body=140mm,210mm,footskip=12mm

,,,,,,

Universität Leipzig
Faculty of Mathematics and Computer Science

Inference of Model Structure Uncertainty in Performance Models for Configurable Software Systems

Master's Thesis

Immanuel Thoke
Born Apr. 29, 1990 in Stollberg i. E.

Matriculation Number 2138070

1. Referee: Prof. Dr. Norbert Siegmund
2. Referee: Prof. Dr. Andreas Maletti

Submission date: June 17, 2024

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Leipzig, June 17, 2024

.....
Immanuel Thoke

Abstract

Performance optimization for configurable software systems is not only an important aspect of software engineering, it restricts how the software is being used on the base of performance evaluations of its configuration options. The lack of domain knowledge in a black-box scenario induces uncertainty in the decision making process, which is why there is a risk that a certain configuration exceeds the user's budget unexpectedly.

To estimate the model structure uncertainty in performance influence models to identify synergistic effects among selected options, this thesis provides the methodology of Gaussian Process Regression, which is the foundation of the Bayesian Optimization Algorithm. The key contribution is the implementation of a variety of different Gaussian process models with shift invariant as well as additive structured kernels that are estimated using exact inference of the marginal log likelihood or the state of the art NUTS sampler for approximate posterior evaluations.

With the presented methods of the Bayesian Approximate Kernel Regression and the distance measure groupRATE, the provided application can identify interactions among configuration options to choose from in a variable selection procedure as a set of constraints to state a dynamic program for finding a best subset selection.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The discipline of software engineering (SE) is typically a multi-staged, distributed technical process to produce software systems (SWS) that work in a specific environment, on a hardware setup that matches the necessary requirements determined by design guidelines that model the system's behaviour based on the physical and functional constraints inferred by interpretations of the user's preferences.

To ensure that the abstract requirements are realized accordingly, performance measures benchmark the output on certain inputs and evaluate if they align with the expected functional aspects. Usually, a SWS has a variety of use-cases and application scenarios to fit a broad spectrum of conventional setups, each having their advantages in executing certain tasks, but likely trade off some performance aspects to optimize for the most used ones. For example, it is common for well-established open source software, like the compiler system LLVM, to work on several hardware architectures in harmony, covering low and high fidelity setups equally well, but it don't compile languages like COBOL, PHP or Smalltalk, which are all quite popular.

To tailor the custom program for individual requirements of an instance to be installed, SWS provide so-called *non-* or *extra-functional configuration options*. These options alter the software's performance and are used to 'tune' the program based on the design decisions of the application scenario. While the outcome of the engineering process, and especially the product owner, needs to guarantee a reliable level of performance for certain configuration setups, testing all possible combinations gets exponentially harder with a growing base of options. That's why in practice these procedures get optimized on a bounded set of features and setups, a 'default option set', while they could work on a broader range of it, but is labeled as 'unintended usage' and excluded from warranty. Derivation from these default options could yield unexpected performance behaviour due to unobserved bugs or edge cases, that improve the

latter. Not only for professional usage, this introduces a source of uncertainty: Deciding for a set of configuration options to aim for a reliable behaviour of the SWS requires knowledge about the expectancy of its performance. If there is no information about the intended use-case there is a risk that it deviates from extrapolated behaviour, called a *performance feature/bug*.

As these extrapolations are grounded in past experiences of the user, the domain knowledge is essential to reason about root causes of possible interactions among configuration options that lead to a posteriori unexpected behaviour, but could be inferred as *expected variance* when retrieving new data. Without any prior domain knowledge, this is called a *black box scenario*, because the user has to rely on the sampled data from previous performance evaluations alone, without any insights like source code or control flow charts. In these situations, a derived argument about expected behaviour between sampled configurations is inherently uncertain due to the unknown data generation procedure. In fact, any scientific approach which tries to infer a model from measurements alone can be interpreted as a statistical model, which introduces *aleatoric uncertainty* (due to measurement errors) and *epistemic risk* about the validity of the model.

A common approach to infer functional behaviour from data that incorporates statistical methods used to estimate the uncertainty of interacting configuration options is *surrogate modeling*. Surrogate models simulate an isomorph system behaviour from which a submodel selection can be sampled to evaluate the performance of certain configurations and derive predictions of new configurations of interest as if they were produced by the surrogated model.

A Bayesian surrogate model can state a framework for model structure uncertainty, where hierarchical dependencies between configuration options are hidden in the data, but cannot be recovered without additional assumptions about the context in which different activation patterns of configuration options occur. The surrogate chosen for the described problem is *Gaussian Process Regression* (GPR). GPRs are useful to model the coefficients of variations in a multidimensional distributional model where the data generator is emulated by an inner product structure, called a *kernel* to estimate covariances of each variational subgroup as a *mixture of models*, which simulate the coupling of software and hardware to mediate the variability of performance 'footprints' a given configuration space on an abstract machine, represented by the model over the data, can have. The inherent uncertainty quantification of this Bayesian statistical model allows for the estimation of confidence regions of interactions

To verify if the approach is suited for solving the described problems, three research questions are stated:

RQ$_0$: How to quantify and verify the uncertainty of interactions?

RQ$_1$: How does the approach perform on different data complexity?

RQ$_2$: Which approach performs best across different data sets?

This thesis shows how GPR as a Bayesian surrogate model can reduce the exponential overhead of an exhaustive search needed to find any 'best' combination for a given set of restrictions to a problem with polynomial probabilistic bounds. This is the foundation for solving the second order optimization problem computationally efficiently with any metaheuristic, stating an optimal decision rule for the inclusion whether a multidimensional aspect is significantly relevant to the domain. The implemented code can be found under `https://git.informatik.uni-leipzig.de/SWS/lehre/abschlussarbeiten/energy-influence-model-structure-uncertainty`.

Chapter 2

Background

For rigorous explanations, the model theoretic foundations are referred to Zeigler et al. 2019 [30]. This chapter will show how data from a black-box scenario is interpreted as a I/O relation observation $IORO = (T, X, \Omega, Y, R)$, with T samples, X the input and Y the output with the constraints that $\Omega \subseteq (X, T)$, the set of allowable input segments and $R \subseteq \Omega \times (Y, T)$, the I/O relation, where $(\omega, \rho) \in R \Rightarrow dom(\omega) = dom(\rho)$. For this IORO it needs to be shown that a statistical learning procedure can estimate a set of functions $\hat{F}$, such that there is a I/O function estimation

$$I\hat{O}\hat{F}O = (T, X, \Omega, Y, \hat{F}) \tag{2.1}$$

where T, X, Y, Ω are the same as for $IORO$ and $\hat{F}$ is a set of I/O functions with the constraint that

$$\hat{f} \in \hat{F} \Rightarrow \hat{f} \subseteq \Omega \times (Y, T) \text{ is a function,} \tag{2.2}$$

and if $\rho = \hat{f}(\omega)$, then $\mathrm{dom}(\rho) = \mathrm{dom}(\omega)$. Further, we motivate how stochastic processes can state a function morphism $I\hat{O}\hat{F}O' = (T', X', \Omega', Y', \hat{F}')$ is a pair (g, k) where

1. $g : \Omega' \to \Omega$
2. $k : (Y, T) \to_{\text{onto}} (Y', T')$
3. For each $\hat{f}' \in \hat{F}'$ there is an $\hat{f} \in \hat{F}$ such that $\hat{f}' = k \circ f \circ g$, i.e., for all $\omega' \in \Omega', \hat{f}'(\omega') = k(\hat{f}(g(\omega')))$.

to surrogate the SWS's performance.

2.1 Performance (Influence) Models of (Highly) Configurable Software Systems

The scientific subject of *Software Engineering* not only involves theoretical and empirical reasoning about all the *technical aspects* of planning, constructing, and shipping application-ready software products. Especially analyzing its requirements sceptically, while guiding a part or the entire lifecycle of these products, requires verified knowledge about the *'environment'* a software system is *embedded* in, which is usually referred to as the *domain* [55] [7].

Especially the concept of *product line engineering* addresses these complex-valued aspects of how diversity in use-case specific requirements is represented and realized in the variability of applications in the domain landspcape [28] [52]. The domain can be thought of a having a characteristic *taxonomy* of hierarchical architectural design *schemes* or *templates*, sometimes referred to as an 'ontology' [1], to build tailored applications [91] top-down or bottom-up to fit any use-case in this field that the software product should satisfy and can be used reliably.

A feature-oriented methodology structures and organizes 'the whole product-line process' [28]. To 'align' the user's requirements with functional and non- or extra-functional model specifications, a set of features is selected to determine a configuration at compile-time, which yields a valid customized program [90] [37] as the instance of a particular design scheme that 'behaves' differently according to the concrete conditions and circumstances present while running.

In terms of a structural (equation) model [24], the features refer to the endogenous variables of the system while the exogenous variable can be expressed as the *performance* of the latter. This models a systematic objective to observe how the configuration and certain inputs shape the possible state transitions, which can result in uncertain output measures, depending on the concrete realization of a system's domain instance. Hence, a set of features to deduce how to optimally *scale* the SWS isn't sufficient. The software needs to be tested on real hardware to estimate a configuration's actual performance. For example, if we want to optimize the trade-off between execution time and memory consumption, a valid configuration needs to ensure that the constraints of one don't impair an opposed feature. Misalignments between requirements and specifications can lead to *performance gaps* between expected and realized performance.

This is called the *feature interaction problem* [28]. '[F]eatures interact

[1] The term originates in aristotelic philosophy and metaphysics [29] but is introduced as a way to describe the generative semantics of a domain and is used in the W3C Semantic Web, [20]

if their simultaneous presence in a configuration leads to an unexpected behaviour, whereas their individual presences do not' [90]. This can happen, for example when the software isn't optimized for the underlying hardware and yields surprising performance (dis-)advantages or occur because of an edge case, which hasn't been covered in the expected scope and thus hasn't been tested and can't be predicted from the model without new data. In other words, a feature interaction alters the system's performance unexpectedly in some way and can trigger an escalation procedure to manage and resolve associated risks [28].

The task of performance optimization is an integrated task of software engineering. Especially in real time applications, complex network structures or dynamical systems, where we face a multilayered structure of possible performance influences [2] and interference, even small perturbations can lead to global instabilities in the future, we need to ensure that a system's performance can be controlled through each phase of its existence to minimize the risk of system failure and build robust SWS.

Systems Science aims to reduce and limit the complexity of these transdisciplinary aspects of software engineering [25] to reflect biases that occur frequently when aiming for multidimensional optimization of processing performant software development processes as consequences of constraints set by an *analytical performance model* [54].

2.1.1 The problem of measuring performance and testing its influences

To illustrate the difficulty of the problem to provide credible performance expectations for an industry-standard open source software, zoom out until the socio-technical dimension of the domain can be seen to have a look at different *'territories'* and how agile software development is specialized across different branches [59]. Reflecting the recent technological advantages in the fields of 'Artificial Intelligence' (AI), even slow moving protocol standardization processes are facing disruptions and companies, once considered 'too big to fail', seem to fumble. Like features can interact under certain conditions, technical interoperabilities change frequently making it surprisingly hard to care for all the corner cases of unintended use *and* pushing forward the accessibility of core assets. Reducing the complexity of different tasks that need to be done only once or repeatingly to fulfill a service-level-agreement (SLA) in order to solve deep problems is no black art, but it requires 'time in the tank'.

Whereas the user is responsible for monitoring its SWS to ensure controlled

[2] see Queuing Theory

output procedures while it's running, the developer and product owner are responsible for reliable performance influences, that configurations can supply expectedly at a given setup description. The software's *configuration space* should not only cover selecting features for high-res solutions, unlocking its full performance potential, but cover a variety of predefined configuration scenarios, providing not only a global but also different customary optima, that fit to canonical option schemes [47].

While most of the cases to trace back a performance bug need a *root-cause analysis* on the code basis [100], there are several scenarios when a *White-Box approach* is either not feasible or simply not possible. One the one hand it requires access to all the information incorporated (not only the code, but also documentation, domain knowledge etc.), on the other hand, on the long run features get deprecated or adopted which, likely alters some variability, then called 'innovation', as a new product line gets introduced.

While this is a continuous process, there are overlapping performance features across different versions of configuration options. But without proper labels or other identifiable metadata, we need to measure 'both' configurations separately, to decide, if there is a smooth continuation.

This also limits interpretability, as we can't speak about something, which isn't defined correctly by contextual annotations, if the context could be decisive to whether a configuration setup is tested or not. To be less uncertain, we need code-testing to estimate the risk of performance escalation during an update procedure in a continuous integration scenario. Following test-driven development guidelines can control risk impact to some extent.

That said, no performance variance is not expected, but underestimating the cross-concerns in software testing can either neglect *hidden improvements* or lead to *performance bugs*.

These aspects can exceed the project's *budget* if opportunity costs for features that should be discontinued a posteriori aren't estimated sufficiently. However, this requires a more flexible model to cover the dynamics of situations with leaky insights.

In such cases, a *Performance Influence Model* (PIM) [52] [90] [60] , which accounts for all possible measurable influences of feature interaction in a *Black Box* scenario by stating a statistical model on the *input-output relations* of the structural model and features under sampling conditions, can be more versatile. To define a PIM, we define a configuration c having k configuration options: $c = \{o_1, o_2, ..., o_k\}$, where o_i is the i-th option, that can be either binary or assigned a numerical value. Denote a configuration's performance Π as a function that *maps* a configuration c from the set of valid configurations C_K to its corresponding scalar performance value $\Pi : C \to \mathbb{R}$ [60].

A PIM can state reasons which information should be acquired next and

helps to extrapolate the risk of worse design decisions, abstracting from the concrete coroutines and pointing out functional and statistical conclusions, which could be less precise, but are a budget-friendly alternative and supports strategic planning.

2.2 Machine Learning for Performance Influence Prediction

We now apply a statistical model to the PIM and utilize machine learning methods to infer function estimates from observations and make performance predictions according to some data updates.

Statistical learning is a process that involves building models to predict or infer patterns from data. It focuses on finding a function that best maps input data to output predictions, balancing the complexity of the model and its ability to generalize well to new, unseen data.

Vapnik [98] describes a statistical learning methodology as an algorithmic approach to reason about abstract theoretical deducible arguments and from which a supervisor can induce conclusions on derived evaluations. Therefore, a *Function Estimation Model* is stated, having

- a *generator* [3] of *random vectors* x, called features, drawn independently from a *fixed* but *unknown distribution* $P(x)$, which reflects the variability of design decisions according to the realized configurations;
- a *supervised output* y for every *input vector* x drawn conditionally from unknown $P(y|x)$ and
- a *'learning machine'*, which can be any (non-)deterministic automaton [4] providing any procedure or coroutine to determine, whether related information is validated as interpretable knowledge about a structure within the model in a set of functions $f(x, \alpha), \alpha \in \Lambda$.

Λ (Lambda) is a set of learning parameters to relate patterns between data points. It likely refers to the Turing-complete lambda-calculus [13]. For example we could use a higher-order logic [5] where the lowest order is just a linear combination of the random vectors and the next order could be some factorization or operations inducing boundary conditions in a measure space,

[3] which can yield a variety of generations

[4] see myhill construction on how to translate a non-deterministic automaton into a deterministic one

[5] See Löwenheim-Skolem's Theorem and for example Frege's predicative logic

clustering the data. First, to find reliable relations between (co-)variables and the response, we define them as a set ℓ of joint probabilities $P(X, Y)$ as the product of $P(X)$ and $P(Y|X)$, a *measure system* .

The *problem of (statistical) learning*, however, is to find a function estimate $\hat{f}_{X,Y}$, which *best* predicts the supervisor's response [98], for which an optimal statistical decision function must exist [105].

As the PIM describes a system's performance in terms of scalar influences of its configuration options on the dependent, exogenous variable, we can predict the possible performance of a configuration, given certain configuration options x^*. But as the sample cases and assumptions under the model are limited due to *known unknowns*, i.e., why the objective is referred to a black box, this introduces *uncertainty* in the prediction of $P(y^*|x^*)$. This can be split into *aleatoric uncertainty* and *epistemic risk* [71].

Aleatoric uncertainty is mainly due to measurement errors, e.g., induced by noise or different confounders influencing the performance signal. A randomized controlled study should address systematic aleatoric uncertainty and filter out or blend experimental conditions which could affect the precision of the true estimator. There is homoscedastic or heteroscedastic noise, depending on whether these conditions are homogeneous or heterogeneous.

Epistemic risk reflects the uncertainty due to missing context information or unverified domain knowledge about conditions under which specific configuration options may have a certain variance of influence on performance, like special software-hardware synergies or optimal performance setups for any use-case, which might deviate from a (tuned) average case. For example a low-res hardware setup probably will likely have higher computation time and vice versa.

That's why it is more likely and computationally efficient to estimate an approximation than the real function, which remains unobserved. The approximation error or discrepancy is measured by a *loss function* $L(y, f(x, \alpha))$ between the response y to a given configuration x and the estimated response $f(x, \alpha)$. To find the best function $f(x, \alpha_0)$ the goal is to minimize the *risk functional*

$$R(\alpha) = \int L(y, f(x, \alpha))dP(x, y), \tag{2.3}$$

'in the situation [of a black-box-scenario] where the joint probability distribution $P(x, y)$ is unknown as the only available information is contained in the training set'.

An early way for an approximate procedure for fitting an assumed function by 'learning' the marginal integral representation of the risk functional is *Laplace's method* [75]. But according Vapnik we can describe statistical learning generally by three problems: The *problem of pattern recognition*, the

problem of regression estimation and the *problem of density estimation.*

If the function $f(x, \alpha_0)$ can be regularized by a L_2 norm, than the *regression function*

$$f(x, \alpha_0) = \int y dP(y|x) \tag{2.4}$$

is one, which minimizes the loss function

$$L(y, f(x, \alpha)) = (y - f(x, \alpha))^2. \tag{2.5}$$

2.2.1 Introducing Bayesian Regression

There is a big debate whether *frequentist* or *Bayesian* statistics are more suitable for different statistical tasks, but applied science should consider which combination of tools solves the research questions 'best'. If the question is about how to express uncertainty quantitatively, it stands to reason to focus on the Bayesian framework first, because it explicitly states 'a coherent axiomatic approach to *inference based on inference of the posterior probability of parameters or models given data*' [79] quantifying uncertainty inherently with explicit conditions formulated. In addition, there are techniques to apply frequentistic notions, like confidence intervals to soundly verify how well the model fits the data, e.g., if effects are significant.

The central formula to that is the platonic *Bayes' Rule*

$$P(M|D) \propto P(D|M)P(M) \tag{2.6}$$

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)} \tag{2.7}$$

which reflects the relationship between an abstract model and its assumptions and the data, from which we want to derive a concrete model structure.

$P(M|D)$ is the *posterior evidence* for model M given data D, $P(D|M)$ is evidence for D given model M, called *likelihood*, $P(M)$ is the *prior probability* of model M and $P(D)$ the probability of the data, which sums or integrates over marginalized sample probabilities.

The prior probability reflects the a priori belief about the distribution(s) of (un-)certainty of a model of probability measures 'before seeing the data' [68]. It is an subjective model [2] of the domain, based on individual or collective perception of knowledge and information asymmetries arising from the (incomplete) retrieval process marked as the 'status quo'. Without having specific domain knowledge, the *Objective Bayes* tries to draw general conclusions about the model's prior stating so-called *reference priors* [19], which are likely to be true and have minimal (false) impact on the posterior inference [2].

The likelihood represents how *likely* it is to observe the data under the assumptions of the model. For example, if the model states that in summer the consumption of ice cream is probably higher than in winter and having samples of orders from a café in the winter containing more orders of icecream than usually in the summer, the probability that the data reflects the model's assumptions is very unlikely.

Ultimately, the posterior contains all information necessary for Bayesian inference. In essence, its probability reflects how much the model's assumptions and arguments are reflected by the data. Regarding the prior probability and the likelihood, it is normalized by the probability of the data. So, if and only if the data can be represented by the model exactly, the *posterior predictive distribution* extends the posterior distribution seeing new data without changing the probability the model is observed given all the data, indicating a high *predictive power* [80].

Depending on the literature different terminology is used to indicate the structural relationship between data and model. Often, the model is represented by a single parameter θ, but there are more expressive formulations. For example, Murphy [80] defines a *generative classifier* of the form $p(y = c|x, \theta) \propto p(x|y = c, \theta)p(y = c|\theta)$, indicating that the response y under the model θ is the prior probability and showing that the posterior is a probability of the response conditioned by the input data and the model! So we can see how the likelihood of the data is reflected in the posterior affected by the model itself. This shows how crucial the choice of prior beliefs is to reason about the resulting posterior and the decisive arguments derived from.

Following Vapnik's Statistical Learning Theory, this thesis focuses on solving the regression problem and the problem of density estimation. *Bayesian Regression* is able to solve these two problems at once by assuming probability density functions (PDFs) of random variables a priori, forming a generative model for the likelihood, and update beliefs accordingly [60].

Generally, the regression problem is defined as

$$Y = f(X) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \tag{2.8}$$

having n observations of data $D = \{(x_i, y_i)\}_{i=1}^{n} \sim p(x, y)$, [79] showing, that the realizations of X and Y are joint probabilities. Knowing this, each sample realizes an instance of the class of functions F, for example

$$f_n(x_n) = \beta^T x_n, \beta \in \mathbb{R}^{\mathrm{p}} \tag{2.9}$$

having only linear functions with scalar coefficients β in a p-dimensional vector space. Therefore the distribution of the response conditioned by β and σ^2,

being a priori normally distributed, is written as

$$y|\beta, \sigma^2 \sim N(X\beta, \sigma^2 I).^6 \tag{2.10}$$

The estimation is usually done by maximizing the likelihood function Lik with respect to $\theta = p(\beta, \sigma^2) = p(\beta|\sigma^2)p(\sigma^2)$

$$Lik(D;\theta) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}}(-\frac{||y_i - \beta^T x_i||^2}{2\sigma^2}) \tag{2.11}$$

or equivalently performed via Ordinary Least Squares (OLS)

$$\arg\max_{\theta} \text{Lik}(D;\theta) \equiv \arg\min_{\beta \in R^p} \left[\frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 \right] \tag{2.12}$$

stating the negative of the log likelihood as

$$L = \sum_{i=1}^{n} (y_i - x_i^T \beta)^2, \tag{2.13}$$

which is consistent to the previous defined loss function [79]. So minimizing the loss function to solve the regression problem is done by maximizing the likelihood of the coefficients, for which the total variance needs to be estimated.

2.2.2 Analysis of Variance and Statistical Interaction

To quantify the uncertain estimates of variables in a parametric regression model, the decomposition of variance is useful to understand how the occurrence of certain parameters affects the response *differently*. As it is *common sense*, [7], that it is expected that different covariates can have different impact on the response, it is reasonable to conclude that their combinations explain a model's variance better, if it is expected that each of them separately has a distinct effect. This is reflected by the *Law of total variance*

$$Var(Y) = E[Var(Y|X)] + Var(E[Y|X]). \tag{2.14}$$

One analytic approach to extract useful information from the explained variance $Var(E[Y|X])$ is Factor Analysis. Correlation between covariances

[6] Note, that this is not a complete bayesian model and for correctly applying the regressor all (un-)known random variables need to be defined by some probability distribution [74] [27].

[7] See Cox's Theorem [6], which explains why a notion of common sense is necessary for sound interpretations of bayesian probabilies.

describe properties of *influence* from some features of the system and are called *latent variable* if they don't match the number parameters. This means, that different parameters explain them in combination for different features. [72] But these factors are aggregated over the population, even marginalized out they don't express uncertainty among the contribution to different factors.

Analysis of Variance (ANOVA) incorporates the exploration of not only the explained variance but the total variance, as analyzing the residuals is important to understand the relation between data and the model, too [74] [72]. More specifically, ANOVA explains how the random variables affect the response variable by assigning a portion of the observed variance to each incorporated RV [1] [72]. To increase the resolution and possibilities to divide the variance into partitions, adding more parameters until the explained variance doesn't increase (much), can help to understand the complexity of the system and how it *grows*. To incorporate many parameters variance in a factor of steps, multivariate methods, like MANOVA (Multivariate Analysis of Variance), need to be applied.

To understand its behaviours, i.e.,how the parameters coefficients vary partially or in total, can be described by their *exchangeability* [70] [27]. Performing frequentistic tests like t- or F-tests investigate, whether a change in *coefficients of variations* $\frac{\sigma}{\mu}$ between groups of variables has a significant change on the outcome. Their results can explain if a parameter is *exchangeable* or *invariant* to one or another [70], for example, if a group of parameters has together has a constant effect size of an single alternative parameter. For instance, a parameter can be *scaling invariant* [21]. In this case one may distinguish between *finite* and *infinite excheangability*, which means that, in the infinite case, adding more random variables of the same parameter family increases the resolution of the explainable variance [8] and thus can be reduced to a single one to explain all of the variance. If they're finitely excheangable, leaving out one decreases the model's explanatory power [27] [9].

If the scale does vary, (and this is likely the case if parameters are finite exchangeably) there can be exogenous and endogenous reasons why this happens. In this case expressivity alone isn't sufficient; we have to investigate whether either different parametric values or changing setups explains the variability.

A few words on this. Imagine a situation where the number of parameters grows faster than the number of factors. This means covariance increases as more parameters overlap to contribute to the same amount of factors [72]. That means that at least some parameters are excheangable and could be replaced by a linear combination of basic parameters and don't explain more variance. Another possibility is the occurrence of so-called *nuisance parameters* [27].

[8] If the search space is non-convex, it can even explain total variance better, but this case should be ignored here for now.

These are unspecified parameters, which do have an impact on the model and need to be estimated consequently if they affect the covariances systematically. In a sense they're a priori 'nonbinary', because they appear as noise in the data as long as they're not observed and distribute a signal if we consider them explicitly in the model.

In a fully black-box scenario a causal model of interaction, as proposed by Pearl [82], can not differentiate between exogenous and endogenous variables, because the exogenous variables, that mediate the interaction between endogenous variables, can not be determined by available domain knowledge. So, the interactions rely solely on the functional differentiation of the related observations between the configuration stated as input and the performance benchmark defined as output. In this case it is useful to distinguish between *moderation* and *mediation* [15] [16].

In moderation *interaction terms* are used to explain additional non-linear and multiplicative errors that may arise from a specific combination of base parameters present in a sample, thus restricting exchangeability [9], because they're necessary to describe the variability of factors determined by a subselection procedure to ensure that the effects are *associated* correctly [60] [26].

In contrast to that, mediation can explain the causal interaction of endogenous variables indirectly in the case of *moderated mediation*, where combined parametric constraints explain deviations in the distribution of effect dependent on the variability of marginal conditions. Nuisance parameters act as moderated mediators as they can mediate the dependent main effects of interacting base parameters, thus concealing them entirely or partially and surrogate confounders or systematic exogenous causal relations.

2.2.3 Model Expansion and Bayesian Hierarchical Modelling

Moderated effects can be observed systematically by expanding the model according to a suggested 'interaction scheme'. Common schemes in NLP and performance prediction of configurable SWS are one- or two-way polynomial interaction [26] [60], whereby the polynomial terms are multiplications of disjunctive pairs or triples of finitely exchangeable base parameters. Especially for simple bayesian models, so-called *varying-coefficients model*, where a single parameter as a vector of coefficients β renders the distribution of a complete dataset, this row-wise extension isn't sufficient to distinguish between moderator and mediator if the factor analysis suggests high correlation between effects, which is known as *multicollinearity* and can lead to the *Simpson's paradox*.

[9] Bernardo and Smith describe non-linearity via partial exchangeability

As Bayesian Inference relies on some a priori knowledge or experience-based intuition about the present conditions under which a parameter certainly varies to quantify uncertainty accordingly, *hierarchical models* define *parameter-dependent conditions* systematically via conditional and joint probabilities.

For example, a simple '2-staged' hierarchical model [27] can be stated as

$$p(\phi, \theta) = p(\phi)p(\theta|\phi)$$

being the joint prior distribution, where θ depends on the unknown parameter ϕ having its own prior distribution $p(\phi)$, called a *hyperprior*. The corresponding joint posterior distribution is

$$p(\phi, \theta|y) \propto p(\phi, \theta)p(y|\phi, \theta) = p(\phi, \theta)p(y|\theta).$$

The hyperparameter ϕ affects y only through θ', which is modeled by the hierarchical dependency. Replacing the nomenclature, this is the simplest way to say that a vector of coefficients β is explained conditionally by the variance σ^2 of the response y as shown in the definition of the 'normal likelihood'.

If all variables are exchangeable, this is sufficient. But as pointed out, interactions are only partially or conditionally exchangeable to explicitly moderate the effect. To avoid mentioned problems, all possible mediators also need to be stratified across a hierarchy of conditional dependencies between interacting parameters, where each case can have their own hyperparameter with a varying probability distributions allowing to distinguish them from moderated effects and distribute the effect size marginally on real interactions only. The so-called *mixed-effect model* can provide an iterative procedure to solve this problem *heuristically* leading to the methodology of 'Variational Bayes'. Alternatively, *empirical Bayes* and weight-space models can apply regularized regression schemes with different norms to distribute *mediated moderation*.

2.2.4 The problem of parametric hierarchies for structural uncertainty estimation

The empirical Bayes approach is especially helpful in fully black-box scenarios, where the prior is uninformative, nuisances are unknown and the focus lies primarily on the moderated effects. But for a complete multivariate analysis the investigation how groups of variables are interacting as mediators on the moderated interaction is impossible if we can't distinguish their effect from the moderated effect. As concluded, estimating the influence of moderated mediators need a hierarchical approach, but without an insightful guess the

hierarchy remains fuzzy and is fragmented across a forest of possible hierarchies. The problem gets even worse if the sample size is small and uncertainty is high due to the spread of possible distribution.

To efficiently compute this problem we need to find an optimal order among the relevant hierarchies, i.e.,shortest paths and intelligent decision making at the furcations, e.g., learnable thresholds.

Estimating the hierarchy of interactions increases the complexity of the model exponentially in the worst case if no selection of specific subsets can reduce this overhead fundamentally. Vapnik introduced the notion of *structural risk minimization* to address these problems of uncertain complexity among the correct extension of the model, where the *VC-dimension*, a measure for the growth of functional complexity, which can represent the complexity of hierarchical interactions, can't be calculated from the base functions [98] and has to be estimated *simultaneously* while estimating the *empirical risk*, which is reflected in the risk functional.

Structural risk minimization (SRM) aims to maximize the effort of reducing the exponential overhead which arises from feedback loops when changing the stratification of parameters in the model. SRM therefore needs to express a notion of *model structure uncertainty*, which can be done by defining how different models are related under certain circumstances, like the different influences of the totality of possible interactions among all subsets of parameters. A *growth function* therefore defines these relations either as a set of sets of possible probability density functions or the class of all boolean operations over the hypothesis of influential parameters [97]. A *generalized growth function* ensures constructive bounds for the involved loss-functions [98] by allowing for the comparison of trade-offs between candidates of moderators for a fraction of variance on specific levels of the hierarchy (e.g., a specific polynomial interaction), which may affect the redistribution of total variance leading to a better or worse model score, while addressing different possibilities of conditional dependence on the vertical axis of the hierarchy. This property is necessary for the optimization routine to have *uniform convergence in probability* [97].

Without limiting the space of possible *latent interaction*, they can conceal the main effects of a moderated interaction completely without being observable. Therefore, the optimization of model structure uncertainty remains unbounded and exponential in overhead if the number of hypotheses doesn't get restricted, e.g., by parameter constraints. To avoid these limitations, a closed model solution is proposed.

2.2.5 Conjugate priors and compound distributions

On paper, without any clue about general preferences or usual tendencies between configuration options based on their semantics, we can suggest any combination of possible interactions influencing endogen parameters that alter the response. However, drawing representative samples from it using analytical methods for a problem where the complexity grows exponentially, is in general infeasible. This is for example a common problem in quantum physics, where the path integral of a n-body problem in a dynamical system is intractable if the data pool is too small. In fact, for any posterior estimation with $p \gg n$, where only approximations apply confidently, this is a major bottleneck.

In such situations it is useful to have a fully bounded, closed model, where we can ensure a priori that all interactions can be evaluated within the model, i.e. eventually observed if and only if reflected by the data. The class of *conjugate distributions* are such models [4]. A *conjugate prior* is a prior distribution where its posterior distribution applies to the same *probability distribution family*, with respect to a given likelihood function [4].

A *conjugate hierarchical model* is one, that can restrict exchangeability or invariance (depending on the parameter type) via conditional dependency of a set of hyperparameters that shape the distribution of the parameter space [27]. If the conjugate distribution is also a *compound distribution*, we can draw conclusions about the latent factors *while* marginalizing over the likelihood [3], which means that some of the residuals can be explained systematically under the assumptions of a mixture distribution. This is necessary for *Multivariate Analysis of Covariance (MANCOVA)*. The first two moments of a compound distribution are precisely derived from the law of total variance and the law of total expectation [14]:

$$E_H[X] = E_G[E_F[X|\theta]] \tag{2.15}$$

$$Var_H(X) = E_G[Var_F(X|\theta)] + Var_G(E_F[X|\theta]) \tag{2.16}$$

Without diving into the details of topology here, remark that F and G are instances of a homotopy H [11] if and only if F and G are conjugated distributions. Then it holds that

> '[i]f the mean of F is distributed as G, which in turn has mean μ and variance σ^2, the expressions above imply $E_H[X] = E_G[\theta] = \mu$ and $Var_H(X) = Var_F(X|\theta) + Var_G(Y) = \tau^2 + \sigma^2$, where τ^2 is the variance of F.' [3]

This is the closing argument for moderated mediation, where a clear hierarchy between the covariance between point estimates (the moderators) on the finite domain $X = [0, \tau]$ and the covariance of a *feature map* $\Phi : X \to F$ (the

moderated mediation) can be modeled, if the inverse problem $\Phi^{-1} : F \to X$ can be solved.

Chapter 3

Related Work

3.1 Performance Models for Configurable Software Systems

The necessity of accurate performance prediction is essential for engineering teams and users to evaluate how the functionality of software systems correspond to the set of constraints of their environmental conditions. The fact that energy consumption is a critical objection on a planet with limited resources and its control a prerequisite of a self-aware society, but still a 'largely untapped potential' in software engineering to date, is pointed out by Siegmund et al. 2020 [89]. That especially runtime performance can be a 'reliable proxy' for energy consumption has been investigated lately by Weber et al. [102]. Thus, it's important to identify 'root causes' of software performance to operationalize their optimization. As this can be a 'lengthy and complex task' [108] and therefore encourages a well structured procedure between software design and performance models to enable automation of such analysis, has been argued by Jing Xu in 2008 [108].

Since then white-box and black-box approaches have been developed to encounter the difficulties of information asymmetry which arise naturally in such situations due to a manifold of reasons, like missing (access to) documentations or the need for cost-sensitive solutions, just to mention a few. White-box approaches focus on insights gained from examining properties of the program at compile- and runtime, especially their control or data flow to identify performance bottlenecks [44] [62], as well as code analysis to locate and debug performance bugs or features [95] [53]. While white-box models source their information from design aspects and performance evaluation, black-box methods rely on existing performance models, e.g., configuration-dependent and historical measurements to predict future evaluations. These predictions are

calculated by the methodology of statistical inference [107] [96] [57] on input parameters and past observations or use simulation techniques like queuing theory [48] [94] to generate synthetic data from which conclusions are drawn.

A central topic to both of them are the non-functional configurations of a SWS that scope the variability of requirements a SWS can apply, which are designed to tailor its performance based on the user's needs. Lillack et al. compared code execution patterns depending on different configuration options using taint analysis, whereas Velez et al. proposed Comprex 'combining insights of local measurements, dynamic taint analysis [...] and compression of the configuration space' to 'understand how and where configuration options and their interactions influence the performance of a system' [99]. This has been investigated further by Weber et al. providing a 'profile and learn approach' to identify '*configuration-dependent performance behaviour* at the *method level*' [101].

3.2 Predicting Interactions of Configuration Options in Black Box Performance Models

To understand how to optimize performance for any application scenario of a software product line it's necessary to gain knowledge about how the variety of options generate different performance evaluations to a certain applied configuration. Black box models can provide cheap clues about where to look for performance gaps first, prioritizing several candidates that can be considered in a white-box analysis afterwards.

Siegmund et al. introduced the framework 'SPLConqueror' [91] [90] [81] to represent and measure non-functional properties of SWS that are used to build configuration-dependent performance-influence models and provide a symbolical learning scheme for computation.

To reduce model complexity to polynomial bounds, feature selection [61] [69] and sampling techniques have been an important branch in this field. Besides solver-based [33] and random sampling [52] [63], 't-wise' coverage was developed to sample interactions between configuration options up to degree t [88] [45] [67]. A sampling strategy for distribution models was proposed by Kaltenecker et al. [34] that utilizes distance measures to characterize the notion of variability in performance evaluations.

Another focus was the dichotomy of models incorporating low or high degree of interaction, because it has been found that low degree interactions are dominating. There exist papers that incorporate both high- and low-order interactions [90] [81], only high-order [57] [58] and only low-order [52] [60].

While subselection of models containing only desired interactions of con-

figuration options, these methods don't take the uncertainty of performance influence into account that occurs frequently among sampled configurations. The successor of DeepPerf proposed a solution to this problem using Bayesian Neural Networks to estimate the distribution of performance influence of large configurations with high degree of interaction considering epistemic as well as aleatoric uncertainty [58]. Dorn et al. [60] presented an empirical bayes approach with an '*automated prior estimation algorithm* to learn performance-influence models in a black box manner without domain knowledge handling aleatoric and epistemic uncertainty among pair-wise interactions.

Several propositions have been made that cover all of these aspects and specialize on a subset of solutions. One of them is DECART, which is based on a classification and regression tree heuristic to build a global prediction model based on recursively sampled local prediction models using 'data-efficient' resampling techniques like cross-validation and bootstrapping up to a certain threshold of prediction error rate. To tune the parameter space they apply random and grid search, as well as Bayesian optimization [69]. Inspired by this work DeepPerf [57] and its successor BDLPerf [58] model a feed-forward neural network to outperform accuracy while relying on big data sets and just randomly sample them. Adding efficient hyperparameter tuning with Bayesian Optimization on the regularized network lead to fast convergence limiting the computational burden of deep neural networks.

Chapter 4

Gaussian Process Regression for Model Structure Uncertainty in Configurable Software Systems

This chapter explains the methodology how to use Gaussian Process Regression for model structure uncertainty. First the structural components of the Gaussian process, the multivariate normal distribution and the covariance matrix get explained in detail. Then the idea how kernels in a stochastic process can simulate the hierarchical dependencies between co-occuring by a surrogated system's morphism. Finally, the methods of the Bayesian Approximate Kernel Regression and the RATE concept are presented, which enable to identify significant interactions between covariates.

4.1 Utilizing the Multivariate Normal Distribution (MVN)

Analyzing more than just a few random independent variables (RVs) in one statistical model is called *multivariate (data) analysis*. Its applications are common in domains, that face complicated or complex relationships among sometimes hundreds or thousands of features to extract (in-)dependencies, factorization, composition and discriminants between certain group structures or spurious associations among causal structures, like biology, macro economics or highly configurable software systems.

Empirical science is interested in describing reality, i.e., the research object, using an ideal but abstract model that explains hypotheses about its behaviour, interpreting the experimental results and updating the model according to the observed information and the errors. Regarding the *principle of maximum en-*

tropy [66], the 'probability distribution which best represents the current state of knowledge about a system is the one with largest entropy[...]'[18] and *Occam's Razor*, starting with a model that only needs minimal assumptions about relations in the data, the MVN is suited for explaining variance exhaustively in a multiple regression procedure.

While the univariate normal distribution is characterized by two parameters of x, the mean μ and its variance σ^2, the generalized MVN given a vector $x \in \mathbb{R}^p$ has a *p-dimensional mean vector* $\mu = E[X] = (E[X_1], E[X_2], \ldots, E[X_p])^T$ and a *(p x p) covariance matrix* $\Sigma_{ij} = E[(X_i - \mu_i)(X_j - \mu_j)] = Cov[X_i, X_j]$, such that its *probability density function* is given by [17] [79]:

$$f(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right).$$

To evaluate the contributions of q *subsets* to the MVN and obtain their *marginal distribution* the joint probabilities (if drawn i.i.d.) can be split like

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, ofsize \begin{bmatrix} q \times 1 \\ (p-q) \times 1 \end{bmatrix},$$

and

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}, ofsize \begin{bmatrix} q \times q & q \times (p-q) \\ (p-q) \times q & (p-q) \times (p-q) \end{bmatrix}.$$

resulting in the marginal and joint distributions

$$x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11}), \quad x_2 \sim \mathcal{N}(\mu_2, \Sigma_{22}), \quad x \sim \mathcal{N}(\mu, \Sigma),$$

In terms of Bayesian inference to calculate the posterior predictive density, the knowledge about the conditional density is necessary:

$$x_1|x_2 \sim \mathcal{N}(\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})[79].$$

4.1.1 Analyzing the Covariance Matrix

The covariance matrix is the 'heart' (latin 'cor') of the MVN in the sense, that their analyzation can yield manifold knowledge about the data, especially interdependencies and patterns of interactions. Further, the information about the covariance can be used for variance reduction. Stating that

$$Var(x_1 - x_2) = Var(x_1) + Var(x_2) - 2\Sigma_{12}[78] \tag{4.1}$$

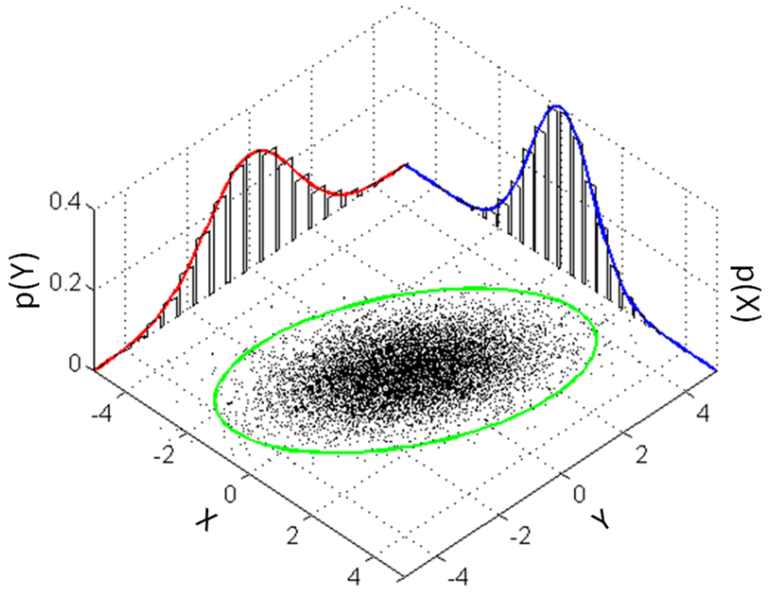


Figure 4.1: marginal distributions and density p(Y), p(X) with their covariance and its cumulative density function [12]

the uncertainty about the expectancy between these RVs is proportional to its covariance. If the covariance of x_1 and x_2 is maximal, the variance between them becomes minimal.

As shown above, for demonstration the covariance matrix can be partitioned into a (2 x 2) matrix:

$$\Sigma = \begin{bmatrix} K & a \\ a^T & b \end{bmatrix}, \tag{4.2}$$

where K forming a group of covariances, while b represents the subject of interest. The diagonal entries of Σ are the variances of K, respectively b, and the perpendicular elements a display the (orthogonal) covariances between K and b [85].

An informative property of the covariance matrix is its relation to the Pearson correlation coefficient, which can be calculated given a random vector X:

$$corr(X) = diag(\Sigma_{XX}))^{-1/2} * \Sigma_{XX}(diag(\Sigma_{XX}))^{-1/2}[5]. \tag{4.3}$$

The inverse of Σ is called the *precision matrix*. It's not always computed with ease, but serves an essential information about the dependency structure: If for features i and j their corresponding entries ij and ji in the precision matrix are 0, then these features are *conditionally independent* [5] and thus don't have additional probability mass.

In optimal cases the covariance matrix is a *positive semi-definite symmetric matrix*, which can be decomposed in its *eigenvalues and eigenvectors*:

$$Kb = \lambda b = U \Lambda U^* \tag{4.4}$$

$det(K - \lambda U) = 0$ if $Kb = \lambda Ub$ [8] [23] These eigenmaps can be utilized to infer latent features with special properties. Imagine you condition on a feature with minimal response, return the covariance of the correlative distribution, compute the 0.95 percentile of the eigenvalues, its corresponding eigenvectors and map them onto the input space to know, which features will move the mixture average the most or not. Spectral clustering for instance is a well established method utilizing these linkages[1].

4.2 Covariance Functions in the Reproducing Kernel Hilbert Space

If it can be validated that the covariance matrix is positive (semi-)definite and the precision matrix between features of choice is not zero, the conditional dependence reflects that the occurrence of this group has a limited effect on the response variable, that can't be explained solely any by (sub-)literal of this feature group. Instead the interaction itself is needed to explain the additional effect. But how can we know, whether the interaction is a representation for the estimated function between input and output? To understand this, the *Reproducing Kernel Hilbert Space* (RKHS) is introduced.

[1] see also https://en.wikipedia.org/wiki/Linear_discriminant_analysis#Multiclass_LDA

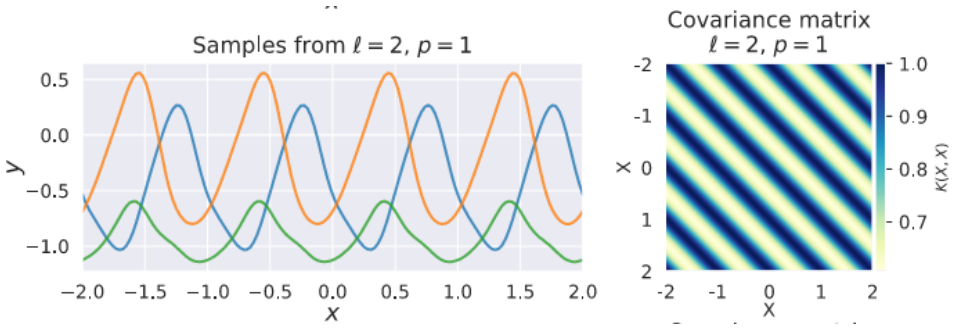


Figure 4.2: samples from a periodic kernel and its covariance matrix

4.2.1 First - What's a 'kernel'?

Signal processing, Graph Theory or supervised machine learning are typical fields utilizing kernel methods with famous achievments like the *Fast Fourier Transform* algorithm, Convolutional Neural Networks and the Support Vector Machine. We can reduce the risk of misinterpretations of this key concept by using a characteristic example from the research domain:

Suppose, you are evaluating a suitable configuration for your application environment, e.g. you want to optimize the energy footprint of your software system based on the current architecture. Maybe you lack in memory capacity, so you wouldn't choose options other than M_0, that increase the demand for memory. At the same time you want to minimize execution time (which might increase energy footprint) and say M_1 is the only option that isn't increasing energy[2] while minimizing execution time. This is a conditional dependency - we choose option M_0 because we can't afford M_1. So if you argue how much additional memory is needed to compensate the estimated execution time and M_0 to M_1 is a continuum, the problem is to find the break even point 't', where the response value shifts [3]. But as there are infinite possibilities it is necessary to find a set of points (called *inducing points* or *splines*) that indicate an exact turning point or reflects shifting signs of a limit, such that you can separate the configuration options, that satisfy the required conditions.

If the conditional dependence is reflected by the occurrence of a third feature as a workaround, that gets activated more often in cases with M_0 and staying in certain bounds , it becomes linearizable because then we have a clear indication of when the classes can be separated.

Suppose we substitute them with three cardinal features A,B,C for which all (commutative) combinations of an associative product have at least a par-

[2]or only with minimal energy overhead like sunk costs that can be relocated else where and don't affect the energy bill

[3]see for example https://en.wikipedia.org/wiki/Phase-change_memory with t states

tial order $AB <= AC <= BC$, searching for at least one single point of strict (partial) order $AB <= AC < BC$, such that C becomes a clear separator if A or B is present. [4] If there is not such a feature C, induce a function of covariance, called a *kernel*, such that for *all samples* there is a set of combinations which translates to a linear separator in this function space, that indicates the membership to a distinct class of features, i.e.,having a deterministic algorithm that permutes feature combinations and *reproduces* the metrical order. [83] In other terms this forms group homomorphisms, a reason why the central operating system of Linux is also-called a 'kernel' having different *modules* that can be equipped ensuring the ACID principle.

Having a kernel, the feature map can be extended by a co-zero map forming structural equations like $a + b = b + c + d$ with b as the zero element. This works well for all kernels reproducing a symmetric positive semi-definite covariance matrix that translates in an eigenmap of the features with *eigenfunctions* $\{\phi_\ell\}_{\ell=1}^{\infty}$ and *eigenvalues* $\{\theta_\ell\}_{\ell=1}^{\infty}$ having a finite integral operator

$$\delta_\phi(x) = \int_{\mathcal{X}} k(x, x')\phi_\ell(x')dx' \tag{4.5}$$

stating *Mercer's Theorem* and resulting in the kernel family

$$k(x, x') = \sum_{\ell=1}^{\infty} \theta_\ell \phi_\ell(x)\phi_\ell(x'). \tag{4.6}$$

4.2.2 The Reproducing Kernel Hilbert Space

Recall that a *Hilbert space* H 'is a vector space equipped with an inner product that induces a distance function for which the space is a complete metric space' [10] over real functions f defined on an index set X. Then H is called a *reproducing kernel Hilbert space* (RKHS) if there exists a function $k : X \times X \to \mathbb{R}$ having the properties

1. for every x, $k(x, x')$ as a function of x' belongs to H, and
2. k has the reproducing property $\langle f(\cdot), k(\cdot, x)\rangle_H = f(x)$ [32].

The mathematical details resulting in this lemma would go beyond the scope of this thesis, but it follows another required statement, the *Representer Theorem*:

Assuming the following *(regularized) empirical risk minimizer*

$$\hat{f} = \arg\min_{f \in \mathcal{H}_k} \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2 + \lambda \|f\|^2_{\mathcal{H}_k}, \tag{4.7}$$

[4] Cauchy-Schwartz inequality

admits a representation of the form

$$\hat{f} = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i), \tag{4.8}$$

In natural language, this means that there is always a kernel that can 'mimic' the function, because the kernel function acts as the inner product of the data.

Going back to Mercer's Theorem and deducing a vector space spanned by the bases having scalar product and emit orthogonality properties through rotation that can indicate conditional independence and disparity in the conjugate transpose. The base function are of form

$$\mathbf{\Psi}(x) = \{\sqrt{\theta_\ell}\phi_\ell(x)\}_{\ell=1}^{\infty} \tag{4.9}$$

returning the *kernel regressor*

$$y = \mathbf{\Psi}^T c + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \tau^2 I). \tag{4.10}$$

For example, $\sqrt{\theta}$ can be thought of the trigonometric *angular resonance frequency* at location x, implying a feature map $\Psi(t) = A[cos(\sqrt{k/m} * t) + isin(\sqrt{k/m} * t)]$ with $\sqrt{\theta} = \sqrt{k/m}$ and $-k/m = c^2$.

Summarizing, the RKHS states that a proper kernel function is sufficient to estimate the empirical function of the surrogate if it is able to create linear combinations that state an unambiguous metrical order among the features.

4.2.3 Boolean satisfiability of polynomial covariances

When working with boolean categorical features the polynomial kernel $K(x, y) = (x^T y + c)^d$ of degree d can be represented as logical statements of the prior or the posterior [5] using Conjunctive Normal Forms (CNFs) to model restrictions among feature sets to compare explicit interactions. This corresponds to performing *Polynomial Regression* without the need to define all the polynomial terms by simply utilizing this kernel function in a Gaussian Process Regression. This way all interactions of degree d as features in a conjunctive clause sum up to 1, while all others stay 0 [49]. Applying boolean masks on NumPy arrays like

```
np.array(
        functools.reduce(
            np.logical_and,
```

[5] For example disjunct or overlapping hypothesises for systematic experiments (subsequent model selection of multi-staged ensembles and scenario techniques.

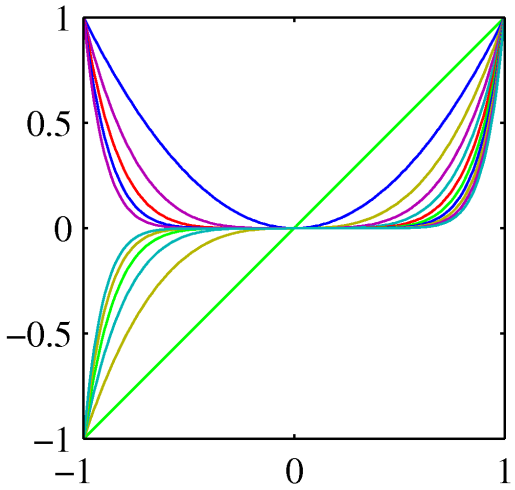


Figure 4.3: polynomial basis functions

```
        [X[:,i] == bool(a),
        X[:,j] == bool(-b)]
        )
)
```

on the row samples can lead to satisfiable models checking for the desired interaction terms only. NLP applications apply $d = 2$ (2-SAT) for common grammatical aspects. MAP-Estimators align with the idea to find the best fit of maximal covariance indicating that the corresponding kernel is a truthful representer explaining why there is covariance.

4.3 Gaussian Process Regression

4.3.1 Using stochastic processes as nonparametric Bayesian models

Like the MVN as a distribution model satisfying the principle of maximum entropy [50, 80], nonparametric models don't need additional assumptions about the structure of the model - even being closed under the features in the data is not required. [6] Instead, the data shapes how the model is fitted as a *mixture of models* by a priori unknown parameters. This means, that 'Bayesian 'nonparametric' models actually have infinitely many parameters which are

[6] Without having domain knowledge the (initial) guess about the choice of parametrics is otherwise itself a NP-hard problem.

given a Bayesian prior.' [40] Having a defined domain as the dimensionality of the features applicated to a general class of model of models, the *multivariate Gaussian mixture model*, that can be extended by new features with new data, infers that the mean and (co-)variance are not parameters of the model itself, but a model over the data. Thus, the features are parameters themselves.

So the terminology of 'nonparametrics' (as well as the term of 'nonfunctional features' in software engineering) is misleading. In fact, the knowledge about a validated model lets us project 'extraordinary' information onto the data about its environment as an embedding of the throughput of the system's state [7] onto the data. This is reflected by *Kolmogorov's Extension Theorem*:

'If a family of *Borel probability measures*, labelled by their corresponding finite index set, μ_V obeys the following consistency relation for all $V \subset X$ and all second finite index sets with $V \subset U$

$$\mu_U(\pi^{-1}_{U\to V}(E)) = \mu_V(E) \tag{4.11}$$

and there is a unique probability measure on the product σ-algebra, μ_X with the property:

$$\mu_X(\pi^{-1}_{X\to V}(E)) = \mu_V(E), \forall E \in \mathcal{B}(\mathbb{R}^{\mathrm{V}}) \tag{4.12}$$

The collection of measures μ_V are called the finite dimension distributions or finite dimensional marginals of the measure μ_X.' [40]

This way, the nonparametric model can be extended by additional dimensions from the feature map with any data update as long as the consistency property of the (projected) family of probability measures is obeyed, i.e.,being part of the same sub vector space V.

Accepting this, a stochastic process can be viewed as an emergent higher order parametric model, if there exist a projected feature map $\pi_{X\to V}$ and a filter $\pi_{U\to V}$ for autocorrelation, that can grow consistently with new data.

4.3.2 Gaussian Process Definition

A *Gaussian Process* (GP) represents a 'probability distributions over functions' [79]

$$p(f) \propto \exp\left(-\frac{1}{2}\|f\|^2_{\mathcal{H}_k}\right), \quad p(f) \geq 0 \ \forall f \in \mathcal{H}, \quad \int_{f\in\mathcal{H}} p(f)\, df = 1, \tag{4.13}$$

that map the input space $\mathcal{X}$ onto a feature space $\mathcal{F}$ that obey consistent rules how new features can be combined or rather *transformed* i.e.,having a mean

[7] which is a important information for estimations of the initial state of a dynamical system, see https://de.wikipedia.org/wiki/Normierbarkeitskriterium_von_Kolmogoroff and https://de.wikipedia.org/wiki/Lokalkonvexer_Raum

$\mu = \{\mu(x_1), \ldots, \mu(x_n)\}$ and covariance $\Sigma_{ij} = K(x_i, x_j)$ to all other row vectors $x_1, \ldots, x_n$ that reproduce the specific similarity, stating $\mu(x) = E[f(x)]$ and $K(x_i, x_j) = E[(f(x_i) - \mu(x_i))(f(x_j) - \mu(x_j))]$:

$$f \sim \mathcal{GP}(\mu(\cdot), K(\cdot, \cdot)). \tag{4.14}$$

So the GP is an 'infinite version of a multivariate Gaussian distribution [...]', where the 'infinite dimensional object [is defined by] finite dimensional projections' [79].

4.3.3 Gaussian Process Regression

This way, choosing a prior $f \sim \mathcal{GP}(0, K)$ with K, a specific kernel function, obtaining a joint Gaussian distribution of latent functions $p(f|X, \theta)$, where θ is a hyperparameter family of the kernel (see next chapter) and likelihood $p(y|f, X, \theta) = \mathcal{N}(f, \sigma^2 \mathbf{I}_N)$, via the marginalization properties of the MVN stating the marginal distribution $Z(\theta) = p(y|X, \theta) = \int p(y|f, X, \theta)p(f|X, \theta)df$, we can derive the posterior distribution of the GP:

$$p(f|y, X, \theta) = \frac{p(y|f, X, \theta)p(f|X, \theta)}{Z(\theta)}. \tag{4.15}$$

Computing the integral of the marginal distribution while maximizing out θ is the standard procedure to obtain the posterior distribution. For prediction, the posterior predictive densitiy is calculated with a validation set $\mathbf{X}_*$ having M observations:

$$p(f_*|X_*, X, y, \theta) = \int p(f_*|X_*, X, f, \theta)p(f|X, y, \theta)df, \tag{4.16}$$

which is a MVN of the form

$$p\left(\begin{bmatrix} f \\ f_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_N & K_{NM} \\ K_{MN} & K_M \end{bmatrix}\right), \tag{4.17}$$

with K_{MN} as the cross-covariance of the training and validation inputs.

Summarizing, the Gaussian Process Regression can be computed using

$$\log Z(\theta) = -\frac{1}{2}\left(y^T(K_N + \sigma_n^2 I_N)^{-1}y\right) + \log(\det(K_N + \sigma_n^2 I_N)) + N\log(2\pi) \tag{4.18}$$

$$\mu_* = K_{MN}(K_N + \sigma_n^2 I_N)^{-1}y. \tag{4.19}$$

$$\Sigma_* = K_M - K_{MN}(K_N + \sigma_n^2 I_N)^{-1}K_{NM}. \tag{4.20}$$

Be aware, that therefore the inversion of the kernel matrix is necessary. This is the computational burden of all GP models, which has $\mathcal{O}(n^3)$ using the 'vanilla' GP with *Cholesky decomposition* [32]. It is shown later that there are useful approximations that speed this process up to $\mathcal{O}(n \log n)$.

4.3.4 The problem of choosing an appropriate kernel

The problem of choosing 'the right' kernel structure is always domain or even application specific, so users have to rely on *best practices* and expansions driven by epistemic certainty about the domain development. As mentioned, in NLP, the (piecewise-) polynomial kernel is often a good choice [49] [39], but there are others, that count as a good initial choice, not only for physical systems. Rasmussen et al. 2006 classifies 'stationary covariance functions', with favourite representatives like the *Radial Basis Function*, which ensures euclidian geometry by any function that is a norm over the input $\phi(x) = ||x||$ and is used for function approximation, which follows by the Representer Theorem, and the *Matérn kernel class* (a results of spatial statistics), which has a hyperparameter for differentiation and the *Rational Quadratic*, which all share, that they have *spectral density power spectrum* and can be represented as a *Fourier transform* as well. Stationary covariances state *shift invariance* $k(\tau) = x - x'$ and can be used to extend a GP having Markov properties [86]. This ensures that the kernels are stationary and may equip a lengthscale hyperparameter, which enables Automatic Relevance Determination (see next chapter). Besides basic stationary kernels, there is a class for *dot product covariances*, that are suited for concatination, e.g. forming kronecker products [50] or second order additive composite kernels [86]. This thesis focuses on vector-valued kernel functions [8], which includes more advanced achievements like the *Brownian Bridge Kernel*, the *Heat Diffusion Kernel*, the *Spectral Mixture Kernel* and the kernel of *Random Fourier Features*. There are also functions that utilize the *Fisher information matrix* for string-typed inputs or non-stationary kernels, that are designed for neural network covariances [32]. Like in clustering, one might go top-down or bottom-up to select fitting kernels and in general any metaheuristic can guide the error to convergence as long as the correct metrics are applied, e.g., the hyperparameter evaluation should be reflected by the likelihood of the kernel assumptions over the data.

4.3.5 Concerns about the hyperparameter familiy θ

For the 'fully bayesian model' each kernel, which can be the composition of 'atomic' kernels, has its own prior called *hyperpriors*. As these parameters 'control the distribution of the parameters [(here the base of the latent function)] at the bottom level' [32], placing priors over hyperparameters is called a *hierarchical model*. This opens the door for the concept of *Automatic Relevance Determination* (ARD). Rather than fixing the smoothnes of the kernel with specific determined values, we estimate them to address the *kurtosis* of

[8] see also Alvarez et al. 2011 for am in-depth analysis of vector-valued kernels

the distribution and care for *heavy tails*. For all covariances with a *lengthscale* using ARD, the inverse of the *featurewise characteristic lengthscales* 'determines how relevant an input is: if the length-scale has a very large value, the covariance will become almost [conditionally] independent of that input, effectively removing it from the inference.' [32] Hyperparameter optimization can lead to overfitting, if the regularization isn't well calibrated, because it increases model complexity significantly. So this should be done with sufficient quantity of data only [9].

4.3.6 A Gaussian hierarchical interaction model

As the SRM principle can be applied to nonlinear dynamics [36], such as a family of functions L over marginal distributions intersecting a RKHS jointly [97], the invariance between these functions define 'kernel interactions' as the surrogate interaction scheme: The estimated interactions get projected onto the domain by the kernel operator, for which the domain is a finite embedding in the infinite-dimensional GP.

For demonstrative purposes, the methodology of a sparse GP regression model is provided, which is tailored for approximative computation primarily, focusing on a hierarchical population model of theorized interactions:

$$\begin{aligned} \tau &\sim p(\tau) \\ f \mid \tau &\sim \mathcal{GP}(0, k_\tau) \\ \sigma^2 &\sim p(\sigma^2) \\ y^{(n)} \mid f, x^{(n)}, \sigma^2 &\sim \mathcal{N}(f(x^{(n)}), \sigma^2) \end{aligned} \tag{4.21}$$

τ (Tau) is the nuisance parameter that accounts for the population model structure and is used for ARD variable selection in hyperparameter optimization. As the functions f that model the GP given the kernel k_τ are distributed given τ, conditional dependence between covariates in the precision matrix of the GP indicates that the intersecting functions are smooth. This defines a kernel interaction in the Gaussian scale mixture distribution $\mathcal{N}(f(x^{(n)}), \sigma^2)$. The distribution model $\mathcal{N}$ could be reparametrized as $\mathcal{N}(\theta^T \phi(x^{(n)}), \sigma^2)$ with $k_\tau = \phi(x^{(n)})$, following that $\theta \mid \tau \sim \mathcal{N}(0, \Sigma_\tau)$, from which the marginal likelihood can be derived as

$$p(D \mid \tau, \sigma^2) = \int p(D \mid \theta, \sigma^2)\, dp(\theta \mid \tau), \tag{4.22}$$

which is a MVN density function and equals

[9] this problem is addressed by the term *Empirical Bayes*, see Murphy 2012, Chapter 5

$$\frac{(1/\sqrt{2\pi\sigma^2})^N \det(2\pi\Sigma_{N,\tau})^{1/2} \exp\left(-\frac{1}{2\sigma^2}Y^TY\right)}{\det(2\pi\Sigma_\tau)^{1/2} \exp\left(-\frac{1}{2\sigma^2}Y^T\phi(X)\Sigma_{N,\tau}\phi(X)^TY\right)}, \tag{4.23}$$

where $\Sigma_{N,\tau}^{-1} := \Sigma_\tau^{-1} + \frac{1}{\sigma^2}\phi(X)^T\phi(X)$.

4.3.7 Structured Additive GP Regression

As to sample all interactions in a GP which has a positive-definite covariance matrix the complexity is still $\mathcal{O}(2^d)$, Siegmund et al. [90] suggested an additive performance interaction scheme to avoid this computational burden. Moreover, additive models 'give rise to interpretable models' [43]. Therefore, the prior must be a sum of d one-dimensional functions:

$$f(x) = f_1(x_1) + f_2(x_2) + ... + f_n(x_n). \tag{4.24}$$

Whereas the basic GP just has one kernel, the additive GP model structure has at least d kernels, which applies for heterogeneous interaction shapes. This is especially useful for external validity, because it enables extrapolation [43]. Furthermore, it reduces the complexity of the regression problem to $\mathcal{O}(n \log n)$ [40].

On paper, this approach can construct a *Toeplitz structure*, where each additive kernel component aligns on the diagonal of the covariance matrix, which is called the 'Generalized Additive GP Regression' [40] and shows, why it's $\mathcal{O}(n \log n)$. More common are so-called *second order additive structures*, where $\binom{d}{2}$ kernels are concatenated pairwisely, while each pair is the product of some base kernel: $k_1k_2 + k_1k_3 + k_2k_3 + ... + k_{d-1}k_d$.

The cross-correlation between these pairwise disjunct covariances explains the degree to which two configuration options - united yet divided - are capable of explaining the behaviour of one another [22]. This is known as *d-th order variance*. Saatci 2011 [86] shows on extent how this can determine on which order of interaction most variance is located and the curse of dimensionality avoided.

4.4 Computational Inference and Implementation

Gaussian Process Regression has become quite popular in the last decade, especially in its context of the Bayesian Optimization Algorithm. Selecting a suited implementation should be seen itself a performance optimization problem as the capabilities to produce valid models depends on the flexibility and

stability of the framework. For this reason, three different Python implementations have been evaluated: PyMC's GP, GPy and GPyTorch. GPy has been discontinued and excluded for that reason, though its methods and optimization routines provided a broad variety of features. While PyMC showed the best runtime performance relying on estimation with the No-U-Turn-Sampler (NUTS), a MCMC sampling technique, and using JAX for hardware acceleration, its restrictions to this inference method and the somewhat 'exotic' tensor framework underneeth, lead to the choice of using GPyTorch and its extension BoTorch, which provides implementations for different heuristics and covers the feature sets of the other frameworks merely completely.

Following Machine Learning Operations (MLOps) design principles, a ML pipeline is implemented to orchestrate the workflow [42]. This includes data extraction, transformation and loading (ETL), as well as model definition, training and storage. To deal with different implementations, modules and versioning, a so-called clean architecture principle [76] is applied. This includes a directory for adapters that bind the different workflow components, the domain directory, where the dataset class and the base model structure is defined and the application directory coding the actual pipeline. The implementation is written in pure Python and relies on reliable and well established frameworks, such as NumPy, Scikit-Learn and PyTorch. All requirements are provided via the Python package managers 'Pip' and 'Conda'.

4.4.1 Basic ML pipeline setup

For comparability, the data ETL process is inherited from the 'ConfigSysProxy' class of the Dorn et al. 2020 'bayesify' framework [60], where the data is extracted from a compressed archive containing the measurements of the datasets. Synthetic data is generated in memory only. The preprocessing includes the possibility to yield experiments for sequential or parallel training and some basic feature engineering methods, like adding virtual interactions or a boolean map to filter subsets of the data, as the implementation is only suitable for binary configuration options for now. The data is then split into a training and a validation set, while a minimal training size is controlled.

All GP models are derived from the GP_Prior domain module, which implements the 'automatic prior estimation' from Dorn et al. 2020 [60], an empirical Bayes method to calculate a weighted linear mean based on regularized linear regression models $\mathcal{M}$ that compute a 'regression spectrum' to estimate weights from the average error of the linear coefficients from each model m_i

on the training set $\bar{\epsilon}(m_i)$:

$$\mathbf{w} = \left\{ \frac{-\bar{\epsilon}(m_i)}{\sum_{j=1}^{|\mathcal{M}|} -\bar{\epsilon}(m_j)} \mid \forall m_i \in \mathcal{M} \right\} \tag{4.25}$$

$$\mu_w(t) = \frac{1}{\sum_{j=1}^{|\mathbf{w}|} w_j} \sum_{i \in \mathcal{I}_{\mathcal{M}}(t)} w_i \tag{4.26}$$

The GP_Prior class is then used as an interface to implement specific adapters, that model the GP based on the experiment setup. Beside the initialization method to calculate the weights for the mean, it just specifies which prior structure is applied to the GP, that is a GPyTorch Mean with a varying coefficient parameter β, and the intercept,

```
import torch
from gpytorch.means import Mean

class LinearMean(Mean):
    def __init__(self, beta, intercept):
        super().__init__()
        self.register_parameter('beta',
            torch.nn.Parameter(torch.tensor(beta)))
        self.register_parameter('intercept',
            torch.nn.Parameter(torch.tensor(intercept)))

    def forward(self, x):
        return (self.beta @ x.t() + self.intercept).squeeze()
```

and kernel module:

```
# excerpt from the kernel module
def get_matern52_kernel(X, ARD=True):
    lengthscale_prior = HalfCauchyPrior(scale=0.1)
    if ARD:
        return MaternKernel(
            nu=2.5,
            lengthscale_prior=lengthscale_prior,
            ard_num_dims=len(X.T))
    else:
        return MaternKernel(
            nu=2.5,
            lengthscale_prior=lengthscale_prior)

def wrap_scale_kernel(base_kernel):
    outscale_prior = GammaPrior(2, 0.15)
    return ScaleKernel(
        base_kernel=base_kernel,
        outputscale_prior=outscale_prior)
```

For now, all kernel setups have a fixed hyperprior setting. For kernels with a lengthscale hyperparameter the half-Cauchy distribution is chosen. This log-scaled prior, also known as the *horseshoe prior*, takes 'heavy tails' of the kurtosis into account, can induce sparsity and thus enables sample efficiency due to global shrinkage control via ARD [87].

All base kernels are wrapped by a ScaleKernel to take the hierarchical population structure into account and is equipped with a Gamma distribution prior. Implemented are the (piecewise-) polynomial kernel, the RBF,

two Matérn kernels, a 'Random Fourier Feature' kernel and one for spectral mixture, which improves prediction away from the data by approximating the spectral density of the mixture of models [103].

The concrete model instance is been chosen based on inference type, i.e.,whether an exact computation of the bayesian regression problem is processed or an approximation scheme of the integral of the MLL is applied.

4.4.2 Exact GP implementation

Gardner et al. 2018 developed the GPyTorch framework, which uses stochastic gradient optimization to calculate an exact distribution of the GP posterior via minimizing the loss of the negative marginal log-likelihood [65]. The inference they introduced is called 'Blackbox Matrix-Matrix' (BBMM), which ,in comparison to the 'vanilla GP' with Cholesky decomposition of the kernel matrix, 'reduces the time complexity of exact GP inference from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. BBMM computes the inferred equations using matrix-vector multiplications iteratively to calculate the *conjugate gradients* of the inverse of the kernel matrix, using a *modified Batched Conjugate Gradients Algorithm* (see Appendix) [86] [64]. The 'MyExactGP' implementation allows for a batched multiouput model inherited from the BatchedMultiOutputGPyTorchModel class, where multiple independent GPs can be trained in parallel, although only sequential training is implemented for now.

The optimization routine is performed with the stochastic gradient descent algorithm 'Adam' [41], that optimizes the loss of the negative MLL, as well as the hyperparameters.

4.4.3 Sparse Axis-Aligned Subspace GP implementation

With the Bayesian Optimization in PyTorch (BoTorch) framework, [77] comes the implementation of the 'Sparse Axis-Aligned Subspace' GP (SAASGP), which also applies a hierarchical prior structure, but relies on inference on a Hamiltonian Monte Carlo sampling of the posterior, the 'No-U-Turn Sampler' (NUTS) [56] (see Appendix) to numerically approximate the posterior distribution based on Markov chains of the marginal distributions. This allows for high dimensional models, where the numbers of parameters can exceed the number of samples by far and still converges to a stationary distribution. Using only shift kernels $k(x-x')$, these Markov chains can always be constructed, as explained in 2.2.

The hierarchical model structure is implemented using Pyro, a probabilistic programming framework. Eriksson and Jankowiak 2021 [46], who introduced the SAASGP used a slightly different hyperprior structure than provided in

the BoTorch library. To match the setup with the ExactGP, the variance parameter is sampled from a Gamma distribution and not from a log-Normal, as described in the paper. In difference to the ExactGP, the SAASGP also has a separare hyperprior for the inverse lengthscale for implicit ARD without the need to opt-in variable selection.

While PyMC already implemented a JAX version of the NUTS sampler to geometrically compute the gradients with 'autograd', the attempt for an own JAX implementation using Numpyro failed due to stability issues with the covariance matrix. This is somewhat problematic, as the strength of this approach is to sample high-dimensional and not only sparse model data. But as the experiments will show, the used real world data sets are in fact sparse, leading to fast convergence and acceptable computation costs.

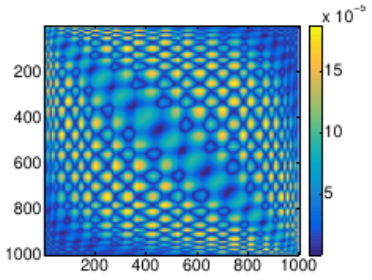
4.4.4 A Generalized Additive Model using Grid Interpolation

The most efficient implementation presented has been developed by Wilson and Nickisch 2015 [104]. Utilizing inducing point methods, they exploit Kronecker and Toeplitz *persymmetric* structures in the covariance matrix that can speed up the inference up to $\mathcal{O}(n + m \log m)$. While Kronecker structures are common in geospatial statistics, they don't scale well on high dimensions [32, 86]. Toeplitz structures are much more common [51]. Actually, all covariance matrices of stationary shift-invariant kernels implemented have a Toeplitz structure, which are constant along their diagonals: $K_{i,j} = K_{i+1,j+1} = k(x_i - x_j)$ [104]. This property can be used to approximate a hierarchical kernel structure, if a grid of inducing points is provided, over which a sparse approximation of the cross covariance between the inducing points can be calculated, using local cubic and inverse distance weights [104], i.e.,interpolate a structured kernel approximately as a finite subset of m covariances over n inducing points. The equidistant inducing points can then be used to tridiagonalize the covariance matrix and explicate its eigenvectors and -values [51] (see also next chapter).

Using GPyTorch's AdditiveStructureKernel along with the GridInterpolationKernel, a second order additive pairwise kernel interaction scheme [10] is built, where a ProductKernel wraps $\binom{d}{2}$ atomic base kernels from the list of all implemented simple kernels permuted over pairwise combinations of input dimensions:

```
# excerpt of the Base Kernels method
def get_base_kernels(X, kernel="linear", ARD=False):
    elif kernel == "piecewise_polynomial":
```

[10] See Duvenaud 2014 [43], which provides different algorithms for additive GP regression.



(c) $|K_{\text{true}} - K_{\text{SKI, 40}}|$

Figure 4.4: Differences in covariance between the additive and a 'simple' kernel model.

```
        base_kernels = [get_piecewise_polynomial_kernel(X, ARD=ARD)
            for item in range(X.shape[1])]

# Additive Kernel
def get_additive_kernel(kernels):
    additive_kernel = kernels[0]
    for kernel in kernels[1:]:
        additive_kernel += kernel
    return len(kernels), additive_kernel

def additive_structure_kernel(X, base_kernels):
    import itertools
    outscale_prior = HalfCauchyPrior(scale=1)
        grid_size = grid.choose_grid_size(X, kronecker_structure=False)
        d_kernels = [ScaleKernel(
            GridInterpolationKernel(
            ProductKernel(k1,k2),
            grid_size=int(grid_size), num_dims=1, active_dims=[i,j]),
            outputscale_prior=outscale_prior,
            num_dims=2,
            ard_num_dims=2) for (i,k1),(j,k2) in
            itertools.combinations(enumerate(base_kernels), 2)]
    num_dims, d_kernels = get_additive_kernel(d_kernels)
    return AdditiveStructureKernel(base_kernel=d_kernels, num_dims=num_dims)
```

This establishes a *generalized additive model* with a linear operator as the grid of inducing points over the sum of smooth second order functions (the pairwise dot product kernels) of covariates [106].

4.5 Methodology for Feature Interaction Identification

It is likely that a certain fraction of combinations of configuration options will have similar performance as SWS get optimized to work on different setups, but often there are edge cases that aren't covered well and could have high influences on the average performance. These cases could be underestimated in random sampling due to selection biases of pseudo-random number generators. We now provide a methodology to answer the first research quesion,

> RQ_0: How to quantify and verify the uncertainty of interactions?

where in principle all interactions among configuration options can be evaluated within a set of samples from the posterior distribution. As interactions are population based stochastic processes, it will be shown theoretically how this can be done with *inverse uncertainty quantification*.

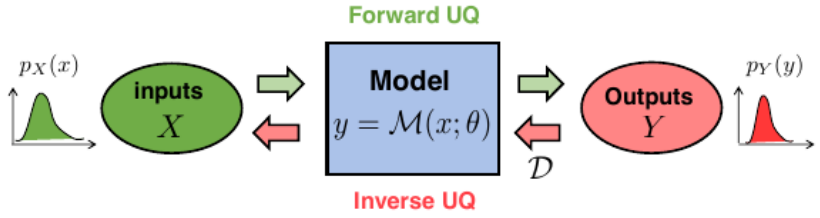


Figure 4.5: Illustration of Uncertainty Quantification (UQ): forward UQ and inverse UQ [110].

4.5.1 Identification of Feature Interaction

In contrast to linear regression models, where the estimated effect sizes of interactions $\tilde{\beta}_{ij}$ are direct results of the coefficients of multiplicative terms of an expansion of base features x_i, x_j, called moderation, non-linear regression models try to explain the latent feature space as a mediator of interaction among the intersecting functions of the domain, which do not correspond directly with a size of effect.

As the kernel function is used to infer dependencies between explanatory variables as a surrogate of feature interaction, transforming the domain into the latent feature space, where each function is part of a orthonormal basis, this procedure is called a *projection* $\text{Proj}(\mathbf{X}, \mathbf{y})$ [38, 92].

Derived from the definitions of the GP, y is represented by the domain of functions $f = [f(x_1), ..., f(x_n)]^T$. Therefore, an *effect size analog* is defined as the determination of coefficients 'that result from projecting the design matrix $\mathbf{X}$ onto the nonlinear response vector $\mathbf{f}$,

$$\tilde{\beta} = \text{Proj}(\mathbf{X}, \mathbf{f}).'[38]$$

Independent from whether an exact or just an approximate solution of the GP posterior is computed, the *generalized approximate kernel model* ensures that the effect size analogs are correct and converge to the exact kernel as sample size increases [84]. Defining the estimated kernel matrix as $\tilde{K} = \tilde{\Psi}^T\tilde{\Psi}$ with $\tilde{\Psi} = [\tilde{\phi}(x_1), ..., \tilde{\phi}(x_n)]$, it can be shown that $\tilde{\beta} = \mathbf{X}^\dagger\tilde{\Psi}^T\mathbf{c}$ with $\mathbf{K}\alpha \approx \mathbf{X}\tilde{\beta}$ even for $p \gg n$ [38]. This means, that *all* interactions present can be denoted to the latent feature space $\mathbf{Z}$, from which follows, that for shift-invariant kernels $k(\mathbf{u}, \mathbf{v}) = k(\mathbf{u} - \mathbf{v})$ that integrate to $\int k(\mathbf{z})d\mathbf{z} = 1$, $\mathbf{z} = \mathbf{u} - \mathbf{v}$ [38].

This can be used for a singular value decomposition (SVD) of the kernel matrix $\tilde{\mathbf{K}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^T$, where $\tilde{\mathbf{U}}$ can be a $n \times q$ matrix of eigenvectors with $q \times q$ eigenvalues $\tilde{\mathbf{\Lambda}}$, where q are the top q eigenvalues. To find a reasonable threshold for q, the explained variance score of the GP is used to define a confidential proportion of the eigenvalues:

```
# part of the evaluation metric module
U, lam, V = np.linalg.svd(X)
explained_var = get_metrics(posterior,
    y_test,
    posterior.mixture_mean.squeeze(),
    type="GP")["explained_variance"]
lam_cumsum = np.cumsum(np.array(lam) / np.sum(np.array(lam))
q = np.where(lam_cumsum >= explained_var)[0][0] + 1
U = U[:, :q]
Lambda = np.diag(np.sort(lam)[::-1])[:q]
```

Finally, the SVD is used to inversely project the latent feature space onto the domain. As the kernel matrix is scaled by the learned factor α, the empirical factors θ of the learned kernel hyperparameters are needed to correctly distribute the latent features to their associated explanatory parameters. For proper translation of the approximated projection with q eigenvalues the correction factor $\mathbf{c}$ is needed, which relates to θ as $\mathbf{c} = (\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\mathbf{T}\tilde{\mathbf{K}}^{-1}\tilde{\mathbf{\Psi}}^\mathbf{T})^{-1}\theta$. As $\alpha = \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{\Psi}}^T\mathbf{c}$ and $\theta = \tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}\alpha$, it can be concluded that

$$g^{-1}(\mu) = \tilde{\mathbf{\Psi}}\mathbf{c} = \tilde{\mathbf{K}}\alpha = \tilde{\mathbf{U}}\theta. \tag{4.27}$$

```
# part of the evaluation metric module
B = np.linalg.pinv(model.X.T) @ U
thetas = torch.diag(Lambda) @ U.T
betas = np.zeros((inverse.shape[0], thetas.shape[0]))
for i in range(betamat.shape[1]):
    betas[:, i] = inverse @ thetas[i, :].T
betas = betas.T
```

This approximation asserts the correct distribution of effect even if the interactions are sparse [38]. It shows how the hierarchical structure of the kernel hyperparameters applies to the population distribution, which is why we can use spectral analysis techniques to estimate sharp contours even if the image is blurred out due to $p \gg n$. That's why a GP with these properties is equivalent to a Hidden Markov Model: The kaleidoscopic throughput of the input output function observation on a group permutation of inputs (see Schur's lemma) is effectively the limit of pattern transformation, i.e.,the latent interaction in the restricted reproducing kernel Hilbert subspace. This can be used to argue for a prioritization of exploration of higher dimensions over exploitation of lower dimensional features which could be confounded by additional components. This can speed up convergence for the overall structural risk minimiation, because the trade-offs could be balanced against cross-entropy relations. That motivates to use Random Fourier Feature Kernel Maps as a possible transform of a shift-invariant kernel from the exponential family and sample chains of Taylor polynoms up to a degree of latent dimension, where the betas don't change significantly. The general structure for Gaussian like kernel interactions is

$$k(\mathbf{u}, \mathbf{v}) = \exp\{-h\|\mathbf{u} - \mathbf{v}\|^2\} = \exp\{-h\|\mathbf{u}\|^2\} \exp\{-h\|\mathbf{v}\|^2\} \exp\{-h\langle \mathbf{u}, \mathbf{v}\rangle\}. \tag{4.28}$$

The third term expands to

$$\exp\{-h\langle \mathbf{u}, \mathbf{v}\rangle\} = \sum_{m=0}^{\infty} \frac{h^m}{m!} (\mathbf{u}^T \mathbf{v})^m \quad \text{where} \quad (\mathbf{u}^T \mathbf{v})^m = \sum_{j \in [p]^m} \left(\prod_{i=1}^{m} u_{ji} \right) \left(\prod_{i=1}^{m} v_{ji} \right), \tag{4.29}$$

which shows, that for binary features boolean maps can be applied to sample subselections.

4.5.2 Classifying significant interaction

To identify which interactions have a significant effect on the response, the methodology of Crawford et al. 2017 [38] introducing Bayesian Approxiamte

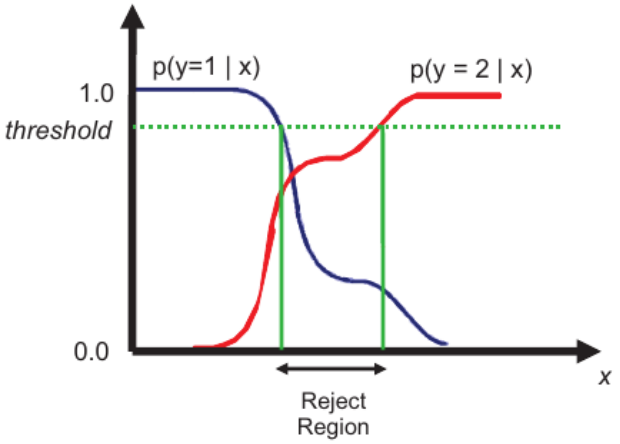


Figure 4.6: Illustration from Murphy 2012 [80] of the 'rejection gap' induced by the threshold

Kernel Regression (BAKR) with variable selection provides the concept of *Posterior Probability of Association Analog (PPAA)*. Therefore, for each value of β_j [11] a 'frequentistic' *p-value*, denoted as $\tilde{\pi}_j$, is calculated, indicating if the effect is associated under the null hypothesis $H_0^{(j)} : \tilde{\beta}_j = 0$:

$$\tilde{\pi}_j = 2\left(1 - \Phi\left(\frac{|\tilde{\beta}_j|}{\hat{\sigma}}\right)\right), \quad j = 1, \ldots, p, \tag{4.30}$$

where Φ is the cumulative distribution function (CDF) of the standard normal distribution. Since σ is unknown, they estimate $\hat{\sigma}$ using the median absolute deviation (MAD) of the β-values: $\hat{\sigma} = \frac{MAD}{0.674}$. These p-values then get sorted in ascending order $\tilde{\pi}_1 \leq \tilde{\pi}_2 \leq \ldots \leq \tilde{\pi}_p$. To avoid the cumulation of type I errors, a bonferroni correction is applied to the significance level λ: $\lambda(j/p)$ [109], to apply for the 'ideal *false discovery rate*'. Now the largest index j^* is calculated to find the rank of the least significant variable $\tilde{\pi}_j \leq \lambda(j/p)$. For this j^*, a threshold z_{j^*} is computed

$$z_{j^*} = \hat{\sigma}\Phi^{-1}\left(1 - \frac{\tilde{\pi}_{j^*}}{2}\right), \tag{4.31}$$

such that we can compare it with all $|\tilde{\beta}_j|$ to mark a significance of interaction.

[11] The index parameter j maps to any finite group of dimensional interaction p. Think of it as $\beta_{ijk\ldots}$

This frequentistic estimation of interaction effect ensures that the observed effects are robust and represent a sufficient statistic from which a generator can be stated to construct a domain language with words $\tilde{\gamma}_j$

$$\tilde{\gamma}_j = \begin{cases} 1 & \text{if } |\tilde{\beta}_j| \geq z_{j^*} \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } j = 1, \ldots, p, \tag{4.32}$$

which guarantees an *optimal decision rule* for which probability to account conditionally, as solicited by Vapnik's statistical learning theory.

4.5.3 A distance measure of interaction

As described in (4.1), the MVN is a conjugate distribution. In such cases, it is useful to describe the influence of specific variables as a *measure of distance* even without the necessity of having a metric on the space of probability distributions, by computing the *relative entropy* between mixture distributions, also known as the *Kullback-Leibler divergence* (KLD).

As the GP with a fixed kernel constructs a restricted RKHS, a measure of distance translates the similarities between covariance eigenvector features bilinearly [31], which means the orientation towards the center from $-\sigma$ or $+\sigma$ is epxlicitly unambiguous if and only if the *local false sign rate*, the analog used in BAKR for the *local false discovery rate* [93], which controls the family-wise error rate - necessary when working with stochastic processes - indicates that. Locality, therefore, requires the conditionalized marginal likelihood for validation of what is evident.

Crawford et al. 2019 applied a KLD to the BAKR effect size analog metric and implemented an 'association mapping case' to mark marginal importance of the β_j associated to the multivariate distribution of the jth variant in the domain of the inputs as a sequence or a group of indices to condition on [73]. This reflects the probability of being removed from the data but not from the model, which could be no difference at all, but is likely due to the similarity to other data for the trained context. In this special case we can reduce the GP to a *quadratic assignment problem* as we can describe the sampling procedure of influential features as a complete bitartite graph of interpretable parameters including the covariance, respective the precision matrix:

$$\tilde{\beta} = \begin{pmatrix} \tilde{\beta}_j \\ \beta_{-j} \end{pmatrix}, \quad \mu = \begin{pmatrix} \mu_j \\ \mu_{-j} \end{pmatrix}, \tag{4.33}$$

$$\theta_j = \begin{pmatrix} -\Lambda_{-j}^{-1}\lambda_{-j} \end{pmatrix}, \tag{4.34}$$

$$\Sigma = \begin{pmatrix} \sigma_j & \sigma_{-j}^T \\ \sigma_{-j} & \Sigma_{-j} \end{pmatrix}, \quad \Lambda = \begin{pmatrix} \lambda_j & \lambda_{-j}^T \\ \lambda_{-j} & \Lambda_{-j} \end{pmatrix}. \tag{4.35}$$

stating the KLD of a feature β_j

$$\mathrm{KLD}(\tilde{\beta}_j) = \int_{\tilde{\beta}_{-j}} \log\left(\frac{\mathcal{P}(\tilde{\beta}_{-j})}{\mathcal{P}(\tilde{\beta}_{-j}|\tilde{\beta}_j)}\right) \mathcal{P}(\tilde{\beta}_{-j})\, d\tilde{\beta}_{-j}, \quad j = 1, \ldots, p, \tag{4.36}$$

$$= \frac{1}{2}\left[-\log|\Sigma_{-j}\Lambda_{-j}| + E(e_{-j}^T \Lambda_{-j} e_{-j}) - 2E(e_{-j})\Lambda_{-j}\theta_j e_j\right.$$

$$\left. -E(e_{-j}^T \Sigma_{-j}^{-1} e_{-j}) + e_j^2 \theta_j^T \Lambda_{-j} \theta_j\right].$$

'More contextually specific questions arise when deciding if a given centrality measure is significant.' [73] Specifically, a notion of characteristica that can reflect a triangle inequality and the symmetries of the bipartitions, without the necessity of positivity, depending on the other features as well.

Chapter 5

Evaluation

5.1 Experimentation Setup

Experiments are set up to answer the empirical questions that arise in the research area while trying to solve the problem *how* the modeled approach is grounded in its application and how the results can be interpreted. From the fields of engineering practices it is known that it requires a calm mind to not mistrust the belief in well-established standards. While experience in the domain teaches us to test not only the default options, it is always necessary to scope the prior belief about what to care for since resources are always limited.

That's why we aim for simplicity and focus on a variety of models investigated, while the experimental workflow is kept minimal. For each setup

- the kernel type and structure,
- the used inference method and
- training sizes

are controlled. Depending on the research question addressed, different metrics are evaluated to estimate the propositions that can be verified. Central to all questions are the relationships and dependencies between the model and the data. The first research question for internal validity is therefore:

> RQ_1: How does the approach perform on different data complexity?

To pave the road to a reasonable answer, synthetic data sets are generated, that reflect different levels of complexity among the input dimensions. Beneath four base features, polynomial terms of the base are added to moderate their interactions, where as the coefficients of each term are distributed randomly. This is easily done with NumPy and Sklearn:

```
# part of the synthetic data generator
X = np.random.randint(2, size=(n_samples, n_features))
X = sklearn.preprocessing.PolynomialFeatures(
    degree=d).fit_transform(X)
coefficients = np.random.rand(X.shape[1]) * 10
y = X @ coefficients + np.random.normal(0, noise, n_samples)
```

For each degree $d = \{2, 3, 4\}$, a set with 1000 data points is generated in memory.

To score the models, we track three different error metrics. The first is the *Mean Absolute Percentage Error* (MAPE). When dealing with uncertainty models, the percentage error of the prediction accuracy is a straightforward interpretation of the model's goodness-of-fit. We adopt the terminology of Dorn et al. 2020 [60] to explain how the MAPE is computed: First, the absolute percentage error (APE) for each configuration $c \in \mathcal{C}$ with the measured performance $\Pi_{\text{true}}(c)$ and predicted performance $\Pi_{\text{pred}}(c)$ is calculated

$$\text{APE}(c) = \frac{|\Pi_{\text{true}}(c) - \Pi_{\text{pred}}(c)|}{\Pi_{\text{true}}(c)}. \tag{5.1}$$

Then, the MAPE is just the average over all APEs:

$$\text{MAPE}(C) = \frac{\sum_{c \in C} \text{APE}(c)}{|C|}. \tag{5.2}$$

As this approach to quantify the uncertainty of interactions is based on MANCOVA, calculating the proportion of explained variance (ESS), closely related the *Coefficient of Determination* (R^2), of the model provides an intuitive measure of how well the model asserts a meaningful structure to the variability in the data:

$$\mathcal{R}^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}, \tag{5.3}$$

where SS_{res} is the residual sum of squares, which measures the total difference of observed values from the predicted values and SS_{tot}, which measures the total difference of observed values from the mean. To take the proportion of training size and number of parameters into account, the adjusted R^2 is given by

$$\mathcal{R}^2_{\text{adj}} = 1 - \frac{(1 - \mathcal{R}^2)(n - 1)}{n - p - 1}. \tag{5.4}$$

This is important, because the number of parameters can improve the $\mathcal{R}^2$ while the complexity of the model doesn't reflect the predictive performance one to one, meaning that a model with less parameters could have similar $\mathcal{R}^2$, but higher $\mathcal{R}^2_{\text{adj}}$.

Another metric useful when working with Bayesian models is the *Bayesian Information Criterion* (BIC):

$$\mathcal{BIC} = \ln(n)p - 2\ln(Lik), \tag{5.5}$$

which balances goodness-of-fit via the likelihood function of the model and its complexity.

After first tests high correlations between ESS and the Root Mean Squared Error (RMSE) could be observed, this score metric has been added as well and is used rather than the BIC in most cases, because the BIC scores remained hard to interprete:

$$\mathrm{RMSE}(\hat{\theta}) = \sqrt{\mathrm{MSE}(\hat{\theta})} = \sqrt{E((\hat{\theta} - \theta)^2)}. \tag{5.6}$$

Internal validity can help to decide if the approach is in principle suited for the domain, but it needs to show evidence whether it can outperform preceding approaches on real world examples. A kernel is a representation of a domain function, and therefore it depends on the system's design if two distinct software systems can be compared using a similar approach. However, there are kernels that try to represent a generalizable family of functions possible to describe an entire domain, such as the Matérn kernel class and additive structure kernels. So we could try to use them on a variety of subject systems if the answer to $\mathbf{RQ}_1$ indicates that a kernel performs best across all levels of complexity measured. That's why the next research question is:

> RQ_2: Which approach performs best across different data sets?

The pool of SWS examined consists of several performance benchmarks on single job measurements under constant load. The 'performance' benchmark is preferred, otherwise 'fixed-energy' is chosen:

SWS	Domain	Options	Benchmark
Apache	web server	19	performance
HSQLDB	Java database	14	performance
LLVM	compiler toolchain	16	performance
PostgreSQL	database system	18	performance
VP8	video encoder	0	performance
x264	video encoder	15	fixed-energy

For all candidates, again the kernel type and structure, as well as the training size and inference method are controlled.

5.2 Results

5.2.1 Internal validity

From each synthetic dataset with polynomial features 180 models have been trained. As all models are stored with their scores in a JSON database, they can be queried using Pandas to find the best performing models under certain circumstances.

Best model with low complexity (polynomial degree 2)

We calculate the top 3 best scoring models for each score. In each score, the simple polynomial kernel with degree 2 has always scored best using exact inference. Because results comparing the MAPE, ESS and RMSE (omitting the BIC) showed the clearest correlation, they are presented here, while more results can be found in the appendix. The ESS showed best performance with only 20 samples, RMSE and MAPE scored lowest with $n = 50$:

idx	kernel	structure	samples	inference	MAPE	ESS	RMSE
0	poly2	simple	20	exact	0.068638	0.994903	1.433441
2	poly3	simple	20	exact	0.068638	0.994903	1.433441
4	poly4	simple	20	exact	0.068638	0.994903	1.433441

idx	kernel	structure	samples	inference	ESS	RMSE	MAPE
36	poly2	simple	50	exact	0.998067	0.932725	0.133901
38	poly3	simple	50	exact	0.998067	0.932725	0.133901
40	poly4	simple	50	exact	0.998067	0.932725	0.133901

idx	kernel	structure	samples	inference	RMSE	ESS	MAPE
36	poly2	simple	50	exact	0.932725	0.998067	0.133901
38	poly3	simple	50	exact	0.932725	0.998067	0.133901
40	poly4	simple	50	exact	0.932725	0.998067	0.133901

That table 2 and 3 are exactly the same is not incorrect. In fact, the scores correlate perfectly here. Notice as well, that the scores are exactly the same for the different polynomial kernels, allthough training time differs (ommited for space reasons). That this is no coincidence, will be shown in the discussion.

Best model with medium complexity (polynomial degree 3)

For degree 3, we see similar results as before:

idx	kernel	structure	samples	inference	MAPE	RMSE	ESS
0	poly2	simple	20	exact	0.055321	1.313854	0.991693
2	poly3	simple	20	exact	0.055321	1.313854	0.991693
4	poly4	simple	20	exact	0.055321	1.313854	0.991693

idx	kernel	structure	samples	inference	ESS	RMSE	MAPE
144	poly2	simple	500	exact	0.994665	1.003015	1.01805
146	poly3	simple	500	exact	0.994665	1.003015	1.01805
148	poly4	simple	500	exact	0.994665	1.003015	1.01805

idx	kernel	structure	samples	inference	RMSE	ESS	MAPE
144	poly2	simple	500	exact	1.003015	0.994665	1.01805
146	poly3	simple	500	exact	1.003015	0.994665	1.01805
148	poly4	simple	500	exact	1.003015	0.994665	1.01805

We see a quite similar picture with the degree 3 model, except that the ESS and RMSE, both with perfect correlation again, scores best at 500 samples each.

Best model with high complexity (polynomial degree 4)

For the model with highest complexity, the situation changes substantially:

idx	kernel	structure	samples	inference	MAPE	RMSE	ESS
51	RFF	simple	50	MCMC	0.056492	1.117017	0.995063
65	matern32	additive	50	MCMC	0.056547	1.119943	0.995045
59	poly4	additive	50	MCMC	0.056640	1.119941	0.995037

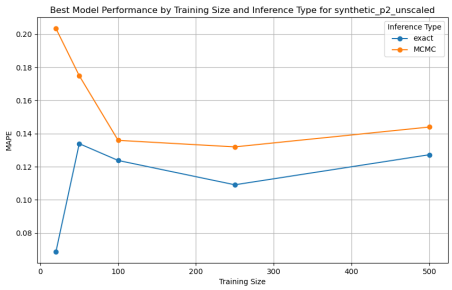
idx	kernel	structure	samples	inference	ESS	RMSE	MAPE
9	RBF	simple	20	MCMC	0.996808	0.844134	0.078918
13	matern52	simple	20	MCMC	0.996576	0.868904	0.077713
17	spectral_mixture	simple	20	MCMC	0.996501	0.883804	0.081767

idx	kernel	structure	samples	inference	RMSE	ESS	MAPE
9	RBF	simple	20	MCMC	0.844134	0.996808	0.078918
13	matern52	simple	20	MCMC	0.868904	0.996576	0.077713
17	spectral_mixture	simple	20	MCMC	0.883804	0.996501	0.081767

Now, the more general kernels arise at the top, along with MCMC inference. For the first time some additive models are under the best models as well. Where we have seen big models for the best RMSE and ESS before, now the smallest models with MCMC score best.

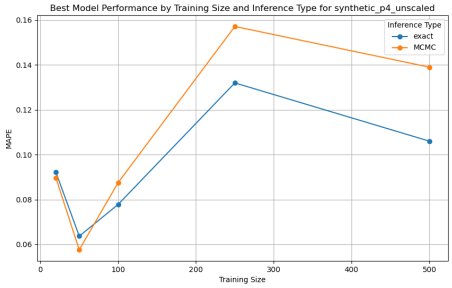
Sample efficiency

Because of the discrepancy between high and low model complexity, in terms of best scoring kernels and inference method, we have a closer look on the sample efficiency of the inferende methods:



(a) training size dependent on MAPE for polynomial model with degree 2

(b) training size dependent on MAPE for polynomial model with degree 3

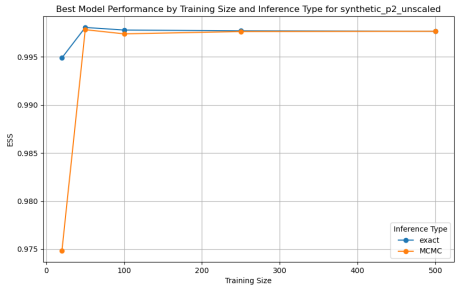


(c) training size dependent on MAPE for polynomial model with degree 4 (beware of the intercept)

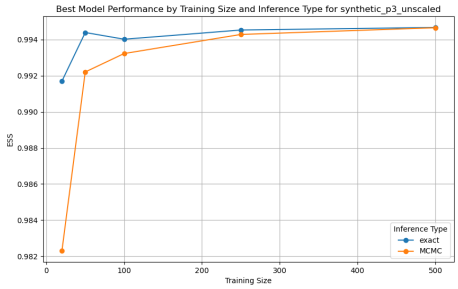
Figure 5.1: Overall caption for the grouped figures

The pattern is quite clear: Exact inference outcompetes MCMC across all training sizes and all levels of data complexity, except for the degree 4 model, where MCMC hits the lowest MAPE ever. With increasing complexity the MAPE increases as well slightly, but we can see an inverse relationship as well: High data complexity is met with high score values on small sample sizes. But this symmetry is broken for the degree 2 model, where the MAPE converges

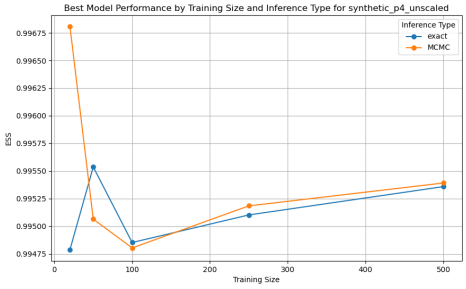
on medium sample sizes. In general, the models with $n = 100$ have the most robust scores. The pattern is almost congruent if we observe the ESS:



(a) training size dependent on ESS for polynomial model with degree 2



(b) training size dependent on ESS for polynomial model with degree 3



(c) training size dependent on ESS for polynomial model with degree 4

Figure 5.2: Overall caption for the grouped figures

Again, the best score is raised by the MCMC inference on more complex data. While ESS is best for exact inference on the lower data complexity, the lead for MCMC in the degree 4 model is only marginal.

To answer $\mathbf{RQ_1}$ - top 10 overall depending on score - correlation in scores, also in subsets

5.2.2 External validity

To evaluate which kernel is the best surrogate to estimate the family of functions under which a SWS is being optimized, the models with the highest scores per SWS are presented:

First run

Dataset	Kernel	Structure	Training Size	Inference	ESS	RMSE	MAPE
Apache	poly2	simple	500	exact	0.99	193.14	0.01
HSQLDB	RBF	additive	500	MCMC	0.91	0.42	0.02
HSQLDB	poly3	simple	500	MCMC	1.00	646.43	0.01
LLVM	poly2	simple	100	exact	0.95	32.35	0.01
PostgreSQL	poly2	simple	500	exact	0.26	49.65	0.01
VP8	poly3	simple	500	MCMC	1.00	275.04	0.01
x264	RFF	simple	250	MCMC	0.91	0.12	0.02

Table 5.1: Top model for each dataset with highest ESS.

Dataset	Kernel	Structure	Training Size	Inference	MAPE	RMSE	ESS
Apache	poly2	simple	500	exact	0.01	193.14	0.99
HSQLDB	RBF	additive	500	MCMC	0.02	0.42	0.91
HSQLDB	poly3	simple	500	MCMC	0.01	646.43	1.00
LLVM	poly2	simple	500	exact	0.01	30.71	0.95
PostgreSQL	poly2	simple	500	exact	0.01	49.65	0.26
VP8	poly2	additive	500	MCMC	0.01	275.25	1.00
x264	RFF	simple	250	MCMC	0.02	0.12	0.91

Table 5.2: Top model for each dataset with lowest MAPE.

Dataset	Kernel	Structure	Training Size	Inference	RMSE	ESS	MAPE
Apache	poly2	simple	500	exact	193.14	0.99	0.01
HSQLDB	RBF	additive	500	MCMC	0.42	0.91	0.02
HSQLDB	poly3	simple	500	MCMC	646.43	1.00	0.01
LLVM	poly2	simple	500	exact	30.71	0.95	0.01
PostgreSQL	poly2	simple	500	exact	49.65	0.26	0.01
VP8	poly3	simple	500	MCMC	275.04	1.00	0.01
x264	RFF	simple	250	MCMC	0.12	0.91	0.02

Table 5.3: Top model for each dataset with lowest RMSE.

This picture is almost perfect! Only the two model setups scoring best for 'VP8' have marginally different RMSE for the 'poly3 - simple' and 'poly2 - additive' models, while ESS and MAPE are equally good. Beyond these minor differences, each data set has a clear kernel candidate and the scores are partly overwhelmingly well, with two models scoring $ESS = 1$ and five models with $MAPE = 0.01$. But we need to look at some BIC scores as well, because the picture changes slightly in the second run.

Second run

Dataset	Kernel	Structure	Training Size	Inference	ESS	BIC	MAPE
Apache	poly3	additive	100	MCMC	0.99	88302.62	0.01
HSQLDB	poly4	additive	100	MCMC	0.80	3397.42	0.03
HSQLDB	poly2	simple	100	exact	0.86	66053.93	0.01
LLVM	poly2	simple	100	exact	0.95	4592.98	0.01
PostgreSQL	poly3	additive	100	MCMC	0.15	24115.59	0.01
VP8	poly2	simple	100	MCMC	1.00	105513.14	0.02
x264	matern32	additive	100	MCMC	0.90	-1699.55	0.02

Table 5.4: Top model for each dataset with highest ESS.

Dataset	Kernel	Structure	Training Size	Inference	MAPE	BIC	ESS
Apache	matern52	simple	100	MCMC	0.01	89014.35	0.99
HSQLDB	RFF	simple	100	MCMC	0.03	3524.88	0.79
HSQLDB	poly2	simple	100	exact	0.01	66053.93	0.86
LLVM	poly2	simple	500	exact	0.01	19381.30	0.95
PostgreSQL	poly2	simple	100	exact	0.01	9348.74	0.12
VP8	poly2	simple	100	MCMC	0.02	105513.14	1.00
x264	poly4	simple	100	MCMC	0.02	-1707.03	0.90

Table 5.5: Top model for each dataset with lowest MAPE.

Dataset	Kernel	Structure	Training Size	Inference	BIC	ESS	MAP
Apache	matern52	additive	20	MCMC	26475.45	0.93	0.02
HSQLDB	poly2	simple	50	exact	172.12	0.67	0.05
HSQLDB	poly2	simple	20	exact	7698.10	0.72	0.01
LLVM	poly2	simple	20	exact	2358.90	0.75	0.01
PostgreSQL	poly2	simple	20	exact	3619.25	-0.40	0.01
VP8	spectral mixture	additive	50	MCMC	71729.08	1.00	0.03
x264	matern52	additive	100	MCMC	-1714.09	0.90	0.02

Table 5.6: Top model for each dataset with lowest BIC.

To answer $\mathcal{RQ}\in_2$, the average scores of each kernel are aggregated over all models. The averaged scores are then ranked and the top scoring kernel are presented:

Dataset	Kernel	Structure	ESS
x264	poly4	additive	0.83
PostgreSQL	RBF	simple	-0.33
VP8	matern52	simple	0.93
HSQLDB	matern32	additive	0.76
HSQLDB	matern52	simple	0.69
Apache	matern52	additive	0.97
LLVM	matern32	simple	0.89

Table 5.7: Best model with best ESS score on average for each dataset.

Dataset	Kernel	Structure	MAPE
x264	matern32	additive	0.03
PostgreSQL	spectral mixture	additive	0.01
VP8	poly2	simple	0.17
HSQLDB	RBF	additive	0.02
HSQLDB	spectral mixture	additive	0.04
Apache	RBF	simple	0.02
LLVM	spectral mixture	additive	0.01

Table 5.8: Best model with best MAPE score on average for each dataset.

Dataset	Kernel	Structure	BIC
x264	poly3	simple	-368.06
PostgreSQL	poly4	additive	12655.18
VP8	matern32	simple	20798007.61
HSQLDB	poly3	additive	35912.72
HSQLDB	matern32	additive	1127.65
Apache	matern32	additive	434348.93
LLVM	matern52	additive	22679.96

Table 5.9: Best model with best BIC score on average for each dataset.

5.2.3 Explorative Posterior Analysis (EPA)

To showcase how the posterior structure of the GP can be analyzed, a Jupyter notebook has been written to visualize some key aspects of the MVNs learned, how interactions can be identified, and what insights can be gained using a kernel structure, that a linear model can not discover. While the EPA focuses on one model, the file can just be swapped to further investigate any of the trained models. As the results of a x264 dataset with simple matern52 kernel on 20 samples using MCMC highlights some really beautiful examples of interaction, they are highlighted here.

The notebook is divided into a prestudy section and the explorative analysis of the GP. The prestudies consist of a correlation matrix that can indicate possible multicolinearity among the features, a linear regression using L1 norm and cross-validation from Scikit-Learn to compare the coefficients of this model with the inverse lengthscales of the kernel hyperparameters later, and a comparison of a linear Principal Component Analysis (PCA) with a KernelPCA.

The KernelPCA reflects clearly how the kernel can separate the non-linear behaviour that the unrotated linear components show, when compared across different dimensions:

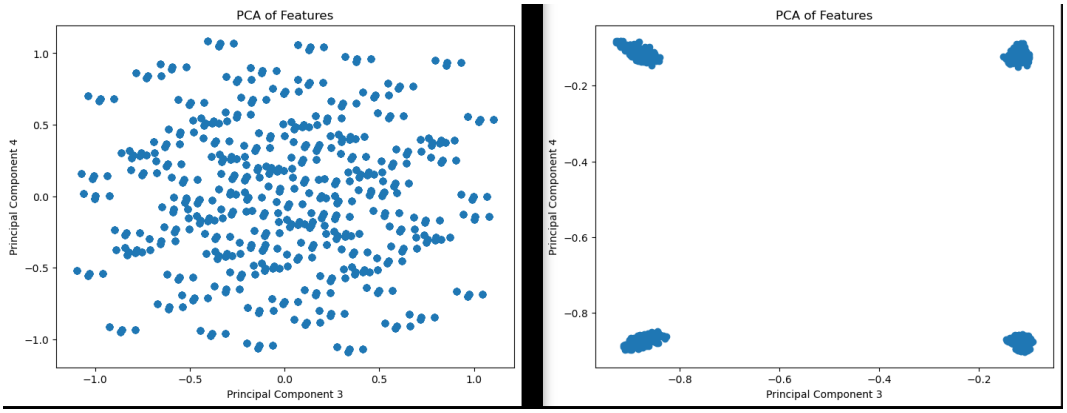


Figure 5.3: linear PCA compared with KernelPCA on pairwise feature dimensions

As the SAASGP provides a batch model, we can plot the mean contour between and covariance matrices for each dimension separately to explore how different features interact hierarchly different:

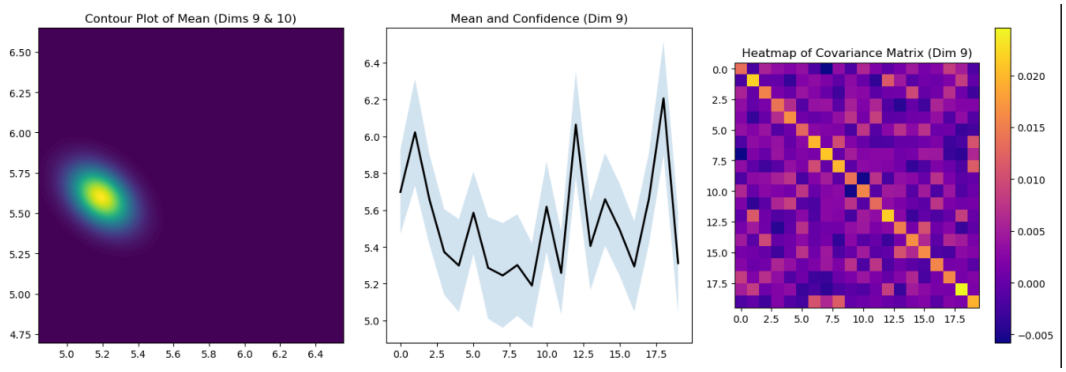


Figure 5.4: Example for a mean contour plot of pairwise dimensions, and one-dimensional mean vector with confidence region together with a heatmap of the covariance matrix.

The contour plot shows a MVN of the 9th and 10th dimension. The bright yellow color indicates high precision for the mean, with rising uncertainty as it diffuses into blue. The confidence region reflects these uncertainties. The covariance matrix shows a yellow diagonal, while some of them are brighter then others, which reflects that each dimension contributes differently to the overall covariance. Off-diagonal the color switches drastically, while some islands can be observed having similar colors as some diagonal elements.

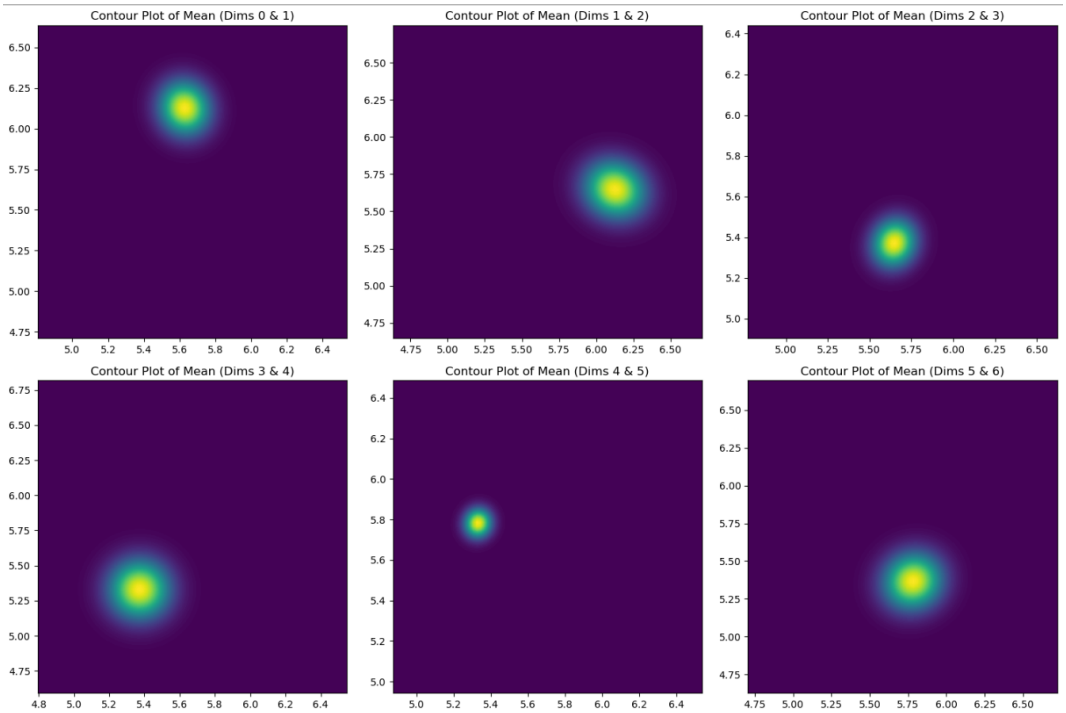


Figure 5.5: pairwise mean contour plots for multiple dimensions

This plot shows how t-wise marginal distributions can be build, each with their own center and covariance contour. Another way to disect the MVN is to plot probability density functions for each dimension separately:

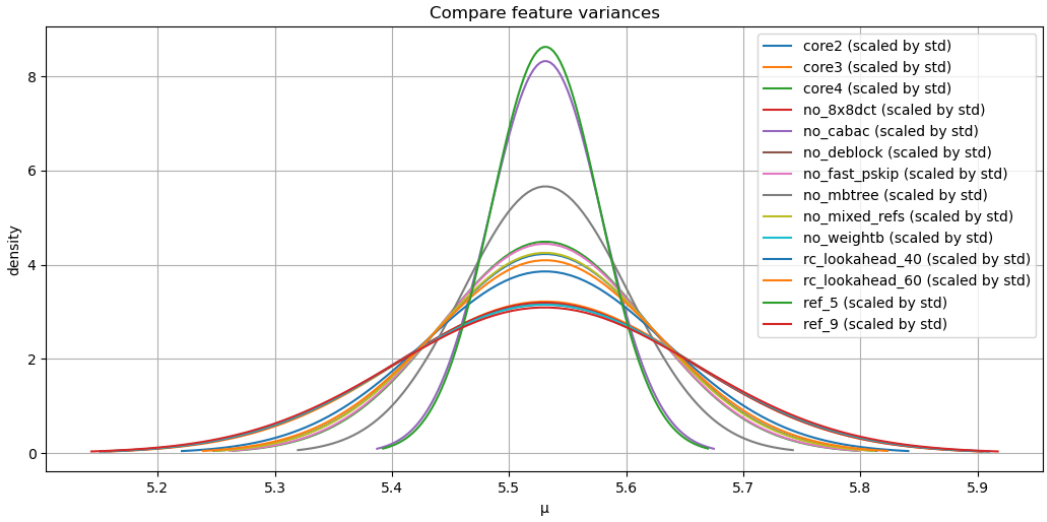


Figure 5.6: PDFs of the x264 model with matern52 kernel

Let's now look at the inverse covariance matrix to indicate conditional dependencies and the decomposition using SVD:

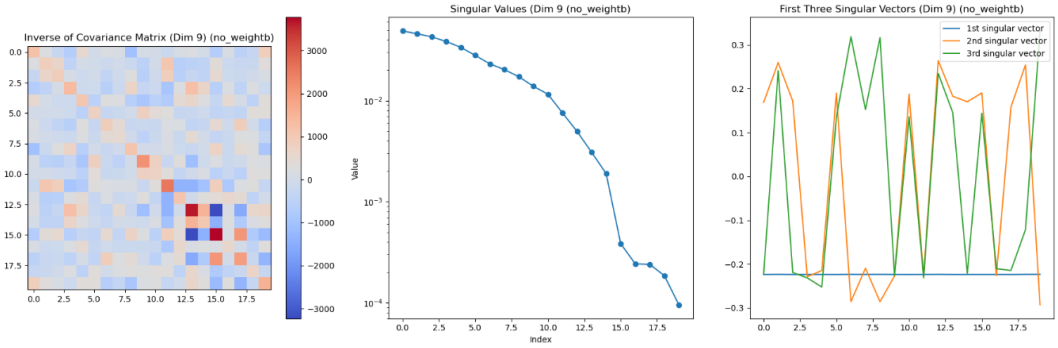


Figure 5.7: precision matrix of single feature dimension, and singular values and the first three singular vectors of the covariance matrix

While most of the entries of the precision matrix are either pale blue or red, indicating weak negative or positive correlation, and thus having almost no conditional dependency (only small change in variance if one feature is opted out or in), the lower right shows some symmetric dark blue points, indicating a strong conditional dependency between these features with inverse influence.

To get a direct association of effect, the low rank matrix of the effect size analog parameters β_{ij} can show on one view which feature dimensions have a synergistic performance or the oppsite effect:

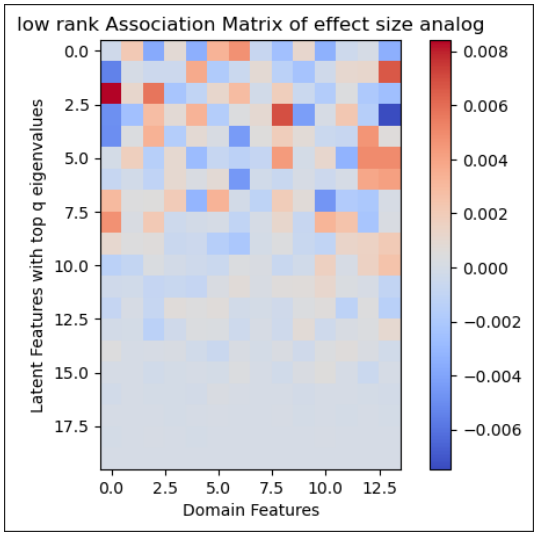


Figure 5.8: low rank Association Matrix of the beta coefficients

As this result is mixed and can point into two directions at different latent dimensions, we use the groupRATE implementation according to 4.5.3, able to use multiple indices $-j$ to subselect likely boolean maps (see 4.2.3) of posterior (predictive) samples to find the interactions that satisfy the user's constraints, and can guide a decision rule for any heuristical aquisition function, for example a compressible tree automata, to rule out co-occurrences that confound the result:

```
# helper functions to build boolean maps to sample interactions
def get_literals_and_interaction(X, features):
    masks = build_masks(X, features)
    literals = get_words(X, masks, clause="xor")
    interactions = get_words(X, masks, clause="and")
    return literals, interactions

def get_opposites_and_interactions(X, features):
    masks = build_masks(X, features)
    opposites = get_words(X, masks, clause="nand")
    interactions = get_words(X, masks, clause="and")
    return opposites, interactions
```

The 'model_inference.py' application therefore gets a list of tuples that take the feature dimension and if they're opted in or out. The grouped subset that applies the boolean function is than used to build the subsequent posterior and calculate the groupRATE. The co-occurrences can be observed looking up the according rows:

```
# partial output of the 'model_inference.py' application
# using SELECTED_FEATURES = [(1,0),(2,1)]
interactions:
 [[0. 0. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0.]
 [0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1.]]
minus_j intersects in X at row: [ 5 15  3  0  8]
groups:
 [[0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1.]
 [0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0.]
 [0. 0. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0.]]
```

```
group rate: tensor([0.2228, 0.1815, 0.1894, 0.2106, 0.1956])
build subsets (can take a while)...
subset shape: (1152, 15)
get subset posteriors...
```

Observe, that we get a RATE distance for each posterior subgroup, which means that it measures marginal distances according to the sample configuration. Finally, the notebook compares posterior mixture distributions with and without the interaction group:

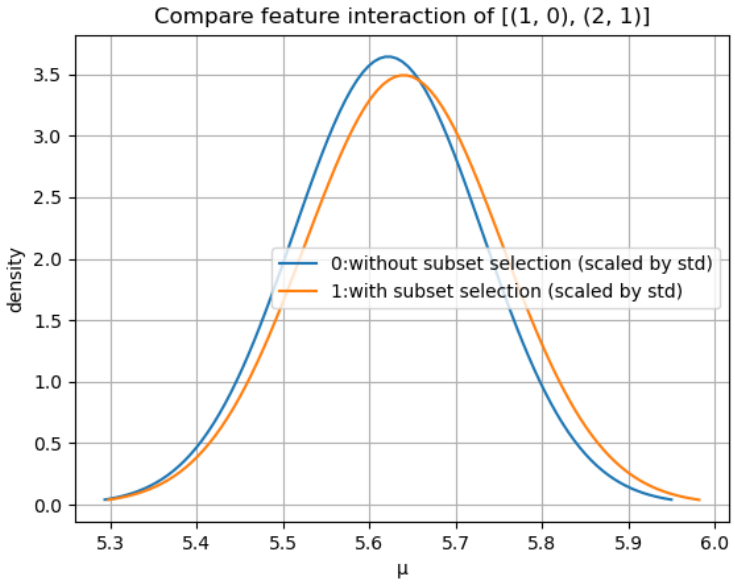


Figure 5.9: Posterior Mixture Distributions with and without subset selection

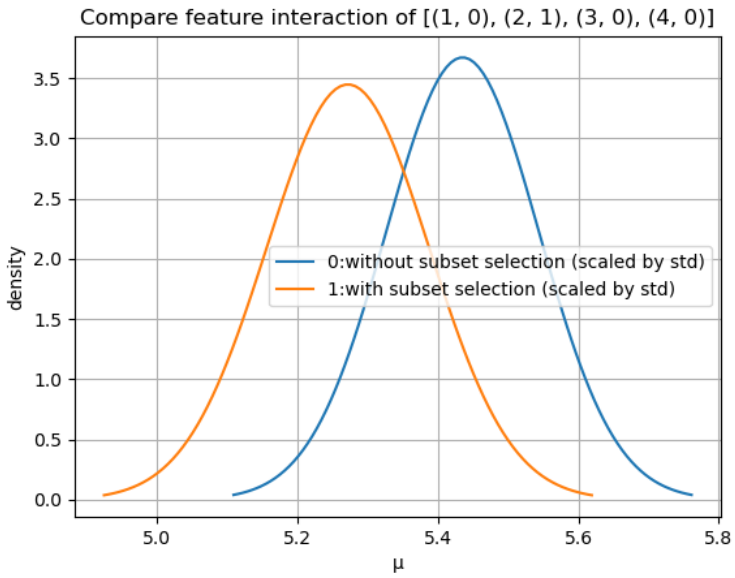


Figure 5.10: Selecting a bigger subset showing bigger divergence.

As there have been issues with plotting the appendix, all supplementary visualizations and results are provided in the GIT repository.

5.3 Discussion

Overall, the results are very promising, that a GP can not only capture non-linear interactions between covariates, but also quantifies the uncertainty to a level of model performance that outcompetes linear regression models and former Bayesian approaches, using a varying coefficient model with only a single coefficient vector. This supports the idea that multivariate statistical models are generally suited better to capture interactions on varying level of granularity, depending on the sample size.

The models from synthetic data clearly show that the prescribed interactions can be estimated exactly with decent precision. In first place, it's quite surprising that the 'poly2' kernel equally good for data with polynomial degree 2 and 3. An explanation because it's probably due to the sparsity inducing hyperpriors. This shows that with an hierarchical approach and the right distributional model one could use a simpler model structure and use it reliably. The conclusion from this is that model structure uncertainty is indeed bounded by the differential kernel operator and can correctly project any distributional model onto the domain while the interpretation depends on the complete model structure. This highlights the flexibility of the GP to 'talk different dialects'

while being able to 'transport the eminent content', which enables the domain expert to choose the hierarchical setup depending on the concrete situation. But validate this hypothesis further experiments are needed to contrast them with results from models without hyperparameter optimization.

Comparing sample efficiency with different scores makes it clear why scoring with MAPE is biased: When increasing the complexity of the synthetic model (see 5.1), the error goes up, while one may expect it to go down, as it's the case for the ESS score. But this just reveals that the complexity hidden in the data grows with sample size and that a certain amount of samples is needed to capture the full latent structure, which is why the MAPE goes up while ESS performs better with growing number of samples.

The results on the various real world SWS verify the internal validation. As the data shows, the models with more samples score better, which is as expected, because higher granularity should capture more nuanced relationships among the features. There is just one major issue with the ESS score of the PostgreSQL data, which is consistently very low and even one time negative, which shouldn't occur. Though the other scores look fine, this should be investigated further soon, because by date there is no plausible explanation for that. It could be an initialization error due to the fact that choosing the training set isn't controlled further and relies solely on the internal procedure of the Scikit-Learn preprocessing methods. That the models show almost perfect scores on some data on a single run, supports the idea to keep the pipeline setup simple to test the performance of the approach handling different scenarios without much fine-tuning. It could be shown that there is even more potential with these type of models if used in an ensemble or with a more advanced iteration procedure and on more complex data.

Speaking of the data, the hypothesis, that software performance (influence) models show rather low dimensional features can be confirmed by interpreting, that having mainly polynomial kernels with degree 2 scoring best, works as a reliable reduction of complexity for most SWS examined. The counter example are the results of the x264 data, which consistently shows best estimations with more complex kernels, namely RFF and matern kernels with additive structure, but even here the EPA shows, that the eigenvalues on the latent feature dimensions fade out fairly quick with high confidence even with a lot of samples, where there is evidence for over 400 latent dimensions but with almost no effect after 100 dimensions, which backs the hypothesis that pair-wise interactions have been found frequently influence performance while the accuracy improves discovering more latent dimensions with more data available.

Reflecting on the average model performance it also payed out to research on generalized additive models and also include the spectral mixture kernel, which makes extrapolation not only possible but also likely, as it shows the

most robust performance across all models on different data sets using different score metrics. It also shows, why matern kernels are a popular choice and that more complex models are likely to score better on average.

Chapter 6

Conclusion

Writing these sentences, a journey comes to an end. Exploring how uncertainty among statistical interactions can be quantified, if they have a noticable influence on performance measures of software systems, led to the discovery of various connections across different subfields of the research process, namely complexity theory, language processing, optimization and measure theory, functional analysis, geometry and multivariate calculus, (Bayesian) statistical inference and system science.

Looking back at what has been accomplished, releases a feeling of satisfaction. Being stuck for a while at the somewhat limited view of statistical interaction as moderated effects, without being aware of the existence of this term, the concept of nuisances in the context of hierarchical models opened a much broader view on the topic. That yielded the understanding of kernel algebra in the context of hyperparameter optimization, which is the key to understand how model structure uncertainty is bounded by the topological sorting of the vertical model structure and the cross sections between a family of probability functions if the population distribution is structured as a stochastic process, which can be estimated by sampling a monte carlo integral of marginal distributions or simply using stochastic gradient descent, all with the axioms of Bayesian inference and the limitation to shift invariant kernels.

The experimental results provide strong evidence that the approach successfully quantifies uncertainty up to a level of performance that it outcompetes linear models and Bayesian approaches restricted to varying coefficient parameters. Thus, the suggested BAKR methodology along with the GPR and the groupRATE distance can give insights on interactions among configuration options in a way that uncertainty can be reduced if local feature evaluations in the posterior distribution are sampled and compared with overlapping subselections.

As the GP's interaction scheme depends on the latent space of 'extra-

functional mediators', that is the represented surrogate, it is determined by the definition of the hierarchical hyperparameters, its posteriors and where smoothness of the covariance matrix is evaluated. We demonstrated that this can be done analytically using gradient descent or with an approximative sample procedure, because it depends on the constraints of evaluation purposes which model structure has to be applied.

The possibility to approximate solutions can help tailor the optimization problem in different stages of fitting. Therefore the model candidates need to be compared among their performance with different parametric complexity and different sample sizes to argue when approximations are sufficient and when it is necessary to dig deeper.

This is the starting point for extended research on the problem that observing possible interactions in higher dimensions is a multidimensional problem itself [1], which is why one has to except a trade-off in risk calculation depending on the user's preferences, whether a risk for choosing an alternative group should be estimated precise or explored in higher dimensions to find a particular order of suggested preferences of configuration options.

To estimate reasonable constraints and verify design decisions for the users, it is required to have preferences between the ability to evaluate higher order interactions (two-way a.s.o.) exactly and just approximative solutions or restrict evaluations to low order interaction only. These tendencies are coupled with the process of serial measurement production, as one can choose for low order on low samples and progress from there. Having only a fixed size of measurements, things are getting easier as one can decide clearly between combinations of richness in interaction and accuracy, where the multi-armed bandit problem can be applied.

The thesis showed why the Bayesian Optimization Algorithm (BOA) in its well established fashion uses GPR as its foundational model structure and how it can solve any second order optimization problem approximately well while keeping the computational cost in polynomial bounds. Concluding, using GPR for inference of model structure uncertainty in performance models for configurable SWS is not only a good choice, but offers access to well-established heuristic problem solving with state of the art machine learning frameworks.

[1] The decisions how to evaluate extensions of probabilities are made *ex ante* [35], when only F is fixed, while P has to be estimated without knowledge about unobserved composition of Ω, e.g. only yet approximately sampled feature combinations.

Appendix A

List of Abbreviations

AI ... Artificial Intelligence

BIC ... Bayesian Information Criterion

BOA ... Bayesian Optimization Algorithm

EPA ... Explorative Posterior Analysis

ESS ... Estimated Sum of Squares

GP ... Gaussian Process

GPR ... Gaussian Process Regression

MAPE ... Mean Absolute Percentage Error

MCMC ... Markov Chain Monte Carlo

ML ... Machine Learning

MVN ... Multivariate Normal Distribution

NLP ... Natural Language Processing

NUTS ... No-U-Turn Sampler

OLS ... Ordinary Least Squares

RBF ... Radial Basis Function

RKHS ... Reproducing Kernel Hilbert Space

RFF ... Random Fourier Features

RMSE ... Root Mean Square Error

SE ... Software Engineering

SPL ... Software Product Line

SWS ... Software System

Appendix B

Gaussian process inference through blackbox matrix multiplication

Algorithm 1 Standard preconditioned conjugate gradients (PCG) [65]

Input: `mvm_A`() – function for matrix-vector multiplication (MVM) with matrix A
        b – vector to solve against
        $P^{-1}()$ – function for preconditioner
Output: $A^{-1}b$

$u_0 \leftarrow 0$ ▷ Current solution
$r_0 \leftarrow \texttt{mvm\_A}(u_0) - b$ ▷ Current error
$z_0 \leftarrow P^{-1}(r_0)$ ▷ Preconditioned error
$d_0 \leftarrow z_0$ ▷ "Search" direction for next solution
for $j \leftarrow 0$ to T **do**
    $v_j \leftarrow \texttt{mvm\_A}(d_{j-1})$
    $\alpha_j \leftarrow (r_{j-1}^T z_{j-1})/(d_{j-1}^T v_j)$
    $u_j \leftarrow u_{j-1} + \alpha_j d_{j-1}$
    $r_j \leftarrow r_{j-1} - \alpha_j v_j$
    if $\|r_j\|_2 <$ tolerance **then return** u_j
        $z_j \leftarrow P^{-1}(r_j)$
        $\beta_j \leftarrow (r_j^T z_j)/(r_{j-1}^T z_{j-1})$
        $d_j \leftarrow z_j + \beta_j d_{j-1}$

        return u_{j+1}

Appendix C

No-U-Turn Sampler

Algorithm 2 Efficient No-U-Turn Sampler [56]

Given $\theta^0, \epsilon, \mathcal{L}, M$
for $m = 1$ to M **do**
  Resample $p^0 \sim \mathcal{N}(0, I)$
  Resample $u \sim \text{Uniform}([0, \exp\{\mathcal{L}(\theta^{m-1}, \frac{1}{2}p^0, p^0)\}])$
  Initialize $\theta^- = \theta^{m-1}, \theta^+ = \theta^{m-1}, r^- = r^0, r^+ = r^0, j = 0, \theta^m = \theta^{m-1}, n = 1, s = 1$
  while $s = 1$ **do**
    Choose a direction $v_j \sim \text{Uniform}(\{-1, 1\})$
    if $v_j = -1$ **then**
      $\theta^-, r^-, -, -, \theta', n', s' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v_j, j, \epsilon)$
    else
      $-, \theta^+, r^+, \theta', n', s' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v_j, j, \epsilon)$
    end if
    if $s' = 1$ **then**
      With probability $\min\{1, \frac{n'}{n}\}$, set $\theta^m \leftarrow \theta'$
    end if
    $n \leftarrow n + n'$
    $s \leftarrow s' I[(\theta^+ - \theta^-) \cdot r^- \geq 0] I[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
    $j \leftarrow j + 1$
  end while
end for
function BUILDTREE$(\theta, r, u, v, j, \epsilon)$
  if $j = 0$ **then**
    Base case – take one leapfrog step in the direction v
    $\theta', r' \leftarrow \text{Leapfrog}(\theta, r, v\epsilon)$
    $n' \leftarrow I[u \leq \exp\{\mathcal{L}(\theta') - \frac{1}{2}r' \cdot r'\}]$
    $s' \leftarrow I[\mathcal{L}(\theta') - \frac{1}{2}r' \cdot r' \geq \log u - \Delta_{\max}]$
    return $\theta', r', \theta', r', \theta', n', s'$
  else
    Recursion – implicitly build the left and right subtrees
    $\theta^-, r^-, \theta^+, r^+, \theta', n', s' \leftarrow \text{BuildTree}(\theta, r, u, v, j - 1, \epsilon)$
    if $s' = 1$ **then**
      if $v = -1$ **then**
        $\theta^-, r^-, -, -, \theta'', n'', s'' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v, j - 1, \epsilon)$
      else
        $-, \theta^+, r^+, \theta'', n'', s'' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v, j - 1, \epsilon)$
      end if
      if $s'' = 1$ **then**
        With probability $\frac{n''}{n' + n''}$, set $\theta' \leftarrow \theta''$
      end if
      $s' \leftarrow s'' I[(\theta^+ - \theta^-) \cdot r^- \geq 0] I[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
      $n' \leftarrow n' + n''$
    end if
    return $\theta^-, r^-, \theta^+, r^+, \theta', n', s'$
  end if
end function=0

Bibliography

[1] Analysis of variance. https://en.wikipedia.org/wiki/Analysis_of_variance. Accessed: 2024-04-28.

[2] Are you a bayesian or a frequentist. https://news.ycombinator.com/item?id=7264131. Accessed: 2024-04-05.

[3] Compound probability distribution. https://en.wikipedia.org/wiki/Compound_probability_distribution. Accessed: 2024-04-26.

[4] Conjugae prior. https://en.wikipedia.org/wiki/Conjugate_prior. Accessed: 2024-04-26.

[5] Covariance matrix. https://en.wikipedia.org/wiki/Covariance_matrix. Accessed: 2024-01-29.

[6] Cox's theorem. https://en.wikipedia.org/wiki/Cox's_theorem. Accessed: 2024-04-08.

[7] Domain of a function. https://en.wikipedia.org/wiki/Domain_of_a_function. Accessed: 2024-02-20.

[8] Eigendecomposition of a matrix. https://en.wikipedia.org/wiki/Eigendecomposition_of_a_matrix. Accessed: 2024-01-29.

[9] Exchangability. https://en.wikipedia.org/wiki/Bayesian_hierarchical_modeling#Exchangeability. Accessed: 2024-04-28.

[10] Hilbert space. https://en.wikipedia.org/wiki/Hilbert_space. Accessed: 2024-02-01.

[11] Homotopy. https://en.wikipedia.org/wiki/Homotopy. Accessed: 2024-04-26.

[12] Illustration of a multivariate gaussian distribution and its marginals. https://en.wikipedia.org/wiki/Multivariate_normal_

distribution#/media/File:MultivariateNormal.png. Accessed: 2024-06-17.

[13] Lambda-calculus. https://en.wikipedia.org/wiki/Lambda_calculus. Accessed: 2024-02-24.

[14] Law of total expectation. https://en.wikipedia.org/wiki/Law_of_total_expectation. Accessed: 2024-04-26.

[15] Mediation (statistics). https://en.wikipedia.org/wiki/Mediation_(statistics). Accessed: 2024-05-07.

[16] Moderation (statistics). https://en.wikipedia.org/wiki/Moderation_(statistics). Accessed: 2024-05-07.

[17] Multivariate normal distribution. https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Definitions. Accessed: 2024-02-19.

[18] Principle of maximum entropy. https://en.wikipedia.org/wiki/Principle_of_maximum_entropy. Accessed: 2024-02-19.

[19] Reference priors. https://en.wikipedia.org/wiki/Prior_probability#Uninformative_priors. Accessed: 2024-04-05.

[20] Resource description framework (rdf). https://www.w3.org/RDF/. Accessed: 2024-06-10.

[21] Scale invariance. https://en.wikipedia.org/wiki/Scale_invariance. Accessed: 2024-04-28.

[22] Spatial cross covariance analysis. https://petrowiki.spe.org/Spatial_statistics#Spatial_cross_covariance_analysis. Accessed: 2024-05-22.

[23] Spektralzerlegung (mathematik). https://de.wikipedia.org/wiki/Spektralzerlegung_(Mathematik). Accessed: 2024-01-29.

[24] Structural equation model. https://en.wikipedia.org/wiki/Structural_equation_modeling. Accessed: 2022-02-21.

[25] System science. https://en.wikipedia.org/wiki/Systems_science. Accessed: 2024-02-19.

[26] Jörg Matthes Andrew F. Hayes. Computational procedures for probing interactions in ols and logistic regression: Spss and sas implementations. *Behaviour Research Methods 2009, 41 (3), 924-936*, 2009.

[27] Hal S. Stern Donald B. Rubin Andrew Galman, John B Carlin. *Bayesian Data Analysis*. Chapman Hall/CRC, 2003.

[28] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer-Verlag Berlin Heidelberg, 2013.

[29] Aristoteles. *Die Kategorien*. Reclam, 2005.

[30] Alexandre Muzy Bernard P. Zeigler and Ernesto Kofman. *Theory of Modeling and Simulation*. Elsevier Inc., 2019.

[31] Salomon Bochner. *Harmonic Analysis and the Theory of Probability*. University of California Press, 1955.

[32] C. K. I. Williams C. E. Rasmussen. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[33] M. Harman C. Henard, M. Papadakis and Y. Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. Proc. Int. Conf. Software Engineering, 2015.

[34] N. Siegmund J. Guo C. Kaltenecker, A. Grebhahn and S. Apel. Distance-based sampling of software configuration spaces. Proc. Int. Conf. Software Engineering, 2019.

[35] Michelle Caprio and Sayan Mukherjee. Extended probabilites and their application to statistical inference. `http://arxiv.org/abs/2111.01050v3`.

[36] George J. Pappas Charis Stamouli, Evangelos Chatzipantazis. Structural risk minimization for learning nonlinear dynamics. *arXiv:2309.16527v1 [eess.SY]*, 2023.

[37] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-Based Software Performance Analysis*. Springer-Verlag Berlin Heidelberg, 2011.

[38] Wood K. C. Zhou X. Mukherjee S Crawford, L. Bayesian approximate kernel regression with variable selection. *ournal of the American Statistical Association, 113(524), 1710–1721*, 2018.

[39] Carl de Boor. *Spline Basics*.

[40] Alexander Graeme de Garis Matthes. *Scalable Gaussian process inference using variational methods*. PhD thesis, University of Cambridge, 2016.

[41] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. International Conference on Learning Representations, 2015.

[42] Niklas Kühl Dominik Kreuzberger and Sebastian Hirschl. Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access*, 2023.

[43] David Kristjanson Duvenaud. *Automatic Model Construction with Gaussian Processes*. PhD thesis, University of Cambridge, 2014.

[44] K.-K. Ma J. S. Foster A. Porter E. Reisner, C. Song. Using symbolic evaluation to understand behavior in configurable software systems. Proc. Int. Conf. Software Engineering (ICSE), ACM, 2010.

[45] Kuldeep S. Meel Eduard Baranov, Axel Legay. Baital: An adaptive weighted sampling approach for improved t-wise coverage. 2020.

[46] David Eriksson and Martin Jankowiak. High-dimensional bayesian optimization with sparse axis-aligned subspaces. 2021.

[47] Performance-influence models for highly configurable systems. August 30 - September 4, 2015.

[48] Marco Gribaudo Riccardo Pinciroli Kishor S. Trivedi Catia Trubiani Fabio Antonelli, Vittorio Cortellessa. Analytical modeling of performance indices under epistemic uncertainty applied to cloud computing systems. *Future Generation Computer Systems 102 (2020) 746-761*, 2020.

[49] Yoav Goldberg and Michael Elhadad. splitsvm: Fast, space-efficient, non-heuristic, polynomial kernel computation for nlp applications. Association for Computational Linguistics, 2008.

[50] Robert B. Gramacy. *Surrogates - Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. CRC Press Taylor Francis Group Boda Raton London New York. A Chapman 6 Hall Book, 2020.

[51] Robert M. Gray. *Toeplitz and Circulant Matrices: A Review*, volume 2 of *Foundations and Trends in Communications and Information Theory*. Now Publishers Inc, Boston-Delft, 2006. Comprehensive review of Toeplitz matrices and their applications.

[52] Jianmei Guo, Krzysztof Czanercki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. Variability-aware performance prediction: A statistical learning approach. 2013.

[53] M. Carbin S. Misailovic A. Agarwal M. Rinard H. Hoffmann, S. Sidiroglou. Dynamic knobs for responsive power-aware computing. Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems, ACM, 2011.

[54] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems. Queuing Theory in Action.* Cambridge University Press, 2013.

[55] J.D. Hintersteiner and A. Nain. Integrating software into systems: An axiomatic design approach. 1999.

[56] Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research 15 (2014) 1351-1381*, 2014.

[57] Hongyu Zhang Huong Ha. Deepperf: Performance prediction for configurable software with deep sparse neural network. 41st International Conference on Software Engineering, IEEE/ACM, 2019.

[58] Hongyu Zhang Huong Ha, Zongwen Fan. Uncertainty-aware performance prediction for highly configurable software systems via bayesian neural networks. *arXiv:2212.13359v1 [cs.SE]*, 2022.

[59] Evaluation of user experience in software product lines derivation process. 2023.

[60] N. Siegmund. J. Dorn, S. Apel. Mastering uncertainty in performance estimations of configurable software system. September 2020.

[61] S. Elflein J. V. Kistowski S. Kounev J. Grohmann, S. Eismann and M. Mazkatli. Detecting parametric dependencies for performance models using feature selection techniques. In *Proc. Int. Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 309–322, 2019.

[62] C. Kästner T. Thüm G. Saake J. Meinicke, C.-P. Wong. On essential configuration complexity: Measuring interactions in highly- configurable systems. Proc. Int. Conf. Automated Software Engineering, ACM, 2016.

[63] M. Myers J. Oh, D. Batory and N. Siegmund. Finding near-optimal configurations in product lines by random sampling. Proc. Europ. Software Engineering Conference and ACM SIGSOFT Symp. Foundations of Software Engineering, 2017.

[64] K. V. Shenoy J. P. Cunningham and M. Sahani. Fast gaussian process methods for point process intensity estimation. 2008.

[65] David Bindel Kilian Q. Weinberger Andrew Gordon Wilson Jacob R. Gardner, Geoff Pleiss. Gpytorch: Blackbox matrix-matrix gaussian process inference with gp acceleration. 2018.

[66] E.T. Janes. Information theory and statistical mechanics. *Physical Review. Series II. 106 (4): 620-630*, 1957.

[67] Paul Gazzillo Jeho Oh and Don Batory. t-wise coverage by uniform sampling. Association for Computing Machinery, 2019.

[68] Robert Tibshirani Jerome Friedman, Trevor Hastie. *The Element of Statistical Learning - Data Mining, Inference and Prediction*. September 2008.

[69] Norbert Siegmund Sven Apel Atrisha Sakar Pavel Valov Krzysztof Czarnecki Andrzej Wasowski Huiqun Yu Jianmei Guo, Dingyu Yang. Data-efficient performance learning for configurable systems. *Empir Software Eng*, (23):1826–1867, 2017.

[70] Adrian F. M. Smith José M. Bernardo. *Bayesian Theory*. John Wiley Sons, LTD, 2000.....

[71] Armen Der Kiureghian. Aleatory or epistemic? does it matter?, 2007.

[72] Wulff Plinke Rolf Weiber Klaus Backhaus, Bernd Erichson. *Multivariate Analysemethoden*. Springer Gabler, 2018.

[73] Daniel E. Runcie Lorin Crawfor, Seth R. Flaxman and Mike West. Variable prioritization in nonlinear black box methods: A genetic association case study. *The Annals of Applied Statistics, 2019, Vol. 13, No. 2, 958-989*, 2019.

[74] Stefan Lang Ludwig Fahrmeir, Thomas Kneib. *Regression - Modelle, Methoden und Anwendungen*. Springer, 2007.

[75] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2005.

[76] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, Boston, MA, 2017.

[77] Daniel R. Jiang Samuel Daulton Benjamin Letham Andrew Gordon Wilson Maximilian Balandat, Brian Karrer and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *arXiv:1910.06403v3 [cs.LG]*, 2020.

[78] Martin Middendorf, 2019. Lecture Notes on 'Diskrete Simulation'.

[79] Sayan Mukherjee. `http://www2.stat.duke.edu/~sayan/561/2015/stat_ml.pdf`. Accessed: 2024-01-29.

[80] Kevin P. Murphy. *Machine Learning - A Probabilistic Perspective*. MIT Press, 2012.

[81] Sven Apel Christian Kästner Norbert Siegmund, Alexander Grebhahn. Performance-influence models for highly configurable systems, 2015.

[82] Judea Pearl. Causality. *Cambridge Press University*, 2009.

[83] Quarianto, Song, and Smola. Kernel sorting. 2017.

[84] A. Rahimi and B. Recht. Random features for large-scale kernel machines. Neural Information Processing Systems, 2007.

[85] Carl Edward Rasmussen. *Evaluation of Gaussian Processes and other methods for non-linear regression*. PhD thesis, University of Toronto, 1996.

[86] Yunus Saatci. *Scalable Inference for Structured Gaussian Process Models*. PhD thesis, University of Cambridge, 2011.

[87] Daniel F. Schmidt and Enes Makalic. Adaptive bayesian shrinkage estimation using log-scale shrinkage priors. *arXiv:1801.02321v1 [math.ST]*, 2018.

[88] Sandro Schulze Sebastian Ruland Malte Lochau Gunter Saake Sebastian Krieter, Thomas Thüm and Thomas Leich. T-wise presence condition coverage and sampling for configurable systems. *arXiv:2205.15180v1 [cs.SE]*, 2022.

[89] Norbert Siegmund, Johannes Dorn, Max Weber, Christian Kaltenecker, and Sven Apel. Green configuration: Can ai help reduce energy consumption of configurable software systems? *IEEE Transactions on Software Engineering*, 2020.

[90] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting performance via automated feature-interaction detection. *International Conference on Software Engineering (ICSE)*, 2012.

[91] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Qual J*, 2012.

[92] Kalaitzis A. Silvia, R. Bayesian inference via projections. *Stat Comput 25, p.739-753*, 2015.

[93] Matthew Stephens. False discovery rates: a new deal. *Biostatistics, Volume 18, Issue 2, April 2017, Pages 275-294*, 2017.

[94] G. Franks M. Woodside T. A. Israr, D. H. Lau. Automatic gener- ation of layered queuing software performance models from commonly available traces. Proc. Int. Workshop Software and Performance, 2005.

[95] On Debugging the Performance of Configurable Software Systems: Developer Needs and Tailored Support. On debugging the performance of configurable software systems: Developer needs and tailored tool support. In *Proceedings of the International Conference on Software Engineering*. IEEE, 2022.

[96] S. Eismann S. Kounev V. Ackermann, J. Grohmann. Blackbox learning of parametric dependencies for performance models. MOD-ELS Workshop, 2018.

[97] A. Ya. Chervonenkis V. N. Vapnik. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability Its Applications. 16 (2): 264*, 1968.

[98] Vladimir N. Vapnik. An overview of statistical learning theory. *IEEE Transactions On Neural Networks, Vol. 10, No. 5*, September 1999.

[99] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. White-box analysis over machine learning: Modeling performance of configurable systems. In *Proceedings of the International Conference on Software Engineering*. IEEE, 2021.

[100] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. On debugging the performance of configurable software systems: Developers needs and tailored tool support. 2022.

[101] Max Weber, Sven Apel, and Norbert Siegmund. White-box performance-influence models: A profiling and learning approach. In *Proceedings of the International Conference on Software Engineering*. IEEE, 2021.

[102] Max Weber, Christian Kaltenecker, Florian Sattler, Sven Apel, and Norbert Siegmund. Is performance a reliable proxy for energy consumption? 2024.

[103] Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process kernels for pattern discovery and extrapolation. *arXiv:1302.4245v3 [stat.ML]*, 2013.

[104] Andrew Gordon Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *Proceedings of the* $32^{nd} International Conference on Machine Learning, Lille, France.$, 2015.

[105] H. Witting. *Mathematische Statistik*. Teubner Studienbücher, 1966.

[106] Simon N. Wood. *Generalized Additive Models*. Taylor Francis Group, LLC, 2017.

[107] Using regression splines for software performance analysis. Proc. Int. Workshop Software and Performance, 2000.

[108] Jing Xu. Rule-based automatic software performance diagnosis and improvement. ACM, 2008.

[109] Yosef Hochberg Yoav Benjamini. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Royal Statistical Society, Volume 57, Issue 1, 1995, Pages 289-300*, 1995.

[110] Jiaxin Zhanh. Modern monte carlo methods for efficient uncertainty quantification and propagation: A survey. *arXiv: 2011.00680v1 [stat.ME]*, 2020.