 Parametric vs non-parametric density estimation

Due to the large number of configurations and dimensions to test, a compact representation of the hyperparameter space 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. A compact representation of the hyperparameter distributions is especially desirable if later use of such learned distributions (e.g. using the learned distribution as a prior for a new optimization) is intended.

Non-parametric alternatives 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.

### 4.0.6 Parametric distribution learning

The main problem when using parametric estimation comes from the choice of the distribution to model the target random variable. One simple and generally good enough strategy is to use a Gaussian Mixture Model (GMM). The goodness of fit of a sample to a GMM depends in turn on the number of components chosen, since a large number of components will overfit the data, and likewise an overly simplistic model with few components will disregard local properties of the underlying distribution (underfitting).

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.

### Exploration enforcement

At early stages of the learning process, it is very likely that the learned hyperparameter distribution does not fit the true distribution. Because an initial prior distribution is specified for all hyperparameters (as described in section 3.2), which assumes little about the true distribution, it makes sense to start off by retrieving and evaluating hyperparameter values drawn from the prior distribution.

Furthermore, even in later stages of the learning process, new local properties of the true distribution may be discovered, that would render the current mixture model invalid or less fit to describe the observed data. This would suggest that more exploration of the hyperparameter space (more sampling from the prior) is needed.

Since there are two distributions describing the hyperparameter distribution, the learned distribution is not trustworthy for prediction at early stages, and exploration of a wide range of values for the hyperparameter is convenient at some points, sampling from both distributions is proposed. The main idea is to sample the prior distribution more frequently when the learned mixture model does not fit the observations well, and progressively "let go" of the prior as the predictions from the learned mixture model do fit the new observations.

The $p$-value obtained from the *one-sample Kolmogorov-Smirnov test* (K-S test) of new observations against the learned distribution is used here to measure the goodness of fit, and deviations of the new observations from the learned distribution beyond an $\alpha$ value are penalized. This means that when disagreements between prediction and observations happen, the frequency with which the prior distribution is used for sampling new hyperparameter values increases, and likewise when the predictions and observations agree (within $\alpha$ tolerance), more weight (relative sampling frequency) is given to the learned distribution, proportional to the $p$-value.

### 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* $\lambda$ 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* $\lambda$ 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:

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

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

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

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

### 4.0.7 The distribution learning algorithm

The hyperparameter distribution learning process is summarized as follows:

**Algorithm 3** Simultaneous hyperparameter learning and sampling

**procedure** HYPERPARAMETERESTIMATION(prior, $\alpha$, penalty)
  $w_{prior} \leftarrow 1$
  $w_{learned} \leftarrow 0$
  **while** stopping condition not met **do** ▷ Timeout reached, manual user abort, ...
    $r \leftarrow$ random(0, 1)
    **if** $r \leq w_{prior}$ **then**
      distribution $\leftarrow$ prior
    **else**
      distribution $\leftarrow$ learned
    **end if**
    value $\leftarrow$ sample(distribution)
    **if** K-S(value, learned) $> \alpha$ **then** ▷ Assess goodness-of-fit
      $w_{learned} \leftarrow$ decrease($w_{learned}$)
      $w_{prior} \leftarrow (1 - w_{learned})$ ▷ Increase prior sample frequency
    **else**
      $w_{prior} \leftarrow$ decrease($w_{prior}$)
      $w_{learner} \leftarrow (1 - w_{prior})$ ▷ Decrease prior sample frequency
    **end if**
    learned $\leftarrow$ combine(value, learned) ▷ Apply equation 4.2
    learned $\leftarrow$ simplify(learned) ▷ Apply equation 4.8
  **end while**
**end procedure**

# CHAPTER 5
# Hyperparameter distribution learning

Applying a SML algorithm on different sets of data requires, in general, different values of the hyperparameters 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 some 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.

## 5.1 Learning the general prior distribution

A training process is required to learn the general prior distribution for a given hyperparameter. The general prior distribution should be learned from evaluations of the hyperparameter on different sets of data, so as not to overrepresent one single problem.

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, since the performance index 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.

The algorithm evaluates at each iteration if a more complex GMMs (mixtures of more com-

**Algorithm 4** General prior distribution learning

```
function LEARN_GENERAL_PRIOR(training_datasets, threshold)
    training_values ← ∅
    general_prior ← initial GMM
    for dataset D in a group of 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
                append(training_values, λ)
            end if
            general_prior ← fit a GMM to training_values using EM algorithm
            temp_prior ← fit a more complex GMM to training_values using EM algorithm
            if relative likelihood of temp_prior with respect to general_prior > threshold then
                general_prior ← temp_prior
            else
                simpler_prior ← fit a simpler GMM to training_values
                if AIC(simpler_prior) ≤ AIC(general_prior) then
                    general_prior ← simple_prior
                end if
            end if
        end while
    end for
    return general_prior
end function
```

ponents) fits the data significantly better than the current GMM, by calculating the *relative likelihood* of the current, simpler model with respect to the more complex one. The relative likelihood of the current model with respect to the more complex one is the probability that using the current model minimizes the information loss (fits better the data). Only when such probability is very low the more complex model is adopted. This ensures that the complexity of the GMM representing the data only increases when the simpler model does not fit the data well. Likewise, if a simpler GMM fits the data better, it is adopted (without imposing a threshold in this case).

## 5.2 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, and can be used as a prior for starting to sample candidate values of the hyperparameter 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, an alternative way to "fix" the prior must be used.

To achieve this, an alternative prior can be used when there is less confidence that the general prior fits the data. Uniform priors are useful for this, since they do not encode assumptions of the distribution and therefore promote exploration of the hyperparameter space. Using a general prior helps speed up the optimization process because it samples values that are expected to perform well, as opposite to using a uniform prior (random search) which relies only in chance to find good hyperparameter values.

# CHAPTER 6
# Evaluation of results

Fusce vehicula, tortor et gravida porttitor, metus nibh congue lorem, ut tempus purus mauris a pede. Integer tincidunt orci sit amet turpis. Aenean a metus. Aliquam vestibulum lobortis felis. Donec gravida. Sed sed urna. Mauris et orci. Integer ultrices feugiat ligula. Sed dignissim nibh a massa. Donec orci dui, tempor sed, tincidunt nonummy, viverra sit amet, turpis. Quisque lobortis. Proin venenatis tortor nec wisi. Vestibulum placerat. In hac habitasse platea dictumst. Aliquam porta mi quis risus. Donec sagittis luctus diam. Nam ipsum elit, imperdiet vitae, faucibus nec, fringilla eget, leo. Etiam quis dolor in sapien porttitor imperdiet.

## 6.1 Results on standard Machine Learning datasets

Cras pretium. Nulla malesuada ipsum ut libero. Suspendisse gravida hendrerit tellus. Maecenas quis lacus. Morbi fringilla. Vestibulum odio turpis, tempor vitae, scelerisque a, dictum non, massa. Praesent erat felis, porta sit amet, condimentum sit amet, placerat et, turpis. Praesent placerat lacus a enim. Vestibulum non eros. Ut congue. Donec tristique varius tortor. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nam dictum dictum urna.

## 6.2 Results on biological data

Phasellus vestibulum orci vel mauris. Fusce quam leo, adipiscing ac, pulvinar eget, molestie sit amet, erat. Sed diam. Suspendisse eros leo, tempus eget, dapibus sit amet, tempus eu, arcu. Vestibulum wisi metus, dapibus vel, luctus sit amet, condimentum quis, leo. Suspendisse molestie. Duis in ante. Ut sodales sem sit amet mauris. Suspendisse ornare pretium orci. Fusce tristique enim eget mi. Vestibulum eros elit, gravida ac, pharetra sed, lobortis in, massa. Proin at dolor. Duis accumsan accumsan pede. Nullam blandit elit in magna lacinia hendrerit. Ut nonummy luctus eros. Fusce eget tortor.

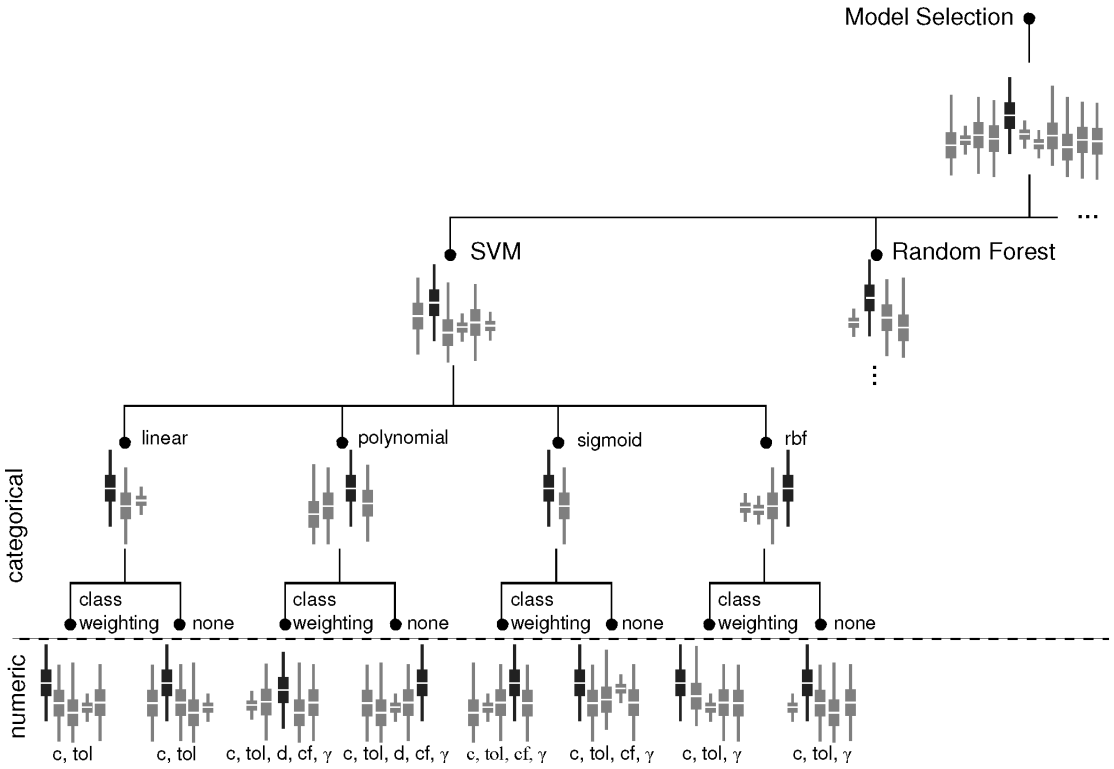
# CHAPTER 7
# Conclusion

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

## 7.1 Future work

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

# 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.

[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.

[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.

[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.

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

_____

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

**Author(s)**

Last name _____ First name _____

**Supervising lecturer**

Last name _____ First name _____

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.

_____ _____

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