Advanced Simulation and Machine Learning

Paul Erhart, Andreas Ekström, Arkady Gonoskov

Dec 17, 2024

CONTENTS

These are the lecture notes for the course TIF345/FYM345 Advanced Simulation and Machine Learning. The accompanying jupyter notebooks can be found in this gitlab repository.

Tip

You can also download a PDF version of these lectures notes.

Python packages

Throughout the course and these lecture notes we will use several Python packages, including, e.g.,

- `pymc`: Probabilistic Programming in Python
- `emcee`: Affine Invariant Markov chain Monte Carlo (MCMC) Ensemble sampler
- `corner`: For generating "corner" plots
- `GPy`: A Gaussian Process (GP) framework in Python
- `scikit-learn`: an extensive library for machine learning algorithms
- `scipy`: Fundamental algorithms for scientific computing in Python

Part I

Main

CHAPTER

ONE

FUNDAMENTALS

Key take-away messages

- We cannot expect a theory to be complete in the sense that it will describe and predict all phenomena. In that sense, a theory always comes with some probability for being false and this probability should never be exactly 0 or 1. (This statement should be handled with care, and we will elaborate this during the lectures and in the subsequent chapters.)
- Models are to a theory what tactics are to a strategy.
- Point or interval estimates, without acknowledging that models (like theories) are incomplete, i.e., there is a so-called model discrepancy, often lead to overconfident inference and prediction.

Discussion

Reflect on the quotes below. To which extent do you agree?

> The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work - that is, to correctly describe phenomena from a reasonably wide area. Furthermore, it must satisfy certain esthetic criteria - that is, in relation to how much it describes, it must be rather simple. - John von Neumann
>
> All models are wrong, but some are useful. - George E. P. Box
>
> A model that is wrong can only be useful if we acknowledge the fact that it is wrong. - J. Bryn-jarsdóttir & A. OHagan
>
> A first-rate theory predicts; A second-rate theory forbids; A third-rate theory explains after the event. - A. I. Kitaĭgorodskii
>
> Theories provide a unified framework, categorization, and the joint language used for discussions, and models can be used to study aspects of a theory, increase understanding, and provide intuition. - S. Hartmann

1.1 The scientific method

The use of probability theory to quantify uncertainty plays a central role in science and the scientific method for generating new knowledge about the universe. The hope is to learn more about the Universe we live in. Before we can elaborate on this topic we must briefly discuss the anatomy of science in terms of data, theories, and models.

1.1.1 Data

Let us start with the *data* obtained through a measurement process, e.g., an experiment in a laboratory or an observation of some astronomical event. It is obviously so that all data are equipped with uncertainties of various origin. Surely you can think of several examples. Given some data $\mathcal{D}$, one would immediately ask what this data can tell us about future data $\mathcal{F}$. This is a central question that we often seek an answer to. Here, future data refers to any data that we have not included in our past inference. It can be data that exists, but we neglected, it can be data that does not exist and/or is impossible to find, or data that we expect to take at a later stage. Regardless, it is uncertain for sure, and can described as a conditional probability $p(\mathcal{F}|\mathcal{D})$. The question is: How does one go from the rather abstract probability $p(\mathcal{F}|\mathcal{D})$ to something that we can more easily interpret?

The answer, of course, is to develop a *theory* T.

1.1.2 Theories

The development of a theory requires imagination, the discovery of patterns and/or symmetries, disruptive ideas, a scoop of luck, and possibly some collaboration. In essence, a theory allows us to state the following

$$p(\mathcal{F}|T\mathcal{D}) = p(\mathcal{F}|T).$$

Although it is on a very high-level, this equality tells us something important about the role of a theory in science. That is, given a theory, it is possible to forget about all previous data. This knowledge is now encapsulated by the theory itself. For example, based on the numerous experiments where one observed falling bodies and rolling balls, it is now possible to employ mathematics and computers to make a quantitative (probabilistic) statement regarding the bending of a metal beam or the tides around the globe; all based on, e.g., Newton's theories. In physics, a theory is very often some framework that postulates, or deduces from some axioms, a master equation that governs the spacetime dependence of a system of interacting bodies, e.g., Einstein's field equations in the general theory of relativity or Heisenberg's equations of motion in quantum mechanics.

A theory always comes with some probability, and, besides for purely logical statements, this probability should never be exactly 0 or 1. In such cases, no evidence/data will ever have any influence[1]. As such, *all theories are wrong* in the sense that they are not correct with absolute certainty. This is of course a provocative statement that is designed to draw attention to the fact that all theories can be improved or replaced, and we do this all the time using the scientific method. In physics we are particularly aware that theories often have a specific (energy) domain of applicability. Indeed, most theories in physics can be interpreted as effective (field) theories. Such theories are designed for describing physics in a specific energy-domain without having to worry about what goes on at, e.g., very high energies. This construct enables us to obtain meaningful descriptions for, e.g., the hydrogen atom without having to worry about the details of the bottom-quark. There is a vast amount of literature and plenty of diverging opinion on all of the above. For more about the fruitfulness of having a tower of effective (field) theories, i.e., linking descriptions across several domains, open questions about reductionism and emergence, and philosophical speculations about a fundamental theory of everything see the recent paper by [1].

[1] This is sometimes called Cromwell's rule.

> Leave a little probability for the moon being made of green cheese; it can be as small as 1 in a million, but have it there since otherwise an army of astronauts returning with samples of the said cheese will leave you unmoved.
>
> – Dennis Lindley (statistician)

As you know from previous courses, we can use use Bayes' theorem to evaluate and compare the probability of two theories; the theory T and its complement $\bar{T}$

$$\frac{p(T|\mathcal{F},I)}{p(\bar{T}|\mathcal{F},I)} = \frac{p(T|I)}{p(\bar{T}|I)} \cdot \frac{p(\mathcal{F}|T,I)}{p(\mathcal{F}|\bar{T},I)}.$$

That is, we update our probability ratio for the theory, sometimes referred to as the odds for the theory, given some future data $\mathcal{F}$ and our background knowledge I, sometimes called knowledge base. The priors, and their probability ratio also depends on our background knowledge. Now, if the likelihood ratio is greater than one, i.e., that the future data is highly probable given T but not $\bar{T}$, then we say that the probability for T increases. Consequently, we have more evidence in support for the theory T. The difficult part is to define the priors and the complement $\bar{T}$ of a theory, and this might drastically change the probability ratio on the left-hand side.

Even if a physical theory is mathematically well defined, its complement might not be. It becomes elusive, if not impossible, to understand the probability for future data $\mathcal{F}$ if the theory T is false. For example, what is the complement of Newton's theory of gravity? Certainly not Einstein's theory since the former is obtained as a low-mass limit of the latter and that the two together would not constitute every possible theory for gravity.

To make inferences quantitative, we need *models*. They are to theories what tactics are to strategy. The tactics (model) apply to a particular situation whereas the strategy (theory) describes the overall method or approach. We always try to avoid tactics without strategy like we always prefer to have a theory to explain or motivate our models. However, sometimes we have to build models without the support of a fundamental theory. It might lead us forward or be sufficient for the task at hand, but the satisfaction of understanding or explaining the patterns in the data only comes with the general explanation of how things work for a class of phenomena.

1.1.3 Models

To understand experimental data on, e.g., the discrete excitation energy of an atomic nucleus, we can adopt various models in harmony with quantum theory and the Schrödinger equation. Each model M will depend on a set of parameters θ. Based on the same future data $\mathcal{F}$ we can now begin to compare several models with each other and quantitatively express the uncertainties of the models as well as their parameters. In the long run, this also tells us something about the probability of the underlying theory T, i.e., its explanatory and predictive power. To that end, in this course you will study various types of models and learn about a few statistical and machine learning methods to infer patters and probabilities.

We refer to a computational implementation and evaluation of a model $M(\theta)$ as a simulation. The simulation generates data and represents the physical system as described by the model in accordance with the theory. We can use this setup for many things. Either employ a simulation to virtually represent selected aspects of a physical system in a domain that cannot be reached by experiments, e.g., for extraterrestrial conditions, and/or compare the simulation data with experimental data to do hypothesis testing, model comparison, model checking, experimental design and in the end learn about the limits of the model and uncertainty of the theory. Multiple opportunities emerge to help us in the process of refining and developing theories.

This interplay between theories and models, via simulation and experimental data, constitutes the backbone of the scientific method. The progress is often incremental, but nevertheless, there is progress and we continuously generate new knowledge that we use to improve our theories, define new models, and perform new experiments.

In this course we will study the simulation of physical systems as described by some theory and use machine learning methods to evaluate and compare models.

1.2 Machine learning

Machine learning (*ML*) can play an important role in the development of models and the analysis of data. Indeed, in the era of abundant and complex data in science as well as industry, from, e.g., particle accelerators, telescopes, and sensors mounted on self-driving cars, data patterns might only be recognizable through the use of *ML*.

In science, compared to, e.g., industry, there often exists a fundamental difference in regards to the desired foundation of a model. As mentioned above, physicists seek an underlying theory (strategy) to explain and *predict* future data. *ML* mostly does the exact opposite. A model derived using *ML* methods will be agnostic at best, but mostly opaque, when it comes to *explaining or understanding* of the data. With the *ML* approach we extract nearly all of the "intelligence" from data at hand. It remains a challenge for, e.g., supervised learning methods to replace a theory. Indeed, to successfully generalize beyond some training data is limited by the bias-variance effect. Nevertheless, the *ML* approach is powerful and there exist several cases where the results are excellent. Refs. [2] and [3] provide a more general introduction to *ML* in the physical sciences. *ML* is a vast topic, and in this course you will encounter several different methods such as Gaussian processes, neural networks, and advanced optimization algorithms to generate/analyze/emulate/improve models of physical systems.

1.3 Simulating reality

In this chapter we will quantify the link between simulations and measured data of some real process. Before moving on, let us mention, as a curious side note, one can of course ask the question, i.e., whether reality *itself* is a simulation [4]. The observable consequences of this are explored further in [5].

Assume that we have N_d observations $\{d_i\}_{i=1}^{N_d}$ of some physical system, i.e., measurement data, gathered in a dataset $\mathcal{D}$. Let us further denote control variables with x. They define the particular instances of the physical systems, e.g., a temperatures, locations or some kinematic settings such as scattering energies and angles. Each observation, d_i, is accompanied with some measurement error, e.g.,

$$d_i = \zeta(x_i) + \varepsilon_i, \tag{1}$$

where $\zeta(x_i)$ denotes the *true* or *real* value at the control variable x_i and ε_i is a measurement error. In general, we will try link our model $M(\theta; x)$ to reality via

$$\zeta(x) = M(\theta_T; x) + \delta(x), \tag{2}$$

where $\delta(x)$ is the model discrepancy and θ_T is the true but unknown value of the model parameter. Here, we assume that the model parameter has some theoretical meaning that transcends the particular model we are working with, has scientific value within the theory underlying the model, and is important for extrapolation. The model discrepancy term is defined as the difference between the simulator output, i.e., the computer evaluated model, and reality. This terms originates from the fact that our models always neglect one or more known physical effects and most likely one or more unknown physical effects. We combine these results to obtain a link between observations and the model, namely

$$d_i = M(\theta_T; x_i) + \varepsilon_i + \delta(x_i). \tag{3}$$

In the process of evaluating the model via some computer realization, i.e., a simulator, we will introduce additional error terms due to finite precision in the representation of numbers and method errors due to the choice of algorithm employed to solve the mathematical equations associated with the model. It requires domain specific knowledge to assess which error will dominate and how they relate to the model discrepancy. In this chapter we will assume that the experimental error and the model discrepancy in particular are the dominating sources of uncertainty, i.e., that we have negligible numerical errors.

A priori, we do not know the true value θ_T. As we will see below, it is important to incorporate $\delta(x)$ already during the calibration stage, i.e., when we try to learn reasonable values for θ from the dataset $\mathcal{D}$. Ignoring $\delta(x)$ will lead to biased and/or overconfident parameter estimates and predictions. In general, the assumption of having *zero* uncertainty is

a rather extreme position to take. *Any* finite uncertainty estimate is far more realistic. Although it might be prohibitively complex to incorporate finite and non-zero uncertainties everywhere in computer simulations it is important to be aware their existence.

So, we have a situation where we are uncertain about model parameter θ and the model discrepancy δ, but we have some background knowledge, or ideas/assumptions, about the system we are studying, we are physicists after all. To make the most of our inferences of future data we should adopt a Bayesian approach.

1.4 The simple machine

In theory we have the luxury of constructing a reality from scratch. For the purpose of explaining the concept and importance of a model discrepancy term in this chapter we will generate some artificiak data from a hypothetical process. We follow closely the work presented in [6]. We imagine a reality where we have a process called *the simple machine*. It returns work $\zeta(x)$ given some effort x according to

$$\zeta(x) = \frac{\theta_T x}{1 + 0.05x}.$$

Here, θ_T is an intrinsic physical property or parameter of our machine that is of interest to the machine designers or anyone studying the machine itself. In this example, we will set θ_T to some value of our liking, then forget about it and pretend that we do not know anything about the underlying reality of the simple machine. This allows us to test how well different procedures for extracting θ_T work.

1.4.1 Generating data

We will generate some data by constructing a (theoretical) measurement process and subsequently try our best to infer a probability distribution for the possible values of θ_T.

Learn more and get active

Using the Jupyter Notebook `simple_machine.ipynb` provided in the course repository you can reproduce all the figures in this chapter. Please have a look and try yourself.

We define the overall precision of the measurement process by assuming independently and identically distributed (*i.i.d.*) measurement errors ε_i that all follow a normal distribution with zero mean and variance σ_e^2. In a commonly used notation in statistics, we write $\varepsilon_i \sim \mathcal{N}(0, \sigma_e^2)$.

The fact that the variance of the error term is constant with respect to variation in x is called homoscedasticity, i.e., the errors are *homoscedastic*. The opposite, i.e., non-constant variance in the error term is called heteroscedasticity (try to say that fast three times in a row). We will focus on *homoscedastic* errors, and treat σ_e as an unknown.

We make a measurement of the output d_i, given some input x_i, by evaluating the following expression

$$d_i = \zeta(x_i) + \varepsilon_i.$$

In Figure 1 we plot a dataset $\mathcal{D}$ with $N_d = 10$ measurements at equidistant $x \in [0.2, 4.0]$ and $\sigma_e = 0.02$, similar to the analysis presented in [6].

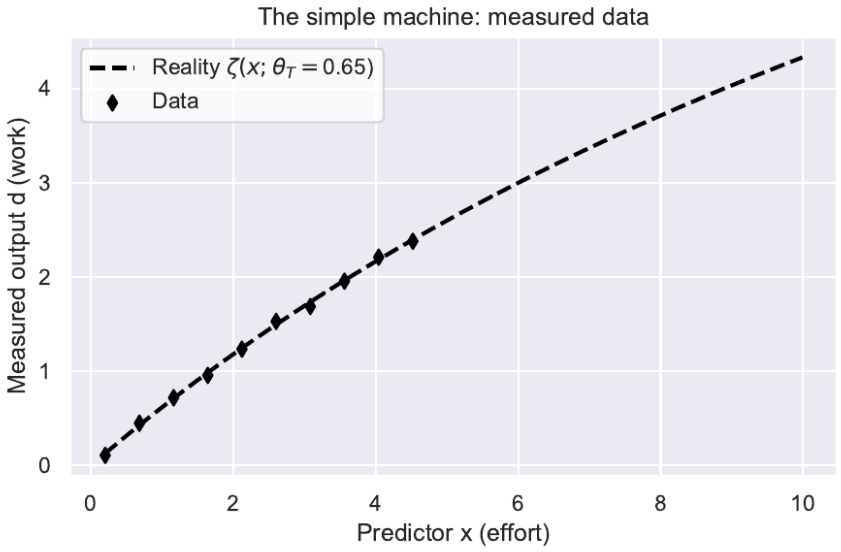


Fig. 1: $N_d = 10$ measurements of the work produced by the simple machine.

1.4.2 Our model and GLMs

In this example we will operate with an embarrassingly simple (linear) model

$$M(\theta; x) = \theta x.$$

We can still motivate the model from what we know about classical mechanics and the laws of motion. Still, we have neglected the effects of friction, which is a clear example of a model discrepancy $\delta(x)$. There might be several reasons for having this and other model discrepancies. It can be difficult to model friction, e.g., require complex numerical operations outside our budget (time-wise or money-wise), or we might be unaware that friction exists(!). Nevertheless, we believe enough in our model to use it for estimating the true physical parameter θ_T inherent to the design of the machine.

Our model is linear in θ and a trivial case of a general linear model (*GLM*). A *GLM* with N_p parameters $\theta = [\theta_0, \theta_1, \theta_2, \dots, \theta_{N_p-1}]^T$ that is evaluated N_d times, i.e., at N_d values for the control variables $[x_0, x_1, \dots, x_{N_{d-1}}]^T$, is linear if we can write each model output i

$$M_i(\theta) \equiv M(\theta, x_i) = \sum_{j=0}^{N_p-1} \theta_j f_j(x_i),$$

where $f_i(x)$ are called basis functions. We can write this in matrix form

$$\mathbf{M} = \Phi\theta,$$

where

$$\mathbf{M} = \begin{bmatrix} M(\theta, x_0) \\ M(\theta, x_1) \\ \vdots \\ M(\theta, x_{N_d-1}) \end{bmatrix}, \qquad \Phi = \begin{bmatrix} f_0(x_0) & \dots & f_{N_p-1}(x_0) \\ f_0(x_1) & \dots & f_{N_p-1}(x_1) \\ \vdots & \ddots & \vdots \\ f_0(x_{N_d-1}) & \dots & f_{N_p-1}(x_{N_d-1}) \end{bmatrix}$$

For polynomial basis functions $f_j(x) \equiv x^j$ we recover the linear model

$$M(\theta, x) = \sum_{j=0}^{N_p-1} \theta_j x^j$$

In *ML* terminology we refer to the independent variables θ_i as weights, the sample or control variable x_i as features, and $\theta_0 \equiv 1$ as the bias.

In our case, the design matrix is given by the vector of predictors, i.e., $\Phi = x$. You encountered *GLM*s already during the course TIF285 *Learning From Data*. In many cases such models allow us to derive analytical results, which is exactly what we will do first.

Using Eqs. (1)-(2) and ignoring the model discrepancy for now, we can relate the data to our model via

$$\mathcal{D} = \Phi\theta + \varepsilon,$$

where Φ denotes the design matrix, θ a vector of parameters and ε are errors following a multivariate normal distribution. This is a very typical situation, and we will also assume $\varepsilon_i \sim \mathcal{N}(0, \sigma_e \mathbf{1})$.

1.4.3 Estimating θ_T

We will attempt to learn about θ_T using the following four approaches:

- Ordinary least squares (*next section*)
- Bayesian parameter estimation without any model discrepancy (*chapter on bayesian inference*)
- Bayesian parameter estimation with a model discrepancy (*chapter on bayesian inference*)
- Bayesian parameter estimation with a constrained model discrepancy (*chapter on bayesian inference*)

The take-away is that that without any model discrepancy term, regardless of the amount of data, you will never recover θ_T, even within any reasonable uncertainty bands. In fact, even as you increase the amount of data, you will only become more certain of the wrong estimate of θ_T.

1.4.4 The OLS estimate

To find the value for the physical parameter that makes our model fit best to the data we have, we can setup the standard least-squares objective (or loss) function

$$\chi^2(\theta) = (\Phi\theta - \mathcal{D})^T(\Phi\theta - \mathcal{D}),$$

and then minimize with respect to θ. That is, we try to find

$$\hat{\theta} = \arg\min \chi^2(\theta).$$

This is equivalent to finding the ordinary least squares (*OLS*)[1] estimator $\hat{\theta}$ for the physical parameter, and is given by $\hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T\mathcal{D}$. This is the familiar *normal equation* for the least squares problem.

Exercise

Derive the normal equation.

If the case of *heteroscedastic* errors, gathered in a data covariance matrix $\mathbf{E}$, we have to use weighted least squares (*WLS*) with normal equations according to

$$\hat{\theta} = (\Phi^T\mathbf{E}^{-1}\Phi)^{-1}\Phi^T\mathbf{E}^{-1}\mathcal{D}.$$

From the perspective of a physicist, regression deals with (least) distances in a vector space. We repeatedly encounter scalar products such as $\Phi^T \cdot \mathcal{D}$. *Heteroscedastic* errors distort the data space, and the inverse data covariance matrix $\mathbf{E}^{-1}$

[1] Sometimes *OLS* is referred to as linear least squares or simply linear regression.

takes the role of a metric in this space. To preserve the scalar product, we must redefine it by replacing $\mathbf{A} \cdot \mathbf{B} \rightarrow \mathbf{A}\mathbf{E}^{-1}\mathbf{B}$. For more on this, see e.g., [7].

In our numerical simulations of the simple machine, we find the following *OLS* point estimates for θ:

	$N_d = 10$	$N_d = 30$	$N_d = 60$
$\hat{\theta}$	0.54	0.54	0.57

They should be compared with the true value $\theta_T = 0.65$. Clearly, our estimates for θ_T are completely wrong, and they will not improve even if we increase the amount of measured data.

We can use the regression residuals $\hat{\varepsilon} = \hat{\mathcal{D}} - \mathcal{D}$, where $\hat{\mathcal{D}}$ is the data prediction based on the *OLS* estimate, to estimate the variance in our sample

$$s^2 = \frac{\hat{\varepsilon}^T \hat{\varepsilon}}{N_d - N_p},$$

where N_p denotes the number of parameters in the model, i.e., $p = 1$. If our model is identical to the process from where we draw the data, i.e., reality, then in the limit of $N_d \rightarrow \infty$, our estimator $s^2 \rightarrow \sigma_e^2$.

OLS estimators are unbiased, i.e., the expected value for $\hat{\theta}$ equals the θ in the model. However, and this is important, with *OLS* we do not obtain a probability density function (pdf) for $\hat{\theta}$. We only get a point estimate, i.e., a single value. To partly remedy this, it is possible to define a so-called confidence interval (*CI*) I_α for $\hat{\theta}$ with upper (U) and lower (L) limits according to

$$I_\alpha(\theta) = [L, U] : p(L(F) \leq \theta \leq U(F) | F \sim \theta) = 1 - \alpha,$$

where $\alpha \in [0, 1]$. This basically states that if we sample future data F from the model governed by the parameter θ, there is a $1 - \alpha$ probability that the interval $[L(F), U(F)]$ contains θ. Strictly speaking, this only applies if the regression model is identical to the real process that generates the data.

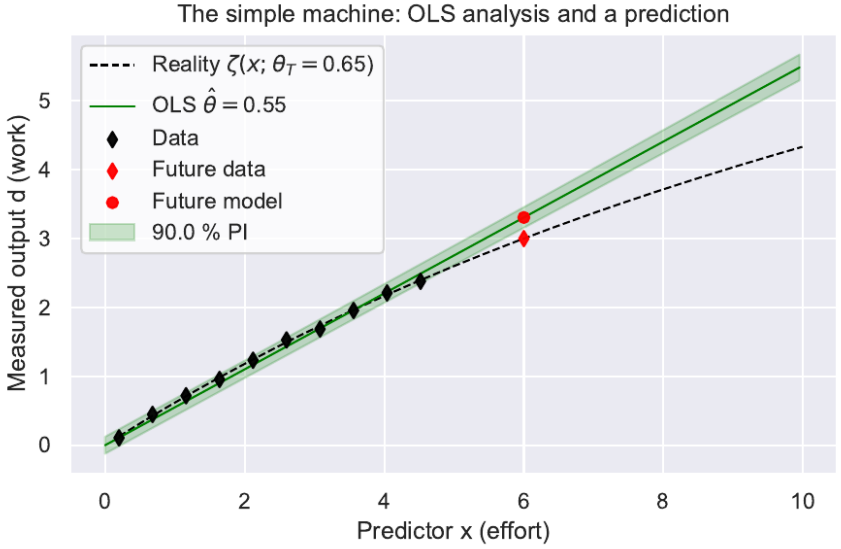


Fig. 2: $N_d = 10$ measurements (black diamonds) of the work produced by the simple machine (dashed black line). The resulting prediction (solid green line) is based on a linear model with an *OLS* point estimate of the model parameter $\hat{\theta}$. The green band indicates the 90% *PI*. A future data point ends up within the *PI*. However, this is not a robust result.

In the next *chapter* we will use a Bayesian strategy to derive an alternative expressions for extracting $I_\alpha(\theta)$. For now we remind you of the standard expression for extracting the *CI* for the parameters in a *OLS* analysis of a *GLM* with an

unknown normal *i.i.d.* error term $\mathcal{N}(0, \sigma_e^2 \mathbf{1})$

$$I_\alpha(\theta) = [\hat{\theta} - \sqrt{s^2 * (\Phi^T \Phi)^{-1}} \cdot t_{N_d - N_p}(1-\alpha/2) \leq \theta \leq \hat{\theta} + \sqrt{s^2 * (\Phi^T \Phi)^{-1}} t_{N_d - N_p}(1-\alpha/2)]$$

where $t_{N_d-N_p}(q)$ is the inverse of the cumulative distribution function for the Student's t-distribution. In the limit $(N_d - N_p) \to \infty$, and $\alpha = 0.05$, we have $t_{N_d-N_p}(1-\alpha/2) \approx 1.96$. With $\alpha = 0.1$ we get $t_{N_d-N_p}(1-\alpha/2) \approx 1.64$.

Exercise

Explain why it is reasonable that $t_{N_d-N_p}(1-\alpha/2)$ is smaller for larger values of α.

In the *OLS* analysis of our model for the simple machine we obtain the following ($\alpha = 0.1$) 90% *CI*s

	$N_d = 10$	$N_d = 30$	$N_d = 60$
$\hat{\theta}$	0.54	0.54	0.55
$I_{0.10}(\theta)$	$[0.51, 0.57]$	$[0.53, 0.56]$	$[0.55, 0.56]$

The above result strengthens our conclusion that we are becoming more confident of a biased (wrong) point estimate for θ_T as we increase the data.

Generally speaking, there is no reason to expect that the value θ in our simple model should converge to that of θ of the true model used for data generation since the models are different. However, we here have in mind a common situation in physics: the former model is ought to be simple enough, whereas the latter provides a correction to be more accurate. Thus, our observation here is that a narrow *CI* constructed for the *OLS* is not to be misinterpreted as the evidence of the simple model correctness. A narrow *CI* rather indicates that in case the data is generated according to the simple model with some value of θ^*, the *OLS* will likely give close $\hat{\theta}$ with the *CI* that includes θ^* (with the probability for which the *CI* is determined). By contrast the Bayesian parameter estimation provides a way to determine the *CI* that is driven by our prior knowledge about the model and measurement errors.

Similar to a *CI*, we can compute the prediction interval (*PI*). It defines in what range future individual observations should fall. The difference $\mathbf{f}$ between some future data $\mathcal{F}$ and our estimate of the future data $\hat{\mathcal{F}}$ is

$$\mathbf{f} = \mathcal{F} - \hat{\mathcal{F}} = \tilde{\Phi}\theta_T + \varepsilon - \tilde{\Phi}\hat{\theta},$$

where $\tilde{\Phi}$ is the design matrix corresponding to the future data, i.e., for new values of the predictor x. One can show that the sample variance of $\mathbf{f}$ is given by

$$\text{Var}(\mathbf{f}) = s^2(\mathbf{1} + \tilde{\Phi}(\Phi^T\Phi)^{-1})\tilde{\Phi}^T.$$

The resulting $(1-\alpha)$ *PI* for the new data is given by

$$\hat{\mathcal{F}} \pm t_{N_d-N_p}(1-\alpha/2) \cdot \sqrt{\text{Var}(\mathbf{f})}.$$

In the *next chapter* we will use a Bayesian strategy to derive the above expressions for the *PI*.

This bias in the point estimate $\hat{\theta}$ is clearly visible also in a prediction, see Figure 2.

Exercise

Use the code in the notebook to simulate multiple measurements of a linear process to better understand the meaning of a *confidence interval* (*CI*).

1.5 Bayesian parameter estimation

As you known from previous courses, e.g., TIF285 *Learning from data*, we can use Bayes' theorem to compute the posterior probability for some model M given data $\mathcal{D}$

$$p(M|\mathcal{D}I) = \frac{p(\mathcal{D}|MI)p(M|I)}{p(\mathcal{D}|I)}. \tag{4}$$

We let I generically denote the proposition *any other information*. Although unconditional probability does not exist, in these notes we will sometimes omit this important condition.

We refer to $p(\mathcal{D}|M)$ and $p(M)$ as the likelihood and the prior, respectively. The denominator $p(\mathcal{D})$ is called the marginal likelihood. We can use our model to link control variables (predictors) $\mathbf{X}$ (see Section 1.4.2) via the design matrix Φ, to data $\mathcal{D}$ according to

$$\mathcal{D} = \Phi\theta + \varepsilon.$$

We will assume *i.i.d.* data errors $\varepsilon = \{\varepsilon_i\}_{i=1}^{N_d}$ such that the error for each datum is normally distributed with the same variance, i.e., $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$.

We can use Bayes' theorem to express

$$p(\theta, \sigma^2|\mathcal{D}) = \frac{p(\mathcal{D}|\theta, \sigma^2)p(\theta, \sigma^2)}{p(\mathcal{D})}.$$

If we assume a finite variance σ^2, the principle of maximum entropy dictates the least informative constraint for the probability of the data conditioned on the parameters, i.e., the likelihood, is given by a normal distribution. In our case we have *i.i.d.* errors parametrized by a common variance σ^2

$$p(\mathcal{D}|\theta, \sigma^2) = \mathcal{N}(\mathcal{D}|\Phi\theta, \sigma^2\mathbf{1}) = \left(\frac{1}{2\pi\sigma^2}\right)^{N_d/2} \exp\left\{-\frac{1}{2}\frac{(\mathcal{D}-\Phi\theta)^T(\mathcal{D}-\Phi\theta)}{\sigma^2}\right\}.$$

The evidence, or marginal likelihood, is defined as

$$p(\mathcal{D}) = \int p(\mathcal{D}|\theta, \sigma^2)p(\theta, \sigma^2)\, d\theta d\sigma^2.$$

We must also define a prior $p(\theta, \sigma^2)$ to reflect our *a priori* knowledge about θ and σ^2 before looking at the data $\mathcal{D}$. That will be one of the main topics during the *next chapter*.

CHAPTER

TWO

BAYESIAN INFERENCE

Key take-away messages

- We can obtain a closed-form expression for the posterior if we use a prior that is conjugate to the likelihood.
- For a certain non-informative and conjugate prior the Bayesian parameter estimates reproduce the least squares result.
- Reality is more complicated than conjugate priors. In many cases we can use Monte Carlo algorithms to sample the posterior.
- A Gaussian process can be used to characterize the unknown model discrepancy.
- With a model-discrepancy term we find more realistic parameter estimates.
- No free lunch: only realistic prior information yields realistic parameter estimates, and predictions are *always* challenging.

Before reading this chapter

Refresh your memory of the following concepts:

- Linear regression
- Bayes theorem
- Markov Chain Monte Carlo
- A Gaussian process with a squared exponential kernel

Much of this material was covered in TIF285 *Learning from data*. However, we will tie these concepts together to better understand the importance of acknowledging model discrepancies.

2.1 Recap: the simple machine

In this chapter, we will keep using data from *the simple machine*. It returns work $\zeta(x)$ given some effort x according to

$$\zeta(x) = \frac{\theta_T x}{1 + 0.05x}.$$

We have data for N_d different values of the control parameter x, and θ_T is an intrinsic physical property of our machine that is of interest to the machine designers or anyone studying the machine itself. In this example, we will set θ_T to some value of our liking (0.65), and then forget about it and pretend that we do not know anything about the underlying reality

of the simple machine. This way we can test how well different procedures for extracting θ_T works. This chapter is based on the work presented in [6]. The dataset is a column vector $\mathcal{D} = [d_0, d_1, \dots, d_{N_d}]^T$ in our case. All data points have been generated with homoskedastic errors via

$$d_i = \zeta(x_i) + \varepsilon_i,$$

where $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ and $\sigma = 0.02$. We will in general treat σ as an unknown at the outset.

Our (linear) model is given by

$$M(\theta; x) = \theta x,$$

and we relate our model to reality via the data according to

$$d_i = M(\theta; x_i) + \varepsilon_i^d + \delta(x_i).$$

As we will see in this chapter, it is important to incorporate $\delta(x_i)$ during the calibration stage, i.e., when we try to learn reasonable values for θ from the dataset $\mathcal{D}$.

2.2 A conjugate prior for estimating θ_T

We ended Chapter 1 with Bayes' theorem for expressing the posterior of interest, i.e.,

$$p(\theta, \sigma^2|\mathcal{D}) = \frac{p(\mathcal{D}|\theta, \sigma^2)p(\theta, \sigma^2)}{p(\mathcal{D})},$$

in terms of a normal likelihood that we will use throughout this chapter

$$p(\mathcal{D}|\theta, \sigma^2) = \mathcal{N}(\mathcal{D}|\Phi\theta, \sigma^2\mathbf{1}) = \left(\frac{1}{2\pi\sigma^2}\right)^{N_d/2} \exp\left\{-\frac{1}{2}\frac{(\mathcal{D} - \Phi\theta)^T(\mathcal{D} - \Phi\theta)}{\sigma^2}\right\}. \tag{1}$$

Now we will continue by defining the prior *PDF* $p(\theta, \sigma^2)$. First, we will explore a so-called conjugate prior. A prior is conjugate to the likelihood if the posterior and the prior belong to the same family of probability distributions, e.g., normal, Poisson, exponential, gamma, etc. The advantage of using conjugate priors is that we can obtain a closed form expression for the posterior *PDF*. This is of course only a convenience. There is no guarantee that our prior belief or information can be encoded mathematically as a conjugate prior. Later we will setup problems that require numerical evaluation using, e.g., Markov Chain Monte Carlo (*MCMC*) methods. Most of the derivations of the analytical expressions are relegated to appendices.

2.2.1 The normal-inverse-gamma distribution

A normal-inverse-gamma (*NIG*) distributed prior is conjugate to a normal likelihood with unknown θ and σ^2, and is defined as

$$p(\theta, \sigma^2) = p(\theta|\sigma^2)p(\sigma^2) = \mathcal{N}(\theta|\mu_0, \sigma^2\Sigma_0) \cdot \mathcal{IG}(\sigma^2|\alpha_0, \beta_0) \equiv \mathcal{NIG}(\theta, \sigma^2|\mu_0, \Sigma_0, \alpha_0, \beta_0),$$

i.e., the product of the normal distribution

$$\mathcal{N}(\theta|\mu_0, \sigma^2\Sigma_0) = \frac{1}{(2\pi)^{N_p/2}|\sigma^2\Sigma_0|^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(\theta - \mu_0)^T\Sigma_0^{-1}(\theta - \mu_0)\right\}, \tag{2}$$

and the inverse-gamma (*IG*) distribution

$$\mathcal{IG}(\sigma^2|\alpha_0, \beta_0) = \frac{\beta_0^{\alpha_0}}{\Gamma(\alpha_0)}\left(\frac{1}{\sigma^2}\right)^{\alpha_0+1} \exp\left\{-\frac{\beta_0}{\sigma^2}\right\},$$

leading to

$$\mathcal{NIG}(\theta, \sigma^2|\mu_0, \Sigma_0, \alpha_0, \beta_0) = \frac{\beta_0^{\alpha_0}\sigma^{-2(\alpha_0+1+N_p/2)}}{(2\pi)^{N_p/2}\Gamma(\alpha_0)|\Sigma_0|^{1/2}} \exp\left\{-\frac{1}{\sigma^2}\left[\beta_0 + \frac{1}{2}(\theta-\mu_0)^T\Sigma_0^{-1}(\theta-\mu_0)\right]\right\}. \tag{3}$$

Note that the *NIG* distribution is a joint *PDF* for (vector) θ and (scalar) σ^2. The so-called normal-inverse-Wishart distribution is a conjugate prior to a multivariate normal with unknown mean and covariance matrix.

Representative examples of these *PDF*s are shown in Fig. 1, Fig. 2, and Fig. 3.

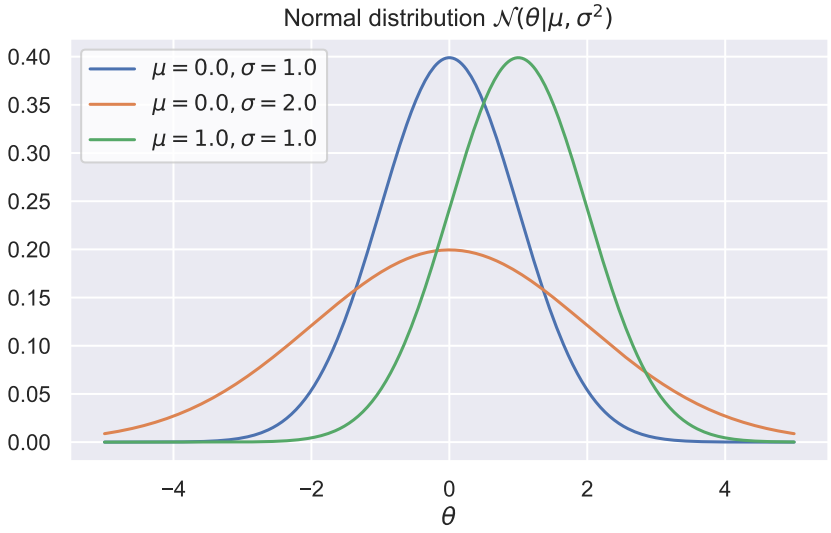


Fig. 1: Univariate normal distributions.

Learn more and get active

You can use the provided Jupyter Notebook (`some_stats_distributions.ipynb`) to explore the probability distributions we will encounter in this chapter.

We note that the prior for θ is defined by the prior mean μ_0 and prior covariance Σ_0. Let us comment briefly on the characteristics of the inverse-gamma distribution. It is a two-parameter *PDF* defined for positive real numbers. The shape-parameter $\alpha_0 > 0$ governs the height of the *PDF*. A larger α_0-value also implies thinner tails. The scale-parameter $\beta_0 > 0$ governs the width of the *PDF*. For $X \sim \mathcal{IG}(\alpha, \beta)$ we have

$$\text{mean} = \frac{\beta}{\alpha - 1} \quad (\alpha > 1), \quad \text{mode} = \frac{\beta}{\alpha + 1}.$$

The inverse-gamma *PDF* is conjugate to a normal likelihood and appears most often in Bayesian inference studies where the variance is unknown.

The parameters that govern the prior *PDF* are called hyperparameters, and a prior for a hyperparameter is called a hyperprior. Clearly, we have several hyperparameters, and based on our prior belief about the system, we will assign definite values to the hyperparameters $(\mu_0, \Sigma_0, \alpha_0, \beta_0)$. This is equivalent to a delta-function hyperprior.

At first sight, the *NIG* prior appears clumsy but its exponential structure fits like a glove with the form of the normal likelihood and enables a closed-form expression for the resulting posterior. Combining the *NIG* prior for θ and σ^2 with

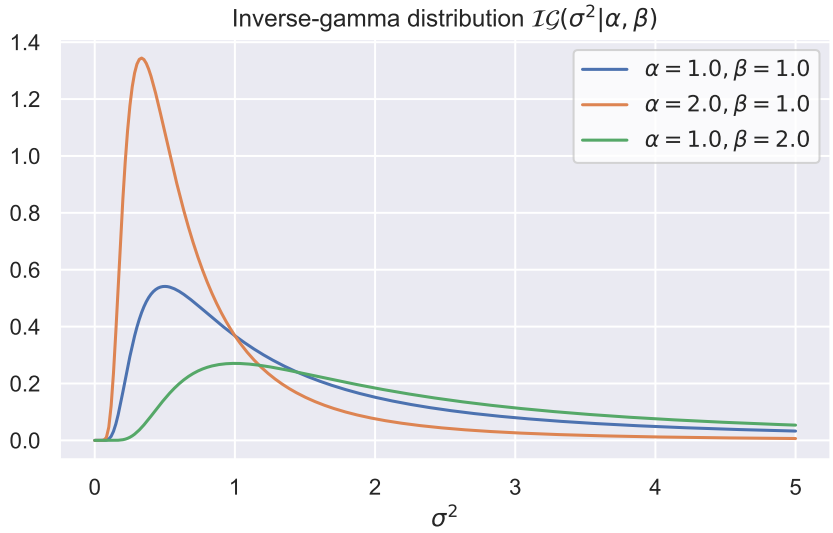


Fig. 2: Univariate inverse-gamma distributions.

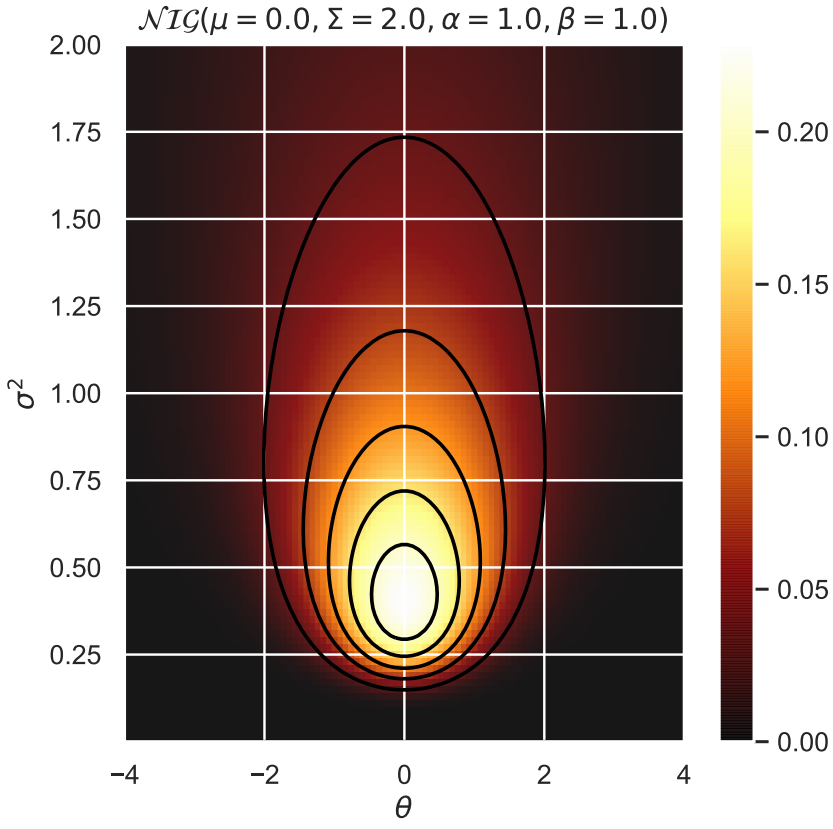


Fig. 3: A bivariate Normal-inverse-gamma distribution.

the normal likelihood in Eq. (1) one can show (see *Appendix*) that the posterior *PDF* is given by

$$p(\theta, \sigma^2|\mathcal{D}) = \mathcal{NIG}(\theta, \sigma^2|\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta}) \tag{4}$$

with

$$\begin{aligned}
\tilde{\mu} &= [\Sigma_0^{-1} + \Phi^T\Phi]^{-1}[\Sigma_0^{-1}\mu_0 + \Phi^T\mathcal{D}] \\
\tilde{\Sigma} &= [\Sigma_0^{-1} + \Phi^T\Phi]^{-1} \\
\tilde{\alpha} &= \alpha_0 + N_d/2 \\
\tilde{\beta} &= \beta_0 + \frac{1}{2}\left(\mathcal{D}^T\mathcal{D} + \mu_0^T\Sigma_0^{-1}\mu_0 - \tilde{\mu}^T\tilde{\Sigma}^{-1}\tilde{\mu}\right).
\end{aligned}$$

In effect, the prior parameters $(\mu_0, \Sigma_0, \alpha_0, \beta_0)$ learn from the data, and the updated posterior parameters are given by $(\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta})$. Using the material in the *Appendix of this chapter* one can also show that the marginal distributions are given by

$$\begin{aligned}
p(\theta|\mathcal{D}) &= \int \mathcal{NIG}(\theta, \sigma^2|\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta})\, d\sigma^2 = \mathcal{T}_{2\tilde{\alpha}}(\theta|\tilde{\mu}, (\tilde{\beta}/\tilde{\alpha})\tilde{\Sigma}) \\
p(\sigma^2|\mathcal{D}) &= \int \mathcal{NIG}(\theta, \sigma^2|\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta})\, d\theta = \mathcal{IG}(\sigma^2|\tilde{\alpha}, \tilde{\beta}).
\end{aligned} \tag{5}$$

Let us comment on this result. We have now shown that the posterior *PDF* for the vector of (linear) model parameters follow a multivariate $t-$distribution. The $t-$distribution $\mathcal{T}_\nu(\theta|\mu, \Sigma)$ has three parameters: ν (degrees of freedom), μ (median, mode, and mean if $\nu > 1$), and Σ (scale matrix). The t-distribution is basically a normal distribution but with heavier tails. This is clearly due to the marginalization of the unknown variance of the underlying normal *PDF*. In fact, in the limit $\nu \to \infty$ we find $\mathcal{T}_\nu(\mu, \Sigma) \to \mathcal{N}(\mu, \Sigma)$, see Figure 4.

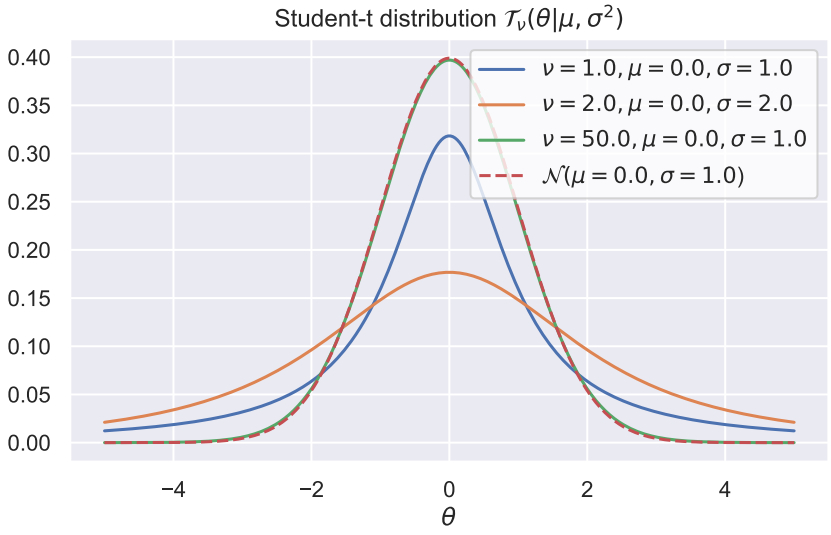


Fig. 4: Univariate Student's t-distributions. Notice how the t-distribution approaches a normal distribution as we increase the number of degrees of freedom ν.

Following the reasoning in the Appendix one can also derive the posterior predictive distribution (PPD), i.e., the *PDF* for future predictions $\tilde{\mathcal{F}}$ given the model M and a set of new inputs for the control variable $\tilde{\mathbf{X}}$. The new inputs also give rise to a new design matrix $\tilde{\Phi}$ (also known as sensing matrix). Our model M is uncertain, i.e., its parameters are represented by a joint posterior *PDF* informed by past data. We obtain the posterior predictive *PDF* by marginalizing over the uncertain

model parameters that we just inferred,

$$\begin{aligned} p(\tilde{\mathcal{F}}|M,\mathcal{D}) &= \int \mathcal{N}(\tilde{\mathcal{F}}|\tilde{\Phi}\theta,\sigma^2\mathbf{1})\mathcal{NIG}(\theta,\sigma^2|\tilde{\mu},\tilde{\Sigma},\tilde{\alpha},\tilde{\beta})\,d\sigma^2 d\theta \\ &= \mathcal{T}_{2\tilde{\alpha}}(\tilde{\mathcal{F}}|\tilde{\Phi}\tilde{\mu},(\tilde{\beta}/\tilde{\alpha})(\mathbf{1}+\tilde{\Phi}\tilde{\Sigma}\tilde{\Phi}^T)) \end{aligned} \tag{6}$$

The scale of the t-distribution contains two terms; the first one is $\tilde{\beta}/\tilde{\alpha}\mathbf{1}$ and is directly proportional to the sample variance, i.e., the data error, and the second term $\tilde{\beta}/\tilde{\alpha}\tilde{\Phi}\tilde{\Sigma}\tilde{\Phi}^T$ is directly proportional to the model parameter covariances learned from the data.

2.2.2 The non-informative limit of the NIG prior

In our first application of the posterior in Eq. (4) we will reconnect with the ordinary least squares (OLS) analysis in *Chapter 1*. This way we also build up some intuition for the general *NIG* posterior. It is possible to obtain the *OLS* result if we tune the *NIG* prior to $\mu_0 = 0$, $\alpha_0 \to -N_p/2$, $\beta_0 \to 0$, and let $\Sigma_0 \to \infty\mathbf{1}$. This particular choice of prior parameters leads to a so-called uninformative limit for the *NIG* prior. Indeed, intuitively we formulate the least informative prior conditioned on θ, and one can show that this choice of parameters also leads to a so called Jeffreys prior $p(\sigma^2) \propto 1/\sigma^2$ for the variance. Some refer to these priors as objective since they are invariant under coordinate transformations. We will not delve into this topic further. For this choice we obtain a posterior

$$p(\theta,\sigma^2|\mathcal{D}) = \mathcal{NIG}(\theta,\sigma^2|\tilde{\mu},\tilde{\Sigma},\tilde{\alpha},\tilde{\beta}) \tag{7}$$

with

$$\begin{aligned} \tilde{\mu} &= [\Phi^T\Phi]^{-1}[\Phi^T\mathcal{D}] \equiv \hat{\theta} \\ \tilde{\Sigma} &= [\Phi^T\Phi]^{-1} \\ \tilde{\alpha} &= (N_d - N_p)/2 \\ \tilde{\beta} &= (\mathcal{D}^T\mathcal{D} - \tilde{\mu}^T\tilde{\Sigma}^{-1}\tilde{\mu})/2. \end{aligned}$$

Indeed, the posterior mean coincides with the *OLS* point estimate $\hat{\theta}$. Furthermore, one can also show that $\tilde{\beta}$ is proportional to the *OLS* sample variance

$$\tilde{\beta} = \frac{N_d - N_p}{2}s^2, \quad \text{where} \quad s^2 = \frac{(\mathcal{D}-\Phi\hat{\theta})^T(\mathcal{D}-\Phi\hat{\theta})}{N_d - N_p} \tag{8}$$

Exercise

Derive Equation (8).

Thus, we can also extract the *OLS* data variance as $s^2 = \tilde{\beta}/\tilde{\alpha}$. In effect the *OLS* coincides with the uninformative limit of Bayesian linear regression. This limit is useful also for validating a numerical implementation of the closed-form expressions for the parameter posterior, which in turn is useful for validating, e.g., a Markov Chain Monte Carlo (*MCMC*) analysis of a more complicated case with non-conjugate priors. Overall, the upshot of the Bayesian approach is of course that it allows us to straightforwardly incorporate any prior knowledge we might have. In fact, we are obliged to use whatever information we have. Next, we will study the simple machine.

2.3 A conjugate prior for the simple machine

As in Chapter 1 we assume that we have a dataset $\mathcal{D}$ consisting of N_d data points. For now we settle with $N_d = 10$, see Figure 5

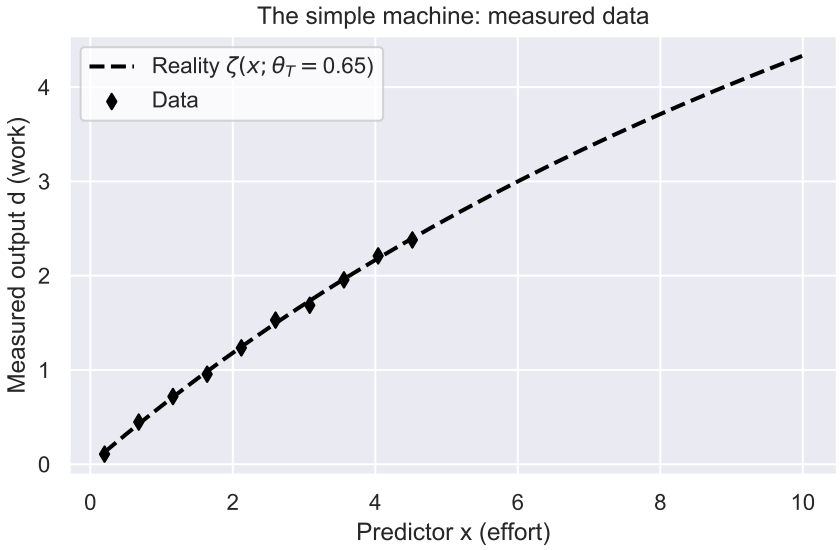


Fig. 5: Measurement of the work produced by the simple machine for $N_d = 10$ different values of the effort x.

Let us explore Bayesian parameter estimation of the simple machine using the analytical expressions for the parameter posterior that we introduced above.

2.3.1 An uninformative prior

We begin by defining an improper *NIG* prior $p(\theta, \sigma^2|\mu_0, \sigma^2\Sigma_0, \alpha_0, \beta_0)$ by setting $\mu_0 = 0$, $\alpha_0 \to -N_p/2$, $\beta_0 \to 0$, and let $\Sigma_0 \to \infty\mathbf{1}$. We can then evaluate Eq. (7) exactly and plot the result, see Figure 6.

Exercise

Use the notebook for the simple machine to reproduce Figure 6.

We can also verify that the posterior *PDF* in Figure 6 corresponds to the *OLS* result. Indeed, the posterior mean $\mu = \hat{\theta} = 0.55$ and according to Eq. (8)we recover the *OLS* sample variance $s^2 \approx 0.0045$ from the posterior values of $\alpha \approx 4.5$ and $\beta \approx 0.02$.

One of the key advantages with a Bayesian analysis is that we obtain a *PDF* for the parameters themselves. In a traditional regression analysis based on frequentist statistics, we only handle probability distributions for the data. To say something about the parameters we had to conjure up, e.g., a *confidence interval* (*CI*). Once we have the posterior *PDF* $p(\theta, \sigma^2|\ldots)$ we know *everything* there is to know about the parameters themselves. If we now marginalize over σ^2 according to Eq. (5), we recover a t-distribution for θ. From this we can extract the symmetric limits $[\theta_L, \theta_H] = [0.537, 0.565]$ that capture 90% of the corresponding probability density for θ. This is the so-called Bayesian credible interval (*BCI*) for θ. In this case it agrees exactly with the 90% *CI* we obtained in Chapter 1, see Figure 7.

We emphasize that although the results from the *OLS* analysis and Bayesian parameter estimation agree exactly when we use an improper *NIG* prior, it is very important to remember the fundamental difference in how we interpret the results.

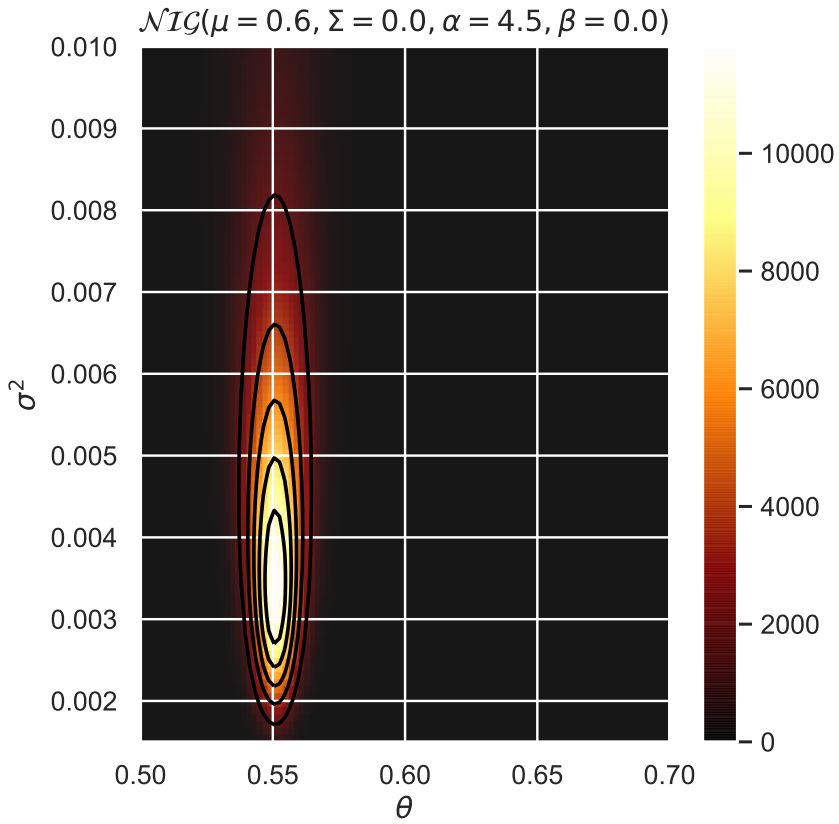


Fig. 6: Posterior *PDF* for the simple machine. The result is based on an improper *NIG* prior and the *OLS* result can be inferred from the Bayesian posterior in this case.

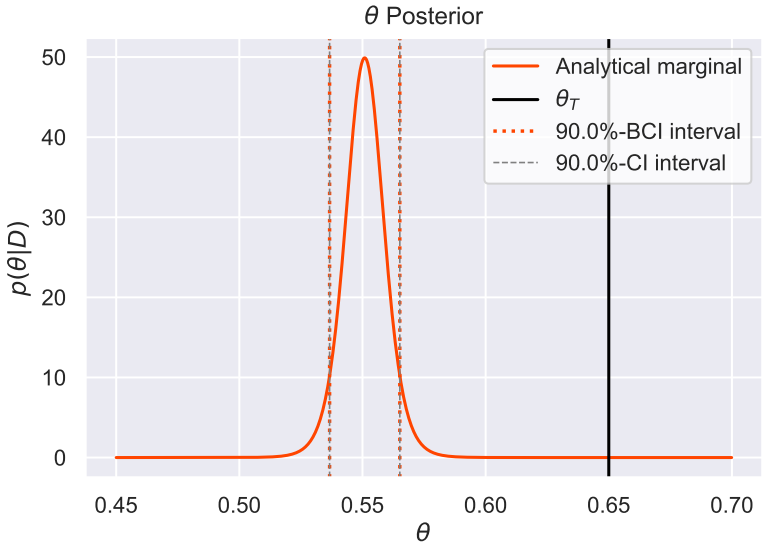


Fig. 7: The marginal *PDF* of the posterior full for the simple machine. The result is based on an improper *NIG* prior and the *OLS* result can be inferred from the Bayesian posterior in this case. Clearly, the 90%-degree of belief (*DOB*) interval falls right on top of the 90%-*CI* from the *OLS* analysis.

Discussion

Which advantages/disadvantages of using a Bayesian/Frequentist approach, respectively, can you think of?

2.3.2 An informative prior

Let us now try an informative prior. Assuming that we have built the machine ourselves, or have some other information about what to expect from it, we decide to put $\mu_0 = 1$. Based on our knowledge about the measurement process we set a prior mean and mode for the *IG PDF* to 0.5^2 and 0.2^2[1]. From our experience with constructing the machine we also know the limits of the work it can output and based on our linear model we tune Σ_0 such that we obtain a variance in θ equal to 2^2 The *NIG* prior is plotted in Figure 8.

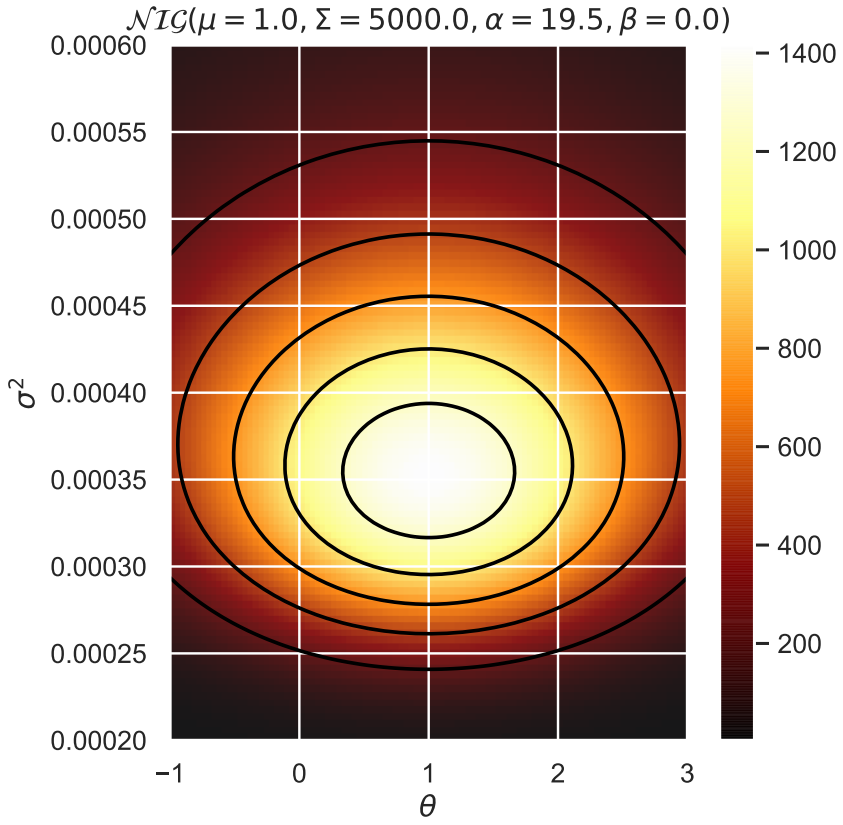


Fig. 8: Informative prior *PDF* for the simple machine. Since this is a textbook example, we know that the true values for θ and σ^2 are hiding at $(\theta, \sigma^2) = (0.65, 0.02^2)$.

Given our choice of prior and likelihood, we can evaluate the posterior analytically, see Figure 9 and Figure 10 for the θ marginal.

Exercise

[1] The mean μ and mode m for the *IG* distribution are given by $\mu = \beta/(\alpha - 1)$ and $m = \beta/(\alpha + 1)$.
[2] The marginal variance of θ depends on α and β, for which we have very small numbers in our case.

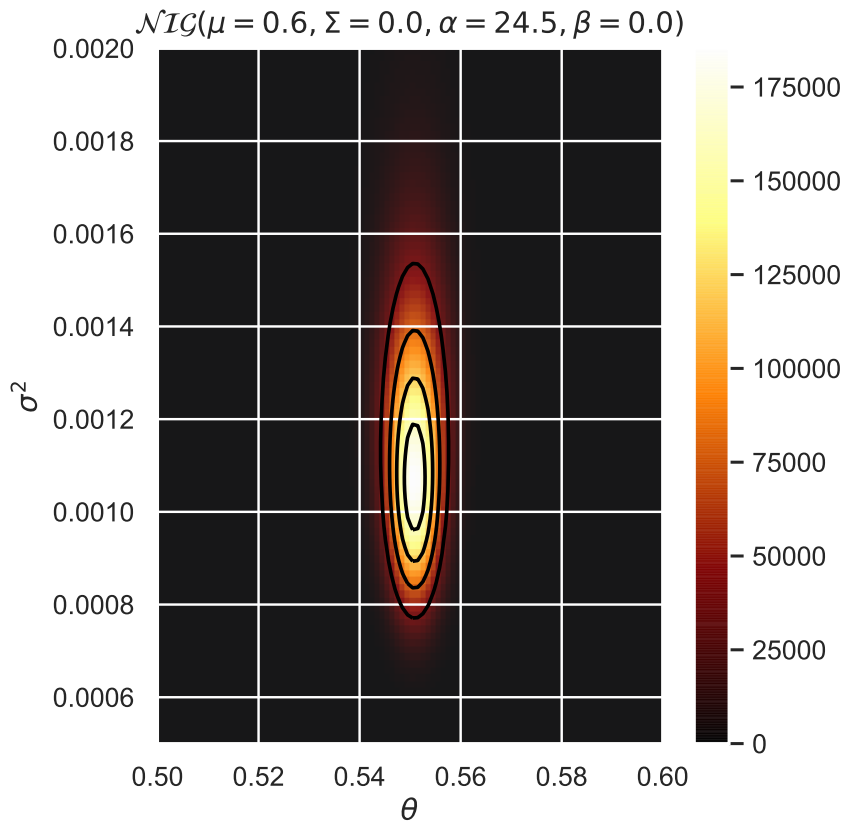


Fig. 9: Joint posterior *PDF* for the simple machine. The result is based on the informative *NIG* prior in Figure 8 and the $N_d = 10$ data points in Figure 5.

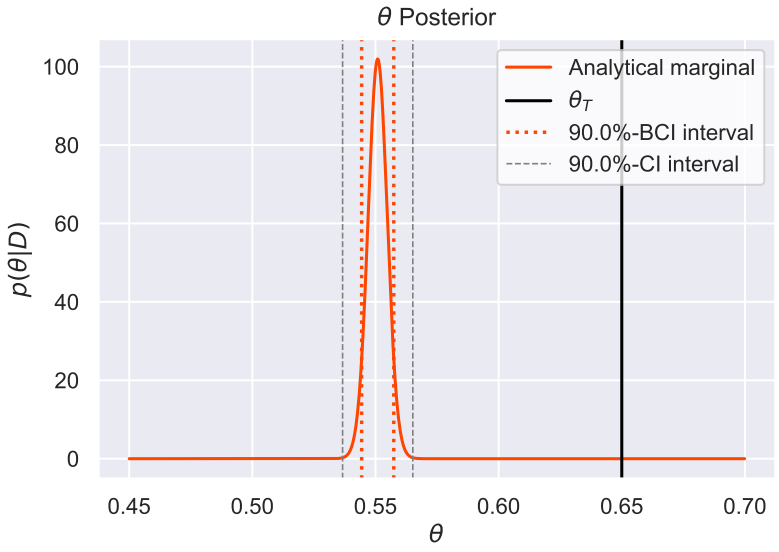


Fig. 10: The marginal *PDF* of the posterior based on the informative prior *PDF*. In this case, the 90%-*DOB* interval does not coincide with the 90%-*CI* from the *OLS* analysis.

What will happen with the posterior *PDF* in Figure 9 if we were to increase the amount of data? You can examine this case using the notebook for the simple machine.

2.3.3 The posterior predictive

Equipped with a joint posterior *PDF* $p(\theta, \sigma^2|\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta})$ it is straightforward to make model predictions for future data and summarize the result as a *PDF*. We call this *PDF* the posterior predictive distribution (*PPD*), and in our case the *PPD* is given by a t-distribution defined in Eq. (6). In this section we will explore predictions for new values $\tilde{x}$ based on the joint posteriors we obtained in the previous section.

As mentioned, when we employ an improper uninformative *NIG* prior we obtain a posterior *PDF* for θ and σ^2 that match the *OLS* result. Similarly, it turns out that the 90%-*BCI* for the corresponding *PPD* matches the 90%-*PI* obtained from the *OLS* analysis. You can verify this using the notebook for the simple machine. Let us instead turn our attention to the predictions based on the parameter posterior in Figure 9 that was obtained using the informative prior in Figure 8. To make this specific, we will look at the prediction for the machine output at the predictor value $\tilde{x} = 6.0$. For our linear model, this trivially yields the design matrix $\tilde{\Phi} = 6.0$. Evaluating Eq. (6) yields the *PPD* t-distribution plotted in Figure 11.

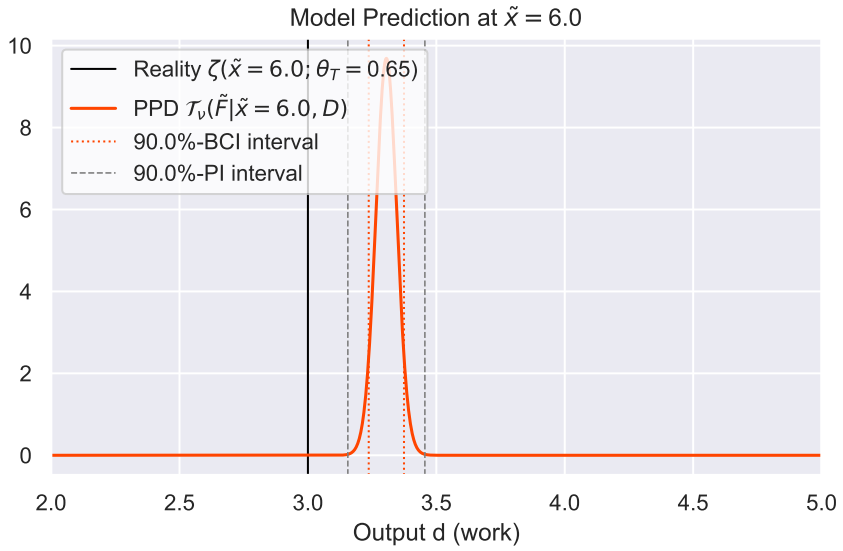


Fig. 11: Posterior predictive distribution at $\tilde{=}6$ based on an informative *NIG* prior and $N_d = 10$ data points.

Repeating this exercise for $\tilde{x} \in [0, 10]$ yields the 90% *BCI*-band for the range of predictions plotted in Figure 12.

Discussion

Is the result in Figure 12 a better result than the *OLS* prediction (see, e.g., Figure 2)? What do you think will happen when you increase N_d from 10 to 30 or 60? Try it out in the notebook.

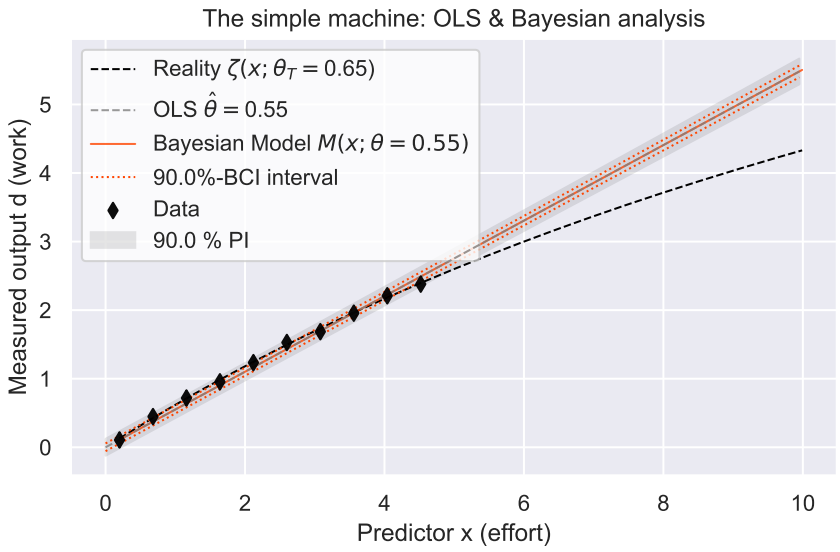


Fig. 12: Posterior predictive distribution for $x \in [0, 10]$. Clearly, the 90%-*BCI* is smaller than the 90%-*PI*. This is only due to our prior.

2.4 MCMC analysis of the simple machine

We can also use Markov Chain Monte Carlo (*MCMC*) to sample the parameter posteriors. Unless we use strictly conjugate priors it will not be possible to obtain closed-form expressions for the posteriors. At any rate, numerical evaluation of the posteriors requires less work by us, and more work by the computer. A quite comfortable tradeoff! As always, applying a numerical algorithm, e.g., *MCMC*, on a test case where we have analytical results enables use to i) work up some intuition about the *MCMC* and ii) eliminate bugs in our numerical code. For *MCMC*, in this section we will employ the affine-invariant ensemble sampler. In this course we will rely on the Python implementation called `emcee`. It would have worked just fine to use a standard Metropolis-Hastings *MCMC* method as well. However, the `emcee` affine-invariant sample is quite efficient and the Python package is easy to handle and rather versatile. For a detailed presentation of this algorithm and the Python package see Ref. [8]. Basically, a collection of `nwalkers` walkers will traverse the `ndim`-dimensional parameter space in a particular way that fulfills the *MCMC* requirement of detailed balance to produce a unique equilibrium distribution that is reached in the limit of a large number of computation steps that is also independent of the initial conditions. We will use *MCMC* to sample the logarithm of the posterior, i.e.

$$\log[p(\theta, \sigma^2|\mathcal{D})] = \log[p(\mathcal{D}|\theta, \sigma^2)] + \log[p(\theta, \sigma^2)] - \log[p(\mathcal{D})].$$

For parameter estimation we only need to evaluate the log-likelihood and the log-prior. The log of the marginal likelihood is an overall normalization constant that is of less importance to us right now. It will return when we discuss model comparison. From the definitions of the normal likelihood and the *NIG* prior, we have

$$\log\left[p(\mathcal{D}|\theta, \sigma^2)\right] \propto -\frac{N_d}{2}\log(\sigma^2) - \frac{1}{2}\frac{(\mathcal{D} - \Phi\theta)^T(\mathcal{D} - \Phi\theta)}{\sigma^2},$$
$$\log\left[p(\theta, \sigma^2)\right] \propto \log\left[\mathcal{N}(\theta|\theta_0, \sigma^2\Sigma_0)\right] + \log\left[\mathcal{IG}(\sigma^2|\alpha_0, \beta_0)\right].$$

Exercise

Verify the expressions for the log-posterior. What is the practical reason for working with logarithms in the first place?

Check out the notebook for how to handle the `emcee` *MCMC* interface and to define the log-prior, log-likelihood, and log-posterior. By default, the notebook will initiate 20 walkers in 2 dimensions (since we have two parameters, θ and σ^2)

and let each walker take thousands of steps. The code should not take more than a minute or two to run. The result will be a chain of Markov steps. The posteriors that we will sample are comparatively easy to handle, and we have a rather clear picture of the end result. The perpetual question in most *MCMC* analyses is: *Are my chains sufficiently converged?* We will not explore this topic further here but it is an aspect that we will reexamine in the *Chapter on model averaging* and that you will also encounter in the assignments. Here, we will only leave you with the advice: *Let your* MCMC *code run for as long as you can.* Indeed, there are many reasons for why your Markov chains appear to have converged when they in fact have not [9].

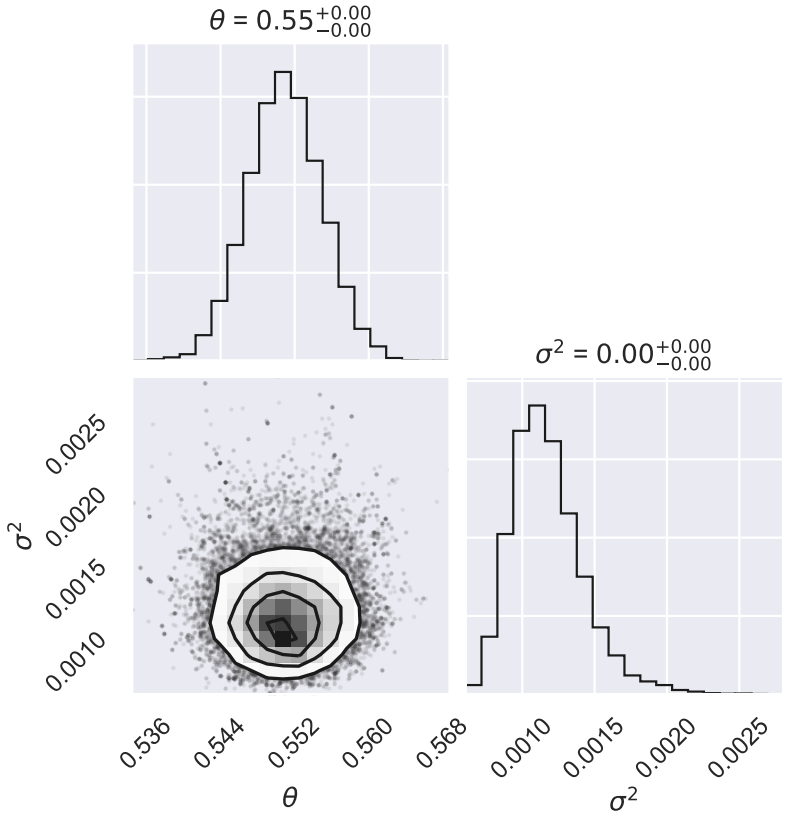


Fig. 13: Corner plot of the *MCMC* samples posterior corresponding to the analytical result in Figure 9

In this section we will use the Python package `corner` to visualize multidimensional *PDF*s in terms of one and two-dimensional marginalized *PDF*s, see Figure 13 for an example. In the notebook you will find code to compare the *MCMC* sampled marginal distributions for θ and σ^2 with the analytical results. You should convince yourself that the numerical approach of using *MCMC* indeed works very well. This will be important for the next section when we incorporate the model discrepancy term and have no way of obtaining the posterior *PDF*s analytically.

2.5 A Gaussian process model for the discrepancy

We do not know exactly how to formulate the model-discrepancy term. If we did, we would probably try to include part of it directly in the model M. Since we are uncertain about the discrepancy term it must inevitably be modeled probabilistically.

Exercise

Even if we know how to incorporate (part of) δ into M there might sometimes exist valid reasons for not doing so. Can you think of any?

All we know is that given some predictor variable x and the true parameter value θ_T, whatever that means, our model will disagree with reality. We most likely have some prior knowledge or belief regarding the magnitude of the discrepancy. This might be a highly subjective belief, or there is a community consensus on what is likely missing from the model. To this end, we will use a supervised-learning approach to infer a *PDF* of the model-discrepancy $\delta(x)$. That is, we will utilize data and a prior to find a posterior for $\delta(\cdot)$.

2.5.1 Gaussian process

Insofar, our inference analysis has been centered on obtaining *PDF*s for parameters, e.g., θ. We will now use the concept of Gaussian processes (*GPs*) to describe random functions. You encountered them already in TIF285 *Learning from data*, but let us repeat some key ideas very briefly.

We will assume that the model discrepancy $\delta(x)$ is normally distributed and posit a *GP*

$$\delta(x) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')),$$

with a mean function $m(\mathbf{x})$ and a covariance function $k(\mathbf{x}, \mathbf{x}')$. From a Bayesian perspective, this corresponds to defining a prior for our model discrepancy function. By using a *GP* we have avoided a specific functional form for δ. We only specify a mean value of all possible values for the model-discrepancy $\delta(x)$ at each $x \in [0, 10]$. A common choice will be $m(x) = 0$ for all x, i.e., we have

$$\mathbb{E}[\delta(\mathbf{x})] = \mu = m(\mathbf{x}) = 0.$$

In practice we must always work with finite-dimensional arrays. Here, $\mathbf{x}$ is an $N+1$-dimensional column vector with values $x_i \in [0, 10]$ for $i = 0, \ldots, N$. We also specify the covariance between model discrepancy functions at different values of x, e.g., x_1 and x_2. That is, we must define the covariance $\text{cov}(\delta(x_1), \delta(x_2)) = \mathbb{E}[(\delta(x_1) - \mathbb{E}[\delta(x_1)])(\delta(x_2) - \mathbb{E}[\delta(x_2)])]$, which we can also generalize to a covariance matrix across the domain of predictor values $\mathbf{x}$

$$\text{cov}(\delta(\mathbf{x}), \delta(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}').$$

This expression should be understood as a matrix, i.e., we get an $(N+1) \times (N+1)$ covariance matrix, and for, e.g., $\mathbf{x} = \mathbf{x}'$ we can write

$$k(\mathbf{x}, \mathbf{x}) \equiv \mathbf{K}_{\mathbf{xx}} = \begin{bmatrix} k(x_0, x_0) & k(x_0, x_1) & \ldots & k(x_0, x_N) \\ k(x_1, x_0) & k(x_1, x_1) & \ldots & k(x_1, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_N, x_0) & k(x_N, x_1) & \ldots & k(x_N, x_N) \end{bmatrix}.$$

The covariance function must be positive definite. In this chapter we will consider only a single type of covariance function[1] called the squared exponential (*SQE*) with matrix elements

$$k(x_i, x_j) = \sigma_{GP}^2 \exp\left\{-\frac{1}{2\ell_{GP}^2}(x_i - x_j)^2\right\}.$$

[1] The covariance function is sometimes referred to as a *kernel* function.

Here, the variance (σ^2_{GP}) and length scale (ℓ^2_{GP}) of the *GP* are determined explicitly by setting values for these so-called hyperparameters. In this chapter we will learn them from the data as well. However, we will use *MCMC* to sample the parameter- and hyperparameter-spaces simultaneously. The *SQE* covariance is very common, it is very smooth in the sense that it is infinitely differentiable.

Once the mean and covariance functions are specified for a vector $\mathbf{x}$, we can draw random function values $\mathbf{f} = f(\mathbf{x})$ from the *GP* by sampling a multivariate normal distribution

$$p(\mathbf{f}|\mathbf{x}) = \mathcal{N}(\mathbf{f}|\mu, \mathbf{K_{xx}}).$$

In Figure 14 we have drawn 250 functions from a *GP* with $\mu = \mathbf{0}$ and a covariance matrix $\mathbf{K_{xx}}$ with $\sigma^2_{GP} = 1$ and $\ell_{GP} = 1$.

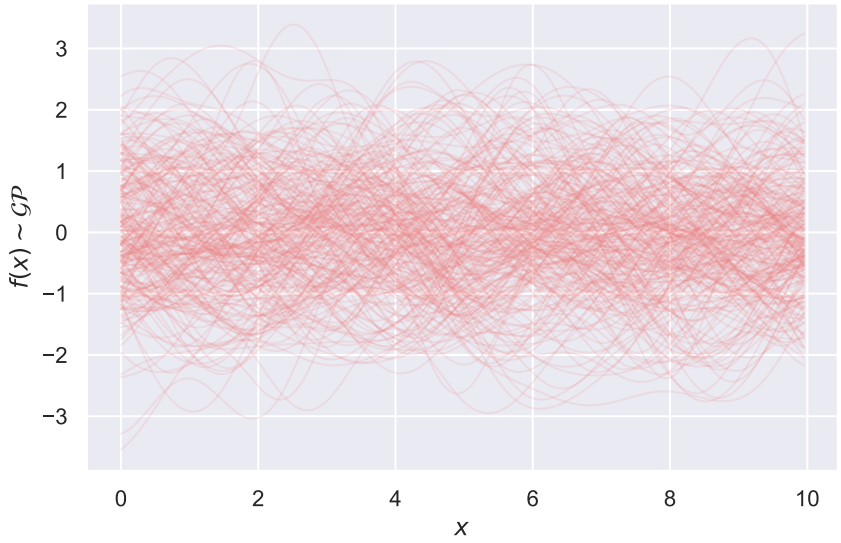


Fig. 14: 250 draws from an *SQE GP* with prior $\sigma^2_{GP} = \ell_{GP} = 1$.

For making plots, we define a vector $\mathbf{x} = [x_0, x_1, x_2, \dots, x_N]^T$ of length 200 with equidistant points in the interval $[0, 10]$. This provides a dense enough grid to make the resulting functions appear smooth.

Exercise

Use the notebook to draw samples from the GP prior. Can you explain what happens in the limits of large and small values for the variance σ^2_{GP} and length scale ℓ_{GP}?

It is straightforward to numerically include the effect of the $\delta(\mathbf{x})$-term in the *MCMC* sampling of θ. All we have to do is to draw a samples of the $\delta(\mathbf{x})$ term from the GP and add to each evaluation of the model $M(\theta; \mathbf{x})$. The use of a GP will introduce two additional parameters; θ^2_{GP} and ℓ_{GP}. We will use an inverse-gamma prior $p(\sigma^2_{GP}|\alpha_{0,s-GP}, \beta_{0,s-GP})$ for σ^2_{GP} and a gamma prior $p(\ell|\alpha_{0,l-GP}, \beta_{0,l-GP})$ for ℓ, with associated hyperpriors. We continue to use a proper *NIG* prior Eq. (3) for the parameters (θ, σ^2). As before, we will learn the four parameters $(\theta, \sigma^2, \sigma^2_{GP}, \ell_{GP})$ from the data $\mathcal{D}$ via Bayes' theorem

$$p(\theta, \sigma^2, \sigma^2_{GP}, \ell_{GP}|\mathcal{D}, I) = p(\mathcal{D}|\theta, \sigma^2, \sigma^2_{GP}, \ell_{GP}|, I) \cdot p(\theta, \sigma^2|I) \cdot p(\sigma^2_{GP}|I)p(\ell_{GP}|I),$$

where I denotes all additional information such as the hyperpriors and model specification M. In the following we will demonstrate one example of how to include additional knowledge about $\delta(x)$ and study the resulting impact on our inference about θ. All steps can be reproduced using the notebook.

2.5.2 Minimal prior information

First, we have to impose the hyperprior on the variance and length scale of the *GP*. We do not know much about σ^2_{GP}, but we will open up for the possibility that our model M might have some non-negligible deficiencies, based on the order of magnitude of the work produced by the machine, and set an inverse-gamma prior with mean 0.6^2 and mode 0.3^2 for σ^2_{GP}. We are more certain about the length scale of the *GP*, and the model discrepancy function is most likely rather smooth. We use a gamma prior with mean 3 and variance 0.5. The *GP* hyperpriors are shown in Figure 15.

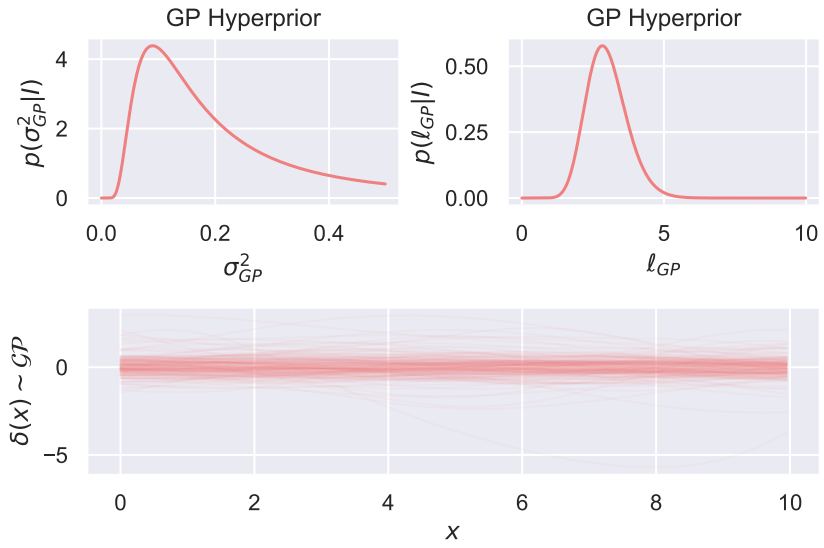


Fig. 15: (Bottom) 200 draw from a *GP* prior with an *SQE* kernel with variance and length scale according to an inverse-gamma (top left) and gamma hyperprior (top right). See text for details.

We will keep the same prior *PDF* for the parameters θ, σ^2 as before, i.e., the one shown in Figure 8. We will also keep the same dataset for the simple machine as before ($N_d = 10$).

A short *MCMC* run with 5000 *MCMC* steps using `emcee` (`ndim=4`, `nwalkers=200`) starting at $[\theta, \sigma^2, \sigma^2_{GP}, \ell] = [1, 10^{-3}, 2 \cdot 10^{-2}, 2.5]]$ takes about 10 minutes on a typical laptop. In this chapter we do not focus on *MCMC* (non-)convergence diagnostics. A brief inspection of the trace and removal of some burn-in indicates that we have obtained a reasonably reliable posterior *PDF*[2]. The corner plot is shown in Figure 16.

We notice immediately that the marginal *PDF* for θ is significantly wider now that we have included this rather unconstrained model discrepancy term, see Figure 17.

But most importantly, the *PDF* is also rather robust as we increase the number of data N_d.

Exercise

What is the significance of having a robust *PDF* $p(\theta|\mathcal{D})$ with respect to the number of data N_d?.

[2] Longer *MCMC* chains would certainly produce an even more reliable result.

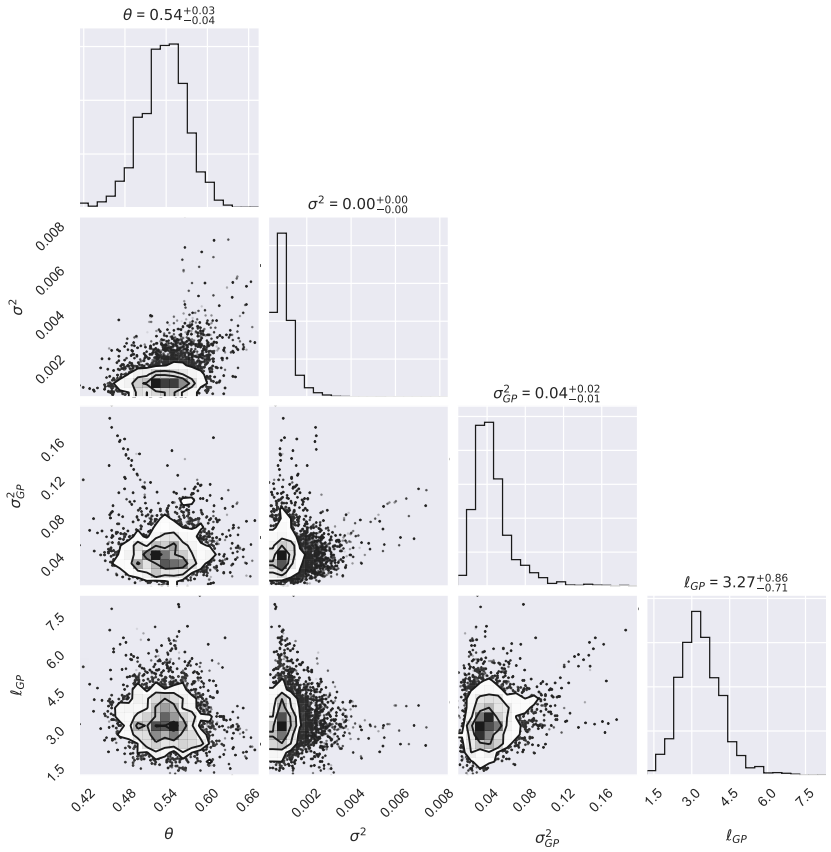


Fig. 16: Posterior *PDF* for the simple machine with $N_d = 10$. Parameter prior: $\mathcal{NIG}(\theta, \sigma^2 | \mu_0 = 1, \Sigma_0 = 5000, \alpha_0 = 19.5, \beta_0 = 7.4 \cdot 10^{-3})$. *GP* prior: *SQE* kernel with hyperprior $\mathcal{IG}(\sigma^2_{GP} | \alpha_0 = 1.67, \beta_0 = 0.24)$ and $\mathcal{G}(\ell_{GP} | \alpha_0 = 18, \beta_0 = 0.17)$

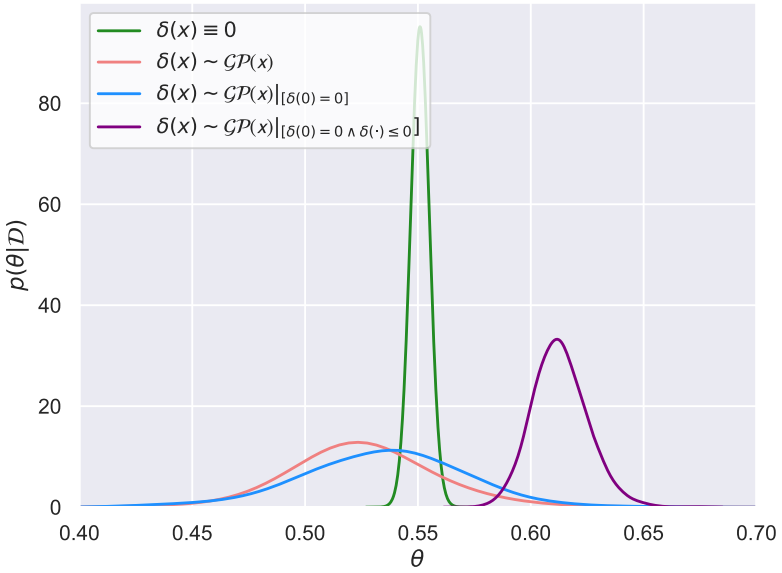


Fig. 17: Marginal posterior *PDF* $p(\theta|\mathcal{D})$ using various model discrepancy terms $\delta(x) \sim \mathcal{GP}$. See the text for details.

2.5.3 Medium prior information

If we believe that the lack of friction in our model M is a significant part of the model discrepancy, then we also know on fairly certain grounds that we should have $\delta(x = 0) = 0$. This condition is something we should incorporate in the model discrepancy. Fortunately, *GPs* are very flexible and we can easily condition our *GP* on this extra bit of knowledge.

Let us be explicit. Until now, we have drawn $\delta(\mathbf{x})$ from $\mathcal{GP}(\mathbf{0}, \mathbf{K_{xx}})$. If we instead assume a jointly normal *GP* for the values of the model discrepancy function at the $N_\star$ predictor values $\mathbf{x}_\star$ and N predictor values $\mathbf{x}$ (some vector where we choose to evaluate the *GP*), we can use the assumed covariance structure to write

$$p\left(\begin{array}{c}\delta(\mathbf{x}) \\ \delta(\mathbf{x}_\star)\end{array}\middle|\begin{array}{c}\mathbf{x} \\ \mathbf{x}_\star\end{array}\right) \sim \mathcal{N}\left(\begin{array}{c}\mathbf{0} \\ \mathbf{0}\end{array}, \begin{array}{cc}\mathbf{K_{xx}} & \mathbf{K_{xx_\star}} \\ \mathbf{K_{x_\star x}} & \mathbf{K_{x_\star x_\star}}\end{array}\right).$$

where, e.g., $\mathbf{K_{x_\star x}}$ is shorthand for the $N_\star \times N$ covariance matrix block with entries $K_{ij} = k(x_{\star,i}, x_j)$ etc. We now seek the conditional distribution $p(\delta(\mathbf{x})|\delta(\mathbf{x}_\star))$ for $\delta(\mathbf{x})$ given $\delta(\mathbf{x}_\star) = \delta_\star$, where $\delta_\star$ is an $N_\star \times 1$ column vector. One can prove that marginal and conditional distributions of two jointly normal distributions are themselves normal distributions.

Suppose $\mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2)$ are jointly normal functions with mean vector and covariance matrix

$$\mu = \begin{bmatrix}\mu_1 \\ \mu_2\end{bmatrix}, \quad \Sigma = \begin{bmatrix}\Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22}\end{bmatrix} \qquad \Sigma^{-1} \equiv \Lambda = \begin{bmatrix}\Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22}\end{bmatrix},$$

where we suppressed the dependence of some predictor variables $\mathbf{x}_1$ and $\mathbf{x}_2$ throughout i.e., $\mathbf{f}_1 = f(\mathbf{x_1})$ etc., then the marginal *PDF*s are given by

$$p(\mathbf{f}_1) = \mathcal{N}(\mathbf{f}_1|\mu_1, \Sigma_{11})$$
$$p(\mathbf{f}_2) = \mathcal{N}(\mathbf{f}_2|\mu_2, \Sigma_{22})$$

and the conditional *PDF* is given by

$$p(\mathbf{f}_1|\mathbf{f}_2) = \mathcal{N}(\mathbf{f}_1|\mu_{1|2}, \Sigma_{1|2}),$$

where the conditional mean and covariance are defined by

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{f}_2 - \mu_2)$$
$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} = \Lambda_{11}^{-1}$$

This equality for multivariate normal *PDF*s facilitates closed-form (numerical) evaluation of conditioned *GPs*. This is a highly useful feature of this type of stochastic process.

In our case, we obtain $p(\delta(\mathbf{x})|\delta(\mathbf{x}_\star)) = \mathcal{N}(\delta(\mathbf{x})|\mu_{\delta|\delta_\star}, \Sigma_{\delta|\delta_\star})$ with

$$\begin{aligned}\mu_{\delta|\delta_\star} &= \mathbf{K}_{\mathbf{x}\mathbf{x}_\star}\mathbf{K}_{\mathbf{x}_\star\mathbf{x}_\star}^{-1}\delta_\star \\ \Sigma_{\delta|\delta_\star} &= \mathbf{K}_{\mathbf{x}\mathbf{x}} - \mathbf{K}_{\mathbf{x}\mathbf{x}_\star}\mathbf{K}_{\mathbf{x}_\star\mathbf{x}_\star}^{-1}\mathbf{K}_{\mathbf{x}_\star\mathbf{x}}\end{aligned} \tag{9}$$

This expression, along with the *SQE* covariance function, is explicitly coded in the Jupyter notebook. The above expressions enable us to condition our *GP* for $\delta(\mathbf{x})$ on the known value $(\mathbf{x}_\star, \delta(\mathbf{x}_\star)$, e.g., $\mathbf{x}_\star = [0]$ and $\delta(0) = 0$. It turns out that this property/knowledge of the model discrepancy does not alter the posterior much, albeit it nudges the θ mode towards $\theta_T = 0.65$, see Figure 17.

2.5.4 Maximum prior information

We know that friction amounts to a monotonic decrease of the machine output, i.e., $\delta(x) \leq 0$ and the derivative $\delta'(x) \leq 0$ for all $x \geq 0$. This is clearly something that we should try to build into the model discrepancy. Unfortunately, global inequality constraints remove most of the computational benefits of the *GP*. However, we can incorporate point wise derivative constraints such that $\delta(x_i) < 0$ for a small set of points. A slightly more advanced method for handling monotonicity in a more general *GP* setting is presented in [10]. Most importantly we will incorporate $\delta'(0) = 0$. This actually define the physical meaning of θ as the slope of the true process $\zeta(x)$.

Exercise

Why is it most likely sufficient to include the constraint $\delta(x) \leq 0$ for ~ 2 values of x?

Since differentiation is a linear operation, i.e., the derivative of a linear combination of functions is equal to the linear combination of derivatives, the derivative of a Gaussian process is another Gaussian process. Thus we can use *GPs* to make predictions about derivatives, and also to make inference based on derivative information. The key step is to define the covariance matrix Σ of the joint distribution of the function $\mathbf{f}$ and its first derivative $\mathbf{f}^{[1]}$ with respect to $\mathbf{x}$

$$p\left(\begin{array}{c|c} \mathbf{f} & \mathbf{x} \\ \mathbf{f}^{[1]} & \mathbf{x}_\star \end{array}\right) \sim \mathcal{N}\left(\begin{array}{c} \mathbf{0} \\ \mathbf{0} \end{array}, \begin{array}{cc} \mathbf{K}_{\mathbf{x}\mathbf{x}}^{[00]} & \mathbf{K}_{\mathbf{x}\mathbf{x}_\star}^{[01]} \\ \mathbf{K}_{\mathbf{x}_\star\mathbf{x}}^{[10]} & \mathbf{K}_{\mathbf{x}_\star\mathbf{x}_\star}^{[11]} \end{array}\right).$$

In these expression we have consistently used the shorthand $K^{[\cdot]}$ for denoting various derivatives of the covariance function $k(x_i, x_j)$. The superscript $[00]$ denotes the zeroth derivative with respect to the first and second variable, whereas $[01]$ denotes zeroth derivative with respect to the first variable and the first derivative with respect to the second variable etc. The matrix elements of the first and mixed derivatives of the joint *SQE* covariance matrix are given by

$$\begin{aligned}\left(\mathbf{K}_{\mathbf{x}\mathbf{x}}^{[01]}\right)_{ij} &= \frac{\partial}{\partial x_j}k(x_i, x_j) = \frac{\partial}{\partial x_i}k(x_j, x_i) \\ \left(\mathbf{K}_{\mathbf{x}\mathbf{x}}^{[10]}\right)_{ij} &= \frac{\partial}{\partial x_i}k(x_i, x_j) = \frac{\partial}{\partial x_j}k(x_j, x_i) \\ \left(\mathbf{K}_{\mathbf{x}\mathbf{x}}^{[11]}\right)_{ij} &= \frac{\partial^2}{\partial x_i \partial x_j}k(x_i, x_j) = \frac{\partial^2}{\partial x_j \partial x_i}k(x_i, x_j)\end{aligned}$$

We can now jointly draw functions and their derivatives $\frac{df}{dx}$ from this distribution, see Figure 18 for an example. Try this in the notebook for yourself. All necessary functions are already coded, but make sure you understand the strategy as

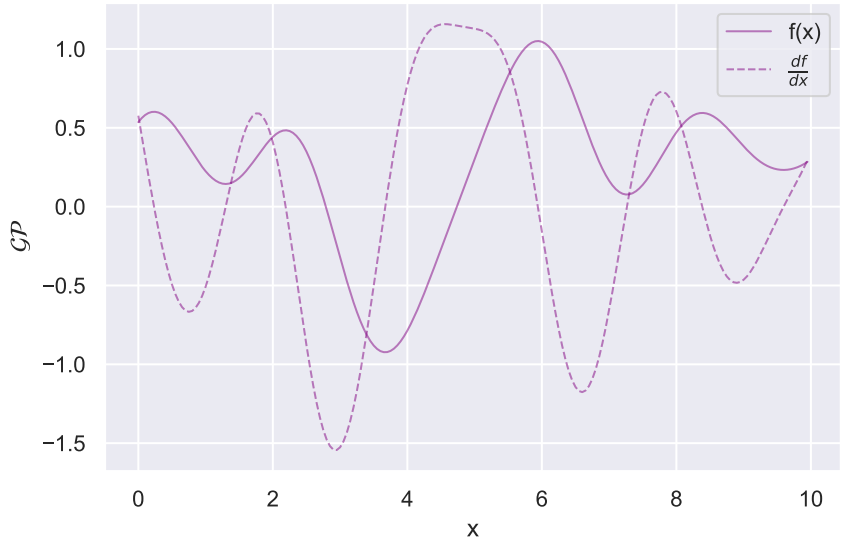


Fig. 18: Jointly normal function $f(x)$ and its first derivative $df(x)/dx$ from a *GP* with an *SQE* kernel where $\sigma^2_{GP} = \ell_{GP} = 1$.

well as the functionality in Python to devise matrices from vectors using the built-in operators for taking outer products and sums.

As before, we seek the joint distribution for $\delta(x)$ and $\delta'(x)$ (now using the conventional notation for the first derivative) that is also *conditioned* on a set of known $\delta(\mathbf{x}_\star)$ values. We could also setup the conditional distribution of $\delta(x)$ conditioned on known values for $\delta'(x)$, however we choose the former this time. We will do this by straightforwardly differentiating the mean and covariance in Eq. (9) with respect to $\mathbf{x}$, not $\mathbf{x}_\star$. Switching back to the superscript notation we obtain

$$\begin{aligned}
\mu^{[1]}_{\delta|\delta_\star} &= \mathbf{K}^{[1]}_{\mathbf{x}\mathbf{x}_\star}\mathbf{K}^{-1}_{\mathbf{x}_\star\mathbf{x}_\star}\delta_\star \\
\Sigma^{[01]}_{\delta|\delta_\star} &= \mathbf{K}^{[01]}_{\mathbf{x}\mathbf{x}} - \mathbf{K}_{\mathbf{x}\mathbf{x}_\star}\mathbf{K}^{-1}_{\mathbf{x}_\star\mathbf{x}_\star}\mathbf{K}^{[01]}_{\mathbf{x}_\star\mathbf{x}} \\
\Sigma^{[10]}_{\delta|\delta_\star} &= \mathbf{K}^{[10]}_{\mathbf{x}\mathbf{x}} - \mathbf{K}^{[10]}_{\mathbf{x}\mathbf{x}_\star}\mathbf{K}^{-1}_{\mathbf{x}_\star\mathbf{x}_\star}\mathbf{K}_{\mathbf{x}_\star\mathbf{x}} \\
\Sigma^{[11]}_{\delta|\delta_\star} &= \mathbf{K}^{[11]}_{\mathbf{x}\mathbf{x}} - \mathbf{K}^{[10]}_{\mathbf{x}\mathbf{x}_\star}\mathbf{K}^{-1}_{\mathbf{x}_\star\mathbf{x}_\star}\mathbf{K}^{[01]}_{\mathbf{x}_\star\mathbf{x}}.
\end{aligned}$$

A jointly-normal model-discrepancy, and its derivative, that is also constrained to reproduce known discrepancy values can now be drawn from the multivariate normal *PDF* $\mathcal{N}(\delta|\bar{\mu}, \bar{\Sigma})$ with mean vector and covariance matrix

$$\bar{\mu} = \begin{bmatrix} \mu_{\delta|\delta_\star} \\ \mu^{[1]}_{\delta|\delta_\star} \end{bmatrix}, \quad \bar{\Sigma} \begin{bmatrix} \Sigma^{[00]}_{\delta|\delta_\star} & \Sigma^{[01]}_{\delta|\delta_\star} \\ \Sigma^{[10]}_{\delta|\delta_\star} & \Sigma^{[11]}_{\delta|\delta_\star} \end{bmatrix}.$$

In Figure 19 we show an example of a *GP* that was conditioned on (noiseless) values at $(\mathbf{x}_\star, \delta_\star) = ([0, 2, 7], [0, -3, 4])$, and its derivative.

We impose the $\delta'(0) = 0$ constraint by conditioning $\delta([0, 0.01, 0.05]) = \delta_\star = [0, 0, 0]$. The monotonicity constraint is approximately enforced by only allowing $\delta(x)_{\delta|\delta_\star}$ draws with $\delta'(x_1) < 0$ and $\delta(x_4) < 0$, where $x_1 = 0.68$ and $x_4 = 2.12$ are the first and fourth predictor value in the $N_d = 10$ data set. This last step is implemented with an 'if' statement inside the log-likelihood function that we sample using `emcee`.

Sampling the posterior requires now a few more steps than before. The introduction of more constraints appear to induce more correlated steps and a lower acceptance rate. For the results presented here, we took 15,000 *MCMC* steps using `emcee` (`ndim=4`, `nwalkers=200`) starting at $[\theta, \sigma^2, \sigma^2_{GP}, \ell] = [1, 10^{-3}, 2 \cdot 10^{-2}, 2.5]$ which takes about 60 minutes to finish on a typical laptop. The corner plot for the final posterior is shown in the Figure 20.

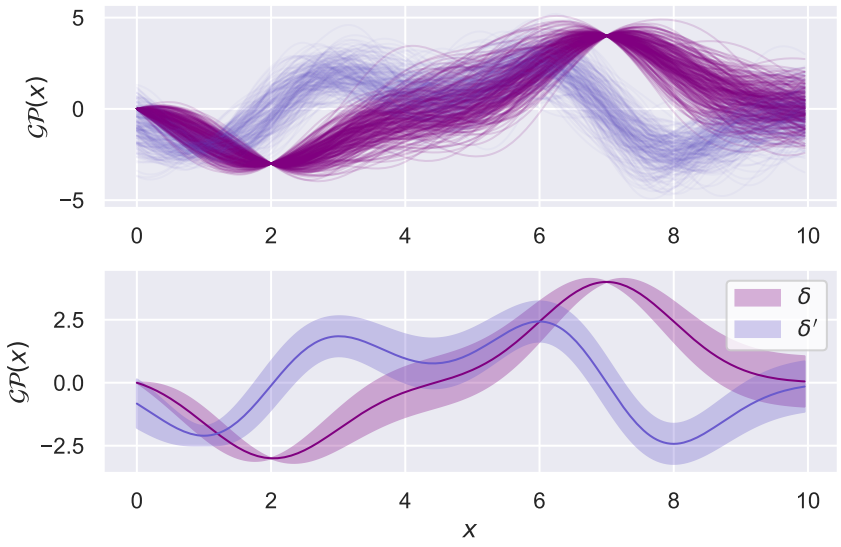


Fig. 19: 250 jointly normal draws of a function $f(x)$, conditioned to pass through three locations, and the corresponding first derivatives $df(x)/dx$. The *GP* is based on an *SQE* kernel where $\sigma^2_{GP} = \ell_{GP} = 1$.

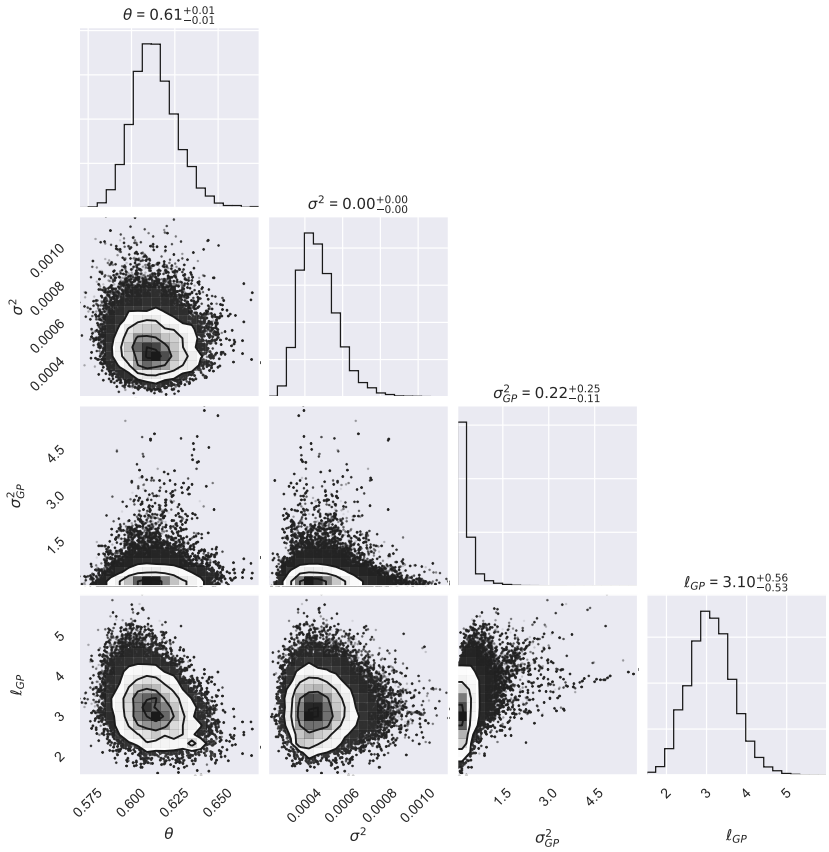


Fig. 20: Posterior *PDF* for the simple machine with $N_d = 10$. Parameter prior: $\mathcal{NIG}(\theta, \sigma^2|\mu_0 = 1, \Sigma_0 = 5000, \alpha_0 = 19.5, \beta_0 = 7.4 \cdot 10^{-3})$. *GP* prior: derivative-constrained *SQE* kernel with hyperprior $\mathcal{IG}(\sigma^2_{GP}|\alpha_0 = 1.67, \beta_0 = 0.24)$ and $\mathcal{G}(\ell_{GP}|\alpha_0 = 18, \beta_0 = 0.17)$

The marginal *PDF* for θ approaches the true value θ_T now that we are starting to build in realistic prior information about the model discrepancy, see Figure 17. We end by drawing 500 realizations from the Bayesian calibration of the model including a maximally constrained model discrepancy, see Figure 21.

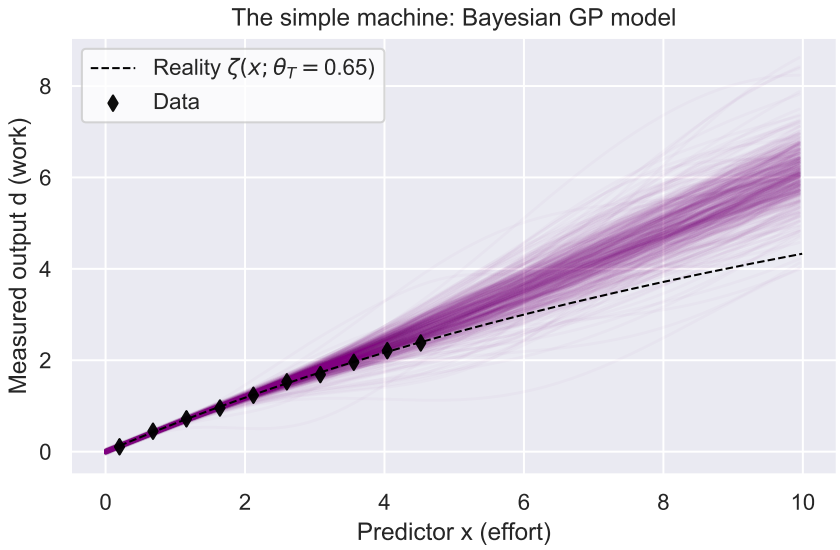


Fig. 21: Our model for the simple machine. Calibrated using a *NIG* prior for the parameters and a constrained discrepancy term that was modeled using a derivative-conditioned *GP* based on an *SQE* covariance function.

Discussion

What are your conclusions and comments about the model that we now have at our disposal?

2.6 What did we learn?

To make realistic inferences about physical parameters we need to introduce a physically motivated model discrepancy term. We saw first hand that it is only when introducing *GP*s constrained on physically relevant prior information (derivatives in our case) that the posterior credible intervals cover the true value. The introduction of a model discrepancy does not necessarily improve predictions. In order to obtain a realistic extrapolation we need realistic prior information about the discrepancy term also in the vicinity of the corresponding control variables for the data that we wish to predict. Or improve our theories and construct better models such that the discrepancy term gets smaller.

2.7 Appendix: Bayesian linear regression

This appendix summarizes one way for deriving a closed form expression for the posterior when facing a normal likelihood, a linear model, and a *NIG* prior. We apply the final expression, Eq. (4), in the main text.

To derive this, we begin by combining the likelihood and the prior in Eq. (1) and Eq. (3), respectively. Applying Bayes'

theorem immediately tells us that the posterior is proportional[1] to

$$\begin{aligned} p(\theta, \sigma^2|\mathcal{D}) \propto & \frac{\beta_0^{\alpha_0}}{(2\pi)^{(N_p+N_d)/2}\Gamma(\alpha_0)|\Sigma_0|^{1/2}} \\ & \times \frac{1}{\sigma^2}^{[\alpha_0+(N_p+N_d)/2+1]} \\ & \times \exp\left\{-\frac{1}{\sigma^2}\left[\beta_0 + \frac{1}{2}\underbrace{\left((\theta-\mu_0)^T\Sigma_0^{-1}(\theta-\mu_0) + (\mathcal{D}-\Phi\theta)^T(\mathcal{D}-\Phi\theta)\right)}_{\equiv E}\right]\right\}, \end{aligned} \tag{10}$$

where we also defined a shorthand E for part of the exponent. Expanding the products in E gives

$$\begin{aligned} E = & \theta^T\Sigma_0^{-1}\theta + \mu_0^T\Sigma_0^{-1}\mu_0 - 2\theta^T\Sigma_0^{-1}\mu_0 \\ & \mathcal{D}^T\mathcal{D} + \theta^T\Phi^T\Phi\theta - 2\theta^T\Phi^T\mathcal{D} \end{aligned}$$

Let us collect the terms that are linear and quadratic in θ.

$$E = \theta^T(\Sigma_0^{-1} + \Phi^T\Phi)\theta - 2\theta^T(\Sigma_0^{-1}\mu_0 + \Phi^T\mathcal{D}) + \mu_0^T\Sigma_0^{-1}\mu_0 + \mathcal{D}^T\mathcal{D}$$

One can show the following matrix identity[2] for a symmetric matrix $\mathbf{A}$ and vectors $\mathbf{x}$ and $\mathbf{a}$

$$\mathbf{x}^T\mathbf{A}\mathbf{x} - 2\mathbf{x}^T\mathbf{a} = (\mathbf{x} - \mathbf{A}^{-1}\mathbf{a})^T\mathbf{A}(\mathbf{x} - \mathbf{A}^{-1}\mathbf{a}) - \mathbf{a}^T\mathbf{A}^{-1}\mathbf{a}. \tag{11}$$

If we assume that our design matrix Φ is invertible, i.e., that there are no linearly dependent columns, then the product $\Phi^T\Phi$ is a symmetric matrix[3]. Then we can apply the identity in Eq. (11) to our exponent term E. This will allow us to rewrite E such that the structure of a normal exponential plus a constant becomes apparent. This will be the first step towards identifying the *NIG* structure of the posterior in Eq. (10). We obtain

$$\begin{aligned} E = & \left(\theta - [\Sigma_0^{-1} + \Phi^T\Phi]^{-1}[\Sigma_0^{-1}\mu_0 + \Phi^T\mathcal{D}]\right)^T [\Sigma_0^{-1} + \Phi^T\Phi]\left(\theta - [\Sigma_0^{-1} + \Phi^T\Phi]^{-1}[\Sigma_0^{-1}\mu_0 + \Phi^T\mathcal{D}]\right) \\ & - [\Sigma_0^{-1}\mu_0 + \Phi^T\mathcal{D}]^T[\Sigma_0^{-1} + \Phi^T\Phi]^{-1}[\Sigma_0^{-1}\mu_0 + \Phi^T\mathcal{D}] + \mu_0^T\Sigma_0^{-1}\mu_0 + \mathcal{D}^T\mathcal{D}. \end{aligned}$$

Now, only the first term depends on θ. Using the tilde-shorthand defined in Eq. (4), we can write the entire exponential factor in Eq. (10) as

$$\exp\left\{-\frac{1}{\sigma^2}\left[\tilde{\beta} + \frac{1}{2}(\theta - \tilde{\mu})^T\tilde{\Sigma}(\theta - \tilde{\mu})\right]\right\}. \tag{12}$$

To obtain the full posterior *PDF* in Eq. (4) we have to normalize the expression in Eq. (10). For this we have to compute the marginal likelihood integral given by

$$\begin{aligned} p(\mathcal{D}) &= \int p(\mathcal{D}|\theta, \sigma^2)p(\theta, \sigma^2)\, d\theta d\sigma^2 \\ &= \int p(\mathcal{D}|\theta, \sigma^2)\mathcal{N}(\theta|\mu_0, \sigma^2\Sigma_0)\mathcal{IG}(\sigma^2|\alpha_0, \beta_0)\, d\theta d\sigma^2. \end{aligned} \tag{13}$$

We will start with the integral over θ and compute the likelihood $p(\mathcal{D}|\sigma^2)$. The product of two normal distributions can be re-written by completing the square, yielding an integrand consisting of a single normal distribution, with a known normalization constant. As an alternative, we will demonstrate another route. This requires the following known relation: For a normally distributed vector $\mathbf{x} \sim \mathcal{N}(\mu_x, \Sigma_x)$, the linearly transformed vector $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{N\times N}$, is also normally distributed with $\mathbf{y} \sim \mathcal{N}(\mathbf{A}\mu_x + \mathbf{b}, \mathbf{A}\Sigma_x\mathbf{A}^T)$. For our linear model $\Phi\theta + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2\mathbf{1})$ we have independently assumed

$$\theta \sim \mathcal{N}(\mu_0, \sigma^2\Sigma_0)$$

[1] Indeed, it is not yet normalized.

[2] This is the matrix analog of *completing the square* that is used over and over for working out integrals and products of exponential *PDF*s.

[3] Remember that for a symmetric matrix we have $\mathbf{A}^{-1} = (\mathbf{A}^{-1})^T$.

which then implies

$$\mathcal{D} \sim \mathcal{N}(\Phi\mu_0, \sigma^2(\mathbf{1} + \Phi\Sigma_0\Phi^T)).$$

Thus we obtain the *PDF* for the σ^2-conditional marginal likelihood

$$p(\mathcal{D}|\sigma^2) = \mathcal{N}(\mathcal{D}|\Phi\mu_0, \sigma^2(\mathbf{1} + \Phi\Sigma_0\Phi^T)).$$

This is a normal distribution, as expected. Multiplying this with the remaining inverse-gamma prior for σ^2 and integrating over σ^2 will yield the marginal likelihood. We perform this integration next. However, we will do the slightly more agnostic integral $\mathcal{I}$ with an unspecified covariance $\sigma^2\Sigma_0$. In the following expressions we suppress the zero subscript for α and β.

$$\begin{aligned} I &= \int \mathcal{N}(\mathcal{D}|\mu, \sigma^2\Sigma_0)\mathcal{I}\mathcal{G}(\sigma^2|\alpha, \beta)d\sigma^2 \\ &= \frac{\beta^\alpha}{(2\pi)^{N_d/2}\Gamma(\alpha)|\Sigma_0|^{1/2}} \\ &\quad \times \int \left(\frac{1}{\sigma^2}\right)^{\alpha+1+N_d/2} \exp\left\{-\frac{1}{\sigma^2}\left[\beta + \frac{1}{2}(\mathcal{D}-\mu)^T\Sigma_0^{-1}(\mathcal{D}-\mu)\right]\right\} d\sigma^2 \end{aligned}$$

To de-clutter this expression, we write the integral $\mathcal{I}$

$$\mathcal{I} = A\int_0^\infty \left(\frac{1}{\sigma^2}\right)^C \exp\left\{-\frac{B}{\sigma^2}\right\} d\sigma^2$$

with

$$\begin{aligned} A &= \frac{\beta^\alpha}{(2\pi)^{N_d/2}\Gamma(\alpha)|\Sigma_0|^{1/2}} \\ B &= \beta + \frac{1}{2}(\mathcal{D}-\mu)^T\Sigma_0^{-1}(\mathcal{D}-\mu) \\ C &= \alpha + 1 + N_d/2. \end{aligned}$$

A change of variables $t = B/\sigma^2$ leads to

$$\begin{aligned} \mathcal{I} &= A\int_0^\infty t^{C-2}B^{1-C}\exp\{-t\}\,dt \\ &= AB^{1-C}\int_0^\infty t^{C-2}\exp\{-t\}\,dt \\ &= AB^{1-C}\Gamma(C-1). \end{aligned}$$

If we now restore the original notation and consolidate factors we obtain

$$\begin{aligned} \mathcal{I} &= \frac{\beta^\alpha\Gamma(\alpha+N_d/2)}{(2\pi)^{N_d/2}\Gamma(\alpha)|\Sigma_0|^{1/2}}\left[\beta + \frac{1}{2}(\mathcal{D}-\mu)^T\Sigma_0^{-1}(\mathcal{D}-\mu)\right]^{-(\alpha+N_d/2)} \\ &= \frac{\beta^\alpha\Gamma(\alpha+N_d/2)}{(2\pi)^{N_d/2}\Gamma(\alpha)|\Sigma_0|^{1/2}}\beta^{-(\alpha+N_d/2)}\left[1 + \frac{1}{2\beta}(\mathcal{D}-\mu)^T\Sigma_0^{-1}(\mathcal{D}-\mu)\right]^{-(\alpha+N_d/2)} \\ &= \frac{\Gamma(\alpha+N_d/2)}{\pi^{N_d/2}\Gamma(\alpha)|2\alpha\frac{\beta}{\alpha}\Sigma_0|^{1/2}}\left[1 + \frac{(\mathcal{D}-\mu)^T(\frac{\beta}{\alpha}\Sigma_0)^{-1}(\mathcal{D}-\mu)}{2\alpha}\right]^{-(\alpha+N_d/2)}. \end{aligned}$$

The final *PDF* in the last line is called a (multivariate) t-distribution. If we introduce the notation $\nu = 2\alpha$ and $\Sigma = (\beta/\alpha)\Sigma_0$, we obtain the standard expression for the *PDF* of a t-distributed vector quantity $\mathbf{Y}$ (with p components, i.e., we also set $N_d = p$),

$$\mathcal{T}_\nu(\mathbf{Y}|\mu, \Sigma) = \frac{\Gamma((\nu+p)/2)}{\pi^{p/2}\Gamma(\nu/2)|\nu\Sigma|^{1/2}}\left[1 + \frac{(\mathbf{Y}-\mu)^T\Sigma^{-1}(\mathbf{Y}-\mu)}{\nu}\right]^{-(\nu+p)/2}.$$

The marginal likelihood can be expressed as

$$\begin{aligned} p(\mathcal{D}) &= \int \mathcal{N}(\mathcal{D}|\Phi\mu_0, \sigma^2(\mathbf{1}+\Phi\Sigma_0\Phi^T))\mathcal{IG}(\sigma^2|\alpha_0,\beta_0)d\sigma^2 = \\ &= \mathcal{T}_{2\alpha_0}(\mathcal{D}|\Phi\mu_0, \frac{\beta_0}{\alpha_0}(\mathbf{1}+\Phi\Sigma_0\Phi^T)). \end{aligned} \tag{14}$$

We now have all the pieces of the puzzle, i.e., Eqs. (10), (12), and (14), to form the normalized posterior using Bayes' theorem

$$p(\theta,\sigma^2|\mathcal{D}) = \frac{p(\mathcal{D}|\theta,\sigma^2)p(\theta,\sigma^2)}{p(\mathcal{D})}.$$

To evaluate the right-hand side, we begin by re-writing the denominator to match the form of the numerator.

$$\begin{aligned} p(\mathcal{D}) &= \mathcal{T}_{2\alpha_0}(\mathcal{D}|\Phi\mu_0, \frac{\beta_0}{\alpha_0}(\mathbf{1}+\Phi\Sigma_0\Phi^T)) \\ &= \frac{\Gamma(\alpha_0+N_d/2)}{\pi^{N_d/2}\Gamma(\alpha_0)|2\alpha_0\frac{\beta_0}{\alpha_0}(\mathbf{1}+\Phi\Sigma_0\Phi^T)|^{1/2}} \\ &\quad\times\left[1+\frac{\left(\mathcal{D}-\Phi\mu_0\right)^T(\frac{\beta_0}{\alpha_0}(\mathbf{1}+\Phi\Sigma_0\Phi^T))^{-1}\left(\mathcal{D}-\Phi\mu_0\right)}{2\alpha_0}\right]^{-\alpha_0-N_d/2} \end{aligned} \tag{15}$$

We can use the generalized matrix determinant lemma[4] to evaluate the determinant in the denominator of the first factor in Eq. (15),

$$\begin{aligned} |2\alpha_0\frac{\beta_0}{\alpha_0}(\mathbf{1}+\Phi\Sigma_0\Phi^T)|^{1/2} &= 2\beta_0^{N_d/2}|\mathbf{1}+\Phi\Sigma_0\Phi^T|^{1/2} \\ &= 2\beta_0^{N_d/2}|\Sigma_0^{-1}+\Phi^T\Phi|^{1/2}|\Sigma_0|^{1/2} \\ &= 2\beta_0^{N_d/2}\frac{|\Sigma_0|^{1/2}}{|\tilde{\Sigma}|^{1/2}}, \end{aligned}$$

where we used the definition of $\tilde{\Sigma}$ according to Eq. (4).

The exponential term of the unnormalized posterior, Eq. (12), is written compactly using the tilde-notation defined in Eq. (4). We therefore need to express the second factor of the marginal likelihood in Eq. (15) using the same shorthand. In fact, one can show the following equality.

$$\left(\mathcal{D}-\Phi\mu_0\right)^T(\mathbf{1}+\Phi\Sigma_0\Phi^T)^{-1}\left(\mathcal{D}-\Phi\mu_0\right) = \mathcal{D}^T\mathcal{D}+\mu_0^T\Sigma_0^{-1}\mu_0-\tilde{\mu}^T\tilde{\Sigma}^{-1}\tilde{\mu} \tag{16}$$

To do so requires a few intermediate steps. So let us repeat them here. We will manipulate the expression above such that the right-hand equals the left-hand side. Begin by expanding the last term on the right-hand side

$$\begin{aligned} \tilde{\mu}^T\tilde{\Sigma}^{-1}\tilde{\mu} &= [\Sigma_0^{-1}\mu_0+\Phi^T\mathcal{D}]^T\tilde{\Sigma}^T\tilde{\Sigma}^{-1}\tilde{\Sigma}[\Sigma_0^{-1}\mu_0+\Phi^T\mathcal{D}] \\ &= \mu_0^T\Sigma_0^{-1}\tilde{\Sigma}\Sigma_0^{-1}\mu_0+\mathcal{D}^T\Phi\tilde{\Sigma}\Phi^T\mathcal{D}+2\mathcal{D}^T\Phi\tilde{\Sigma}\Sigma_0^{-1}\mu_0. \end{aligned}$$

To get the second equality we also used that $\tilde{\Sigma}^T = \tilde{\Sigma}$. The above leads to the following expression for the entire right-hand side in Eq. (16).

$$\mathcal{D}^T(\mathbf{1}-\Phi\tilde{\Sigma}\Phi^T)\mathcal{D}+\mu_0^T(\Sigma_0^{-1}-\Sigma_0^{-1}\tilde{\Sigma}\Sigma_0^{-1})\mu_0-2\mathcal{D}^T\Phi\tilde{\Sigma}\Sigma_0^{-1}\mu_0 \tag{17}$$

[4] $|\mathbf{A}+\mathbf{BCD}^T| = |\mathbf{C}^{-1}+\mathbf{D}^T\mathbf{A}^{-1}\mathbf{B}||\mathbf{C}||\mathbf{A}|$

We can use the Woodbury identity[5] twice to re-express the middle factor of the second term using mainly the design matrix. This will enable us to form the product $\Phi\mu_0$ that we find in the left-hand side of Eq. (16).

$$\begin{aligned}\Sigma_0^{-1} - \Sigma_0^{-1}\tilde{\Sigma}\Sigma_0^{-1} &= \Sigma_0^{-1} - \Sigma_0^{-1}[\Sigma_0^{-1} + \Phi^T\Phi]^{-1}\Sigma_0^{-1} \\ &= (\Sigma_0^{-1} + \mathbf{1}(\Phi^T\Phi)^{-1}\mathbf{1})^{-1} \\ &= \Phi^T\Phi - \Phi^T\Phi(\Sigma_0^{-1} + \Phi^T\Phi)^{-1}\Phi^T\Phi \\ &= \Phi^T(\mathbf{1} - \Phi\tilde{\Sigma}\Phi^T)\Phi\end{aligned}$$

In the second equality we used Woodbury "backwards" to deflate the sum of prior covariances, and in the third equality we used it "forwards" to inflate the expression again but this time in terms of the square of the design matrix.

We now find the common factor $(\mathbf{1} - \Phi\tilde{\Sigma}\Phi^T)$ in all terms of Eq. (17) except the last one. However, we can re-write $\Phi\tilde{\Sigma}\Sigma_0^{-1}$.

$$\tilde{\Sigma}\underbrace{[\Sigma_0^{-1} + \Phi^T\Phi]}_{\tilde{\Sigma}^{-1}} = \mathbf{1} \Rightarrow \tilde{\Sigma}\Sigma_0^{-1} = \mathbf{1} - \tilde{\Sigma}\Phi^T\Phi \Rightarrow \Phi\tilde{\Sigma}\Sigma_0^{-1} = (\mathbf{1} - \Phi\tilde{\Sigma}\Phi^T)\Phi$$

With this in place, we can re-write Eq. (17) as a familiar sum of squared residuals

$$\left(\mathcal{D} - \Phi\mu_0\right)^T(\mathbf{1} - \Phi\tilde{\Sigma}\Phi^T)\left(\mathcal{D} - \Phi\mu_0\right).$$

We now use the Woodbury identity one last time to re-write the middle factor

$$(\mathbf{1} + \Phi\Sigma_0\Phi^T)^{-1} = \mathbf{1} - \Phi(\Sigma_0^{-1} + \Phi^T\Phi)^{-1}\Phi^T = \mathbf{1} - \Phi\tilde{\Sigma}\Phi^T,$$

and we have (finally) demonstrated the equality in Eq. (16).

Going back to the expression for the marginal likelihood in terms of the multivariate t-distribution, we now have

$$p(\mathcal{D}) = \frac{\Gamma(\alpha_0 + N_d/2)|\tilde{\Sigma}|^{1/2}}{\pi^{N_d/2}\Gamma(\alpha_0)2\beta_0^{N_d/2}|\Sigma_0|^{1/2}}\left[\mathbf{1} + \frac{1}{2\beta_0}\left(\mathcal{D}^T\mathcal{D} + \mu_0^T\Sigma_0^{-1}\mu_0 - \tilde{\mu}^T\tilde{\Sigma}^{-1}\tilde{\mu}\right)\right]^{-\alpha_0 - N_d/2}.$$

Using the definitions of $\tilde{\alpha}$ and $\tilde{\beta}$ according to Eq. (4) yields

$$p(\mathcal{D}) = \frac{\Gamma(\tilde{\alpha})|\tilde{\Sigma}|^{1/2}\beta_0^{\alpha_0}}{(2\pi)^{N_d/2}\Gamma(\alpha_0)|\Sigma_0|^{1/2}}\tilde{\beta}^{-\tilde{\alpha}}. \tag{18}$$

Several factor cancel when we divide the un-normalized posterior in Eq. (10) with the above, and the final expression for the posterior *PDF* is given by the *NIG* distribution given in Eq. (4):

$$\begin{aligned}p(\theta, \sigma^2|\mathcal{D}) =& \frac{\tilde{\beta}^{\tilde{\alpha}}}{(2\pi)^{N_p/2}\Gamma(\tilde{\alpha})|\tilde{\Sigma}|^{1/2}} \\ &\times \frac{1}{\sigma^2}^{[\tilde{\alpha}+N_p/2+1]}\exp\left\{-\frac{1}{\sigma^2}\left[\tilde{\beta} + \frac{1}{2}\left(\theta - \tilde{\mu}\right)^T\tilde{\Sigma}\left(\theta - \tilde{\mu}\right)\right]\right\}\end{aligned}$$

Using Eq. (14) we obtain the posterior marginal for θ

$$p(\theta|\mathcal{D}) = \int \mathcal{NIG}(\theta, \sigma^2|\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta})\, d\sigma^2 = \mathcal{T}_{2\tilde{\alpha}}(\theta|\tilde{\mu}, (\tilde{\beta}/\tilde{\alpha})\tilde{\Sigma}),$$

and the posterior marginal for σ^2 follows straightforwardly from integration

$$p(\sigma^2|\mathcal{D}) = \int \mathcal{NIG}(\theta, \sigma^2|\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta})\, d\theta = \mathcal{IG}(\sigma^2|\tilde{\alpha}, \tilde{\beta}).$$

[5] $(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{C}^{-1} + \mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1}$

CHAPTER

THREE

MODEL SELECTION

Key take-away messages

- p-values and traditional significance testing can at best only reject a null hypothesis.
- In a Bayesian formulation we can compare several hypotheses.
- The marginal likelihood (Z) can be used for model comparison and model selection. In fact, it is the gold-standard for such analyses.
- However, it is very often impossible to evaluate Z analytically, and almost always challenging to extract Z numerically.
- There exists a range of different so-called information criteria to approximate model scores.

Before reading this chapter

Refresh your memory of the following concepts:

- Ordinary least squares (*OLS*)
- Weighted least squares (*WLS*)
- Marginalization of *pdf*s
- Posteriors and Bayesian linear regression

3.1 More models

We will stay with the data for the simple machine. However, we will now consider a few more (perhaps physically inspired) models to understand this data. Previously, we used a purely linear model

$$M_0(\theta; x) = \theta_0 x.$$

Let us now assume that we have two more models M_1 and M_2 for the simple machine

$$M_1(\theta; x) = \theta_0 x + \theta_1 x^2$$
$$M_2(\theta; x) = \theta_0 x + \theta_1 x^2 + \theta_2 x^3.$$

It is not an uncommon situation to have more than one model to explain data and predict future data. In physics, very often the models are a bit more complicated than the ones presented above; we will some examples of such models in Chapter 6. However, for the purpose of discussing model selection and associated methods, these simple models will

suffice. The situation of model selection extends to the case of having several machine learning models that have been trained to the same data, and one would like to move forward with only one of the models. The overarching question is: How can we rank and select models?

To make the situation slightly more realistic, we will now assume that we have some quantified experimental uncertainty σ_e ("error bars") in each datum, so called *heteroscedastic* errors. The $N_d = 20$ data we obtained from measurements of the simple machine is shown in Figure 1.

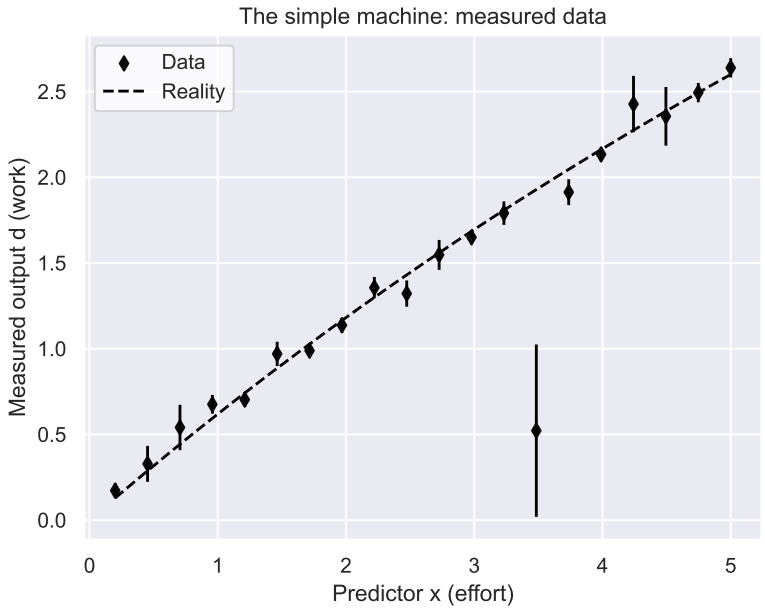


Fig. 1: $N_d = 10$ measurements of the work produced by the simple machine. The measurement process introduced some errors in the data.

We will model the uncertainty of each datum independently from all other data points using a normal distribution $\epsilon_i \sim \mathcal{N}(0, \sigma^2/w_i)$. Here, we assume that data points d_i with smaller variance obtain a larger weight by setting $w_i \propto 1/\sigma_e^2$. As we mentioned already in Chapter 1, the weighted least squares (*WLS*) estimator for θ is given by

$$\hat{\theta} = (\Phi^T \mathbf{E}^{-1} \Phi)^{-1} \Phi^T \mathbf{E}^{-1} \mathcal{D},$$

where the data covariance matrix $\mathbf{E}$ is a diagonal matrix proportional to σ_e^2. For *WLS*, and weighted Bayesian regression, we introduce a diagonal weight matrix $\mathbf{W}$ with entries $(w_1, w_2, \ldots, w_{N_d})$ along the diagonal with $w_i = 1/\sigma_e^2$. The weights are then normalized such that $\sum_i w_i = N_d$. This way, in the case of equal weights the *WLS* analysis reduces to the *OLS* analysis. Clearly, the renormalization of the weights do not impact the *WLS* estimator $\hat{\theta}$, and we could also write

$$\hat{\theta} = (\Phi^T \mathbf{W} \Phi)^{-1} \Phi^T \mathbf{W} \mathcal{D}.$$

Following standard linear regression theory, we can also compute an estimate of the *WLS* variance

$$s^2 = \frac{\hat{\varepsilon}^T \mathbf{W} \hat{\varepsilon}}{N_d - N_p}.$$

This is the variance of the data according to our model. The variance of the *WLS* estimator for the parameters is given by

$$\mathrm{Var}(\hat{\theta}) = s^2 \cdot \Phi^T \mathbf{W} \Phi,$$

which we can use to extract the parameter confidence intervals (see Chapter 1) according to

$$I_\alpha(\theta) = [\hat{\theta} - \sqrt{s^2 \cdot (\Phi^T \mathbf{W} \Phi)^{-1}} \cdot t_{N_d - N_p}(1-\alpha/2) \\ \leq \theta \leq \hat{\theta} + \sqrt{s^2 \cdot (\Phi^T \mathbf{W} \Phi)^{-1}} t_{N_d - N_p}(1-\alpha/2)]$$

where $t_{N_d-N_p}(q)$ is the inverse of the cumulative distribution function for the Student's t-distribution. In the limit $(N_d - N_p) \to \infty$, and $\alpha = 0.05$, we have $t_{N_d-N_p}(1-\alpha/2) \approx 1.96$. With $\alpha = 0.1$ we get $t_{N_d-N_p}(1-\alpha/2) \approx 1.64$. The *WLS* analysis of models M_0, M_1, and M_2 leads to the parameter estimates in Table 1. The *WLS* model predictions, and accompanying prediction intervals are shown in Figure 2.

Table 1: Parameter estimates and 90% confidence intervals for models M_0, M_1, and M_2.

Model	$\hat{\theta}$	$I_{\alpha=0.1}(\theta)$
M_0	(0.545)	$\theta \in ([0.532, 0.558])$
M_1	$(0.619, -0.020)$	$\theta \in ([0.588, 0.650], [-0.028, -0.012])$
M_2	$(0.693, -0.071, 0.008)$	$\theta \in ([0.618, 0.769], [-0.119, -0.023], [0.000, 0.015])$

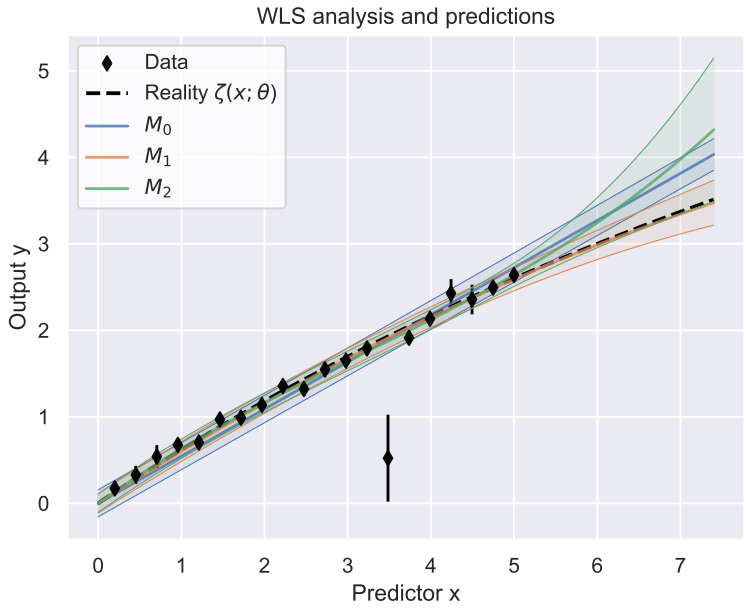


Fig. 2: $N_d = 20$ measurements (black diamonds) of the work produced by the simple machine (dashed black line). The resulting predictions (solid lines) are based on a linear model with an *WLS* point estimate of the model parameters $\hat{\theta}$. The bands indicate the 90% prediction intervals (*PI*s) for each model M_i, i.e., the interval in which a future observation will fall with 90% probability, given the existing data and model.

In the notebook you will find Python code for *WLS* fitting as well as a running example of how to use the Python package `statsmodels` for *WLS*. However, the above information is of little help for ranking and selecting the best model to explain the existing data and hopefully maximize our chances of accurately predicting future data. In this Chapter we briefly will cover some traditional methods based on significance testing and p-values. But this is mainly to set the stage for two other approaches for model comparison:

- Bayes factors $p(M_i|\mathcal{D}, I)$ for model M_i given data $\mathcal{D}$
- Information criteria (*IC*s) for the model given the data. We will look at the Bayesian IC (*BIC*), the Akaike IC (*AIC*), and in the *next chapter* discuss extensions applicable to *MCMC* chains.

3.2 p-values and the null hypothesis

The p-value is the probability for getting a value of the test statistic at least as extreme as that observed solely by chance, provided the null hypothesis $H_0$is true.

- **Null hypothesis** H_0: The fundamental assumption of "no effect". In a sense the opposite of whatever you are trying to measure.
- **Test statistic**: Any appropriate measure/number to quantify the discrepancy between the data and the null hypothesis.

We can never prove H_0 to be true, only reject it or fail to reject it. This is based on the principle of falsification[1]. The *smaller* the p-value the *stronger* the evidence against H_0. Given a regression model M we can setup a null hypothesis

$$H_0 : \text{the predictor } \theta_i \text{ in model } M_j \text{ is zero}$$

and compute the corresponding p-value. This is called a t-test for each parameter of the model. For the null hypothesis we will use a test statistic $f_j = (\hat{\theta}_j - [\theta_j = 0])/\sqrt{\text{Var}(\theta_j)}$, and compute the p-value by integrating the total probability for finding a value at least as extreme as f_j according to the $T_{\nu=N_d-N_p}$ distribution. Remember that the predictor θ_j can deviate from zero with positive or negative values, so we should add both integrated tails of the T-distribution. If the p-value is larger than some threshold α, then we cannot reject the null hypothesis. This can be repeated for all predictors in the model and tell us something about the individual significance of each θ_j. This type of significance testing can be extended to include all predictors. There is nothing wrong with the definition of a p-value. However, it can easily be misinterpreted and misused. The above testing cannot *prove* H_0 or *any* other hypothesis. Also, the common choices of a 0.05 or 0.1 significance level is just a convention. There is absolutely nothing fundamental about it.

The (two-sided) p-values for the parameters in models M_0, M_1, and M_2 are all smaller than the $\alpha = 0.1$ level, except in M_2 where we fail to reject the null for parameters θ_1 and θ_2, see Table 2.

Table 2: Summary of the p-values p_i for the null hypothesis H_0 *the predictor* θ_i *in model* M_j *is zero*.

Model	p_0	p_1	p_2
M_0	$8.4 \cdot 10^{-25}$	•	•
M_1	$6.8 \cdot 10^{-18}$	$4.0 \cdot 10^{-4}$	•
M_2	$1.0 \cdot 10^{-11}$	$2.0 \cdot 10^{-2}$	$8.1 \cdot 10^{-2}$

One must now be very careful to not make any false claims based on the above results. For all parameters in M_0 and M_1 we found that for a true H_0 it is highly unlikely for the test statistics of the collected data to exhibit a value at least this large. For model parameter θ_2 in M_2 we found insufficient evidence to reject a true H_0. The large p-value for this parameter (0.081) is *not* the probability that H_0 is true. To interpret a p-value as the probability that the null hypothesis is correct is a very common misinterpretation. But we cannot attribute a probability to the hypothesis itself. This is impossible from a frequentist perspective since a model/hypothesis is not a random object with a natural frequency interpretation. One can devise more complex null hypotheses in regression analyses to test the correlation structure of all parameters at the same time. A so-called $F-$test. There also exists so-called likelihood ratio methods for ranking models. However, the models must be nested, as in our case of progressively adding polynomial predictors, for such methods to be applicable.

[1] *All crows are black* is impossible to prove. But we can disprove it by presenting observational data of at least one non-black crow. In a different scenario, we might have counted 1 million crows, all black, and just failed to record a white crow insofar. Thus, we can only say that we *fail to reject/disprove* or *reject/disprove*.

3.3 Bayesian hypothesis testing

In a Bayesian approach we can attach a probability $p(M_i|\mathcal{D}, I)$ to a model M_i itself. We should be clear that $p(M_i|\mathcal{D}, I)$ alone is a meaningless number in an absolute sense, but we can, and will, compare probabilities of multiple models. This is a clear advantage of using Bayesian statistics. But we still have to evaluate

$$p(M_i|\mathcal{D}, I) = \frac{p(\mathcal{D}|M_i, I)p(M_i|I)}{p(\mathcal{D}|I)}$$

where the numerator contains the marginal likelihood of the data given model M_i (and any other information)

$$p(\mathcal{D}|M_i, I) = \int d\theta\, p(\mathcal{D}|\theta, M_i, I)p(\theta|M_i, I).$$

This is a challenging integral to evaluate. Still, it is certainly worthwhile trying. In TIF285 *Learning from Data* you discovered the usefulness of Bayes factors; the ratio $B_{i,j}$ of marginal likelihoods of model M_i and model M_j

$$B_{i,j} = \frac{p(\mathcal{D}|M_i, I)}{p(\mathcal{D}|M_j, I)} = \frac{p(M_i|\mathcal{D})}{p(M_j|\mathcal{D})} \cdot \frac{p(M_j|I)}{p(M_i|I)}.$$

Computing the marginal likelihood of the data $p(\mathcal{D}|M_i, I)$, given a model and other background information I, enables us to do straightforward model comparison where we also account for the complexity of each model. This is the gold-standard in terms of evaluating (data) support for a model.

3.3.1 The Jeffrey-Lindley paradox

In the notebook `many_models.ipynb` you will also find a running example of the so-called *Jeffrey-Lindley* paradox [11]. This is an academic exercise to highlight the somewhat absurd situation that can appear when Bayesian and frequentist hypothesis testing yield contradictory results. It will also demonstrate the fact that frequentist tests tend to reject null hypotheses almost systematically in very large samples. In the example in the notebook you will find that if you obtain, e.g., 50,400 heads when tossing a coin 100,000 times the coin is certainly not fair according to traditional hypothesis testing. Whereas in a Bayesian setting you find that the probability null hypothesis H_0 : *the coin is fair* is far greater than the probability of the alternative hypothesis H_1: *the coin is not fair*. One can argue that there is no paradox since the Bayesian and the frequentist approaches answer two different questions. The frequentist finds that H_0 is a poor explanation for the observation, whereas the Bayesian finds that H_0 far better explanation for the observation than H_1. We will run the example during the lecture.

3.3.2 The marginal likelihood with a NIG prior

We will now evaluate the marginal likelihood: i.e., the probability $p(\mathcal{D}|M_i, I)$ of the data given a model and any other information, using a normal likelihood and a normal inverse-gamma (*NIG*) prior. Since we now also include the data errors, gathered in the data covariance matrix $\mathbf{E}$, our likelihood will be slightly different compared to the one used in Chapters 1 and 2.

$$\begin{aligned} p(\mathcal{D}|\theta, \sigma^2) &= \mathcal{N}(\mathcal{D}|\Phi\theta, \sigma^2\mathbf{W}) \\ &= \frac{1}{(2\pi\sigma^2)^{N_d/2}|\mathbf{W}|^{-1/2}} \exp\left\{-\frac{1}{2}\frac{(\mathcal{D}-\Phi\theta)^T\mathbf{W}(\mathcal{D}-\Phi\theta)}{\sigma^2}\right\}. \end{aligned}$$

The parameter prior remains the same, i.e.

$$\begin{aligned} \mathcal{NIG}(\theta, \sigma^2|\mu_0, \Sigma_0, \alpha_0, \beta_0) = &\frac{\beta_0^{\alpha_0}\sigma^{-2(\alpha_0+1+N_p/2)}}{(2\pi)^{N_p/2}\Gamma(\alpha_0)|\Sigma_0|^{1/2}} \\ &\times \exp\left\{-\frac{1}{\sigma^2}\left[\beta_0 + \frac{1}{2}(\theta-\mu_0)^T\Sigma_0^{-1}(\theta-\mu_0)\right]\right\} \end{aligned}$$

The introduction of $\mathbf{W}$ into the likelihood does not change anything but the normalization, and the fact that the scalar product between the design matrix and the data vector must be taken with respect to $\mathbf{W}$. Thus, we can easily modify the marginal likelihood in Eq. (18) in the *Appendix of the Chapter on Bayesian inference* and include $|\mathbf{W}|^{1/2}$ in the denominator as well

$$Z_M = p(\mathcal{D}|M) = \frac{\Gamma(\tilde{\alpha})|\tilde{\Sigma}|^{1/2}\beta_0^{\alpha_0}}{(2\pi)^{N_d/2}\Gamma(\alpha_0)|\Sigma_0|^{1/2}|\mathbf{W}|^{-1/2}}\tilde{\beta}^{-\tilde{\alpha}}.$$

Here, we also defined the very common notation Z_M for the marginal likelihood of model M given data $\mathcal{D}$. This expression is available in the notebook accompanying Chapters 3 and Section 4. You will immediately find that Z values fluctuate wildly across different choices for the model M. This is expected since the normal likelihood is exponential in the data and the model response. Thus, a poor model will yield orders of magnitude lower Z values compared to a better one. For this reason, one instead computes the logarithm $\ln(Z_M)$. This also helps to avoid numerical under- and over-flows.

Let us turn to the evaluated Z for the models M_0, M_1, and M_2 of the simple machine. To compute Z, we need to decide on a specific parameter prior. In a realistic setting this would of course depend on the (*your*) prior knowledge (*belief*) about the simple machine and the models in question. In the present case, we can pretend that we expect model parameters of order unity. There could for instance be theoretical reasons for this prior. We therefore use a (normal) prior for θ with mean $\mu_0 = \mathbf{0}$ and parameter covariance matrix $\text{diag}(\Sigma_0) = 10$, the latter to indicate that we are rather uncertain about the values of θ Let us pretend that we know a bit more about the variance parameter σ^2 associated with the error model for the measurement. For now, we will use the inverse-gamma prior with mean 0.1^2 and mode 0.05^2, i.e., $a_0 = 5/3$ and $b_0 = 2/300$ to keep the mode of σ^2 rather close to 0.05^2. Just like in Chapter 2, you can mimic an uninformative prior and recover the *WLS* result. Using an uninformative (improper) prior yields an unnormalizable posterior and is hence of no value for studies involving the marginal likelihood Z. In practice, it ruins much of the Bayesian approach. In Table 3 you will find the marginal likelihoods and log maximum-likelihoods from the Bayesian analysis of models M_0, M_1, and M_2 using datasets of length $N_d = 20$ in Figure 1.

Table 3: Bayesian parameter modes and marginal likelihoods for models M_0, M_1, and M_2. The rightmost column shows the log maximum likelihood values.

Model	$\tilde{\mu} = \hat{\theta}_{\text{WLS}}$	$\tilde{\mu} = \theta_{\text{MAP}}$	$\ln(Z_{M_i})$	$\ln p(D\|\theta_{\text{ML}}, M_i)$
M_0	$[0.545]$	$[0.545]$	8.27	15.10
M_1	$[0.619, -0.020]$	$[0.615, -0.019]$	9.62	22.24
M_2	$[0.693, -0.071, 0.008]$	$[0.658, -0.049, 0.005]$	6.19	24.08

Based on these results we can conclude that

- as expected, the log maximum likelihood value will keep increasing as we introduce more parameters. This measure does not incorporate the so-called Occam penalty for using a more complex model. Indeed, with more parameters we can fit almost anything. However, this usually comes at the expense of explanatory capability of the model.
- M_1 has the largest $\ln(Z)$ value, but not by much, and we get the following (log) Bayes factors: $\ln(BF_{1,0}) = 1.35$, $\ln(BF_{1,2} = 3.43)$, and $\ln(BF_{0,2} = 2.08)$. Citing [12]:

  The Bayes factor is a summary of the evidence provided by the data in favor of one scientific theory, represented by a statistical model, as opposed to another.

  So, contrary traditional null hypothesis rejection, we actually recover evidence *in favor* of a *model*. Following [12] we say that we have positive to strong evidence for selecting M_0 or M_1 over M_2, but not any substantial evidence to select M_0 in favor of M_1. The (subjective) Table 4 can be used as a guide to interpret Bayes factors.

Table 4: Kass' and Raferty's interpretation of Bayes factors.

$2\ln(BF_{i,j})$	$BF_{i,j}$	**Interpretation**
> 10	> 150	Decisive evidence for M_i
6 to 10	20 to 50	Strong evidence for M_i
2 to 6	3 to 20	Positive evidence for M_i
0 to 2	1 to 3	Not worth more than a bare mention

We end this section by mentioning that Bayesian regression, with all the bells and whistles that come with it, can be applied even in the case of $N_d < N_p$. In such data-deficient cases standard regression methods fail. In Bayesian regression the priors effectively regulate the otherwise ill-posed regression problem. In the case of very little data, your prior will also influence the posterior more. One can also show that the methods of Ridge regression and *LASSO* are nothing but normal Bayesian regression with a normal and Laplace parameter prior, respectively (also see Chapter 5.7). Such regression functionality, as well as linear combinations of Ridge and *LASSO* methods (called elastic net regression), can be found in most machine learning packages, e.g., `scikit-learn`.

3.3.3 The posterior predictive

As in Chapter 2, we can compute a posterior predictive distribution for future data $\tilde{\mathcal{F}}$ given some model M_i with parameters calibrated using data $\mathcal{D}$. In the case of a normal likelihood, with weights $\mathbf{W}$ (the same weights as in *WLS*), and a *NIG* parameter prior, we obtain a *NIG* parameter posterior given by

$$p(\theta, \sigma^2|\mathcal{D}) = \mathcal{NIG}(\theta, \sigma^2|\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta})$$

with

$$\begin{aligned}
\tilde{\mu} &= [\Sigma_0^{-1} + \Phi^T\mathbf{W}\Phi]^{-1}[\Sigma_0^{-1}\mu_0 + \Phi^T\mathbf{W}\mathcal{D}] \\
\tilde{\Sigma} &= [\Sigma_0^{-1} + \Phi^T\mathbf{W}\Phi]^{-1} \\
\tilde{\alpha} &= \alpha_0 + N_d/2 \\
\tilde{\beta} &= \beta_0 + \frac{1}{2}\left(\mathcal{D}^T\mathbf{W}\mathcal{D} + \mu_0^T\Sigma_0^{-1}\mu_0 - \tilde{\mu}^T\tilde{\Sigma}^{-1}\tilde{\mu}\right).
\end{aligned}$$

We obtain the posterior predictive *pdf* by marginalizing over the uncertain model parameters that we just inferred,

$$\begin{aligned}
p(\tilde{\mathcal{F}}|M_i, \mathcal{D}) &= \int \mathcal{N}(\tilde{\mathcal{F}}|\tilde{\Phi}\theta, \sigma^2\mathbf{1})\mathcal{NIG}(\theta, \sigma^2|\tilde{\mu}, \tilde{\Sigma}, \tilde{\alpha}, \tilde{\beta})\, d\sigma^2 d\theta \\
&= \mathcal{T}_{2\tilde{\alpha}}(\tilde{\mathcal{F}}|\tilde{\Phi}\tilde{\mu}, (\tilde{\beta}/\tilde{\alpha})(\mathbf{1} + \tilde{\Phi}\tilde{\Sigma}\tilde{\Phi}^T)).
\end{aligned}$$

The scale of the $\mathcal{T}$-distribution contains two terms. The first one is $\tilde{\beta}/\tilde{\alpha}\mathbf{1}$ and is directly proportional to the sample variance, i.e., the data error, and the second term $\tilde{\beta}/\tilde{\alpha}\tilde{\Phi}\tilde{\Sigma}\tilde{\Phi}^T$ is directly proportional to the model parameter covariances learned from the data. In the case of uninformative priors, the sum of these terms contributes to the total *WLS* prediction interval. Given the priors we defined above, the Bayesian (weighted) posterior predictive looks very similar to the *WLS* prediction interval, see Figure 3.

Exercise

Use the notebook `many_models_pymc.ipynb` to extract the parameter posteriors, posterior predictives (and the 90% credibility intervals) for the three models M_0, M_1, and M_2 of the simple machine. But this time you should use an inverse-gamma prior with $a_0 = b_0 = 1$ for σ^2. Can you explain (qualitatively) how this prior choice impacts the results?

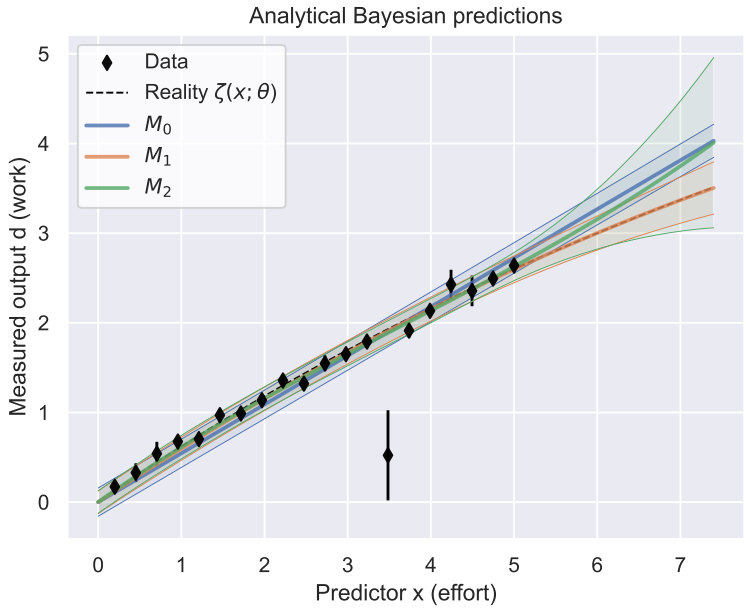


Fig. 3: 90% Bayesian credibility intervals for the posterior predictive distributions ({term}:ppds) of models M_0, M_1, and M_2.

3.4 Information criteria

If we can compute the log maximum likelihood for the models we consider, then we can easily extract various scores for each model that will help us in the rank/select process. Below, we cover two of the most common so-called information criteria (*IC*); the Bayesian IC (*BIC*) [13] and the Akaike IC (*AIC*) [14]. They only apply in the large-data limit. In our case, with $N_d = 10$ data, they are not valid and we will see this first hand. Most *IC*s have similar structure. Some deviance, i.e., a log predictive density of the data, that is corrected for bias due to the number of parameters in the model. We have

$$\begin{aligned}\text{BIC} &\equiv 2\ln[p(\mathcal{D}|\theta_\star)] - N_p \ln(N_d)\\ \text{AIC} &\equiv 2\ln[p(\mathcal{D}|\theta_\star)] - 2N_p.\end{aligned}$$

Here, we define the scores such that a larger value indicates a better model. This is not always the case. Often the *IC* scores are multiplied with an overall minus sign. However, in these notes we choose to be consistent with the definition of the marginal likelihood. One can argue that the *AIC* is designed for out-of-sample prediction, i.e., geared towards ranking models based on the estimated predictive performance, whereas *BIC* is not. The *BIC* will often select the simpler model and *AIC* the more complex model. The *AIC* will also reduce to a likelihood ratio for comparing models with the same number of parameters. Remember, both scores are *approximations*. The approximations require at least $N_d \to \infty$. The resulting *AIC* and *BIC* scores for models M_0, M_1, and M_2 using the same dataset as before, i.e of length $N_d = 20$, is reported in Table 5.

Table 5: Bayesian parameter modes, marginal likelihoods, and BIC, AIC scores, for models M_0, M_1, and M_2 and with $N_d = 20$.

Model	$\tilde{\mu} = \hat{\theta}_{\text{WLS}}$	$\tilde{\mu} = \theta_{\text{MAP}}$	$\ln(Z_{M_i})$	$\ln p(D\|\theta_{\text{ML}}, M_i)$	*AIC*	*BIC*
M_0	$[0.545]$	$[0.545]$	8.27	15.10	28.20	27.20
M_1	$[0.619, -0.020]$	$[0.615, -0.019]$	**9.62**	22.24	40.49	38.49
M_2	$[0.693, -0.071, 0.008]$	$[0.658, -0.049, 0.005]$	6.19	**24.08**	**42.17**	**39.18**

The results change if we introduce more data. In Table 6 you can find a comparison between all model scores when using $N_d = 100$ data points from the simple machine. All other conditions remain the same.

Table 6: Bayesian parameter modes, marginal likelihoods, and BIC and AIC scores, for models M_0, M_1, and M_2 and with $N_d = 100$.

Model	$\tilde{\mu} = \widehat{\theta}_{\text{WLS}}$	$\tilde{\mu} = \theta_{\text{MAP}}$	$\ln(Z_{M_i})$	$\ln p(D\|\theta_{\text{ML}}, M_i)$	*AIC*	*BIC*
M_0	$[0.548]$	$[0.548]$	64.22	72.764	143.52	140.91
M_1	$[0.641, -0.025]$	$[0.640, -0.025]$	**105.36**	123.08	**241.37**	**236.16**
M_2	$[0.651, -0.032, 0.001]$	$[0.643, -0.027, 0.000]$	100.80	**123.25**	239.65	231.84

A few comments are in order. First, the value of the log maximum likelihood $\ln p(D|\theta_{\text{ML}}, M_i)$ will always be the highest for the model with most parameters. With a more complex model it is easier to fit all data. However, at one point the model will be calibrated to reproduce the noise in the data, which leads to a biased inference of the parameter values and unreliable extrapolations. The log marginal likelihood is guarded against overfitting by construction. The evidence integral

$$Z_{M_i} = p(\mathcal{D}|M_i, I) = \int d\theta \, p(\mathcal{D}|\theta, M_i, I) p(\theta|M_i, I).$$

contains two factors, a likelihood and a prior. With more parameters, i.e., a more complex model, the volume of the parameter space increases, and the fraction of the space where model fits better to the data might not be in proportion, in which case we obtain a lower value of the (log) marginal likelihood. One way to see this is to consider the univariate case, assuming a unimodal likelihood with a uniform prior, and then approximate the evidence integral as the best-fit likelihood times some characteristic interval $\Delta_{\theta|\mathcal{D}}$ across which the likelihood is non-negligible, i.e.,

$$Z_{M_i} \approx p(\mathcal{D}|\theta_{\text{ML}}, M_i, I) p(\theta_{\text{ML}}|M_i, I) \Delta_{\theta|\mathcal{D}}.$$

We can express a normalize and uniform prior over some finite, but wide, interval Δ_θ as $p(\theta|M_i, I)\Delta_\theta = 1$. From which we obtain

$$Z_{M_i} \approx p(\mathcal{D}|\theta_{\text{ML}}, M_i, I) \frac{\Delta_{\theta|\mathcal{D}}}{\Delta_\theta}.$$

This expression provides reveals how the ratio of the likelihood times the relevant part of the parameter space divided by the model prior, i.e., the model complexity, regulates the evidence. Simply put, you have to pay a price if you 'waste' too much parameter space. In the multivariate case and the derivation of the BIC this becomes even more obvious. A critical comment is also in place. Two different parameter priors might yield very different values for the evidence an similar mean values of the estimated parameters.

Finally, we should point out that our three models we are working with are a bit peculiar. First of all, they are *nested*, i.e., $M_0 \subset M_1 \subset M_2$, which can be exploited in the evaluation of Bayes factors via the so-called Savage-Dickey ratio (not discussed in this course). Second, the simple machine can be Taylor expanded, and if we keep taking data across some range of x-values,it will resolve the higher order terms more and more. As such, with increasing data we should see stronger evidence for more complex models. The rate at which this happens will depend on the range of x-values explored and the variance of the data.

3.4.1 The Bayesian Information Criterion

The marginal likelihood can be rewritten as an exponential integral

$$Z = \int d\theta p(\mathcal{D}|\theta) \, p(\theta) = \int d\theta \exp\left\{\ln\left[p(\mathcal{D}|\theta) \, p(\theta)\right]\right\}.$$

Assuming that the kernel, i.e., the unnormalized posterior, peaks at $\theta_\star$ we Taylor expand around this point

$$\begin{aligned} \ln\left[p(\mathcal{D}|\theta) \, p(\theta)\right] &\equiv f(\theta) \\ &\approx f(\theta_\star) + (\theta_\star - \theta) \, \nabla_\theta f(\theta)|_{\theta=\theta_\star} + \frac{1}{2}(\theta_\star - \theta)^T \mathbf{H}_\theta f(\theta_\star - \theta) + \ldots, \end{aligned}$$

where $\mathbf{H}_\theta f$ is the Hessian matrix of f evaluated at $\theta = \theta_\star$. We are expanding around an extremum (maximum), so the second term is zero and the Hessian is negative definite. We obtain

$$Z \approx \exp\{f(\theta_\star)\} \int d\theta \exp\left\{-\frac{1}{2}\delta^T(-\mathbf{H}_\theta f)\delta\right\},$$

where $\delta \equiv (\theta_\star - \theta)$. To evaluate the Gaussian integral we simply remind ourselves of the normalization constant of the multivariate normal distribution, which gives us

$$Z \approx \exp\{f(\theta_\star)\} \sqrt{\frac{(2\pi)^{Np}}{|-\mathbf{H}f|}} = p(\mathcal{D}|\theta_\star)\, p(\theta_\star) \sqrt{\frac{(2\pi)^{Np}}{|-\mathbf{H}f|}}.$$

This is the well-known Laplace approximation of the marginal likelihood. It applies when the posterior is unimodal and falls off at least exponentially. Twice the logarithm of the marginal likelihood can therefore be approximated as

$$2\ln(Z) \approx 2\ln[p(\mathcal{D}|\theta_\star)] + \ln[p(\theta_\star)] + N_p \ln(2\pi) - \ln(|-\mathbf{H}f|).$$

The first term is the log maximum-likelihood. If we assume a flat (improper) prior $p(\theta) \equiv 1$ the element of the (negative) Hessian of the 'log posterior' f is given by

$$-(Hf)_{ij} = -\frac{\partial^2}{\partial\theta_i\theta_j} \ln[p(\mathcal{D}|\theta)]|_{\theta=\theta_\star}.$$

This is sometimes referred to as the Fisher information matrix. Assuming independent and identically distributed data we can express the likelihood as a product, which turn into a sum when logarithmized, and for large N_d we can write

$$-(Hf)_{ij} = -\frac{\partial^2}{\partial\theta_i\theta_j} \sum_i \ln[p(d_i|\textbf{?})] = -N_d \frac{\partial^2}{\partial\theta_i\theta_j} \mathbb{E}(\ln[p(d_i|\textbf{?})]) = N_d I_{ij}.$$

Thus, we have $\ln(|-\mathbf{H}f|) = \ln(|N_d\mathbf{I}|) = \ln(N_d^{N_p}) + \ln(\mathbf{I})$. In the large-$N_d$ limit, twice the logarithm of the marginal likelihood is approximately given by

$$2\ln(Z) \approx 2\ln[p(\mathcal{D}|\theta_\star)] - N_p \ln(N_d).$$

The right-hand side is referred to as the *BIC* or the Schwarz criterion. A larger *BIC* score should be interpreted as a larger marginal likelihood, i.e., stronger evidence for a better model to describe the data. As $N_d \to \infty$, the *BIC* will identify the best model (asymptotic consistency). For smaller N_d, it will often pick models that are too simple.

The *BIC* score can be easily calculated for models where we can obtain the log-maximum-likelihood. It will measure the efficiency of the model capability for describing measured data while also penalizing a larger number of parameters. Strictly speaking, the *BIC* score is only be applicable for unimodal model posteriors where we also have a large amount of data. Naturally, the first condition is difficult to verify. We end by commenting on the name. The *BIC* is not truly Bayesian since it explicitly uses the maximum likelihood instead of the posterior distribution. The *BIC* is not an information criterion in the traditional sense since it is not based on the Kullback-Leibler (*KL*) divergence. This will be discussed in the next section.

3.4.2 The Akaike Information Criterion

Akaike used the *KL* divergence to derive the so-called *AIC* score for ranking competing models. The *AIC* for model M_i is given by

$$\text{AIC} = 2\ln[p(\mathcal{D}|\theta_\star)] - 2N_p,$$

where $\theta_\star$ is the maximum-likelihood estimator for θ and N_p is the number of parameters in model M_i. Here we also employed the same sign-convention as for the *BIC* score, implying that one should select the model that maximizes the

AIC score. The opposite sign-convention is not uncommon. The *BIC* and *AIC* have the same overall structure, and only differs in how they penalize model complexity. In the large N_d limit, the *BIC* score penalizes model complexity stronger than the *AIC* score, and therefore tends to select simpler models for $N_d \geq 8$.

The *KL* divergence is a measure of how a probability distribution q is different from a reference probability distribution p. In essence, the *KL* divergence from q to p is defined by

$$D_{KL}(p||q) = -\int dx\, p(x) \ln\left(\frac{q(x)}{p(x)}\right) = \int dx\, p(x) \ln\left(\frac{p(x)}{q(x)}\right)$$

This measure quantifies how much information we loose by approximating the reference distribution p with q. It is always positive, and zero only for $p = q$. This measure is intimately related with the (Shannon) entropy, which in a discrete representation is given by

$$H = -\sum_i p(x_i) \ln p(x_i).$$

Here, the possible outcomes $x_1, x_2, ..., x_n$ of an event is associated with a probability $p(x_i)$[^3]. As a side note, the uniform probability distribution maximizes this entropy. In an information-theoretic language, this can be interpreted as a maximum amount of surprise in the sense that any other distribution holds a lesser amount of surprise since it clearly also would reduce H. In the extreme case of, e.g., $p(x_2) = 1$ and $p(x_i) = 0$ for all $i \neq 2$, then $H = 0$ and there no surprise at all.

The (discrete) *KL* divergence emerges if we modify the Shannon entropy with a sign and insert the $q(x)$ distribution

$$D_{KL}(p||q) = \sum_i p(x_i)(\ln p(x_i) - \ln q(x_i)) = \mathbb{E}_p(\ln(p)) - \mathbb{E}_p(\ln(q))$$

where the last step highlight that we simply evaluate the difference of the p-expectation values of the logarithmized distributions. The term $\sum_i p_i \ln(q_i)$ is called the cross entropy. Have the *KL* divergence to be a difference between entropy and cross-entropy, we ensure a zero divergence for the case $q \equiv p$. Note that it is an asymmetric measure in the sense $D_{KL}(p||q) \neq D_{KL}(q||p)$.We now go back to our original problem and assume that we can approximate the distribution for the data p with two different models q_1 and q_2. We keep the p/q notation to highlight that we are referring to probability distributions for the true data distribution. We can now proceeded to use the *KL* divergence for model comparison purposes. To this end we calculate the *KL* divergences $D_{KL}(p||q_1)$ and $D_{KL}(p||q_2)$

$$D_{KL}(p||q_k) = \mathbb{E}_p(\ln(p) - \mathbb{E}_p(\ln(q_k)), \quad k = 1, 2. \tag{1}$$

To be specific, in the event of $D_{KL}(p||q_1) < D_{KL}(p||q_2)$, then model q_1 would be deemed better than model q_2. To make quantitative use of this approach we need to evaluate $D_{KL}(p||q_k)$. Fortunately, the first term on the right-hand in Eq. (1) side is a constant independent of q_k, and we only have to compare $\mathbb{E}_p(\ln(q_k))$, i.e., the cross-entropies for the two models. A *larger* $\mathbb{E}_p(\ln(q_k))$ will be obtained for the better model. However, we do not know p, over which we take the expectation values. We only have data and models that in turn depend on parameters θ. Akaike managed to prove, via several Taylor expansions, that in the large-data domain the relative cross-entropy, i.e., the relative *KL* divergence with respect to the true distribution p, can be estimated by the *AIC* score

$$\text{AIC} = 2\ln[p(\mathcal{D}|\theta_\star)] - 2N_p.$$

There exists a vast literature on the derivation and application of the *AIC* score, and various modifications of it. Given the structure of the penalty term, you notice that *AIC* will favor more complex models compared to *BIC*.

3.5 What did we learn?

We often have multiple models available for explaining/predicting data. Despite the fact that *all models are wrong*, we should strive to employ measures/scores for model comparison. The gold standard is to compute the marginal likelihood $\log(Z)$. That being said, it is often challenging or even impossible to reliably compute this quantity, and in practice there is no silver bullet for ranking models. One should therefore try to analyze the problem from multiple angles using, e.g., *AIC* and *BIC*. Should these *IC* scores point to slightly different models, then this *range* of models should be kept when going forward in the analysis.

In the midst of all model selection and scoring one should not forget to scrutinize the model content and try to account for neglected content via the model discrepancy. All models should also be checked thoroughly, e.g., by comparing (visually or through some measure) how well they actually reproduce known data.

CHAPTER

FOUR

MODEL AVERAGING

Key take-away messages

- Most of the time, we need to sample models numerically using, e.g., *MCMC*.
- There exist useful information criteria for cases when you only have *MCMC* sample chains of the posterior.
- Sequential Monte Carlo with likelihood tempering or nested sampling can in some cases extract marginal likelihoods.
- Bayesian model averaging is one possible technique for combining several model predictions.

Before reading this chapter

Refresh your memory of the following concepts:

- The marginal likelihood
- Markov Chain Monte Carlo (*MCMC*)
- Weighted averages

4.1 Recap: the marginal likelihood

The central quantity throughout this chapter is the log marginal likelihood $\ln(Z_{M_i})$ for model M_i

$$Z_{M_i} = p(\mathcal{D}|M_i I) = \int d\theta \, p(\mathcal{D}|\theta M_i I) \, p(\theta|M_i I).$$

Henceforth, we will omit the explicit mentioning of any possible condition on any other information I.

4.2 Deviance information criterion

We will continue using the $N_d = 20$ data from the simple machine presented in Chapter 3. However, in this chapter we will explore numerical sampling techniques for estimating the marginal likelihood, and in case we cannot do so, we will show how to use the posterior samples themselves to compute the so-called Deviance[1] information criterion (*DIC*) [15]. This will attribute the model with a score that can be used for ranking/selection/comparion and more. The *DIC* can be interpreted as a Bayesian version of *AIC* where one replaces the maximum likelihood estimator with the posterior mean

[1] Deviance is any statistic that measures the goodness-of-fit.

θ_{mean}, and the number of parameters N_p is replaced with a data-based correction. As such, this score can be applied to models with an unknown number of parameters.

The *DIC* is computed from the S samples of θ in the posterior distribution $p(\theta|\mathcal{D}M_i)$,

$$\text{DIC} = 2\ln[p(\mathcal{D}|\theta_{\text{mean}})] - 2p_{\text{DIC}},$$

and p_{DIC} denotes the effective number of parameters defined as

$$p_{\text{DIC}} = 2\left(\ln p(\mathcal{D}|\theta_{\text{mean}}) - \mathbb{E}_{\text{posterior}}[\ln p(\mathcal{D}|\theta)]\right).$$

This term is calculated from the posterior samples straightforwardly as

$$p_{\text{DIC}} \approx 2\left(\ln p(\mathcal{D}|\theta_{\text{mean}}) - \frac{1}{S}\sum_{s=1}^{S} \ln p(\mathcal{D}|\theta_i)\right).$$

The effective number of parameters measures the necessary penalty in the deviance score. We note that this might become negative if the likelihood mean is far away from the mode. In fact, the *DIC* is only valid when the mean is a good summary of the posterior. According to [15], *[…] a negative p_{DIC} indicate substantial conflict between the prior and data, or where the posterior mean is a poor estimator (such as a symmetric bimodal distribution).*

There exists a whole family of *IC*s that can be used with posterior samples, see, e.g., [16] for a recent critical analysis. In the notebook you will find several examples of how to use the Python module `pymc` for probabilistic programming to evaluate Bayesian posteriors. This package, and its accompanying analysis package `ArviZ`, contain additional criteria, such as the Watanabe-Akaike IC (*WAIC*) and approximate leave-one-out cross validation (*LOOCV*) methods. The *WAIC* is actually a cheaper approximation to *LOOCV*. We will not discuss these methods further, but you can try them out in `pymc`.

In Table 1 we summarize the *DIC* scores and p_{DIC} for models M_0, M_1, and M_2 defined in Chapter 3 (informative prior and $N_d = 20$ data as before). To compute the *DIC* scores, we obtained samples from the parameter posterior using the `emcee` ensemble sampler. Please see the notebook `many_models_pymc.ipynb` for details.

Table 1: DIC-scores for the simple machine.

Model	$\tilde{\mu} = \theta_{\text{MAP}}$	$\ln(Z_{M_i})$	$\ln p(D\|\theta_{\text{ML}}, M_i)$	AIC	BIC	*DIC*$[p_{\text{DIC}}]$
M_0	$[0.545]$	8.27	15.10	28.20	27.20	26.59[1.7]
M_1	$[0.615, -0.019]$	**9.62**	22.24	40.49	38.49	38.09[2.3]
M_2	$[0.658, -0.049, 0.005]$	6.19	**24.08**	**42.17**	**39.18**	**38.59[3.1]**

4.3 Sampling the marginal likelihood

As you know, it is straightforward to run an *MCMC* algorithm to draw samples from the model posterior, but it is very challenging to know when to stop. In practice, we can never be sure of having reached a sufficiently converged representation of the sought distribution.

To begin with, the dimensionality of the parameter space might be quite large. This rules out any quadrature-based method or even brute force sampling. The posterior might also be curved and/or sharply peaked. It is in general not unlikely that much, or nearly all, of the integral value is concentrated to a very small (unknown) region of the space.

In this chapter we will employ so-called likelihood tempering to evaluate $\ln(Z_M)$. Intuitively, this method proceeds by introducing an inverse temperature scale β, where for $\beta = 0$ we only sample, using some Monte Carlo method, from the normalized prior. This is most often easier than sampling from the posterior. The trick is then to slowly lower the

temperature, i.e., gradually increase the value of β to 1, whereby the effects of the likelihood becomes more apparent[1]. Yet, at sufficiently high temperatures it is still somewhat easy to traverse the space even with a *MC* method, but we will most likely employ some *MCMC* method. Between each temperature step β_i it becomes more and more apparent where the dominant features of the posterior resides and we will be able to concentrate our *MCMC* samples to that parameter region, or regions. In the end, when $\beta = 1$ and the approach has been sufficiently slow, we will integrate the posterior accurately. This gives us the marginal likelihood as well as the posterior distribution. Tempering is no silver bullet since any *MC* integration, at every temperature step, in multidimensional spaces is challenging. The tempering method will eventually become insufficient for practical use. Another algorithm for computing $\ln(Z_M)$ is called nested sampling, see, e.g., [17]. A version of this approach, called MultiNest [18], has been demonstrated to outperform thermodynamic integration; a Python implementation of this technique can be found here.

In this course we will use the tempering method available in `pymc`. It is easy to use and certainly sufficient for our needs. In general, computing the marginal likelihood in many dimensions is a challenging problem, and a topic at the absolute frontier of computational Bayesian research. Mathematically, for the tempering method we write the integral at temperature step β

$$Z_{M_i}(\beta) = p(\mathcal{D}|M_i\beta) = \int d\theta p(\theta|M_i)\, p(\mathcal{D}|\theta M_i)^\beta = \int d\theta f(\theta,\beta).$$

Differentiating $\ln[Z_{M_i}(\beta)]$ with respect to β yields

$$\begin{aligned}\frac{d}{d\beta}\ln[Z_{M_i}(\beta)] &= \frac{1}{Z_{M_i}(\beta)}\int d\theta \frac{d}{d\beta} f(\theta,\beta) \\ &= \int d\theta \frac{f(\theta,\beta)}{Z_{M_i}(\beta)} \frac{d\ln[f(\theta,\beta)]}{d\beta} \equiv \mathbb{E}_\beta\left[\frac{d(\ln[f(\theta,\beta)])}{d\beta}\right].\end{aligned} \tag{1}$$

The last step defines the expectation value $\mathbb{E}_\beta[(\cdot)] = \frac{1}{Z_{M_i}}\int d\theta(\cdot)\, p(\theta|M_i)\, p(\mathcal{D}|\theta M_i)^\beta$. Integrating the left-hand side of Eq. (1) with respect to β can be evaluated as

$$\begin{aligned}\int_0^1 d\beta \frac{d}{d\beta}\ln[Z_{M_i}(\beta)] &= \ln[Z_{M_i}(\beta=1)] - \ln[Z_{M_i}(\beta=0)] \\ &= \ln[p(\mathcal{D}|M_i)] - \ln[1],\end{aligned}$$

which gives us the key relation in this thermodynamic integration approach

$$\ln[p(\mathcal{D}|M_i)] = \int_0^1 d\beta \mathbb{E}_\beta\left[\frac{d(\ln[f(\theta,\beta)])}{d\beta}\right].$$

The integrand on the right-hand side can be expressed once we take the derivative with respect to β

$$\frac{d(\ln[f(\theta,\beta)])}{d\beta} = \frac{d}{d\beta}\ln[p(\theta|M_i)\, p(\mathcal{D}|\theta M_i)^\beta] = \ln[p(\mathcal{D}|\theta M_i)],$$

and we finally obtain

$$\ln[p(\mathcal{D}|M_i)] = \int_0^1 d\beta \mathbb{E}_\beta\left[\ln\left[p(\mathcal{D}|\theta M_i)\right]\right].$$

We can evaluate the one-dimensional β-integral using, e.g., the trapezoidal rule for discrete $[\beta_1, \beta_2, \ldots, \beta_M]$

$$\ln[p(\mathcal{D}|M_i)] \approx \sum_{j=2}^{M}(\beta_j - \beta_{j-1})\frac{Y_j + Y_{j-1}}{2},$$

[1] This is closely related to the *simulated annealing approach* that we will encounter in the *Chapter on global optimization techniques*.

where each Y_j is a β-averaged log-likelihood

$$Y_j \approx \frac{1}{N_s} \sum_{s=1}^{N_s} \ln[p(\mathcal{D}|\theta_s M_i)]$$

where θ_s are drawn from some chain S_β of *MCMC* samples of the β-posterior $p(\theta|M_i)\, p(\mathcal{D}|\theta M_i)^\beta$. One could try and approach this by setting up a discrete set of ~ 10 (inverse) temperatures $\beta_j \in [0,1]$, and run *MCMC* samples with different β-values in parallel, then compute Y_k values and perform the thermodynamic integral. A more dynamic approach, which we will use here (via the `pymc` Python module), is to initialize β at zero, and draw N_s samples from the $(\beta = 0)$-tempered posterior, i.e., the prior. For each sample, compute a normalized weight $w_\beta^{(s)}$ as the ratio of the tempered likelihoods at β_{j+1} and β_j, where β_{j+1} is chosen such that the so-called effective sample size (*ESS*) $N_s[\sum_{s=1}^{N_s}(w_{\beta_j}^{(s)})^2]^{-1}$ equals $N_s/2$ (an adjustable parameter of the method). The *ESS* value always lie in the range 1 to N_s, and a very small value indicates that the samples are very degenerate. Each sample is now attributed with a normalized weight ($\sum_s w_{\beta_j}^{(s)} = M_s$). The β_{j+1}-posterior is now sampled for a few more steps (an adjustable parameter of the method) using *MCMC* chains that are initialized at points with the highest weights. Samples with low weights are eliminated. This way, we gradually identify and *MCMC* sample the region(s) in space that contribute most to the marginal likelihood integral. At $\beta = 1$, a final run of the MCMC chains yields a collection of N_s samples from the parameter posterior.

The details behind this so-called tempered sequential Conte Carlo (*SMC*) algorithm are given in [19]. We will employ the version available in `pymc`. See the notebook for details about implementation. As usual, this will not always work, i.e., be of operational use in finite time. At the core of the method we have some *MCMC* whose convergence properties will deteriorate for very complicated posteriors and/or very large-dimensional spaces.

For the analysis of the $N_d = 20$ simple machine data using models M_0, M_1, and M_2, we can converge the *SMC* algorithm rather well. It is a bit of an overkill since the posterior is unimodal, but at least we obtain a numerical approximation to the $\ln(Z_{M_i})$ values, see Table 2 for Bayesian parameter modes and analytical as well as *SMC* marginal likelihoods, and *AIC*/*BIC*/*DIC*-scores for models M_0, M_1, and M_2 and using $N_d = 20$ and an informative prior.

Table 2: Marginal likelihoods for the simple machine.

Model	$\tilde{\mu} = \theta_{\text{MAP}}$	$\ln(Z_{M_i})$	*SMC* $\ln(Z_{M_i})$	$\ln p(D\|\theta_{\text{ML}}, M_i)$	AIC	BIC	*DIC* $[p_{\text{DIC}}]$
M_0	$[0.545]$	8.27	8.81	15.10	28.20	27.20	26.59[1.7]
M_1	$[0.615, -0.019]$	**9.62**	**11.00**	22.24	40.49	38.49	38.09[2.3]
M_2	$[0.658, -0.049, 0.005]$	6.19	7.55	**24.08**	**42.17**	**39.18**	**38.59[3.1]**

A plot of the `pymc` traces for model M_1 is shown in Figure 1. Results for the other models can be generated using the notebook.

Equipped with the *SMC* chains, we can easily sample the posterior predictive for, e.g., models M_0 and M_1, our the two best candidates, see Figures 2 and Fig. 3.

4.4 Bayesian model averaging

Several different theoretical models can provide realistic, or equally adequate, descriptions of experimental data. Often, one selects the better model according to some pre-determined criteria, e.g., model fit to calibration data or predictive capabilities for a subset of data etc. All predictions are then based on this single model. There are several downsides to this approach. Most importantly, selecting a single model can lead to overconfident inference since it ignores the apparent model uncertainty in favor of the particular distributions and assumptions inherent to the model of choice. In our example we would pick model M_0 based on the marginal likelihoods, and model M_1 based on the information criteria. The latter scores are not reliable in this low-data domain so we could disregard them based on that. From the similar values of the marginal likelihoods of M_0 and M_1 we realize that the evidence in favor of M_0 over M_1 is *not worth more than a bare*

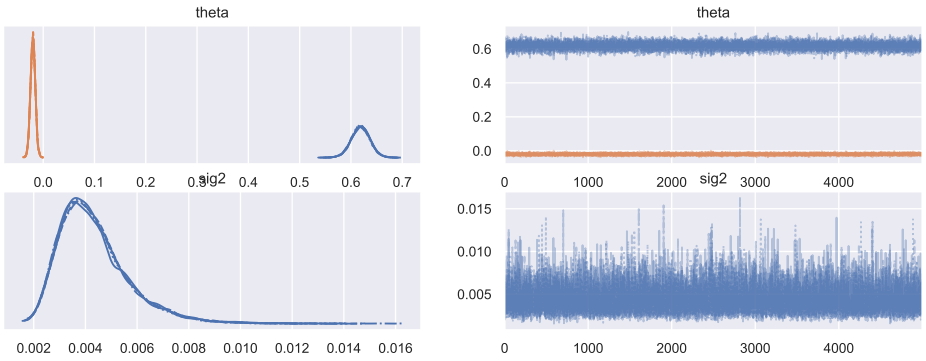


Fig. 1: The final (likelihood tempered) *SMC* traces for the three parameters of model M_1. Left panels: Kernel density estimate (*KDE*) plots for each parameter value, proportional to the probability of how likely that parameter value is. Right panels: Individual sampled values at each step during the final, i.e., $\beta = 1$, sampling.

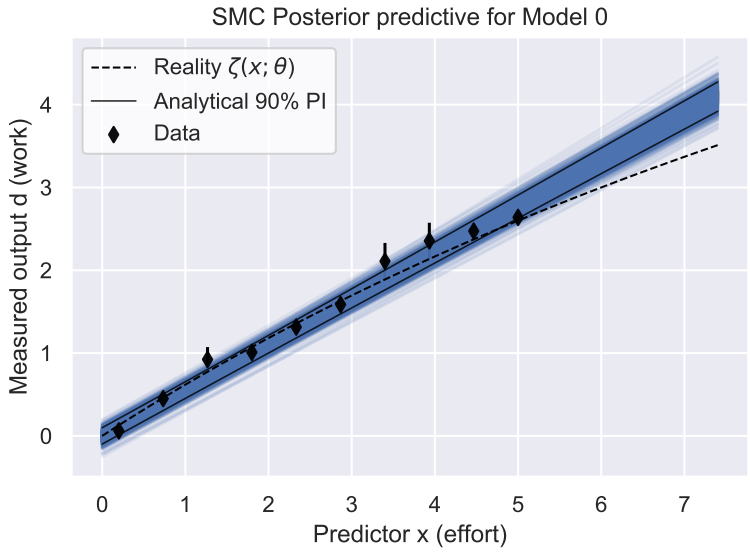


Fig. 2: Posterior predictive plots for models M_0. Each panel contains 1000 random draws from the model posterior density.

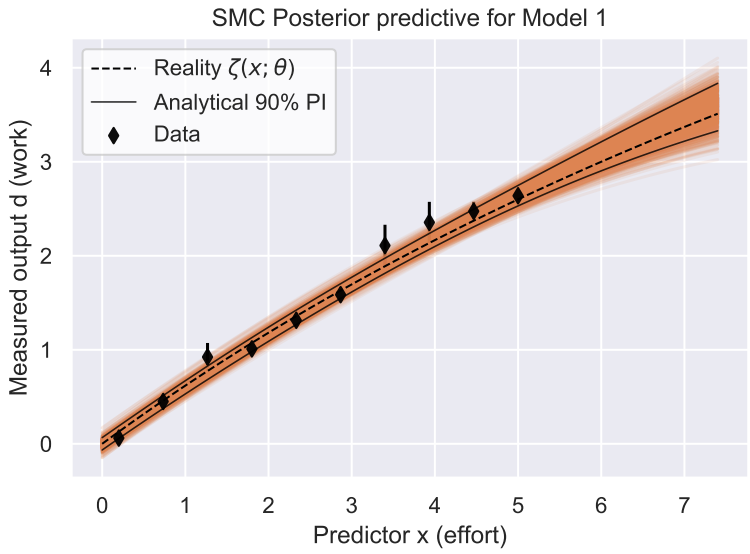


Fig. 3: Posterior predictive plots for models M_1. Each panel contains 1000 random draws from the model posterior density.

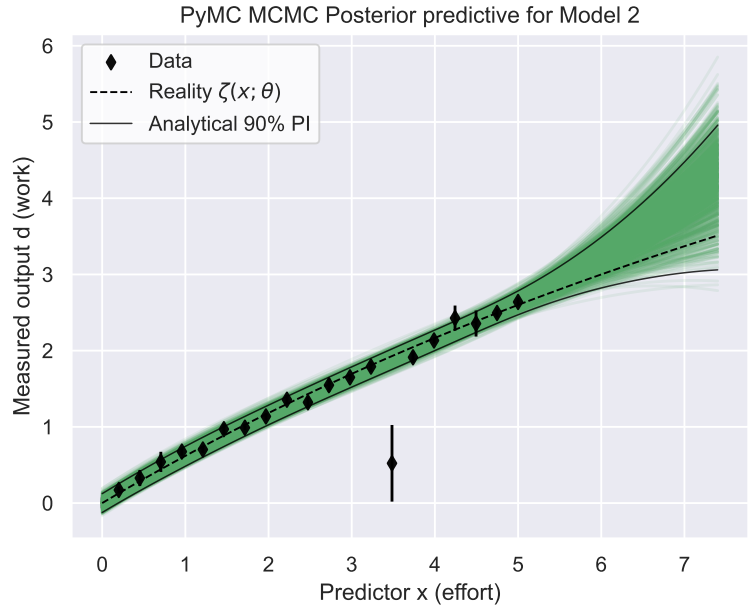


Fig. 4: Posterior predictive plots for models M_2. Each panel contains 1000 random draws from the model posterior density.

mention. We clearly do not have enough data to disentangle these two models. How do we proceed if we cannot select only M_0 or M_1? There is certainly information present in the parameter posteriors from the other two models, see Figure 5, so it would make sense to keep at least both models.

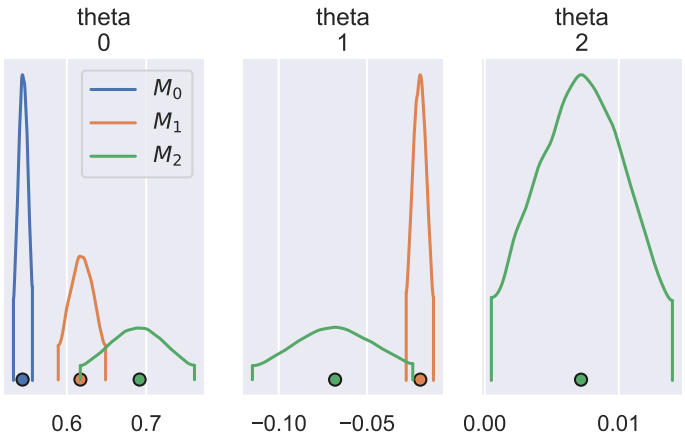


Fig. 5: Parameter posteriors M_0, M_1, and M_2 obtained via *SMC* sampling with 5000 samples in the final tempering step. The distributions cover a 90% credibility interval around the mode marked with a dot.

Bayesian model averaging (*BMA*) [20] offers a solution to this dilemma; instead of choosing a single winning model, we select the *entire* set of available models $\mathcal{M}$, i.e., M_0, M_1, and M_2 and average all subsequent predictions across this set.

The marginal likelihoods (or the *BIC*-score) allows us to compute the probability $P(M_i|D)$, i.e., the probability of model M_i being true given the observed data $\mathcal{D}$. We simply use Bayes' theorem Eq. (4) to write

$$p(M_i|\mathcal{D}) = \frac{p(\mathcal{D}|M_i)p(M_i)}{\sum_{j=0}^{K-1} p(\mathcal{D}|M_j)\, p(M_j)}.$$

This expression also requires us to reveal our prior belief $p(M_i)$ in each of the models we consider, i.e., the relative probability of model M_i being true, without conditioning on the data. Either we have some information that leads us to believe more in some models, or we simply assign a uniform prior over the set of K considered models

$$p(M_i) = (K)^{-1}.$$

There are several subtleties regarding model priors and model averaging. First and foremost, we must acknowledge that the set of models will (most likely) not contain the true data-generating process. So, what to expect when we average a set of *wrong models*? Certainly not the right answer! At least not for the right reasons. Still, acknowledging the fact that our models are wrong, one can hope to generate a more conservative uncertainty estimate if one deploys a method to account for the inference across several models. There exist many ways to *average* over models. For the interested reader, we refer to a recent review [21] where the many avenues of model averaging are elaborated in great detail.

We will follow a simple and traditional *BMA* strategy here, and assign $p(M_i) = 1/3$ for $i = 0, 1, 2$. In our case we could also use the exact $\ln[p(\mathcal{D}|M_i)]$ values that can be obtained analytically, but to maintain some realism we will instead use the marginal likelihoods from the *SMC* runs. To evaluate the posterior model probabilities we need to compute

$$\ln\left[p(M_i|\mathcal{D})\right] = \ln\left[p(\mathcal{D}|M_i)\right] + \ln\left[p(M_i)\right] - \ln\left[\sum_{j=0}^{K-1} \exp\{\ln[p(\mathcal{D}|M_j)\, p(M_j)]\}\right]$$

since all marginal likelihoods are evaluated on a logarithmic scale. This was done in order to avoid underflow and overflow problems. The normalization term is a log-sum-exp expression which must be handled with care numerically. One way is to pull out the largest exponent x_m and compute

$$\ln\left[\sum_j \exp\{x_j\}\right] = x_m + \ln\left[\exp\{x_0 - x_m\} + \exp\{x_1 - x_m\} + ... + 1 + ... + \exp\{x_{K-1} - x_m\}\right].$$

This evaluation of the log-sum-exp is more stable to under/overflow problems that might occur with brute-force evaluation.

In the end, we obtain the following posterior model probabilities

$$p(M_0|\mathcal{D}) = 0.099 \qquad p(M_1|\mathcal{D}) = 0.874 \qquad p(M_2|\mathcal{D}) = 0.028.$$

The model probability $P(M_i|\mathcal{D})$ allows for straightforward weighted averaging of the posterior distribution for *any* quantity of interest, e.g., the posterior predictive for future data $\mathcal{F}$

$$p(\mathcal{F}|\mathcal{D}) = \sum_{j=0}^{K-1} p(\mathcal{F}|M_j, \mathcal{D})\, p(M_j|\mathcal{D}).$$

In the figure below we show draws from the posterior predictive (in grey) extending up to $x \approx 7$, i.e., beyond the domain where we calibrated the models.

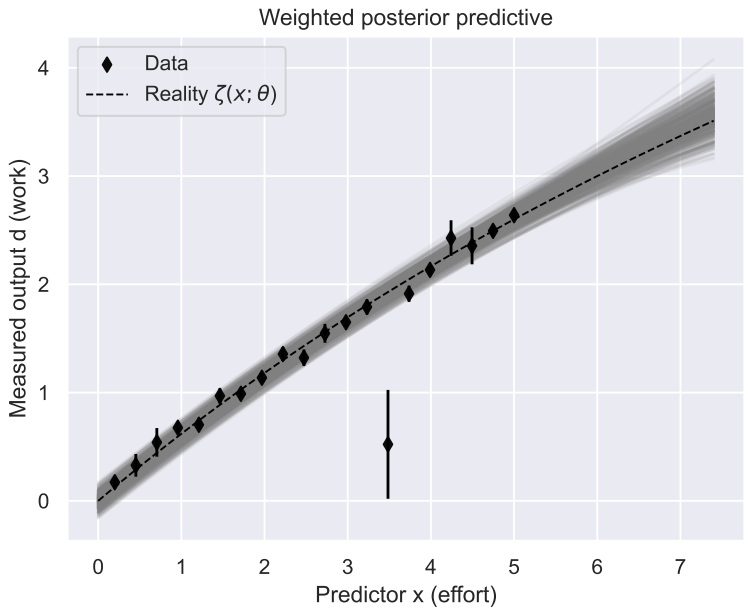


Fig. 6: Bayesian average of the posterior predictive distributions for M_0, M_1, and M_2.

It is possible to average, e.g., the posterior predictive distributions, see Figure 6 or the model parameters themselves, i.e.,

$$p(\theta_i|\mathcal{D}) = \sum_{j=1}^{K-1} p(\theta_i|M_j, \mathcal{D})\, p(M_j|\mathcal{D}),$$

see Figure 7.

This average is one way to estimate the posterior *PDF* of a parameter in the presence of model uncertainty. However, we did not build in any prior knowledge about the model discrepancy term like we did in Chapter 2. Still, the model average does provide a more conservative estimate of the physical parameter θ_T.

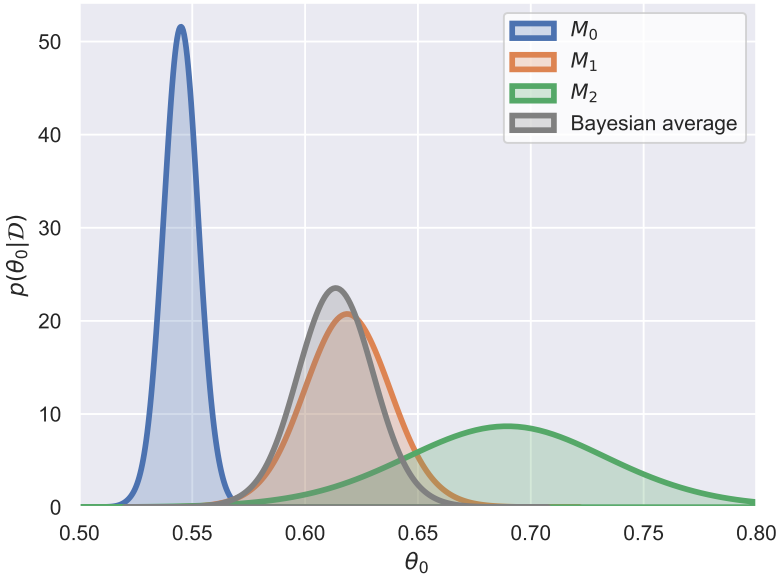


Fig. 7: Bayesian average of the parameter posterior for θ_0 in models M_0, M_1, and M_2.

4.5 What did we learn?

We often have multiple models available for explaining/predicting data. One possible strategy for handling such situations is to employ a *Bayesian model averaging* approach whereby we combine and use all available models for inference and predictions. This method relies on computing marginal likelihoods for all models (or atleast a BIC/AIC/…-score) and placing priors on each model. Both of are challenging to quantify. The former is computationally challening, whereas the latter (the model prior) is conceptually difficult to define. Nevertheless, the resulting model average provides a more conservative estimate of the model uncertainty. Indeed, acknowledging the existence of multiple models implicity indicates that there are model discrepancies. There are many pitfalls in model averaging. The marginal likelihoods of multiple models can differ exponentially, and the resulting average can easily become dominated by a single model. But in a situation where many different models describe the data equally, the average prediction is worth considering.

CHAPTER

FIVE

ADVANCED REGRESSION

Key take-away messages

- Error correlation can be important in many applications and can be handled via the error correlation matrix.
- Ridge regression, Bayesian ridge regression, and automatic relevance detection (*ARD*) regresssion provide means to combat overfitting, where the latter two include automated hyperparameter estimation. Which approach is best suited can be sensitive to problem size since the techniques incur different computational costs.
- Robust regression provides a means to deal with outliers. It can be practically achieved using the Huber regressor.
- Cross validation (*CV*) is a frequentist technique for measuring the predictive power of a model, which can be useful for computational reasons as a complement/alternative to Bayesian techniques.
- Sparse models are models that are constructed on a large parameter space but only few parameters are non-zero. They can be obtained using feature selection algorithms such as the *LASSO*, automatic relevance detection (*ARD*) regression or recursive feature elimination (*RFE*).

In previous courses (in particular "Learning from Data") as well as the first chapters of these lecture notes you have already learned about techniques such as *OLS*, *LASSO* or ridge for solving such linear problems and their relation to Bayesian statistics. Here, we will recap these techniques and develop them further.

Try it out yourself!

The techniques described in this chapter are demonstrated in several jupyter notebooks that can be found in the `notebooks/advanced-linear-regression` directory of the `tif345` repository.

Tip

The book by Murphy is an excellent starting point if you want to dig deeper [22]. The most relevant chapters are 7 "Linear regression" (in particular 7.4 and 7.5) and 13 "Sparse linear models" (in particular 13.3 and 13.7).

Note

The notation used in scikit-learn differs from the one used in these lecture notes. The two notations are related as follows

- design or sensing matrix: $\Phi \to X$
- vector of parameters/regression weights: $\theta \to w$
- data: $d_i \to y_i$, $\mathcal{D} \to y$

The scikit-learn notation is similar but not identical to the one used by Murphy. For example Murphy expresses the residual as $w^T x - y$. x is thus a column vector representing the state (set of control variables) associated with a measurement. The non-bold y indicates a single data point.

5.1 Ridge regression and beyond

5.1.1 Ridge regression

A problem that is commonly encountered in any form of regression is overfitting. This can occur, e.g., with *OLS* if the model has sufficiently many *DOF*s to "reproduce" the noise in the data as demonstrated in Section 5.1.1. A closer inspection of the coefficients of the polynomial that is obtained by *OLS* shows them to be spread out over a rather large range of parameters.

Following the Bayesian logic, it is hence indicated to add a prior for the parameters and as often the simplest choice is a Gaussian one with zero mean and a variance of λ^2,

$$p(\theta) = \prod_j \mathcal{N}(\theta_j|0, \lambda^2) = \mathcal{N}(\theta|0, \lambda^2).$$

Note that here λ^2 applies for *all* parameters in the same way. Also note that since we choose λ^2 in principle *a priori* we create an *informative* prior. In practice one then finds "good" values for λ^2 using, e.g., cross-validation (*CV*; see Sect. 5.5.1).

To find the solution to the ridge optimization problem, one needs to minimize the following loss function

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(d_i - (\theta_0 + \Phi_i \theta)\right)^2 + \alpha ||\theta||_2^2,$$

where $\alpha = \sigma^2/\lambda^2$, $||\theta||_2^2 = \theta^T \theta$ is the ℓ_2-norm of the parameter vector and θ_0 is the intercept. For $\alpha = 0$ this expression reduces to the one for *OLS*. The technique introduced in this fashion is most commonly referred to as ridge regression and, as illustrated in Fig. 1, it indeed leads to a marked improvement of the smoothness of the model. This is also apparent from the coefficients themselves, which are distributed over a much smaller range now.

Try it out yourself!

If you want to explore robust regression yourself you can do so using the `ridge-regression-1.ipynb` and `ridge-regression-2.ipynb` notebooks. There is an even a notebook with an interactive widget that allows you to add and remove data as well as change the hyperparameters of ridge, Lasso, and *OLS* dynamically.

5.1.2 Bayesian regression and Bayesian ridge regression

The hyperparameter associated with ridge regression is commonly tuned "by hand", using, e.g., a *CV* score to find a suitable value. In Bayesian regression techniques this approach is refined further by including the regularization parameters themselves in the estimation (optimization) procedure. This is commonly achieved by introducing uninformative priors for the hyperparameters.

Following this logic, the output is assumed to be Gaussian distributed around $\Phi\theta$

$$p(\mathcal{D}; \Phi, \theta, \sigma) = \mathcal{N}(\mathcal{D}|\Phi\theta, \sigma^2)$$

where σ is a random variable that can be estimated from the data.

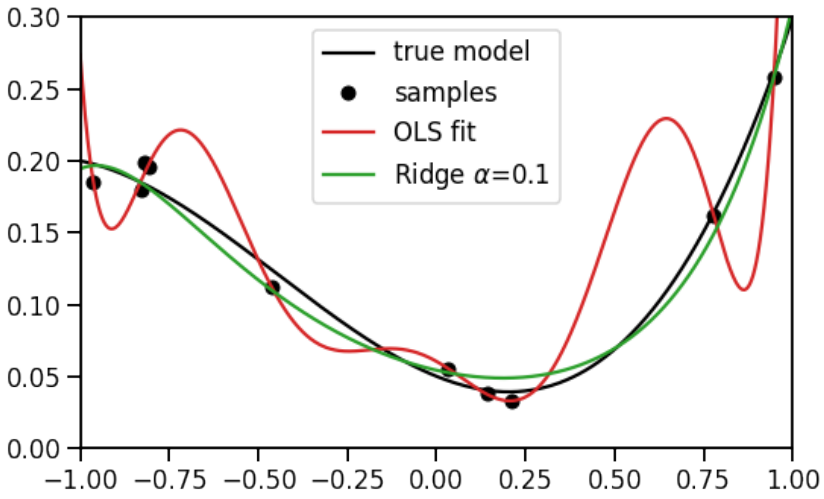


Fig. 1: Comparison of linear models obtained using different regression techniques against noisy data. The *OLS*-fitted model exhibits overfitting. See `ridge-regression.ipynb` for details.

In Bayesian ridge regression we choose the prior for the regression weights θ to be given by a *spherical* Gaussian such that

$$p(\theta|\lambda) = \mathcal{N}(\theta|0, \lambda^{-1}I).$$

The respective priors for the hyperparameters σ and λ are taken as Gamma distributions and hence as conjugate priors to the Gaussian (see Sect. 2.2). And the optimal values for σ and λ are estimated along with θ by maximizing the log marginal likelihood. There are additional hyperparameters associated with the Gamma distributions for the priors for $\sigma and \lambda$, which are chosen to be non-informative. In practice this means that default values are used that typically need no adjustment.

5.1.3 Automatic relevance detection (ARD) regression

One can of course increase the complexity of the probabilistic model further. For example, one can replace the assumption of *spherical* Gaussian in Bayesian ridge regression with an *elliptical* Gaussian. In other words, instead of assuming the covariance matrix to be unity scaled by a scalar, it is assumed to be diagonal but with arbitrary elements on the diagonal. This means we assume the prior for the regression weights to be

$$p(\theta|\Lambda) = \mathcal{N}(\theta|0, \Lambda^{-1}),$$

where Λ is positive definite and diagonal, i.e., $\Lambda = [\lambda_1, \lambda_2 \ldots \lambda_{N_p}]$.

The number of new hyperparameters scales with the number of regression weights $N_p = \|\theta\|_0$ as there are two new hyperparameters associated with the gamma distributions used for the priors of each individual parameter.

In *ARD* one also commonly includes a pruning step, where all parameters for which the statistical significance is too low are set to zero. In the `scikit-learn` implementation this is handled through the specification of "threshold" $\lambda_{\text{threshold}}$. Any parameter for which the associated λ_i is above this threshold is set to zero[1].

The increase in convenience provided by the "automated" hyperparameter estimation comes at a computational cost. This is illustrated in Figure 7 for a fixed problem size (N_p = const.) but variable number of data points N_d. Yet for many small to moderately sized problems the computational cost is still almost negligible.

[1] The `scikit-learn` documentation specifically states that $\lambda_{\text{threshold}}$ is the "[t]hreshold for removing (pruning) weights with high precision". Here, *precision* means the inverse variance.

Try it out yourself!

In project 2a you will be using *ARD* regression as one of the methods for optimizing the parameters of a cluster expansion (*CE*).

5.2 Robust regression

You have already seen that *OLS* can be derived by realizing that basic linear regression corresponds to a model of the form

$$p(\mathcal{D}|\Phi,\theta,\sigma) = \mathcal{N}(\mathcal{D}|\Phi\theta,\sigma^2),$$

where Φ is the design or sensing matrix. Each row of this matrix corresponds to an input vector (e.g., a cluster vector in the case of a *cluster expansion* or a displacement product vector in the case of a *force constant expansion*). θ are the regression weights (e.g., the *ECI* in the case of a *CE* or the symmetry reduced and constrained parameters of a *FC* expansion), $\mathcal{D}$ is the prediction (e.g., the energy in a *CE* or a force component in a *FC* model) and σ^2 is a variance representing the noise in the input data.

When applied to a set of data with some normal noise and sufficiently many data points *OLS* works well, as expected. If there are, however, outliers present and the model has sufficiently many *DOF*s (i.e., parameters), the *OLS* fit fails badly. This behavior is related to the assumption of a Gaussian likelihood that underpins *OLS* (as well as *LASSO* and ridge (Section 5.1.1), also see Table 1). One strategy to avoid problems with outliers is to switch from the Gaussian distribution for the likelihood to a distribution with a so-called "heavy tail", e.g., a Laplace or Student distribution, both of which admit a higher probability for outliers than a Gaussian distribution.

If one adopts a Laplace distribution one obtains for the likelihood

$$p(\mathcal{D}|\Phi,\theta,b) \propto \exp\left(-\frac{1}{b}|\mathcal{D}-\Phi\theta|\right).$$

Compared to the Gaussian distribution the argument of the exponential thus contains the absolute deviation between model prediction and target data, rather than the square. This form leads to a discontinuity in the first derivative, as a result of which it is hard to optimize. It is, however, still possible to achieve a solution by using linear programming.

A more pragmatic solution is available via the Huber regressor, which optimizes the squared residual for errors that lie within a specified range and the absolute loss for outliers. The loss function is defined as

$$\mathcal{L} = \sigma + \sum_i^{N_d} H_\epsilon\left(\frac{\Phi_i\theta - d_i}{\sigma}\right) + \alpha\|\theta\|_2^2,$$

where Φ_i denotes the i-th row of Φ and

$$H_\epsilon(z) = \begin{cases} z^2 & \text{for} \quad |z| \leq \epsilon \\ 2\epsilon|z| - \epsilon^2 & \text{otherwise} \end{cases}.$$

If the scaled residual $(\Phi_i\theta - d_i)/\sigma$ is thus smaller than ϵ, one recovers *OLS* if $\alpha = 0$ and ridge regression if $\alpha > 0$. The great advantage of this approach is that both the loss function and its first derivative are smooth and hence one can employ standard smooth optimization techniques.

Note

You can find more examples in the implementation of the Huber regression in scikit-learn.

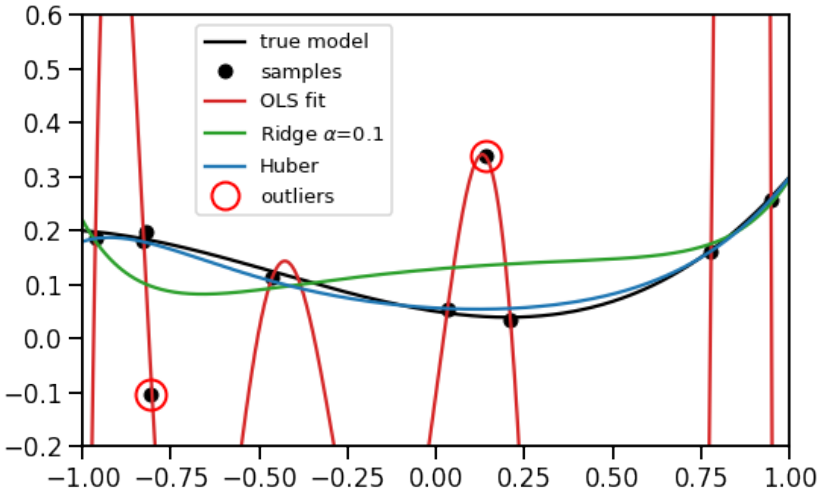


Fig. 2: Comparison of linear models obtained by training against data with outliers using different regression techniques. Only the Huber regressor provides a sensible representation of the original data. See `robust-regression-1.ipynb` for details.

Try it out yourself!

If you want to explore robust regression yourself you can do so using the `robust-regression-1.ipynb` and `robust-regression-2.ipynb` notebooks.

5.3 Error correlation

5.3.1 Recap of earlier key results

In Sect. 1.3 we discussed the relation between model and reality. Specifically we established that the observations $\mathcal{D} = \{d_i\}_{i=1}^{N_d}$ are related to the model $M(\theta; x_i)$ according to

$$d_i = M(\theta; x_i) + \varepsilon_i + \delta(x_i),$$

where θ denotes the parameter vector, x_i is a vector of control variables that specifies the particular state the system to be model resides in, and $\delta(x_i)$ is the in principle unknown model discrepancy. Moreover, in Sect. 1.4, we arrived at the *OLS* estimator (Sect. 1.4.4), which gives a closed solution for the parameter vector $\hat{\theta}$ for (generalized) linear models (Sect. 1.4.2) and arises from the minimization of the loss function $\chi^2(\theta)$.

$$\hat{\theta} = \arg\min_{\theta} \chi^2(\theta).$$

Building on this result, we then discussed the principles of Bayesian parameter estimation (Sect. 1.5). Under the assumption of *i.i.d.* errors parametrized with a common variance σ^2, this allowed us to write the likelihood for such a model as

$$p(\mathcal{D}|\theta, \sigma^2) = \mathcal{N}(\mathcal{D}|\Phi\theta, \sigma^2\mathbf{1}) = \left(\frac{1}{2\pi\sigma^2}\right)^{N_d/2} \exp\left\{-\frac{1}{2}\frac{(\mathcal{D}-\Phi\theta)^T(\mathcal{D}-\Phi\theta)}{\sigma^2}\right\}. \tag{1}$$

We will now build on and generalize this result.

5.3.2 A more general form

Equation (1) readily generalizes to non-linear models, if we use the regression residuals $\hat{\varepsilon} = \hat{\mathcal{D}} - \mathcal{D}$, where $\hat{\mathcal{D}}$ is the data prediction based on our (general non-linear) model. In addition, we can drop the assumption of *i.i.d.* errors and use a general error correlation matrix Σ, in which case we obtain

$$p(\mathcal{D}|\theta, \Sigma) \propto \exp\left\{-\frac{1}{2}\hat{\varepsilon}^T \Sigma^{-1} \hat{\varepsilon}\right\}.$$

Now what could this error correlation matrix Σ encode? Imagine, for example, an experiment, in which we measure some form of spectrum, say an absorption spectrum. The sensitivity of the spectrometer that we are using is most likely going to vary with the wave length of the light and hence the error of the measurement would be different in different parts of the spectrum. We can encode this knowledge in our model by allowing the diagonal elements of the error correlation matrix Σ to assume different values for measurements d_i at different wave lengths. This implies that instead of assuming a *spherical* Gaussian for the error distribution, we allow for an *elliptical* shape.

There can even be situations when the errors of individual measurements d_i are correlated. This is then expressed through non-zero off-diagonal elements of Σ. A specific case where it is crucial to include these effects are diffraction experiments, where the error correlates with the square root of the intensity of the signal.

The structure of the error correlation matrix is illustrated in Figure 3.

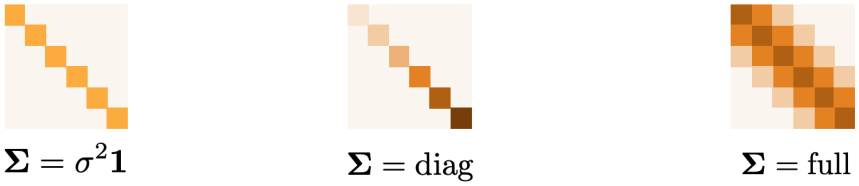


Fig. 3: Schematic illustration of the error correlation matrix for *i.i.d.* errors (left), independent but non-identically distributed errors (center), and correlated errors (right).

Tip

In the lecture we will discuss an explicit example that demonstrates the importance of error correlation in the case of neutron diffraction data.

5.4 Additional considerations

5.4.1 Parameter correlation

It is very common that the parameters of a model exhibit some degree of correlation. Examples that we discuss within this course are *cluster expansions* (see in particular Sect. 6.1.3) and *force constant expansions* (Sect. 6.2.4. The assumption of a *spherical* Gaussian distribution that underlies *ridge* and *Bayesian ridge regression* is therefore poorly motivated. We already saw in Sect. 5.1.3 how one can relax the assumption of a *spherical* Gaussian in favor of an *elliptical* Gaussian in the case of *ARD* regression. In that case, the hyperparameter selection is even handled automatically. For larger problems this can, however, become impractical (see Figure 7) and it certainly not handle parameter *correlation*.

In physics we often knowledge about the system hand ("physical intuition") that translates, e.g., to expectations regarding the magnitude of parameters or even correlation between parameters. In Section 6.1.4, we will explore this concept for the specific case of a cluster expansion. Here, we only focus on the conceptual side.

In the previous chapters and sections, we have considered a variety of different approach to estimating model parameters. Generally these schemes involve a (or several) loss functions (even if we might not always write it out). For the sake of this discussion let us consider the general form of an ℓ_2-regularized regression.

$$\mathcal{L} = \hat{\varepsilon}^T \Sigma^{-1} \hat{\varepsilon} + \theta^T \Lambda^{-1} \theta.$$

Here, the parameter correlation enters via Λ while the *error correlation* is expressed via Σ. It is impractical for most applications to allow either of these matrices to be "free". Rather we typically need to introduce simplified functional forms that convey our "physical" intuition. If the parameters represent for example interactions at different distances r, we could parametrize Λ such that the diagonal elements decrease with increasing interaction distance. For example, we could choose $\Lambda = [\lambda_1, \lambda_2 ... \lambda_{N_p}]$ with $\lambda_i \propto 1/r_i$, where r_i is the distance associated with parameter θ_i. In other words, we expect the associated parameters θ_i to be smaller the longer the distance.

Question

Can you think of other examples?

5.4.2 Physical interpretation of hyperparameters

In this context, it is important to emphasize the specific form of the loss function as specified above. Commonly the loss function for ridge regression is written as

$$\mathcal{L} = \hat{\varepsilon}^T \hat{\varepsilon} + \alpha \theta^T \theta,$$

with only one hyperparameter α. This makes sense. The result for the maximum likelihood estimate generated by this loss function is invariant to arbitrary rescaling of the loss function. What we have lost in this step, however, is insight into the meaning of the hyperparameter. If we write, however,

$$\mathcal{L} = \frac{\hat{\varepsilon}^T \hat{\varepsilon}}{\sigma^2} + \frac{\theta^T \theta}{\lambda^2}$$

with the (scalar) data variance σ^2 and the (scalar) parameter variance $\lambda^2 = \sigma^2/\alpha$, we regain an intuition for the meaning of these parameters. It is therefore useful to keep in mind when using these algorithms that the hyperparameters do in principle carry useful information about our model.

5.5 A brief detour into frequentist statistics

It is useful to note that the regression techniques introduced above via a Bayesian approach, can also be obtained without an explicit Bayesian perspective. Consider the following loss function

$$\mathcal{L}(\theta) = \frac{1}{N_d} \sum_{i=1}^{N_d} \left(d_i - (\theta_0 + \Phi_i \theta)\right)^2 + \alpha_1 ||\theta||_1 + \alpha_2 ||\theta||_2^2.$$

This expression becomes equal to *OLS* for $\alpha_1 = \alpha_2 = 0$, to *LASSO* for $\alpha_2 = 0$ and to ridge regression for $\alpha_1 = 0$. If both α_1 and α_2 are non-zero, the above expression is known as elastic net.

Purely for the sake of optimization it is sufficient to consider the loss functions for these techniques and find by some means (a) suitable value(s) for the hyperparameters. We do not to consciously assign priors to the hyperparameters. In fact, we can often get away with a simple line scan, i.e., vary the respective hyperparameter and observe the change in the quality of the model.

5.5.1 Cross validation

To measure the quality of the model, the most widely employed strategy is *CV*, which is defined in the framework of frequentist statistics. The basic idea is to split the available reference data into sets used for training and testing. In some cases, one can even set apart a further data set for validation. Note that the nomenclature here can differ a bit between different publications and implementations.

Formally, let us assume that there are N_c data cases in the data set $\mathcal{D}$. The data can be split in several different "folds" (groups of cases). We denote the set of cases that belong to the k-th fold by $\mathcal{D}_k$ and all other cases by $\mathcal{D}_{-k}$.

For notational convenience, we denote the parameter vector returned by a learning algorithm $\mathcal{F}$ for a model M applied to a set $\mathcal{D}$ as

$$\hat{\theta}_M = \mathcal{F}(\mathcal{D}, M).$$

Furthermore, we denote with $\mathcal{P}$ a prediction function that takes an input Φ_i and a parameter vector $\hat{\theta}$ and generates a prediction $\hat{\mathcal{D}}$

$$\hat{\mathcal{D}} = \mathcal{P}(x, \hat{\theta}).$$

We can then define the K-fold *CV* estimate of the "risk" of a model M (a frequentist term for model quality) as

$$R(M, \mathcal{D}, K) = \frac{1}{N_c} \sum_{k=1}^{K} \sum_{i \in \mathcal{D}_k} \mathcal{L}(d_i, \mathcal{P}(\Phi_i, \mathcal{F}(\mathcal{D}_{-k}, M))),$$

where $\mathcal{L}$ is the loss function used for training models. Implementing this expression implies that we need to run the fitting algorithm for each fold using each time only the data cases that *do not* belong to the fold $\mathcal{D}_{-k}$. We then measure the performance of the prediction using the data cases that *do* belong to the fold.

There are different strategies to split the data. Some of the more common ones are shown graphically in Fig. 4.

LOOCV is a special case, in which there are as many folds as there are data cases $N_c = K$. When dealing with linear problems and quadratic loss functions (e.g., when using *RMSE* as loss function), the computation of the *LOOCV* score can be simplified. It then only requires a single fit using the full data after which the *LOOCV* score can be computed from the loss function for the training set, corrected for the correlation in the training data.

Try it out yourself!

You can explore cross-validation using the `cross-validation-via-sklearn.ipynb` notebook.

5.5.2 Hyperparameter optimization via CV

Consider now the problem of finding optimal hyperparameters for a particular problem. One approach is to use Bayesian sampling of the hyperparameters directly, which can be achieved "manually" by setting your own prior and posterior. There are also implementations of this approach, see e.g., the Bayesian ridge regression implementation in scikit-learn. Alternatively, this problem can be solved via *CV*, which for large problems can be computationally advantageous.

Try it out yourself!

The `ridge-regression-2.ipynb` notebook notebook demonstrates how to carry out hyperparameter optimization with *CV*.

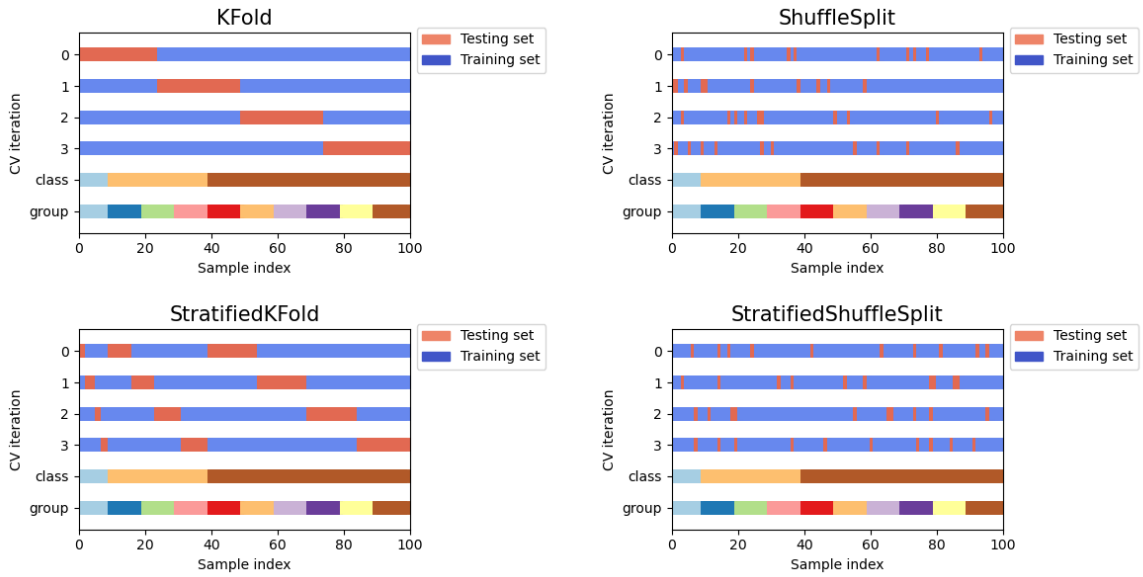


Fig. 4: Strategies for generating folds (splits) for *CV*. Adapted from the documentation of `scikit-learn`. The rows labeled "class" and "data" represent that the underlying data is structured. Samples 1 to 10, 11 to 40 and 41 to 100 belong to different classes of samples, e.g., taken from different sources. The data sets is ordered in this regard and hence it makes a difference whether the data is split up as a whole (as in K-fold) or whether one ensures that the relative size of the samples from the different classes is preserved (as in stratified K-fold). One can also shuffle the data set prior to splitting, using either a simple split or a stratified split.

5.6 Feature selection and sparse models

We now come to the important topic of feature selection and sparse models. Often times the parameter space is much larger than the number of data cases that we have available for training a model. We can, for example, easily obtain such a situation by increasing the cutoffs and expansion order in a *CE* model (see Fig. 5). While in some cases one might be able to compensate for this increase in model complexity by adding more training data, this approach is often not feasible. At the same time, suitable models should not require many non-zero parameters. In other words, as the number of parameters grows most of those that are added should be zero and the task at hand is thus to find out which ones to keep. This is the task of feature selection, which is intended to yield sparse models, i.e., models with few non-zero parameters (features) and hence a sparse populated parameter vector.

In Section 6.1.4, we discussed an approach to achieve this goal by incorporating physical insight via Bayesian priors. But such an approach is not always possible, e.g., because we do not have physical insight or maybe because there are too many cases to consider as in a high-throughput investigation. There is thus a need for automated or at least semi-automated feature selection techniques.

5.6.1 A physicist's perspective

It is helpful to recall that our parameter space is connected to a basis. If we develop a description of a system under study, we usually try to find a basis that is "well suited" for the problem at hand. What this usually means a basis that allows us to represent the system with as few components and unknowns as possible.

If we try to compute the spectra of atoms, a convenient representation of the wave function is in terms of localized hydrogen-like orbitals, while for a periodic solid a periodic basis set such as plane waves is favorable. While it is in principle possible to represent the wave function of an atom in plane waves (since they form a complete basis set), convergence with the number of plane waves (the size of the basis set) is very slow.

The first step in achieving a good representation is thus the selection of a basis set or parameter space, followed by finding (selecting) the basis functions (features) that are the most important. In physics we are accustomed to carrying out these tasks based on physical insight and experience. This approach can be used for example for the construction of *CE*s and *FC* models, where the tunable parameters are the cutoff radii for clusters of increasing order (pairs, triplets etc; see Section 6).

The manual feature selection approach based on *OLS* has been used for example in [23] to construct a high-order *FC* models for silicon (Fig. 5). The *OLS* data (we will discuss the other techniques in the next section) shows an improvement in the *CV*-*RMSE* score up to a third-order cutoff around 4 Å, after which the performance becomes worse again, indicative of overfitting (Fig. 5a). The *OLS* solution is fully dense and the increase in the number of features mirrors the increase in the number of model parameters with increasing cutoff (Fig. 5b). Using the thereby manually selected cutoff of 4 Å, one can predict the lattice thermal conductivity at the same level of accuracy as when using more advanced automated feature selection techniques.

5.6.2 Automated feature selection

Manual feature selection is not always a manageable solution. In this context, feature selection algorithms aim to automate this task in general fashion. There is a range of feature selection algorithms and they can be rather complex in practice. Here, we therefore only consider the basic principles and illustrate how these algorithms work through examples.

A key aspect of any feature selection algorithm is to introduce a penalty for having many non-zero parameters. One way of stating the problem is to simply require the number of non-zero parameters to be as small as possible. Formally, this means that we require the ℓ_0-norm of the parameter vector $||\theta||_0$ to be as small as possible.

The ℓ_0-norm is mathematically challenging since it is by its nature discrete. There are some non-Bayesian algorithms that solve this task iteratively such as *RFE* and *OMP*. A Bayesian approach is provided by the spike and slab model, which is based on assuming a Bernoulli distribution for the prior.

An alternative is to use the ℓ_1-norm, which leads us to *LASSO*, recalling its loss function

$$\mathcal{L}(\theta) = \hat{\varepsilon}^T \hat{\varepsilon} + \alpha_1 ||\theta||_1.$$

While this loss function still has a discontinuous first derivative there are efficient algorithms for its minimization as you have already seen earlier in this course.

It turns out that "bare" *LASSO* has a tendency to overselect, i.e., the solutions are not as sparse as they could be. This is already apparent in Fig. 5a but illustrated more explicitly in Fig. 2, using again for third-order silicon *FC* model already considered above. Here, however, the parameter space was deliberately chosen to be very large by using cutoff radii of 9.65 Å, 9.65 Å, and 2.5 Å for pair, triplet, and quadruplet terms, yielding in total 2525 parameters.

The over-selection of *LASSO* can be at least partly remedied by suitable extensions of the underlying priors and posterior. Probably the two most notable examples are adaptive *LASSO* [24] and group *LASSO*. A simplified way to view these methods, adaptive *LASSO* in particular, is to realize that a single hyperparameter α_1 that is homogeneously applied to all parameters is often not optimal. Rather the hyperparameter can be adapted for each model parameter. In fact, adaptive *LASSO* leads to a notable improvement relative to *LASSO* as shown Fig. 2.

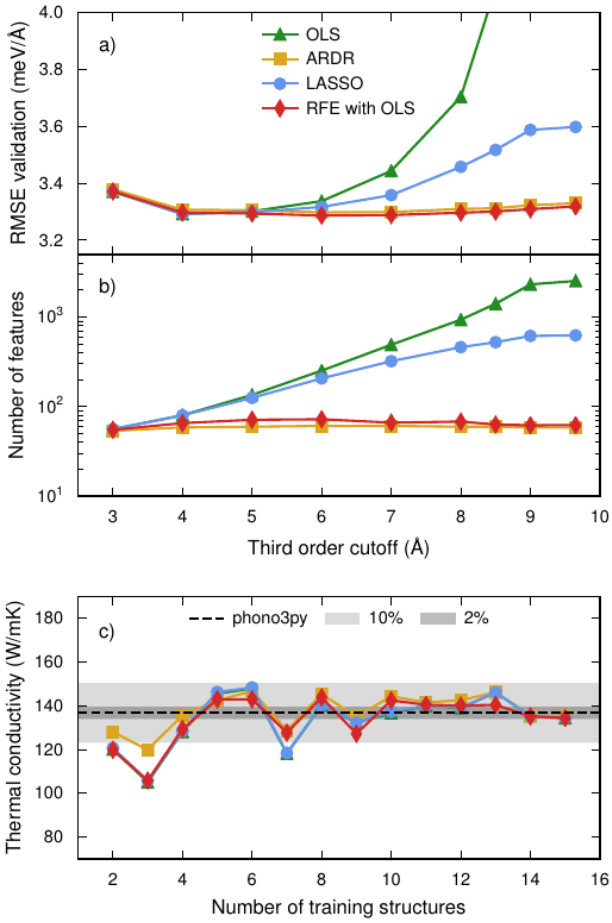


Fig. 5: Extraction of *FC*s by regression in combination with manual feature selection. The goal is to obtain a third-order *FC* expansion for silicon that can be used to predict the (lattice) thermal conductivity (by solving the Boltzmann transport equation). (a) Variation of the *RMSE* over the validation set as obtained via K-fold *CV* as well as (b) the number of features (non-zero parameters) with the third-order cutoff (the pair cutoff is fixed).
© Thermal conductivities at obtained from *FC* expansions using a third-order cutoff of 4 Å.

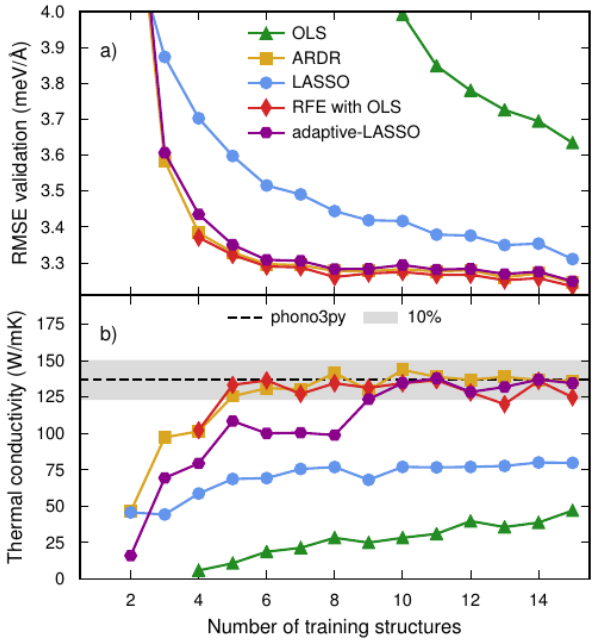


Fig. 6: Extraction of *FC*s by regression with automated feature selection for the same example as in Fig. 5. The hyperparameters were chosen to minimize the *CV*-*RMSE* for each data point. (a) Without minimal constraints on the parameter space all algorithms require more input data in order to achieve converged results. Unsurprisingly this is particularly pronounced for *OLS* but even *LASSO* converges rather slowly. *ARD* regression, *RFE* with *OLS* and adaptive *LASSO* perform, however, much better. (b) Importantly considering the *RMSE* from *CV* alone is insufficient to obtain a reliable model. This is apparent by comparing the prediction for an actual quantity (here the thermal conductivity). Here, only *RFE* and *ARD* regression achieve reasonable results with a very limited amount of data. For reference an explicit numerical evaluation of the third-order *FC*s requires one to calculate the forces of about 100 structures.

The ℓ_2-term in the loss function of ridge regression also imposes a penalty on the parameter vector although it commonly does not induce sparsity (i.e., strictly zero parameters). The underlying idea can, however, be generalized and improved to obtain *ARD* regression (see Sect. 5.1.3), which is a very elegant and powerful method for some applications. Recall that the starting point is similar to ridge regression assuming a prior for the parameter vector that is Gaussian distributed

$$p(\theta) = \mathcal{N}(\theta|0, \Sigma),$$

where the covariance matrix Σ is chosen to be diagonal. In contrast to ridge regression, the variance is, however, allowed to vary for each element of the parameter vector and Gamma-distributed priors are added for each (diagonal) element of Σ. If one follows through with a derivation of the posterior, one obtains a method that is called "sparse Bayesian learning" or automatic relevance detection (*ARD*) regression, which then also includes a pruning step, in which parameters whose numerical significance is below a certain threshold are explicitly set to zero.

ARD regression performs remarkably well for small to intermediate size problems including *CE*s [25] and smaller *FC* models [23]. Thanks to it being rooted in Bayesian statistics it also does not require *CV* in order to obtain estimates for the distribution of errors. Unfortunately it scales more strongly with problem size than other methods, which limits its applicability to large problems (see Fig. 7).

Comparing the convergence of the automated feature selection algorithms with training set size (Fig. 6b) to manual feature selection (Fig. 5c), it is apparent that in the former case all algorithms require more input data in order to achieve converged results. Unsurprisingly this is particularly pronounced for *OLS* but also the convergence of *LASSO* is rather slow. *ARD* regression, *RFE* with *OLS* and adaptive *LASSO* on the other hand perform much better, requiring only slightly more data than in the manual feature selection case.

It is very instructive to also compare the performance of the different models for the prediction for a quantity that is *derived* from the model, such as the thermal conductivity. This is a simple example for a sensitivity analysis, a topic that we will return to in Section 8. Here, *RFE*, *ARD*, and adaptive *LASSO* produces models that succeed in recovering the reference value.[1] The results (Fig. 6a,b) highlight that *CV*-*RMSE* alone is a poor predictor for the ability of a model to predict the lattice thermal conductivity. The *CV*-*RMSE* scores for *ARD*, *RFE* with *OLS*, and adaptive *LASSO* are practically identical, yet the convergence with the number of training structures is much slower in the latter case. Maybe even more striking is that the *CV*-*RMSE* score for *LASSO* in the large training set limit is barely larger than for any of the other feature selection techniques, yet these models still underestimate the thermal conductivity by more than 30%. Even more troublesome is that the prediction appears to have converged with respect to the training set size already for about 6 structures. We will return to this example in Section 8.

It is must, however, be emphasized that the above results should not be generalized. Rather one should be mindful of the aforedescribed effects when approaching new problems.

5.6.3 Scaling of feature selection algorithms

One aspect that can become relevant in practice is the scaling of the different algorithms (Fig. 7). Here, *OLS*, *RFE* with *OLS* and *LASSO* are merely offset by a constant factor. This is the case since the major computational cost in the latter two cases is an *OLS* optimization, which is carried out repeatedly. *ARD* regression on the other hand exhibits a stronger scaling with system size, as a result of which it cannot be reasonably applied for larger problems. As a result, while *ARD* is extremely well suited for constructing *CE* models, it is too expensive for *FC* expansions, for which *OLS* with manual feature selection or potentially *RFE* are well suited.

Try it out yourself!

You can find data and analysis scripts for the silicon third-order *FC* example in the public gitlab repo https://gitlab.com/materials-modeling/hiphive-examples/. The silicon example is located in `examples/Si_bulk`.

---

[1] In this case, one must calculate the forces of about 100 structures to obtain the reference lattice thermal conductivity.

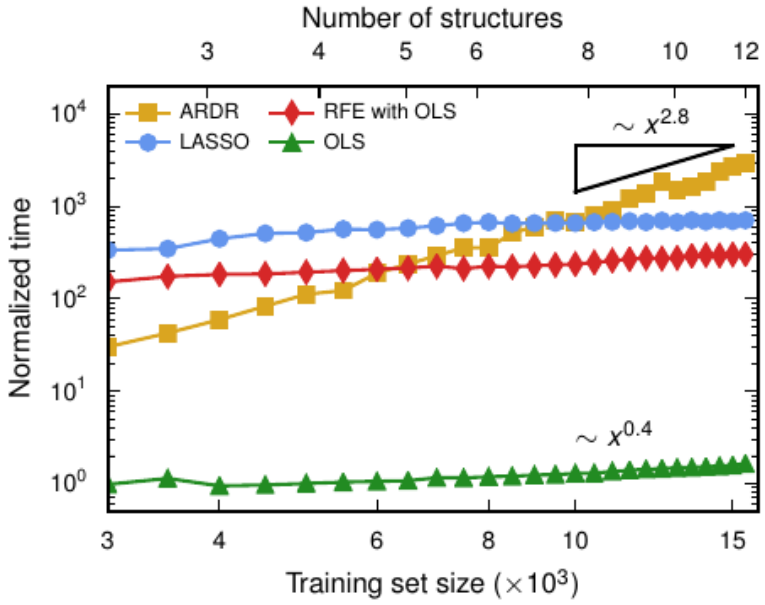


Fig. 7: Scaling of different regression algorithms with the number of *data* points. Relative timings for the computational effort for carrying out a single optimization of a large *FC* model using different regression techniques. Time is normalized using the time for a single *OLS* fit.

5.7 Key take-aways

OLS, ridge, Bayesian ridge, *ARD* regression, and *LASSO* and robust regression are all based on different combinations of likelihood and prior, which arise from different Bayesian considerations. Table 1, which has been adapted and partly extended from Table 7.1 in Ref. [22], provides a helpful overview of how different regression techniques are related in terms of their underlying priors and likelihoods.

Table 1: Overview of important linear regression techniques from a Bayesian perspective. Here, the likelihood refers to the form of $p(\mathcal{D}|\Phi, \theta, \sigma)$ while the prior refers to the form of $p(\theta)$, where θ is the parameter vector.

Method	Likelihood	Parameter prior	Hyperparameter prior
OLS	Gaussian	Uniform	n/a
Ridge	Gaussian	Gaussian	n/a
Bayesian ridge	Gaussian	Gaussian (spherical)	gamma
ARD	Gaussian	Gaussian (elliptical)	gamma
LASSO	Gaussian	Laplace	n/a
Robust regression	Laplace	Uniform	n/a

A similarly compact summary of feature selection methods is difficult. It can, however, be helpful to remember the following: Feature selection generally requires working with a prior for the parameter vector. Where *LASSO* and ridge use the ℓ_1 and ℓ_2-norm of the parameter vector, adaptive (and group) *LASSO* as well as *ARD* regression take a more fine-grained approach that allows for variations in the variance of the parameter prior on a per-parameter basis.

CHAPTER

SIX

TOWARD APPLICATIONS

Key take-away messages (chapter-level)

- To achieve scale bridging one often needs to utilize models of models ("metamodels"). These metamodels provide instructive examples for the use of machine learning in physics. Here, we focus on models that describe interatomic interactions at different levels.
- Some physics-based models can be formulated as linear models and can provide *quantitative* predictions for physical properties while allowing analysis via Bayesian techniques. Examples include cluster expansions (Sect. 6.1) and force constant expansions (Sect. 6.2).
- Historically interatomic interactions have been described via empirical potentials, which have physically motivated functional forms. As a result of their functional form, these models are, however, difficult to fit. As a result they are sometimes referred to as "sloppy models". *ML* techniques such as neural networks can be used to formulate more flexible models that *can* be easier to fit Sect. 6.3).

In the first four chapters, we have worked with a toy example (the "simple machine"), which allowed us to explore and test various aspects of Bayesian statistics and learning with very limited computational effort. In practice, the systems under investigation and their associated models are often more complex and computationally much more demanding. In the second part of this course, we will discuss and work with several such cases.

Examples will be primarily taken from condensed matter and materials physics, which provides several good use cases for the types of models and methods to be discussed here. Most condensed matter physics courses focus on concepts and very simplified model systems that are sufficient for demonstrating phenomena but are ill-suited for capturing the behavior of actual materials. Condensed matter theory has, however, also produced a wealth of methods that enable predictive calculations of (relatively) realistic systems. These methods are rooted in electronic structure theory, with the most widely used approach being density functional theory (*DFT*), which then also serves as starting point to incorporate material specificity in many more advanced techniques, including, e.g., dynamical mean field theory, density matrix theory, GW theory, Bethe-Salpeter equation etc.

DFT and related techniques are so appealing because they in principle merely require a specification of the atomic species and positions in order to compute total energies, forces, wave functions etc. They are therefore often referred to as first-principles or *ab-initio* methods, although some empiricism creeps in via underlying approximations of exchange and correlation effects. *DFT* is the core topic of the course TIF320 ("Computational Materials and Molecular Physics") and is covered in various text books, see, e.g., Ref. [26]. Here, it is sufficient to know that there are methods, rooted in a quantum mechanical description of matter, that allow one to obtain energies and forces (as well as other properties, including spectra) given a certain arrangement of atoms. In the present context, it is furthermore crucial that these methods are computationally extremely demanding. Evaluating energy and forces for a system comprising on the order of about 1,000 electrons takes between roughly 10 minutes and a couple of hours using about 100 cores on a high-performance computing system such as `tetralith` at NSC in Linköping. In addition, the calculations scale strongly with system size and cannot be arbitrarily parallelized. These factors limit the application of these methods for exploring and sampling complex and large configuration spaces. This provides a strong motivation to develop models for interatomic interactions that can reproduce the data from reference calculations within sensible limits (say composition, local structure

etc.) but are computationally much more efficient. We must thus engage in a compromise between reliability, fidelity and computational efficiency. This provides a strong motivation for the utilization of *ML* techniques.

The aforedescribed approach is an example for models of models, or "metamodels", and commonly adopted to enable scale bridging. For example, models that operate on the electronic scale are used to parametrize models that operate on the atomic scale, the latter are used to parametrize mescoscopic model and so on.

This chapter provides background on several physical models that we will discuss in more depth in the lectures and demos. Before we continue please think about the following two questions.

Question 1

Which *DOF*s do you consider to be relevant when sampling a crystalline system, a glass, at a surface or in a multi-component system?

Question 2

Which conditions should models fulfill in order to enable both accurate and efficient sampling of these *DOF*s?

6.1 Alloy cluster expansions

Key take-away messages (section-level)

- Cluster expansions (*CE*s) are lattice-based models that enable one to describe the energetics of crystalline multi-component systems (Sect. 6.1.2).
- The problem of parametrizing these models can be cast in a linear form and is therefore very suitable for linear regression techniques (Sect. 6.1.2) although care must be taken (Sect. 6.1.3).
- Since the parameters in a *CE*s carry physical meaning, we can introduce physical insight in the training process using priors for both the parameters (the diagonal part of the covariance matrix) and the correlation between them (the off-diagonal part of the covariance matrix; Sect. 6.1.4).

Try it out yourself!

Several jupyter notebooks that demonstrate alloy cluster expansions in the context of regression techniques can be found in the `notebooks/alloy-cluster-expansions` directory of the `tif345` repository.

6.1.1 Alloys

In the narrow sense of the term, alloys are mixtures of metallic elements such as bronze, brass or steel. More broadly, however, they refer to crystalline multi-component systems, including also semiconductors (e.g., $Al_xGa_{1-x}N$), and insulators (e.g., zeolites, perovskites). They are ubiquitous in nature and technology, with wide-ranging applications, since the combination of multiple chemical species provides a vast space for emergent behavior and property optimization.

In order to study such systems on the atomistic scale one often resorts to *DFT* calculations, which can provide properties such as the energy, the elastic constants or the atomic displacements of a particular configuration with high accuracy. A

key thermodynamic quantity is the mixing energy per atom of a configuration, which is defined as

$$E_{\text{mix}} = E_{\text{total}}/N - \sum_i n_i E_i,$$

where E_{total} is the energy per atom of the configuration, n_i is the number of atoms of species i present in the configuration, $N = \sum n_i$, and E_i the energy of species i in its reference (typically elemental) form. Working with the mixing energy is often convenient as it is the relevant energy when considering, e.g., the phase stability of mixed systems. Examples for alloy configuration are shown in Fig. 1.

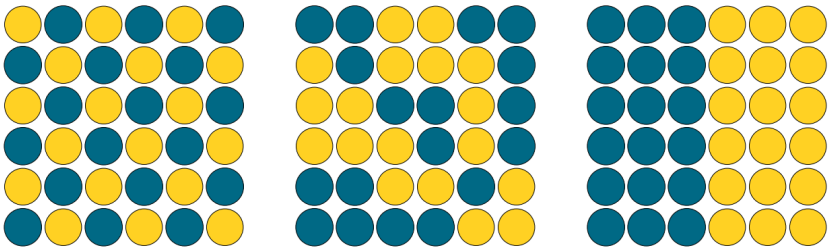


Fig. 1: Examples for alloy configurations at 50% composition on a simple square lattice, including an ordered, a disordered, and an interface configuration.

6.1.2 Cluster expansions

In order to understand the properties of these systems it is pivotal to account for the configurational degrees of freedom (*DOF*s) and the associated entropy contributions. Given the vast number of different configurations, a direct sampling of the compositional configuration space by first-principles techniques (such as *DFT*) is out of the question.[1] One can, however, construct effective lattice Hamiltonians, so-called alloy cluster expansions (*CE*s; often the "alloy" part is omitted), that enable a much more efficient sampling of the compositional configuration space. For illustration, whereas the calculation of the energy of a 100-atom cell with *DFT* typically requires hundreds of core hours, the evaluation of the same configuration using a suitably parametrized *CE* requires only a fraction of a second, equivalent to a speed-up by a factor on the order of 10^6 or more. This dramatic speed-up is enabled by a considerable reduction in complexity (coarse graining) of the original problem, which is a widely used strategy in computational physics and an illustration of the "models of models" approach alluded to above. In the case of *CE*s the reduction in complexity is achieved by replacing the full electron-ion Hamiltonian, which depends on the positions of the electrons ($\{r_i\}$) and ions ($\{R_j\}$) as well as the chemical identities of the ions (i.e., their nuclear numbers $\{Z_j\}$), with a Hamiltonian that limits the ionic positions to a lattice and is only aware of the chemical identities of the components within a (very) limited range of components, typically between 2 and 5.

An illustrating example

The following description aims to convey the core ideas that underpin *CE*s. To this end, we limit ourselves to binary systems, i.e., systems that comprise exactly two different species. The formalism can, however, be extended to an arbitrary number of components in a rather straightforward manner. This requires a more formal and rigorous treatment that is beyond the present scope and not necessary to appreciate the application of statistical learning techniques to these models. The interested reader is referred to, e.g., Refs. [27], [28], [25], and [29], for more details.

The objective of a *CE* is to describe the variation of a property of interest, most commonly the energy, with the chemical configuration, i.e., the distribution of different species over a lattice. Formally, we can describe a particular chemical configuration by a vector σ, the elements of which denote which lattice site is occupied by which species. To be specific, in a binary system, we can denote species A by -1 and species B by $+1$. The interface configuration in Fig. 1 could then be denoted as $\sigma = [-1, -1, -1, -1, -1, -1, +1, +1, +1, +1, +1, +1, ...]$, assuming that the first site is in the upper

[1] If we consider a binary system (A-B) and a lattice consisting of 100 atoms, this yields $2^{100} \approx 10^{30}$ unique ways of occupaying the lattice.

left corner and we enumerate by row. While this is a possible description, it is not a very convenient one. For example, we know that shifting the interface by unit cell to the right (assuming periodic boundary conditions) does not alter any global property of the system (translational symmetry), yet σ would change significantly. This illustrates that a description (more precisely a basis set) that incorporates (or accounts for) the underlying lattice symmetries is strongly preferred.

The Ising model is a very simple example for how this can be achieved. The energy for this case is written as

$$E_{\text{Ising}} = \frac{1}{N} \sum_{ij} J \sigma_i \sigma_j, \tag{1}$$

where the double summation runs over all N sites in the structure subject to the restriction that i and j are *nearest neighbor* pairs. J denotes the interaction strength between species A (i.e., $\sigma_i = -1$ and B (i.e., $\sigma_i = +1$). The Ising model is probably more familiar as the simplest microscopic model for a magnetic system, but it also serves as an (overly) simplistic model for alloys.

In contrast to spin-spin interactions, chemical interactions are typically more long-ranged and exhibit stronger many-body character. To describe the interaction we therefore need to consider not only nearest-neighbor pairs of atoms but also more distant pairs as well as groups involving three or four atoms. The formulation of this insight leads to alloy *CE*s. More rigorously, one decomposes a configuration into "clusters", where a cluster of order k is defined as a list of k sites. Clusters are commonly referred to by order as singlets, pairs, triplets, and so on (Fig. 2).

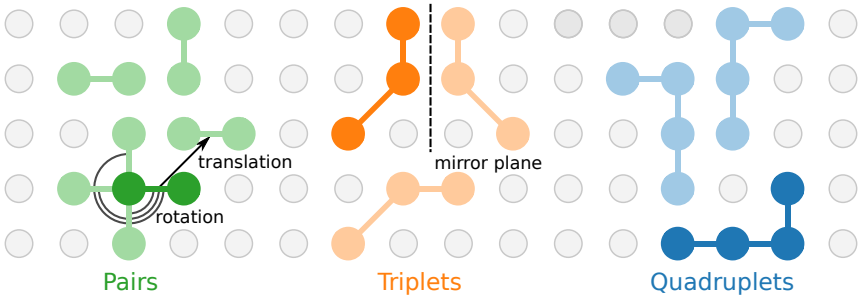


Fig. 2: Examples for pair (green), triplet (orange), and quadruplet (blue) clusters on a square lattice. The representative (symmetry inequivalent) clusters are shown in dark colors, whereas examples for other (symmetry equivalent) clusters in the orbits are shaded.

Extending Eq. (1) we can thus write

$$\begin{aligned} E_{\text{generalized Ising}} = & \underbrace{\frac{1}{N} \sum_{ij}^{\text{1nn pairs}} J_{\text{pair-1}} \sigma_i \sigma_j}_{E_{\text{pair-1}}} + \underbrace{\frac{1}{N} \sum_{ij}^{\text{2nn pairs}} J_{\text{pair-2}} \sigma_i \sigma_j}_{E_{\text{pair-2}}} \\ & + \underbrace{\frac{1}{N} \sum_{ijk}^{\text{1nn triplet}} J_{\text{triplet-1}} \sigma_i \sigma_j \sigma_k}_{E_{\text{triplet-1}}} + \ldots \end{aligned} \tag{2}$$

This goes to show that a particular configuration can be characterized by the average number of times different symmetry equivalent clusters occur in a structure. This is illustrated for a very simple case in Fig. 3. There are for example 7 nearest neighbor A-A pairs, 13 B-B pairs, and 30 A-B pairs. The first term in Eq. (2) then becomes

$$E_{\text{pair-1}} = J_{\text{pair-1}} \left[7 \cdot (-1)^2 + 13 \cdot (+1)^2 + 30 \cdot (-1 \cdot +1) \right] / 50 = -10/50 = -0.2 J_{\text{pair-1}},$$

where we have already included the normalization by the number of clusters. Similarly for the second nearest neighbor

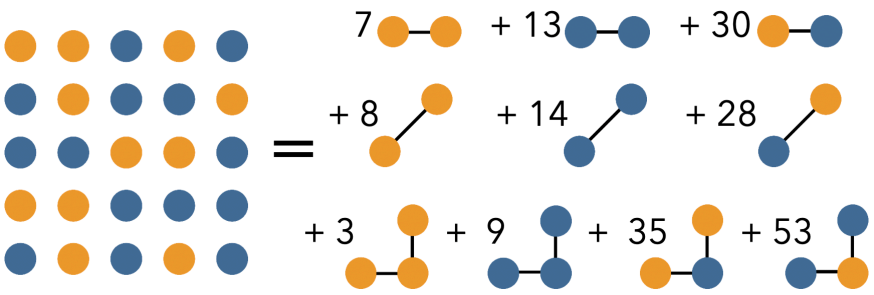


Fig. 3: Example for counting the average number of clusters for a binary alloy on a simple square lattice.

term and the triplet term we obtain

$$\begin{aligned} E_{\text{pair-2}} &= J_{\text{pair-2}} \left[8 \cdot (-1)^2 + 14 \cdot (+1)^2 + 28 \cdot (-1 \cdot +1)\right] / 50 \\ &= -6/50 = -0.12 J_{\text{pair-2}} \\ E_{\text{triplet-1}} &= J_{\text{triplet-1}} \left[3 \cdot (-1)^3 + 9 \cdot (+1)^3 + 35 \cdot (-1^2 \cdot +1) + 53 \cdot (-1 \cdot +1^2)\right] / 100 \\ &= -12/100 = -0.12 J_{\text{triplet-1}} \end{aligned}$$

Such that

$$E_{\text{generalized Ising}} = -0.2 J_{\text{pair-1}} - 0.12 J_{\text{pair-2}} - 0.12 J_{\text{triplet-1}}.$$

CE formalism and formulation as a linear problem

More formally in a *CE* the energy (or other global properties) are written as

$$E_{\text{CE}} = J_0 + \sum_\alpha m_\alpha J_\alpha \Pi_\alpha(\sigma),$$

where the summation runs over orbits α (or symmetry inequivalent clusters; see Fig. 2), m_α denotes the multiplicity of the orbit (i.e., the number of clusters that are symmetry equivalent to α per unit cell), J_α are the effective cluster interactions (ECIs; i.e., generalizations of the J parameters in Eq. (2)), and $\Pi_\alpha(\sigma)$ are symmetry averaged products over σ.

Note that the first term in this summation only involves "products" over individual sites. This "singlet" term corresponds to a measure of the concentration. One often also subsumes also the constant pre-factor J_0 into the summation; this term is then typically referred to as the "zerolet".

A particular configuration is thus mathematically described by a "cluster vector" ξ, the elements of which represent the average number of times a particular cluster occurs in a structure. The energy (or other property) is then simply obtained as the scalar product of the cluster vector ξ with the parameter vector J.

The structure in Fig. 3 yields for example the cluster vector

$$\xi^T = [\underbrace{1}_{\text{zerolet}}, \underbrace{0.12}_{\text{singlet}}, \underbrace{-0.2}_{\text{pair-1}}, \underbrace{-0.12}_{\text{pair-2}}, \underbrace{-0.12}_{\text{triplet-1}}].$$

We can repeat this kind of decomposition for a set of structures, for which we have access to reference data for the target property of interest (e.g., the energy). We can stack the cluster vectors to obtain a matrix $\mathbf{X} = \left[\xi_1^T, \xi_2^T, ...\right]$, each row of which corresponds to one structure, and similarly combine the target property values into a vector $E = [E_1, E_2 ...]^T$. This allows us to write

$$E = \mathbf{X} J,$$

from which it is apparent that extracting the unknown effective cluster interactions J (and thereby parametrizing a *CE* model) is simply a linear problem.

Try it out yourself!

The notebook `introduction-to-cluster-expansions.ipynb` provides a simple introduction to alloy cluster expansions.

6.1.3 Correlation of cluster vectors

From the discussion of the cluster vector above, it is apparent that each element of the cluster vector can *individually* vary between -1 and $+1$. There is, however, correlation built in the underlying structures that implies that not all elements can span this range *independently* of each other. For example, the number of nearest-neighbor A-A pairs is correlated to the number of nearest-neighbor A-A-A and A-A-B triplets. This is illustrated in Fig. 4 for several orbits in a binary alloy on a face-centered cubic lattice. If the orbits were completely uncoupled, the scatter plots on the off-diagonal of this corner plot, would evenly cover the entire square between -1 and $+1$. Moreover, for the singlet, one can generate the extremal values already in 1-atom unit cells. For the first pair, however, while one readily achieve a value of $+1$ in 2-atom cell, in cells with up to 10 atoms one can only reach a negative value of slightly below -0.25 as the value of -1 is only realized in the infinite size limit. The latter case corresponds in fact to an interface between two semi-infinite slabs of A and B, respectively. This illustrates that while the cluster basis set is formally orthogonal, correlations intrinsic to structures along with the finite size dictated by practical considerations, prevents us from sampling the cluster vector space evenly and completely. In practice this implies for example that even in cases that appear over-determined the sampling matrix will in general not provide an even sampling due to hidden correlations between the columns. This situation also occurs in other cases and one should bear this in mind when interpreting the results of regression techniques in terms of being over or under-determined.

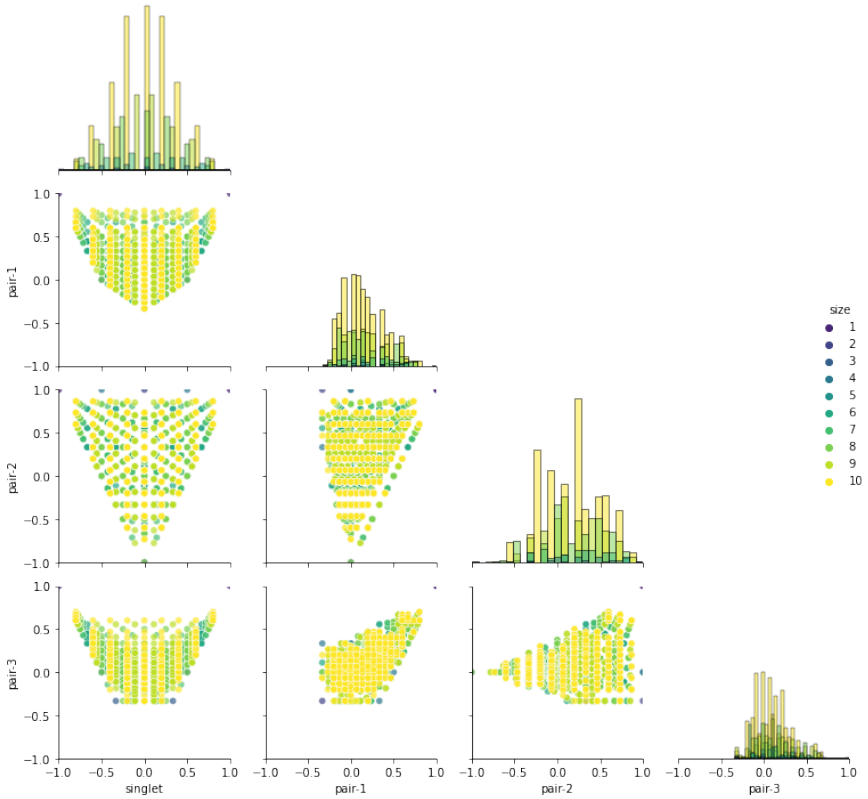


Fig. 4: Correlation between the singlet and the first five pair orbits in a binary alloy on a face-centered cubic lattice.

Try it out yourself!

The figure above was produced using the notebook `correlation-in-cluster-expansions.ipynb`. You can use the notebook to inspect correlations yourself and test other crystal structures.

6.1.4 Incorporating physical insight

One key reason for discussing *CE* models in the present context is that they allow us to demonstrate how we can incorporate physical insight into a model via a Bayesian approach. (The following discussion is based on [30], specifically Sect. III.)

Insight 1: Physical intuition (and experience) tell us that interactions decay with distance and with the number of bodies involved. In the case of *CE*s, we expect the magnitude of the *ECI*s to decay with the radius and the order of a cluster. This "knowledge" can be expressed via the prior for the *ECI*s. Specifically, assuming a normal prior (as commonly done), the variance σ_α^2 associated with prior for the *ECI* for cluster α should decrease with radius and order. $\sigma_\alpha^2 \to 0$ implies that the respective cluster is effectively excluded from the *CE*. Formally, we can choose the prior as

$$p(J|\mathbf{X}) \sim \prod_\alpha \exp\left(-\frac{J_\alpha^2}{2\sigma_\alpha^2}\right),$$

where σ_α^2 should decrease with cluster radius and order.

Insight 2: Intuitively we furthermore expect that the *ECI*s of clusters with a similar shape or topology should be similar. If we consider for example a surface (compare the demonstration notebook referenced below), a nearest-neighbor pair three layers below the surface is symmetrically distinct from a nearest-neighbor pair four layers below the surface. As a result there are two distinct *ECI*s associated with these two clusters. Yet, we intuitively expect these two *ECI*s be numerically rather similar since the respective surrounding of these two clusters are very similar. Specifically, both of them are surrounded by other sites. By contrast, we would not expect a equally strong similarity between the *ECI*s for a nearest-neighbor pair three layers below the surface and a nearest-neighbor pair directly at the surface. This belief can be expressed through a prior such as

$$p(J|\mathbf{X}) \sim \prod_{\alpha,\beta\neq\alpha} \exp\left(-\frac{\left(J_\alpha - J_\beta\right)^2}{2\sigma_{\alpha\beta}^2}\right),$$

where $\sigma_{\alpha\beta}$ quantifies the degree of expected similarity. $\sigma_{\alpha\beta} = 0$ implies no similarity while $\sigma_{\alpha\beta} \to \infty$ enforces $J_\alpha = J_\beta$.

We can combine these priors to obtain a maximum likelihood estimate for J in the known form (see Sect. 2.2),[2]

$$J_{\text{opt}} = \left(\mathbf{X}^T\mathbf{X} + \Lambda\right)^{-1}\mathbf{X}^T E,$$

where $\Lambda = \Gamma_0^{-1}$ is the inverse covariance matrix (see, e.g., Eq. (2)) given by

$$\Lambda_{\alpha,\alpha} = \frac{\sigma^2}{\sigma_\alpha^2} + \sum_{\beta|\beta\neq\alpha} \frac{\sigma^2}{\sigma_{\alpha\beta}}$$

$$\Lambda_{\alpha\beta} = \Lambda_{\beta\alpha} = -\frac{\sigma^2}{\sigma_{\alpha\beta}^2}.$$

It is useful to consider the extremal cases for the elements of Λ, using the reduced values $\lambda_\alpha = \sigma^2/\sigma_\alpha^2$ and $\lambda_{\alpha\beta} = \sigma^2/\sigma_{\alpha\beta}^2$.

1. $\lambda_\alpha \to \infty$: force *ECI* for orbit α to be zero (i.e., remove the orbit from the *CE*)
2. $\lambda_\alpha \to 0$ and $\lambda_{\alpha\beta} \to 0$: all *ECI* values are equally likely; this recovers *OLS*
3. $\lambda_{\alpha\beta} \to 0$: no correlation (coupling) between *ECI*s; this recovers ridge regression if λ_α is the same for all orbits
4. $\lambda_{\alpha\beta} \to \infty$: force two orbits to have the same *ECI*

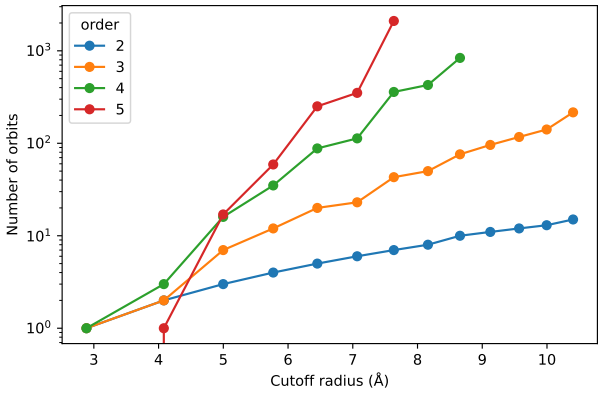


Fig. 5: Number of orbits with cutoff radius for different cluster orders (2=pair, 3=triplets, etc) for a face-centered cubic lattice with a lattice constant of 4.1 Å.

There are different strategies to choose σ_α and $\sigma_{\alpha\beta}$. Generally, σ_α should go to zero for $r \to \infty$ and $n \to \infty$. Also σ_α should decrease faster than the increase in the number of orbits with increasing cutoff radii. Figure 5 shows that the number of orbits n_{orbits} increases approximately exponentially with cutoff radius r_{cut} and order n, i.e., $n_{\text{orbits}} \sim r_{\text{cut}}^{\gamma n}$, where γ is some structure specific parameter. A suitable way to encode the "Insight 1" can therefore be to set [30]

$$\lambda_\alpha(n, r) = (\gamma_1 r)^{-\gamma_2 n}$$

or with more refinement

$$\lambda_\alpha(n, r) = \gamma_1 \left(\gamma_2 r + \gamma_3 + 1\right)^{\gamma_4 n + \gamma_5},$$

where n and r are the number of sites and the radius of orbit α, respectively. λ_α grows with n and r, decreasing the size of the respective *ECI*. Rather than having to find suitable λ_α values for each J_α separately, now one only has to optimize the hyperparameters γ_i. One can easily devise other such schemes [30].

With regard to $\sigma_{\alpha\beta}$, there is even more flexibility. If we stay with the example of a surface discussed above, one can, e.g., implement a scheme that couples the *ECI*s of orbits in the middle of the slab to a *CE* for the bulk system with the same components. Another approach is to introduce a similarity measure that takes into account the neighborhood of the orbits. In our example notebook (see below), we make another even simpler choice. We first identify which orbits we deem to be connected and then use a single $\sigma_{\alpha\beta}$ value for all of these combinations. This parameter is tuned from a very small to a very large value, which demonstrates how *OLS* and exact identity are recovered in the limits, while the lowest *RMSE* score is actually obtained for an intermediate value of $\lambda_{\alpha\beta}$.

Try it out yourself!

The above discussion illustrates the large number of possibilities that the covariance approach provides for incorporating physical insight in model construction. The notebook `ce-with-covariance-matrix.ipynb` illustrates the coupling of *ECI*s for a simple surface. You can use the notebook as a starting point for a deeper exploration of the covariance approach. (The scheme for choosing σ_α described above is a part of Project 2a.)

---

[2] The expression in [30] include a weight matrix $\mathbf{W}$, which can be used to weigh the input structures differently. We have omitted this complication here for simplicity.

6.2 Phonons and force constants

Key take-away messages (section-level)

- Force constant (*FC*) expansions model the interatomic forces of crystalline materials (Sect. 6.2.1).
- The problem of parametrizing these models can be cast in a linear form and is therefore very suitable for linear regression techniques (Sect. 6.2.3).
- The comparison of *FC* expansions and *CE*s showcases different types of hidden correlation and challenges in constructing useful reference data sets (Sect. 6.2.5).).

Try it out yourself!

You can find jupyter notebooks that demonstrate force constant expansions in the context of regression techniques in the `force-constant-expansions` and `sensitivity-analysis` directories of the `tif345` repository.

The vibrational properties of solids are pivotal for a large number of physical phenomena, including phase stability and thermal conduction [31]. In crystalline solids the vibrational motion of the atoms is periodic and commonly described using phonons – quasi-particles that represent collective excitations of the lattice. At the first level of approximation, phonons can be obtained within the harmonic limit, which implies non-interacting quasi-particles with infinite lifetimes. This approach, along with its so-called quasi-harmonic extension, already provides a wealth of information. There are, however, countless examples where anharmonic effects are crucial and must be accounted for in order to capture the correct physical behavior of a system. Notable examples include the lattice contribution to the thermal conductivity or the vibrational stabilization of metastable phases, e.g., the body-centered cubic phase of the group IV elements (Ti, Zr, Hf) or the cubic phase of zirconia. In more general terms, phonon-phonon coupling must be taken into account, which leads to finite phonon lifetimes and temperature-dependent frequencies. Phonons and their role in understanding condensed matter and materials are a standard topic in condensed matter theory and solid state physics. You can consult any of the many text books in this field for a review of this important topic, see e.g., Refs. [32, 33].

6.2.1 Formalism

The analysis of vibrational spectra requires a set of *FC*s, which allows the computation of atomic forces solely based on the displacements of atoms from their reference positions. The harmonic approximation requires only knowledge of the second-order *FC*s and allows one to compute the harmonic phonon spectrum. Third-order *FC*s are required, e.g., for computing the thermal conductivity to the lowest permissible order of permutation theory. *FC*s beyond third-order are required to describe, e.g., metastable systems or the temperature dependence of phonon modes.

Formally *FC*s can be introduced as follows. The potential energy V of a solid can be represented by a Taylor expansion of the potential energy surface (*PES*) in ionic displacements $u = R - R_0$ away from the equilibrium positions R_0

$$V = V_0 + \Phi_i^\alpha u_i^\alpha + \frac{1}{2}\Phi_{ij}^{\alpha\beta} u_i^\alpha u_j^\beta + \frac{1}{3!}\Phi_{ijk}^{\alpha\beta\gamma} u_i^\alpha u_j^\beta u_k^\gamma + \dots ,$$

where the Einstein summation convention applies and

$$\Phi_i^\alpha = \frac{\partial V}{\partial u_i^\alpha}, \quad \Phi_{ij}^{\alpha\beta} = \frac{\partial^2 V}{\partial u_i^\alpha \partial u_j^\beta} \quad \text{etc.}$$

Φ_i, Φ_{ij}, … are the *FC*s corresponding to increasing orders of the expansion. Latin indices i indicate the atomic labels, where the summation is over an infinite crystal lattice, while Greek indices α run over the Cartesian coordinates x, y, z. V_0 is a constant term, which is commonly ignored when dealing with lattice dynamics. The first order *FC* is also often dropped since the expansion in displacements can be made around an equilibrium lattice configuration with vanishing

forces. These two terms are important for some applications but here are considered zero. Truncating the potential after the second-order term results in the conventional harmonic phonon theory, which is analytically solvable and widely used [34, 35].

The forces can be written in terms of the *FC*s as

$$F_i^\alpha = -\Phi_{ij}^{\alpha\beta} u_j^\beta - \frac{1}{2}\Phi_{ijk}^{\alpha\beta\gamma} u_j^\beta u_k^\gamma - \ldots \tag{3}$$

Crucially, this expression that takes the functional form of an interatomic many-body potential is *linear* in the *FC*s. In other words, we can cast the problem of finding the coefficients $\Phi_{ij}^{\alpha\beta}$, $\Phi_{ijk}^{\alpha\beta\gamma}$ etc in the form of a linear equation, where the forces are the target values, the products of displacements form the fit matrix, and the *FC*s are the unknowns. Doing this directly is, however, ill-advised as the *FC* matrices and hence the number of coefficients is large and also unnecessary since there are many symmetries that couple the coefficients.

6.2.2 Using symmetry

The number of *DOF*s in the *FC* expansion scales exponentially with $\mathcal{O}(N^n)$, where N is the number of atomic sites of the supercell and n is the order after which the expansion is truncated. The number of *independent* parameters is, however, much smaller due to the symmetries of the underlying lattice as well as constraints due to the conservation of linear and angular momentum [36]. The number of *numerically relevant* parameters is smaller still due to the decay of the *FC*s with interaction distance, order, and finite many-body interactions.

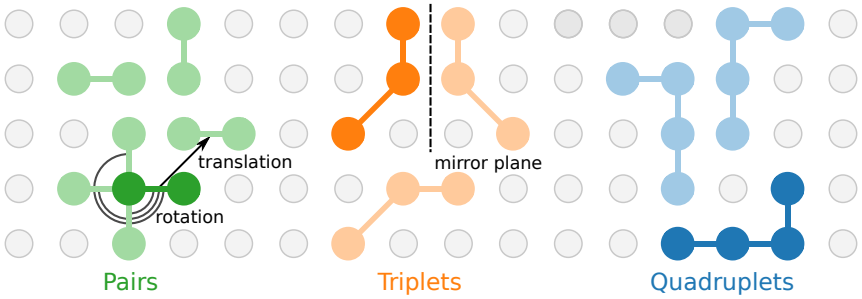


Fig. 6: Examples for one-body (singlet; red), two-body (pair; green), and three-body (triplet; blue) clusters. Dark colors indicate representative cluster whereas lighter colors represent symmetry equivalent clusters that belong to the associated orbit. While the two-body clusters shown here are related by a simple fourfold rotation, the three-body clusters can be mapped onto each by application of a fourfold rotation combined with a mirror operation.

To represent the *FC*s and their inherent symmetries it is convenient to consider clusters of sites $(ij\ldots)$, each of which can be assigned a size, taken for example as the largest distance between any two sites in the cluster. The *FC*s correspond to interactions between the sites forming a cluster; for example, the term Φ_{ijkk} describes a fourth-order interaction in the three-body cluster (ijk). Clusters can be categorized based on the number of sites they comprise and accordingly there are one-body (singlet), two-body (pair), and n-body (many-body) clusters (see Fig. 6). In general a cluster of order n is a multiset of order (or cardinality) n consisting of up to n different atoms. As a result of the locality of the interactions, one commonly truncates the *FC* expansion both in size, order, and cluster size. It is thus possible to create for example a non-central, short ranged, anharmonic pair potential or a long ranged harmonic pair potential with short ranged anharmonic many-body corrections.

The *FC*s obey a number of symmetries, the most fundamental of which stems from the requirement that the differentiation of the total energy and thus Φ must be invariant under a simultaneous permutation P of atomic and Cartesian indices. The *FC*s must further comply with the symmetry of the underlying lattice as expressed by the associated spacegroup. Application of these symmetries reduces the free parameters in the *FC* matrices.

6.2.3 Linear form

By exploiting symmetries as well as sum rules, one can construct a fit (or sensing) matrix $\mathbf{X}_k$ for each input structure k from the $3N$-dimensional displacement vector u. Each row of the matrix corresponds to one atom and one Cartesian direction, and relates the $3N_{\text{at}}$-dimensional vector of predicted forces f to the N_{p}-dimensional parameter vector w ,

$$f_k = \mathbf{X}_k w, \tag{4}$$

where N_{at} and N_{p} denote the number of atoms in the supercell and the number of parameters, respectively. The translational symmetry of periodic systems dictates that *FC*s must fulfill the so-called acoustic sum rules, which can be stated as

$$\sum_j \Phi_{ij}^{\alpha\beta} = 0 \,\forall\, j, \alpha, \beta.$$

This enforces that the three acoustic branches of the phonon dispersion are exactly zero at the Γ-point (i.e., in the long-wave length limit). If this condition is applied to the displacements products in Eq. (3), one unfortunately loses the direct connection between a particular element of the parameter vector w and a particular *FC* term. As a result, in contrat to, e.g., *CE*s (see Section 6.1.4), physical insight related to, e.g., the range of interactions cannot be equally easily incoporated.

When dealing with multiple structures the fit matrices and target vectors for each structure can simply be stacked on top of each other.

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \end{bmatrix} w.$$

Try it out yourself!

You can try out the construction of *FC* models via regression using the `introduction-to-force-constants.ipynb` notebook.

6.2.4 Correlations in the reference data

In the case of *CE*s (see Sect. 6.1.3), we have already observed that the sensing (or fit) matrix can exhibit correlations that impact the regression of the data. These correlations are the result of the underlying physical space and, in the case of *CE*s effectively enter during the translation from an occupation vector into a cluster vector. Similar effects are also at play in the *FC* expansions and possibly even more intricate.

Let us first consider the sensing matrix. A simple yet plausible approach to generating input structures is to draw *i.i.d.* atomic displacements from a normal distribution with a standard deviation on the order of 0.01 Å, which generates suitably small displacements for the harmonic approximation to be valid. As shown in Figure 7, while the distribution of the individual columns is relatively close to a normal distribution, there are pronounced correlations *between* the columns, similar to the case of *CE*s.

Correlations are also caused by the underlying physical interactions themselves. As a direct result of the interaction that we are trying to probe, the forces experienced by nearby atoms are going to correlated to some degree. This can be shown by computing the projection of the force vectors of atom pairs $f_i \cdot f_j$. This is demonstrated for a simple metal (face-centered cubic Cu) with small displacements in Fig. 8. In this case, stronger correlation is only apparent for nearest-neighbor pairs. This can be understood by considering the range of the *FC*s (Fig. 9), which shows that the interaction strength to decay very quickly with distance. This very short-ranged behavior is characteristic for a metal. By contrast, in ionic materials the *FC*s decay (potentially much) more slowly with distance, as a result of the Coulomb interaction and weak screening. In the latter case, one should thus also expect the interatomic forces to be correlated over longer distances.

Try it out yourself!

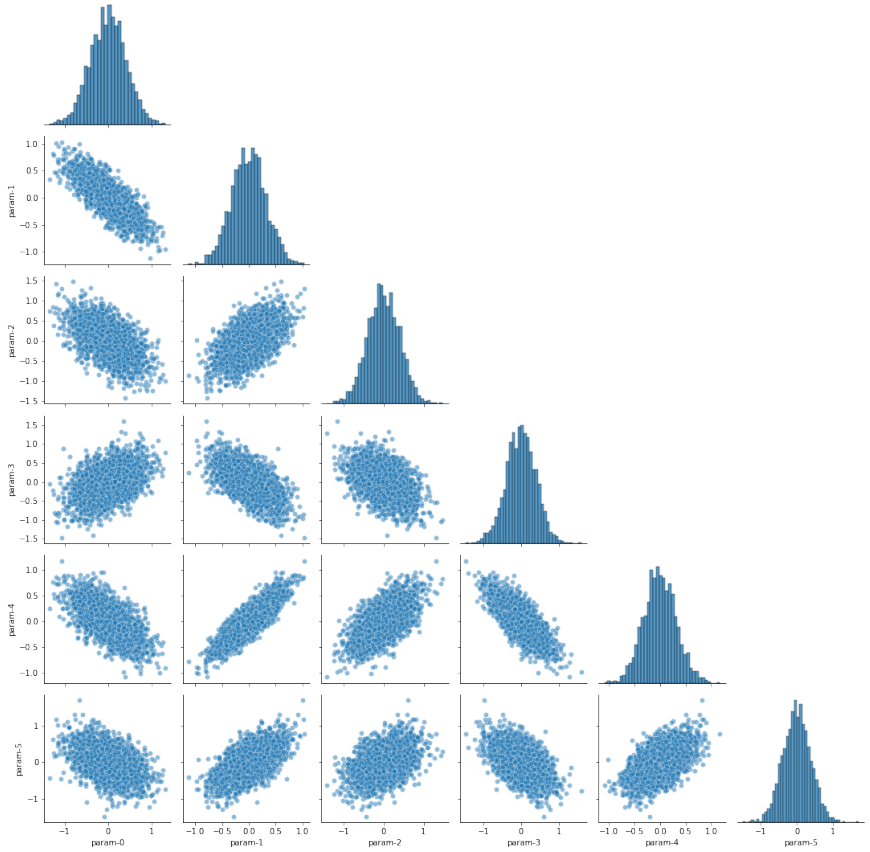


Fig. 7: Correlation between the columns of a sensing matrix for a second-order *FC* expansion, constructed from a structure with random displacements.

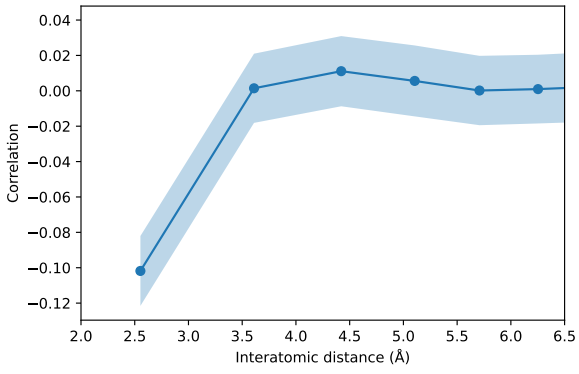


Fig. 8: Correlation between the force vectors of atoms as a function of their distance. The shaded region approximately indicates the error-of-mean.

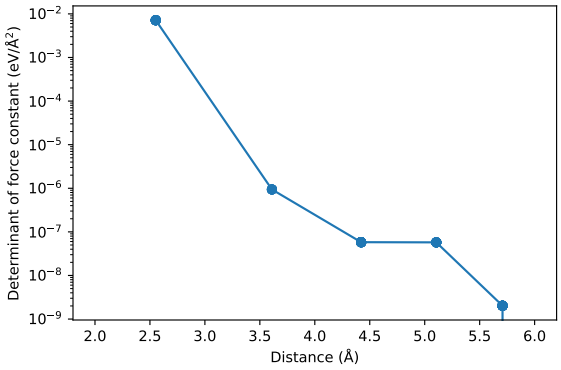


Fig. 9: Variation of the determinant of the second-order *FC* (sub)matrix as a function of pair distance.

You can explore correlation in *FC* models using the `correlation-in-fc-models.ipynb` notebook.

6.2.5 FC expansions vs CEs

It is instructive to compare *CE*s and *FC* expansions in terms of their training behavior.

In the case of *CE*s, one obtains one data point (namely energy) per structure and the original space (occupation vectors) is discrete. The distribution in representation space (elements of the cluster vector, columns of the sensing matrix) is rather feature-rich (see Fig. 4) and extremal values in the sensing matrix are often obtained with small cells (see Sect. 6.1.3), which are computationally comparably inexpensive. Sensing matrices for *CE* models are relatively small, as typical cluster spaces include between 10 and 100 orbits and thus parameters (*ECI*s), while as a result of computational cost and structure enumeration principles (not discussed here), there are usually only on the order of tens to a few hundred structures. In practice, one can therefore use almost any regression algorithm, including direct *MCMC* simulations, and model testing is computationally inexpensive.

In the case of *FC*s, one obtains $3N$ data points (force components) per structure and the original space (i.e., displacements) is continuous. In representation space (displacement products), columns of the sensing matrix are approximately normal distributed but the columns are rather strongly correlated with each other (Fig. 7). Moreover, the target data has built-in (albeit rather weak) correlations as a direct result of the interactions that are being probed (Fig. 8 and Fig. 9). Sensing matrices for *FC* models are much larger than in the case of *CE*s. There are far more parameters, especially in expansions for systems with low symmetry and for increasing expansion order.[1] Situations with hundreds and even several thousand parameters are not uncommon. At the same time, the generation of reference data is usually uncritical. As a result, in realistic situations, one often deals with very large matrices, for which the computational cost of the regression algorithms becomes an important factor [23]. Automated feature selection algorithms (see Sect. 5.6) can therefore become too costly and manual feature selection is a more viable approach.

This comparison illustrates that different models require different approaches and model characteristics are important to keep in mind when selecting a regression approach and analyzing the results.

[1] Here, we only consider harmonic (second order) *FC* expansions but in practice third and fourth-order models are rather common, and in special cases one can even go up to sixth or eight order.

6.3 Interatomic potentials

6.3.1 Empirical potentials

In order to appreciate the advantages (and disadvantages) of *ML* models, it is instructive to consider "classical" models that were (largely) derived based on physical insight. In the context of condensed matter and materials, this case is exemplified by interatomic potentials, i.e., *empirical* models that enable one to compute energy (and forces) for a collection of of atoms that is described by the set of atomic positions $\{r_i\}$ and atom types $\{\sigma_i\}$,

$$E = E[\{r_i\}; \{\sigma_i\}]$$

The simplest form is a pair potential

$$E = \sum_{ij} v_{\sigma_i \sigma_j}(r_{ij}),$$

where $r_{ij} = \|r_i - r_j\|$ is the interatomic distance. Since pair potentials are only suitable for the most simple systems (inert gases, such as Ar, Ne, Xe etc), various other forms have been developed that aim to represent different bonding types (covalent, ionic, metallic etc). In these cases additional terms are added that account for effects such as screening and directionality. In the present context, it is important that for constructing an empirical potential researchers adopt functional forms for the individual terms that are meant to capture the essential physics with as few parameters as possible and are computationally convenient. Consider, for example, a Lennard-Jones potential, one of the simplest pair potentials,

$$v(r)^{\text{LJ}} = 4\epsilon \left[\underbrace{\left(\frac{\sigma}{r}\right)^{12}}_{\text{repulsive branch}} - \underbrace{\left(\frac{\sigma}{r}\right)^{6}}_{\text{attractive branch}} \right].$$

The first term is a steeply rising function that represents repulsion whereas the second more slowly-varying term represents attraction. ϵ and σ control the depth and the position of the potential well, respectively. The important thing to note is how different parameters carry different units and contribute differently and in non-linear fashion.

While the $1/r^6$ dependence of the attractive branch in the Lennard-Jones potential can be motivated (derived is a bit strong) on physical grounds (interaction between a fluctuating and an induced dipole), the repulsive branch is largely chosen for computational convenience ($r^{-12} = (r^{-6})^2$). A physically more sound model is the Morse potential, which in variations is present in many empirical potentials. It can be written in the following form

$$v(r)^{\text{Morse}} = D_e \left(1 - \exp[-a(r - r_c)]\right)^2,$$

where D_e and a control potential depth and decay of the interactions, respectively, while r_c is an auxiliary parameter that sets the cutoff radius (the range). Again, the crucial point to note is that the two "physical" parameters (D_e, a) affect the interaction in rather different ways. This a common feature of empirical interatomic potentials and, if not carefully handled, can lead to strong parameter correlation. In *ML* this type of model are usually referred to as "sloppy models". The latter are not restricted to empirical potentials but also appear, e.g., in biological models. You can read more about sloppy models in Refs. [37, 38].

Try it out yourself!

As a result of their functional form, empirical potentials are tedious to sample using Bayesian techniques. This is demonstrated in the `sample-potential.ipynb` notebook.

CHAPTER

SEVEN

COMPRESSIVE SENSING

Compressive sensing (*CS*; or sampling) is a paradigm in the signal processing field that is concerned with the recovery of sparse signals. It goes against traditional knowledge in signal sensing in so far as it allows one to beat the Nyquist-Shannon theorem, which puts a firm lower limit on the amount of sampling that is required to recover high-frequency signals.

7.1 Connection to basis sets

In the previous lecture we alluded to the role of basis sets in physics and that finding sparse representations is one of the core aspects of developing physical models that are mathematically manageable. A good example for this behavior is the relationship between a signal in frequency domain that is recorded in the time domain. This example also happens to be one of the most common introductory examples in *CS* [39].

Consider a signal $f(t)$ in the time domain, e.g., a sound wave. Commonly such a signal would be dense along the time axis, i.e., if we represent the signal in vector form, where each element corresponds to a different time, almost all elements would be non-zero and continuous (smooth). Sticking with the example of a sound wave, we know, however, that in the frequency domain this signal would be composed of only a few frequency components. We can convert the signal from the time to the frequency domain using a Fourier transformation

$$F(\omega) = \int_0^\infty dt f(t) e^{i\omega t}$$

or in discrete form ($\rightarrow$ *FFT*)

$$F_{\omega_k} = \sum_n f_{t_i} e^{i\omega_k t_i}.$$

This is a basis set transformation and the essential task consists in finding out which coefficients F_{ω_k} are significant (i.e., carry information).

Slightly more generally we are concerned with obtaining information about a signal $f(t)$ by linear functionals that record the values

$$y_k = \langle f | \phi_k \rangle \,,$$

where $\{\phi_k(t)\}$ is a set of basis functions (wave forms is the a term often used in signal processing), e.g., plane waves (as in the example above) or wavelets. The crucial point is that the chosen basis set should enable a *sparse* representation of the signal.

To emphasize the sparse representation consider the *FFT* example in Fig. 1. In this case, the signal recovery is bound by the Nyquist-Shannon theorem, which states that if we measure a signal with a sampling rate f_s, the largest possible frequency we can resolve in said signal is $f_{\text{max}} = f_s/2$.

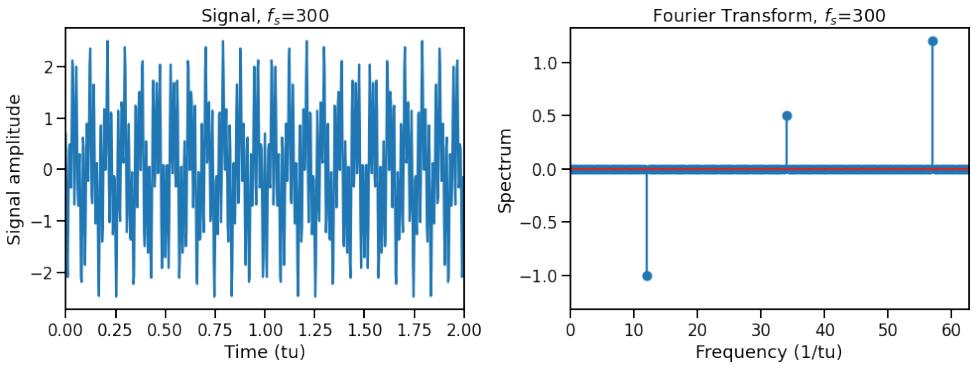


Fig. 1: Top left: A signal in the time domain generated by superimposing three cosines with frequencies of 12, 34 and 57. Top right: *FFT* of the signal on the left, showing the exact recovery of the input spectra. Bottom row: As the top but with normal noise.

7.2 Formulation as a linear problem

We now outline how the signal recovery can be formulated as a linear problem. (Here, we follow the notation from Ref. [39], which provides a good and concise summary of *CS*, in order to make it easier for you to follow that reference.) Let us denote the number of available measurements or samples by m and the dimension of the signal F in the sensing domain as n. The rows of the fit matrix $\mathbf{A}$ (usually referred to as "sensing matrix" in the context of *CS*) are given by the complex transpose of the basis functions, i.e.

$$\mathbf{A} = \begin{bmatrix} \vec{\phi}_1^* \\ \vec{\phi}_2^* \\ \vdots \\ \vec{\phi}_m^* \end{bmatrix}.$$

The recovery of the signal is then equivalent to extracting $\vec{F}$ from

$$\mathbf{A}\vec{F} = \vec{y}.$$

In general this problem is, however, vastly underdetermined since $m \ll n$.

According to Shannnon theory a uniform sampling (similar to *FFT*) allows us to recover the sigmal (bound by the Nyqvist-Shannon theorem). A crucial insight underpinning *CS* is that this limitation can be overcome by an *incoherent* sampling (under certain conditions that have a very high probability of being fulfilled).

7.3 Incoherent sampling

Assume that we are given a pair (Φ, Ψ) of two orthogonal bases in $\mathbb{R}^n$. Sensing is carried out in Φ (in which the signal is dense) while Ψ is used for representing the signal (in sparse form). The coherence of the two bases is defined as

$$\mu(\Phi, \Psi) = \sqrt{n} \cdot \max_{1 \leq k,j \leq n} |\langle \phi_k | \psi_j \rangle|.$$

In *CS* we commonly seek pairs for which the coherence is low as, e.g., in the example outline above where the sensing basis is composed of delta-functions along the time axis and the representation basis is comprised of plane waves.

In practice the signal is often recovered by ℓ_1-norm minimization (leading to *LASSO*), which in this context is often solved by linear programming, a technique that we also mentioned in passing in the context of robust regression. One can, however, also other approaches for solving sparse problems.

It is crucial that *CS* does not provide a guarantee that the signal recovery is successful. It is, however, possible to show that it will do so with very high probability. Let there be $\mathcal{S}$ non-zero coefficients in the representation (one says the basis Ψ is $\mathcal{S}$-sparse) and select m measurements in Φ uniformly *random*. Then if

$$m \geq C \cdot \mu^2(\Phi, \Psi) \cdot \mathcal{S} \cdot \log n,$$

where C is some positive constant, the solution to the minimization problem will be correct with *overwhelming probability*.

The last equation clarifies the role of coherence. The smaller the coherence between Φ and Ψ the fewer samples are needed. Moreover, if $\mu^2(\Phi, \Psi)$ is on the order of one, we see that on the order of $\mathcal{S} \cdot \log n$ samples are needed to achieve recovery. There is a lot more mathematics to this problem to those who are interested and the review by Candes and Wakins provides a good starting point and references.

7.4 Demonstration

To demonstrate that *CS* actually works we consider the example that was already introduced above. These results are summarized in the following figure panels, which were generated using the jupyter notebook `compressive-sensing.ipynb`.

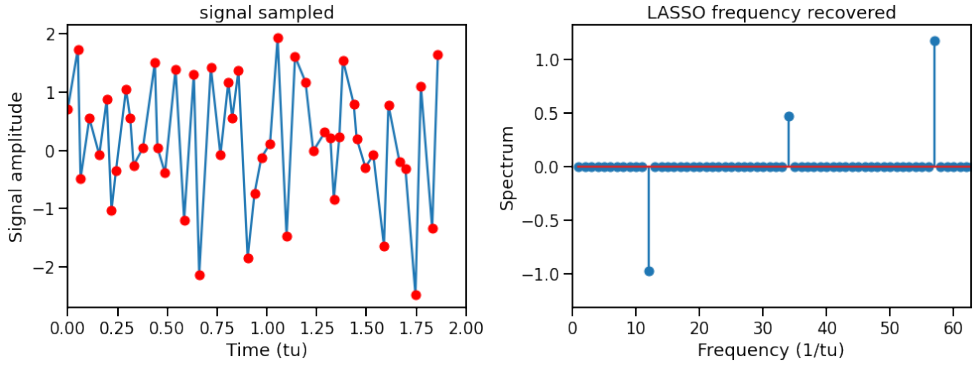


Fig. 2: Signal recovery based on 50 samples spaced at random intervals but at least a distance that complies with a maximum sampling frequency of 80 using *LASSO*.

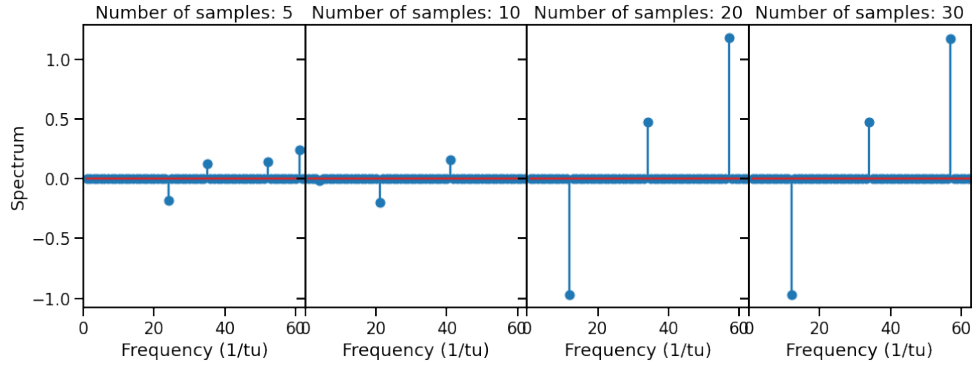


Fig. 3: Signal recovery based on a varying number of samples drawn completely at random from the available time interval using *LASSO*. Already about 20 samples are sufficient in this case to recover the frequencies. About 30 are needed to recover the amplitudes as well.

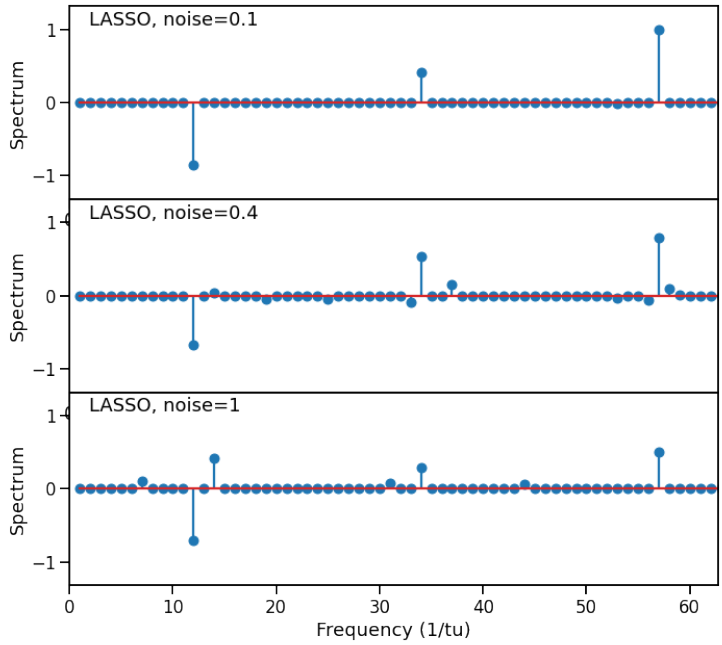


Fig. 4: Signal recovery with *LASSO* in the presence of noise.

7.5 From linear toward global optimization

An interesting application of *CS* (or sparse models) in physics is for basis set reduction. This is illustrated by the following example.

The dielectric function of a material describes its frequency dependent dielectric response. It can be measured experimentally or obtained from electronic structure calculations. For electrodynamic simulations using, e.g., the *FDTD* approach, it can be convenient (depending on implementation) to represent the dielectric function by a series of Lorentzian oscillators, i.e., the dielectric function is written in the form

$$\begin{aligned}\epsilon(\omega) &= \epsilon_\infty + \sum_{n=1}^{N} \frac{x_n \omega_n^2}{\omega_n^2 - \omega^2 - i\omega\gamma_n} = \epsilon_\infty + \sum_{n=1}^{N} x_n L_n(\omega) \\ &= \epsilon_\infty + \mathbf{x} \cdot \Re\mathbf{L}(\text{?}) + i\mathbf{x} \cdot \Im\mathbf{L}(\text{?}),\end{aligned}$$

where ϵ_∞ is the dielectric function of the vacuum. The parameters x_n, ω_n and γ_n represent strength, resonance frequency, and broadening of oscillator n. To represent a particular material we are thus faced with the task of finding these parameters. Due to the non-linear form of this is, however, a non-trivial task. For simpler cases one can adopt a manual fitting approach, choosing a suitable starting point and then minimizing using a least-squares optimization method such as Levenberg-Marquardt (LM) or Broyden-Fletcher-Goldfarb-Shanno (BFGS). This approach is, however, difficult to automatize and impractical if there are many cases to handle.

One approach to circumvent this problem is to rewrite it as a linear form, which by separating the real and imaginary part leads to

$$\begin{cases} \Re\epsilon_r - \epsilon_\infty &= \mathbf{x} \cdot \Re\mathbf{L}(\text{?}) \\ \Im\epsilon_r &= \mathbf{x} \cdot \Im\mathbf{L}(\text{?}) \end{cases}.$$

We then set up a two-dimensional grid of Lorentzians, $L_n(\omega)$, with different resonance frequencies ω_n and broadening values γ_n, it is, however, straightforward to solve this problem. The grid represents a basis set of Lorentzians and we seek to find the combination of these functions that are best suited to represent the actual dielectric function. Since the

raw data is discretized with respect to frequency the problem can thus be seen as a standard matrix equation, $\mathbf{A}\vec{f} = \vec{y}$. Here, $\mathbf{y}$ is a vector containing the real and imaginary parts of the dielectric function stacked on top of each other. The functions $L_n(\omega)$ are stored as rows in $\mathbf{A}$, where each row corresponds to one of the sampled frequencies. Note here that the upper/lower half of the matrix correspond to the real/imaginary part of the functions $L_n(\omega)$.

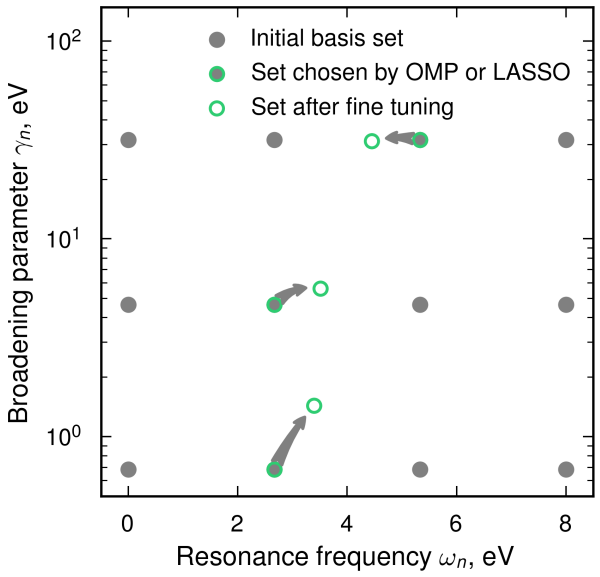


Fig. 5: Overview of fitting method. Gray circles denote the initial basis set of Lorentzian functions. Green circles on top of gray circles represent the functions selected by *OMP* or *LASSO*. The gray arrows indicate the fine tuning of the parameters γ_n and ω_n, via a non-linear least squares method. The final Lorentzians are the hollow green circles.

We are thus faced with the task of solving an underdetermined problem where the number of samples is smaller than the number of variables. For reference, the number of sampled frequencies is roughly 100 while the number of Lorentzians is on the order of 10,000. Two different algorithms are employed here, adaptive *LASSO* and *OMP*. The latter, which was mentioned in passing before, finds solutions to problems of the type:

$$\mathbf{w}_{\text{opt}} = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \quad \text{subject to} \quad \|\mathbf{w}\|_0 \leq N_{\text{req}},$$

where N_{req} is the hyperparameter that specifies how sparse the solution should be [40]. The intuitiveness of this hyperparameter is here an advantage of *OMP* compared to other regularization techniques.

The sparse solution $\mathbf{x}_{\text{opt}}$ yields the optimal strengths for the defined grid of oscillators. Further refinement can then be achieved using a standard local optimizers such as the LM and BFGS techniques mentioned above. It should be noted though that the basis set of Lorentzians is not orthogonal and this is thus not a rigorous application of *CS*. Nonetheless, in the present case, the approach works very well.

From the perspective of computational cost, it is advantageous to force the number of oscillators to be low (the computational cost of the *FDTD* simulations is linear in the number of oscillators). However, from a physical perspective this can be contraindicated since valuable information is lost. To deal with this issue the *BIC* and the *AIC* can be used [41]

$$\text{BIC} = N_{\text{s}} \ln(\text{MSE}) + N_{\text{p}} \ln(N_{\text{s}}) + c$$
$$\text{AIC} = N_{\text{s}} \ln(\text{MSE}) + 2N_{\text{p}} + c$$

Here, N_{s} is the number of samples, i.e., the number of frequencies in the problem at hand, N_{p} is the number of parameters, and MSE is the mean-square error of the fit. The value c is a constant for a data set and a fitting model, i.e., the quantities can only be compared relative to each other for the same data set and fitting model.

In the *BIC* and *AIC* values are shown for the *OMP* and adaptive-*LASSO* fitting methods. The minimum *BIC* values correspond to a reasonable trade off between error and number of oscillators.

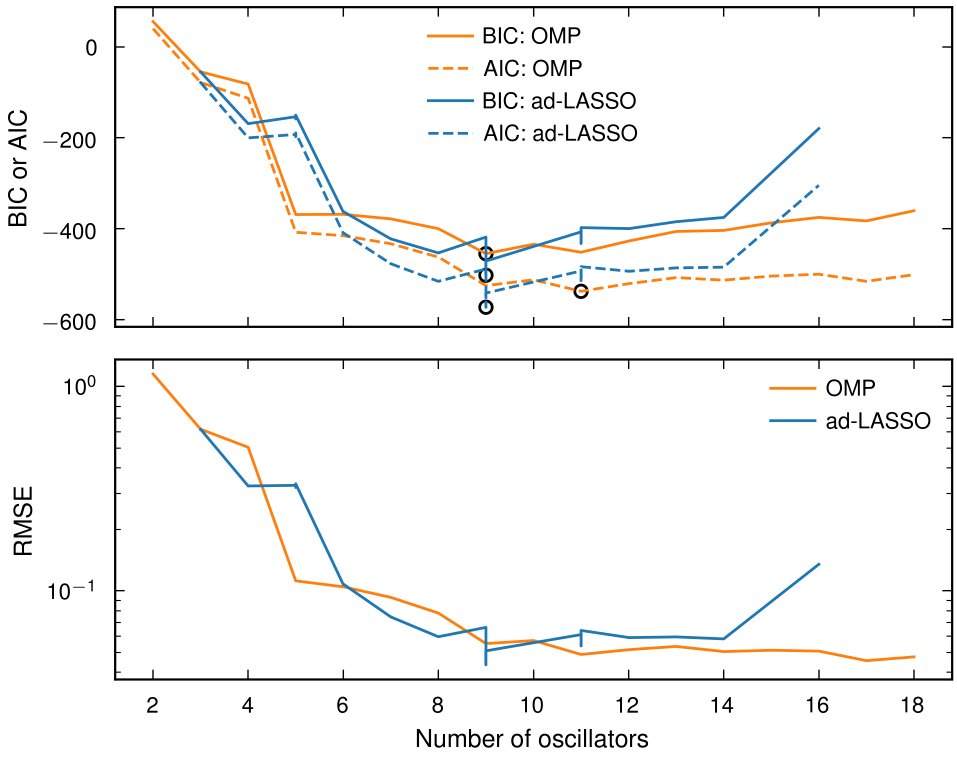


Fig. 6: Proof of concept of the fitting methods in combination with information criteria. In the upper figure *BIC* and *AIC* are shown as a function of the number of oscillators. Minima are marked with black circles. Both *OMP* and adaptive-*LASSO* show good results.

CHAPTER

EIGHT

SENSITIVITY ANALYSIS

Key take-away messages (chapter-level)

- Models are often used to predict properties that are *derived* from the immediate model predictions via other techniques.
- One can estimate the reliability of these *derived* predictions via sampling.
- Typically one must strike a compromise dictated by the computational cost associated with the model.

So far we have considered models as a means to generate data that is available from experiments or from another (physical) model at a (hopefully much) reduced computational cost. In these cases, we usually were interested in a property that is available from both the reference and the model. For example, in the case of the models discussed in Chapter 6 we can obtain the energy or the forces of a structure using both the respective models and, e.g., *DFT* calculations. In practice, however, the property predicted by a model might not be of immediate interest but only needed as input for another method or procedure. Energies and forces (and equivalent properties predicted by other models) have very limited immediate use for interpreting physical phenomena. More commonly we could be interested in dynamic behavior, thermodynamic averages or other *derived* properties. For example in the case of an alloy (see Sect. 6.1), we could be interested in its phase diagram (as opposed to bare energies), in the case of a vibrating crystalline material (see Sect. 6.2), we might want to obtain the phonon dispersion (as opposed to the bare forces), and in the case of a liquid (see Sect. 6.3), the property of interest could be the melting point, the pair correlation function or the dynamic structure factor (as opposed to the bare energy and forces of a single structural snapshot).

Discussion

Can you think of examples from other fields of physics?

In the cases outlined above we commonly cannot access the property of interest *directly* with the underlying reference, since the cost is exorbitant. This is of course the very point of making a model. It does leave us however with the obligation to query the reliability of a model prediction for the derived property. This question is at the heart of sensitivity analysis.

There are various approaches to sensitivity analysis and we have in fact already encountered several cases. For example, *CV* and is a common way to measure the sensitivity of model predictions to variations in the input data. *LOOCV* is particular convenient since it does not require repeated sampling. Its use is, however, limited to linear models. Gaussian processes (*GP*s) are another example of a technique that provides us not only with a prediction of the most likely value but also the reliability of this prediction.

Even in the latter cases it is, however, *a priori* unknown how variations in the immediate predictions of the model translate to derived properties, regardless of whether they are caused by errors in the input data, the selection of the input data,limitations of the form of the model etc.

In general, this is a difficult but crucial question and there is no single solution. In this chapter, we will discuss some examples based on the models introduced in Chapter 6.

8.1 Phonons

A simple example for a derived property that we have already partly encountered is the phonon dispersion. While a *FC* model predicts forces, these are *per se* not useful observables. They enable one, however, to compute the phonon dispersion both in the zero-Kelvin limit using only the second-order *FC*s and at finite temperature taken into phonon renormalization through higher-order terms. In this case, it can be very instructive to explore the sensitivity of the phonon dispersion to the model.

The notebook `bayesian-phonon-dispersion.ipynb` demonstrates both a Bayesian and a frequentist ("bagging") approach to this problem. In both approaches, a series of models is constructed, for each of which one then computes the phonon dispersion. In the Bayesian case, the an *MCMC* trajectory is generated in the usual way using a uniform prior for the parameters. The resulting phonon dispersion (Fig. 1) shows that most modes are insensitive to the sampled variations in the model, with the exception of the longitudinal acoustic modes near the X-point and the transverse acoustic modes along the Γ-K direction near the Γ-point. This indicates that these modes are not well represented by the training data, and hence we obtain an indication as to how to improve our training set, similar as in the case of a *GP*.

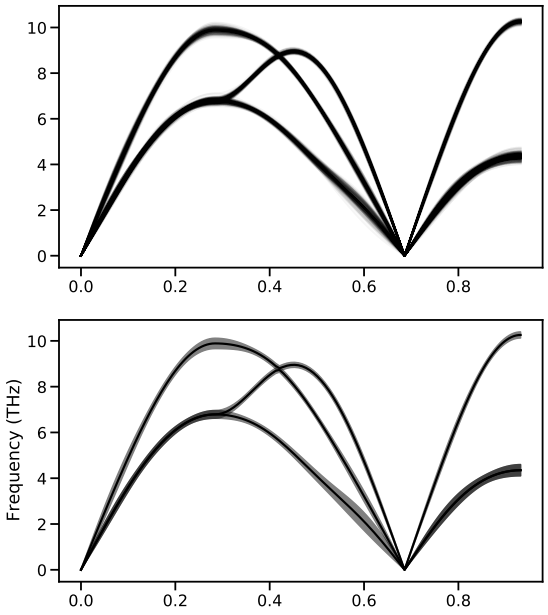


Fig. 1: Phonon dispersion for face-centered cubic Ni obtained via Bayesian regression.

In the frequentist approach, we do not vary the model parameters directly but rather sample over different training sets, which are generated by selecting from the available reference structures using the K-fold cross-validation procedure (Section 5.5.1). The resulting phonon dispersion is almost indistinguishable from the Bayesian case, providing further support for the previous interpretation.

Try it out yourself!

You can use the notebook `bayesian-phonon-dispersion.ipynb` to explore both the Bayesian and the bagging approach for sensitivity analysis.

8.2 Lattice thermal conductivity

We encountered the lattice thermal conductivity as a derived property already in Section 5.6.2. Previously we considered how *CV*-*RMSE* and lattice thermal conductivity vary with the size of the training set (Fig. 6), which already clearly demonstrated that the *CV*-*RMSE* is a poor predictor for the ability of a model to predict this important *derived* quantity. This aspect is even more clearly illustrated when considering model performance as a function of model sparsity (Fig. 6). In this case, the training set is kept fixed but we vary the hyperparameter of the respective algorithms and thereby the number of features. In the case of *ARD*, *RFE*, and to slightly lesser degree adaptive *LASSO*, the minimum in the *CV*-*RMSE* score coincides approximately with the best prediction of the lattice thermal conductivity. Interestingly reducing the number of features even further, that is enforcing increasingly sparser models, leads to a modest reduction in the ability of the models to correctly predict the thermal conductivity. This illustrates the stability of these models to variations in the hyperparameter.

For *LASSO* on the other hand, there is large mismatch between the minimum in the *CV*-*RMSE* score and the best prediction of the thermal conductivity, which as in the case considered before (Fig. 6) is still a notable underestimation of the reference value. Moreover, *LASSO* does not exhibit the type of behavior with respect to feature reduction that is observed for the other cases.

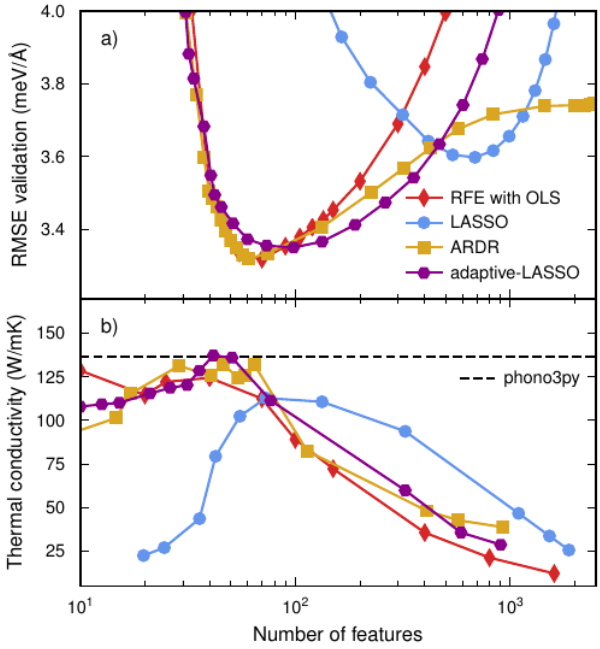


Fig. 2: Extraction of *FC*s by regression with automated feature selection for the same example as in , extending the analysis shown in . By tuning the hyperparameters for each method, we affect both (a) *CV*-*RMSE* and (b) sparsity of the solution. This comparison clearly reveals the ability of different algorithms to find sparse solutions with adaptive *LASSO* and *LASSO* trailing *RFE* and *ARD* regression in this regard.

Try it out yourself!

You can find data and analysis scripts for the silicon third-order *FC* example in the public gitlab repo https://gitlab.com/materials-modeling/hiphive-examples/. The silicon example is located in `examples/Si_bulk`.

8.3 Alloy phase diagrams

As discussed in Chapter 6.1, *CE*s are very efficient models for predicting the energetics of multi-component systems. In practice, we are, however, rarely interested in the energies alone. Rather we care about, e.g., phase diagrams or thermodynamic average of observables such as the heat capacity or site occupancy factors. These kind of properties can be derived from Monte Carlo simulations for sampling thermodynamic ensembles and using the *CE* model to compute the energy changes between different configurations. These simulations are computationally relatively expensive and the results are subject to statistical noise as a direct result of the sampling procedure. As a result, a Bayesian approach for measuring the sensitivity of these predictions (e.g., via *MCMC* simulations) is ill advised. Yet, we have seen above that "bagging" based on *CV* (Section 5.5.1) can be a suitable and computational more efficient alternative.

This approach was taken, e.g., in [25] to obtain an error estimate for the phase boundaries in the Ag-Pd system (Fig. 3). A set of 10 *CE* models was generated by bagging, where each individual model was trained using *ARD* regression. For each of these models the phase boundaries ("Ensemble optimizer, individual CEs" in Fig. 3) were obtained from the free energies extracted from *MC* simulations as a function of composition and temperature. The thus obtained phase boundaries yield critical temperatures between 673 and 803 K, with an average of 742 K, which is only slightly higher than the estimate from a *CE* that was trained against the entire available data set. This illustrates again that *CV*-*RMSE* scores alone are an insufficient measure of the quality of a linear model with respect to a particular application, say the prediction of a phase diagram. Rather a more careful assessment of the quantity of interest must be carried out if an accurate estimate is required. In the case of the Ag–Pd phase diagram considered here, a heuristic approach is to estimate the uncertainty in the critical temperature (and thus the uncertainty in the phase boundary) by converting the *CV*-*RMSE* score to a temperature scale. This yields 2 meV/atom, which is equivalent to about 23 K, and thus comparable to the spread observed. This estimate is, however, unreliable as the spread in the mixing energies predicted here is only indirectly related to the miscibility gap.

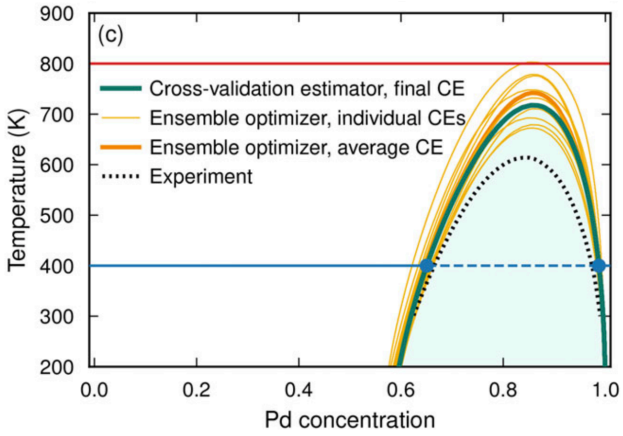


Fig. 3: Phase diagram for Ag-Pd from *CE* models.

For quantities such as site occupancy factors (Fig. 4; [25]) or heat capacities, no heuristic estimates are available. Yet it remains important to determine the sensitivity of predictions to the model uncertainty.

Try it out yourself!

In Project 2a one of the tasks is to determine how sensitive the prediction of the ground state structure to the mode uncertainty. You will thus have to carry out a sensitivity analysis yourself.

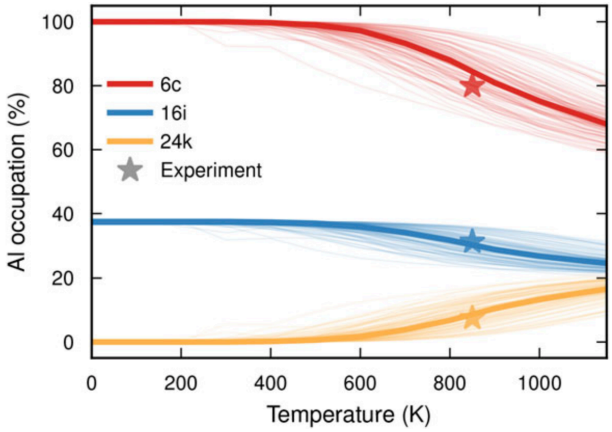


Fig. 4: Al site occupancy factors in the inorganic clathrate Ba8AlxSi46−x with x=12. Simulations were carried out for an ensemble of {termCE models obtained by different 90-10 splits of the reference data set (thin shaded lines) as well as the *CE* model obtained by averaging over the ensemble (bold line).

CHAPTER

NINE

GLOBAL OPTIMIZATION

Key take-away messages

- Global optimization techniques can be grouped into different categories, including (but not limited to) direct (not discussed here), stochastic, heuristic and metaheuristic as well as response surface based methods.
- Stochastic approaches (simulated annealing, parallel tempering) are largely based on Monte Carlo (*MC*) techniques.
- Heuristic and metaheuristic techniques (particle swarm optimization, ant colony optimization, evolutionary algorithms) are often guided by observation of natural systems.
- Bayesian optimization (a response surface based method) using Gaussian processes to generate a cheap model of the true system that can be explored much more quickly.

Global optimization is in general a very hard problem and becomes an increasingly daunting task as the number of dimensions increases. This lecture attempts to provide short introductions to several global optimization techniques, the selection of which has been guided by the proximity to physical problems, the connection to Bayesian techniques and the interest of the lecturer. There are many other techniques and even more variations that have emerged in the literature, which reflects on the difficulty of the problem.

9.1 Stochastic methods

Global optimization is at the heart of physics. For illustration consider a square lattice with N sites, over which we distribute two different species (say black and white spheres or up and down spins). Ignoring symmetry there are 2^N ways to distribute the two different species, e.g., $2^{100} \approx 10^{30}$; if symmetry is taking account this number will drop but still be gigantic. Let us assume that the interaction between the species (or spins) is repulsive. The ground state is thus a system where the two species are separated by sharp boundaries. The number of ways in which we achieve such configurations is extremely small compared to the total number of configurations. Nonetheless we routinely observe such ordered structures in reality. The crucial balance is the one between energy (or information) and entropy. The entropy of a state is given by

$$S = k_B \ln \Omega,$$

where Ω is the number of ways, in which the state can be achieved. Entropy one thus favors disordered structures, whereas energy prefers ordered structures. The thermodynamic potential of the canonical ensemble, the free energy, captures this duality

$$F = U - TS,$$

where T is the temperature and U is the internal energy. In the zero-temperature limit the free energy equals the internal energy whereas with increasing temperature entropy takes over. Stochastic techniques such as simulated annealing and parallel tempering are based on this principle.

9.1.1 Simulated annealing

In the case of a discrete (configuration) space such as the lattice example above, the partition function is given by

$$\mathcal{Z} = \sum_i \exp\left[-E_i/k_BT\right],$$

where the summation runs over all states i. The ensemble is conveniently sampled by *MC* simulations using the Metropolis-Hastings algorithms, according to which moves are accepted with probability

$$\mathcal{P} = \begin{cases} 1 & \Delta E \leq 0 \\ \exp(-\Delta E/k_BT) & \Delta E > 0 \end{cases},$$

where ΔE is the change in energy during the trial move under consideration. In simulated annealing the temperature T is initially chosen to be large and then gradually reduced during the course of the simulation. This procedure "simulates" the procedure by which materials, in particular alloys such as steel are thermally annealed. It can be applied in its immediate physical meaning, e.g., to find the ground states of lattice models.

Practical Example

The notebook `simulated-annealing.ipynb` demonstrates simulated annealing for the case of a square lattice with nearest neighbor interactions.

Discrete spaces such as the lattice model above are relatively simple since their configuration space is comparably small. The procedure can, however, be also used for continuous spaces and in any real-valued loss function. In this general case, the temperature becomes fictitious and k_BT is given in the units of the loss function. The general procedure is illustrated for the empirical potential example that was considered in Sect. Section 6.3.1 in Figs. Fig. 1 and Fig. 2.

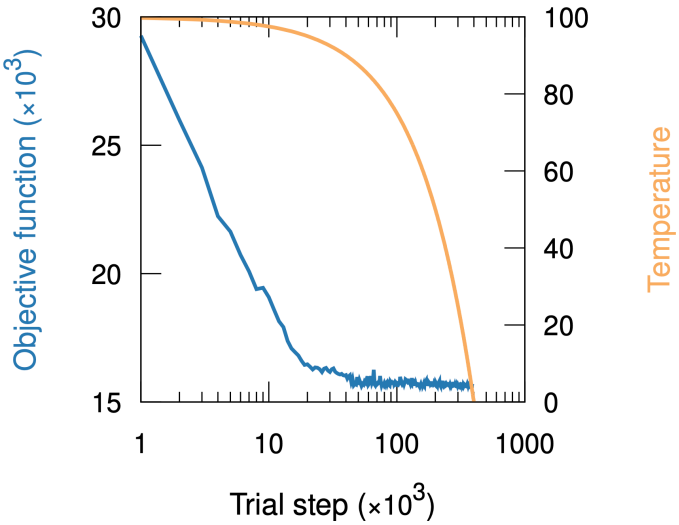


Fig. 1: Loss function and (fictitious) temperature during a simulated annealing based optimization of an empirical potential. Note the similarities between the present approach as well as parameter values and the demo of Bayesian sampling of a potential discussed in Sect. Section 6.3.1. The full example can be found at https://atomicrex.org/advanced_topics/simulated_annealing.html.

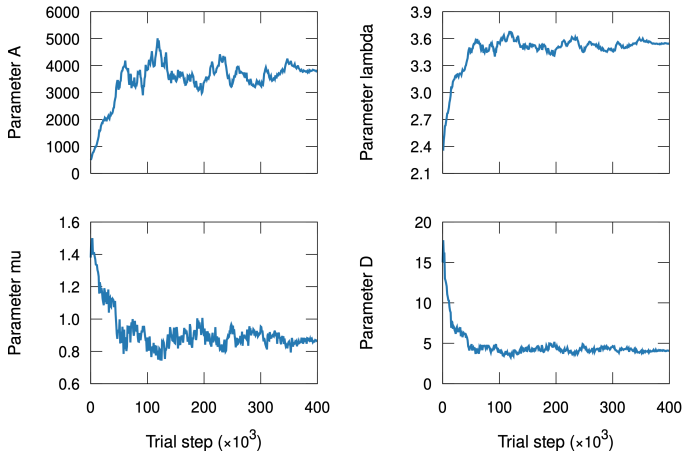


Fig. 2: Variation of the potential parameters during the simulated annealing run. Note the similarities between the present approach as well as parameter values and the demo of Bayesian sampling of a potential discussed in Sect. Section 6.3.1. The full example can be found at https://atomicrex.org/advanced_topics/simulated_annealing.html.

9.1.2 Parallel tempering

An approach closely related to simulated annealing is parallel tempering, which is also known as replica exchange Markov chain *MC* sampling [42]. In this approach N systems are initialized and run at different temperatures. The systems are individually propagated using, e.g., standard canonical *MC* updates. With a certain frequency it is, however, also possible to exchange two states, i.e., swap the configurations between two different temperatures. This exchange occurs with a probability given by

$$\mathcal{P} = \min\left[1, \frac{\exp\left(-\frac{E_j}{k_B T_i} - \frac{E_i}{k_B T_j}\right)}{\exp\left(-\frac{E_i}{k_B T_j} - \frac{E_j}{k_B T_i}\right)}\right] = \min\left[1, \exp\left(\left(E_i - E_j\right)\left(\frac{1}{k_B T_i} - \frac{1}{k_B T_j}\right)\right)\right].$$

This enables a much more comprehensive sampling of the configuration space and hence is more likely to succeed in locating minima in the landscape.

Originally parallel tempering was not designed as a global optimization technique but to provide better access to low-temperature properties and to avoid critical slow-down due to long-range correlation at phase transitions. It can, however, also been used to find ground state configurations.

In order to achieve convergence of the simulated annealing procedure, one must commonly adopt a cooling procedure that follow a schedule of the form (also see above)

$$T_i = \frac{T_0}{\log(i + i_0)}, i \geq 1.$$

Due to this logarithmic slow down, convergence cannot always be achieved and the minimization can stuck in local minima. Here, the parallel tempering approach has advantages since it allows for a geometric temperature schedule [43]. The optimal schedule in this case ensures a constant probability $\mathcal{P}$ for swaps between neighboring temperatures, which is achieved via a geometric progression

$$T_i = R^{i-1} T_{\text{min}}, 1 \leq i \leq N,$$

where R is chosen to achieve an target acceptance value, typically between 0.4 and 0.5 (also see Ref. [44] for additional insight regarding optimal acceptance ratios). R is then set according to a geometric progression (see Ref. [43] for details). The above procedure was used, e.g., to find the lowest energy configurations for complex silicon surfaces [43] (Fig. Fig. 3).

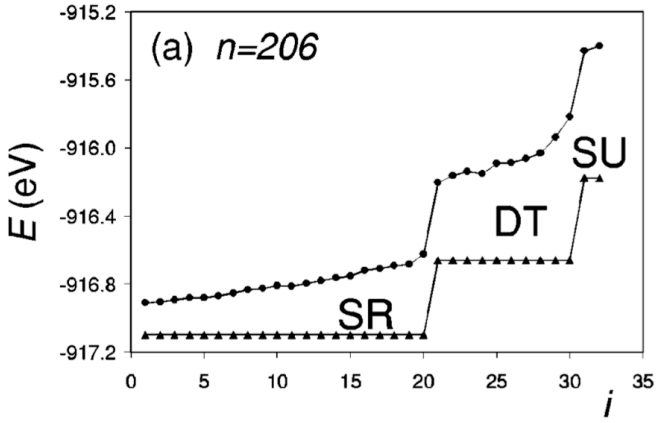


Fig. 3: Left: Energies of 32 replicas (index on x-axis) of a Si(105) slab configuration at the end of the cooling sequence (circles) and after subsequent conjugate-gradient minimization (triangles). Right: Minimum configurations corresponding to the plateaus indicated in the left-hand panel.

9.2 (Meta)heuristic methods

There is a variety of heuristic techniques that are based on observation of natural processes. This includes, e.g., methods inspired by the behavior of animals such as particle swarm optimization (*PSO*) and ant colony optimization (*ACO*) as well as evolutionary algorithms such as genetic algorithms (*GAs*).

9.2.1 Particle swarm optimization

PSO is inspired by the observation of the collective behavior of, e.g., flocks of birds and schools of fish. In these systems no individual has knowledge of the entire system but merely information concerning its immediate surrounding. The entire swarm, however, exhibits very complex and structured behavior. This allows the swarm as a whole to respond better to, e.g., attacks by predators or other forms of danger, food availability or thermal effects (thermal columns).

In *PSO* we start with a swarm of "particles", each of which

- has a position and a velocity,
- knows the value of the loss function for its position,
- remembers its best previous position (according to the loss function) and
- knows the best positions and respective functions of its neighbors.

Neighborhood can either be defined in the beginning and remains the same throughout the optimization ("social" neighborhood) or be defined on some distance measure ("physical" neighborhood).

The propagation of the particles (and thus the swarm) is then governed by the following equations [45]

$$\vec{v}_{i,t+1} = c_1\vec{v}_{i,t} + c_2(\vec{\tilde{x}}_{i,t} - \vec{x}_{i,t}) + c_3(\vec{\tilde{x}}_{i-\mathrm{nbr},t} - \vec{x}_{i,t})$$
$$\vec{x}_{i,t+1} = \vec{x}_{i,t} + \vec{v}_{i,t+1},$$

where $\vec{x}_{i,t}$ and $\vec{v}_{i,t}$ are the position and the velocity of particle i at time step t, $\vec{\tilde{x}}_{i,t}$ is the best position along the trajectory taking by particle i up to time t and $\vec{\tilde{x}}_{i-\mathrm{nbr},t}$ is the best previous position among the neighbors of i up to time t. Note that $\vec{x}, \vec{v} \in \mathbb{R}^n$, where n is the number of degrees of freedom (*DOFs*; typically parameters) of the model. c_1, c_2 and c_3 are coefficients and referred to as social/cognitive confidence coefficients. c_1 quantifies how much the particle trusts itself *now*, c_2 measures how much the particle trusts its *experience* and c_3 determines how much the particle trusts its *neighbors*.

There exist a lot of different strategies for choosing the coefficients, designing the swarm and varying the propagation. If you want to explore this topic further, you can start with Ref. [46]. You can also check out the `PySwarms` package, which provides a Python implementation of *PSO* that can be adapted to different application cases. Another short introduction with dummy code can be found here.

9.2.2 Ant colony optimization

Another heuristic algorithm inspired by a biological system is *ACO*, which allows one to find short and hopefully the shortest path(s) through a graph. Hence any problem that can be cast in this form is in principle amenable to this technique.

As the name suggest, *ACO* is inspired by the foraging behavior of ants, each of which is an individual unit that only operates locally. To mark pathways through their habitat the ants deposit pheromones, which allow ants to follow their path in order to find food. The *ACO* emulates this procedure on a weighted graph. It can be summarized as follows [47]

1. several ants (walkers, agents) are placed at the origin of the graph and find different stochastic pathways through the graph
2. the quality (weighted length) of the paths is compared
3. "pheromone" is deposited along each path, with more pheromone placed on shorter paths
4. another set of ants passes through the graph choosing each next step weighing in both the desirability (typically related to the length) of the path and the amount of pheromone already deposited

This procedure is carried iteratively until some termination condition is fulfilled.

More formally the probability for choosing a step from state i to state j (edge selection) is given by

$$\mathcal{P}_{ij} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_k \tau_{ik}^{\alpha}\eta_{ik}^{\beta}},$$

where τ_{ij} is the amount of pheromone deposited along the edge (i, j) and η_{ij} is the desirability of the edge (i, j) (the a-priori knowledge, which is typically related to the inverse distance), while α and β are two positive constants and the summation in the denominator runs over all possible edges involving i.

The pheromone update step is formalized as follows,

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_k \Delta\tau_{ik}^{k}.$$

Here, $\Delta\tau_{ik}^{k}$ is the amount of pheromone deposited by ant k in the current iteration along the edge (i, k) and it is given by

$$\Delta\tau_{ik}^{m} = \begin{cases} \frac{Q}{L_m} & \text{if ant } m \text{ used edge } (i,k) \text{ during its current tour} \\ 0 & \text{otherwise} \end{cases},$$

where Q is a constant and L_m is the tour length of ant m (or a generalized measure thereof).

ACO is particular useful for solving problems of the traveling-salesman type, i.e., finding shortest pathways throughout networks, e.g., in telecommunication or transport.

9.2.3 Evolutionary algorithms

Evolutionary algorithms emulate the process of biological evolution by implementing processes such as reproduction, mutation, recombination and selection. One considers a "population of individuals" each of which represents a potential solution to the problem. The quality of each solution is measured by its fitness function, which is equivalent to a loss function. The population then evolves by undergoing the processes named above.

In practice there is a lot of freedom in designing the different steps, which makes it difficult to generalize these algorithms. The general algorithm can, however, be summarized as follows:

1. generate a set of initial candidates (the first generation)
2. evaluate the fitness scores and select the best performing candidates
3. subject these candidates to mutation operations as well as "breeding" operations to generate a new generation
4. return to step 2 and repeat

The most expensive part of the computation is often the fitness function, which sets the performance limit for the algorithm. Evolutionary algorithms have a number of limitations such as poor scaling with complexity, an ill-defined convergence condition or a tendency to converge to local minima. It can also be difficult to come up with "good" operations for evolving good candidates.

9.3 Response surface methodology-based approaches

9.3.1 Bayesian optimization

Often times, both in physics and *ML*, the we seek to optimize a function $f(\vec{x})$ that cannot be represented analytically and for which we do not have access to derivatives either. We can merely evaluate the function at a given point $\vec{x}$ and the response is likely also subject to noise. In practice the latter evaluation is often also very expensive, e.g., because it requires time-consuming experiments or calculations. The task is thus to find the optimal $\vec{x}$ with as few evaluations of $f(\vec{x})$. This is where Bayesian optimization comes in [48]. As in other Bayesian techniques that we have discussed in this course, this procedure incorporates prior knowledge over f and operates via a posterior. To this end, one requires a model that approximates f. This *surrogate* model is often achieved via *GPs*. As in the construction of *GPs* one uses an *acquisition function* to determine which point $\vec{x}$ to sample next. These functions balance between exploration (sample unknown regions $\rightarrow$ positions with large uncertainty of f) and exploitation (sample close to extrema $\rightarrow$ locations with large absolute value of f).

The procedure can be summarized as follows

1. find the next sampling point by optimizing the acquisition function
2. sample the loss function (which is possibly subject to noise
3. add the sample to the data and update the *GP*
4. return to 1.

Practical Example

The notebook `bayesian-optimization.ipynb` demonstrates the use of Bayesian optimization to find the minimum of a rugged and noisy function in one dimension. A more in-depth discussion of practical aspects of Bayesian optimization with *GPs* can be found in Ref. [48]. A step-by-step introduction to an implementation in Python (using `NumPy` and `SciPy`) can be found at http://krasserm.github.io/2018/03/21/bayesian-optimization/.

CHAPTER

TEN

APPROXIMATE BAYESIAN COMPUTATION

Recommended literature

- Sisson, Fan, and Beaumont, *Handbook of Approximate Bayesian Computation* (2019)
- Lista, *Statistical methods for Data Analysis*, 2nd Ed. Springer (2018)

10.1 Introduction

As of today, machine learning (and NNs in particular) remains a controversial topic for many physicists because of several concerns. One point is that the idea of supervised learning (being the most practical part of *ML* so far) can be seen as a brute-force version of heuristic approximation. Instead of intelligent deduction of model through a sequence of experiments and logical inferences exploited in physical sciences, *ML* implies the logic of taking generic models with vast amount of parameters and their adjustment by sufficient amount of available data. Although this appears practical in some problems, there are essential limitations, including:

- curse of dimensionality
- explainability
- reliability

The *curse of dimensionality* stands for the difficulty of scaling *ML* models: the number of cases needed for training grows exponentially with the number of quantities to be determined. One interpretation is that current *ML* models tend to "memorize" given cases and approximate states in between by generic rules. Once the number of quantities to be learned increases the coverage of their high-dimensional space with known points becomes rarefied and these generic rules become increasingly inefficient. The problem of *explainability* is associated with the fact that even successful *ML* models appear as "black boxes", i.e., they do not directly provide insights that one can build on, something that appears crucial for the scientific method. Finally, inability to interpret rules learned by *ML* models makes it difficult to control and combat errors. For example, there are many indicative examples in the area of image recognition that show paradoxical errors and exemplifies the problem of *reliability*. Despite ongoing progress in combating the outlined difficulties, we have to accept the fact that in almost any non-trivial task current *ML* models *make* mistakes. How can one then use them in research? There are several options (see, e.g., [49], [50]):

- making quick decisions for controlling equipment, when errors are acceptable;
- processing large data sets and finding candidates verifiable by other means;
- proposing approximate solutions to be fine-tuned by other means;
- processing large data sets for statistical inferences.

The last option may sound controversial: If *ML* models do mistakes, how can we apply them for making inferences that are required to be accurate by nature? This section concerns an approach that provides such a possibility.

10.2 Toy problem: characterization of a leaning Galton board

Let us start from considering a simple toy problem. Suppose we are given with a leaning Galton board shown in Fig. 1a and are asked to model it. According to our heuristic model, k beads propagate through n rows of pegs so that at each peg they bounce to the left with probability θ and to the right with probability $1-\theta$. Under this prior model, our goal is to determine θ based on the results of experiments. This is known as the problem of model calibration.

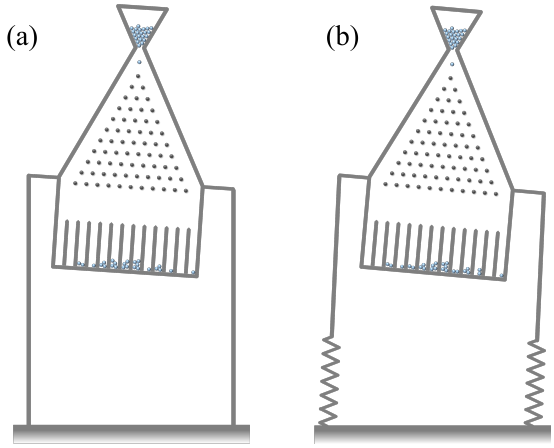


Fig. 1: Schematic representations of the leaning Galton board resting on rigid pillars (a) or springs (b).

We can note that the probability distribution of final bead location x (the cell number) can be computed analytically for each bead:

$$p\left(x \mid \theta\right) = \binom{n}{x} \theta^{x} \left(1-\theta\right)^{n-x},$$

where $\binom{n}{x} = x!/\left(n!\left(n-x\right)!\right)$ denotes the binomial coefficient. We can now compute the posterior distribution given any outcome x_1 of the first bead and the prior $\pi(\theta)$:

$$p\left(\theta \mid x_1\right) = \frac{p\left(x_1 \mid \theta\right)\pi(\theta)}{\int p\left(x_1 \mid \theta'\right)\pi(\theta')d\theta'}.$$

The outcomes are independent and we thus can account for all the beads in a straightforward way (for the sake of the argument we assume for the moment that the beads are enumerated):

$$p\left(\theta \mid x_i, ..., x_1\right) = \frac{p\left(x_i \mid \theta, x_{i-1}, ..., x_1\right) p\left(\theta \mid x_{i-1}, ..., x_1\right)}{\int p\left(x_i \mid \theta', x_{i-1}, ..., x_1\right) p\left(\theta' \mid x_{i-1}, ..., x_1\right) d\theta'}, \;\; i = 2, ..., k;$$

$$p\left(\theta \mid x_k, ..., x_1\right) = \frac{\prod_{i=1}^{k} p\left(x_i \mid \theta\right)\pi\left(\theta\right)}{\int \prod_{i=1}^{k} p\left(x_i \mid \theta'\right)\pi\left(\theta'\right)d\theta'}.$$

In the last expression we exploit the fact that the beads are identical and the probabilities of all events are independent from each other. We thus can account for all the beads in the their resultant position in any order. The maximum likelihood estimator takes the form:

$$\theta^{*} = \arg\max_{\theta} \prod_{i} p\left(x_i \mid \theta\right)\pi\left(\theta\right).$$

We can carry out several experiments to achieve reasonably narrow posterior and determine corresponding θ^{*}. As one can see the solution of this problem is rather straightforward.

Now let us consider a modification of this problem. Suppose that the board is suspended on springs so that the angle of slant is affected by the distribution of landed beads across the cells (see Fig. 1b). Apart from the necessity to model

the effect of beads on the angle of slant, this modification causes a notable methodological difficulty. The probability distributions for the final location of beads are not anymore independent. That is why we need to account for all possible permutations of beads that lead to the observed result of our experiment. This makes our routine intractable. (See another modification of this example in [51]). How can we cope with this difficulty?

A notable fact is that we can still develop a reasonable model and simulate possible outcomes of the experiment for any given θ. It is thus obvious that we should be able to estimate θ based on which value of θ in our simulation yield distributions that are most close to what we observe from experiments. A similar situation appears in many other scientific problems. For example, in cosmology one can be interested in using various images of galaxies (from differenct perspective) to calibrate a model describing certain probabilistic details of galaxies' shapes. In epidemiology one can be interested in using the statistics of disease spreading to determine parameters of a model that can be used to determine the most efficient actions to undertake. One notable, and in many senses historically pioneering, cases is the determination of the Higgs boson existence based on the measurements of secondary particles and radiation appeared from the collisions of energetic contra-propagating protons. In the following section we formalise the approach of Approximate Bayesian Computations (*ABC*) that are ought to deal with such statistical inferences.

10.3 Problem statement and basic thoughts

Let us first generalize the formulation of the problem to be solved. We assume that we want to model a complex system (or process) through a sequence of experiments or acquired measurements $y^{(E)} \in Y$. We assume that we have a model with parameter θ, i.e., a function $M(\theta)$ that can generate hypothetical outputs $y^{(M)} = M(\theta)$ for any given value of θ (to highlight this possibility $M(\theta)$ is sometimes referred to as *generative* model). Our prior assumption is that this model can provide a good enough (useful for our needs) description of the system in question under appropriate choice of parameter θ.

Finally, to make the task more versatile, we assume that both the process and the model are non-deterministic (otherwise the task boils down to the inversion of $M(\theta)$ at $y^{(E)}$). Make it worse, we assume that the output of the model depends on some latent parameter z that varies from experiment to experiment in an uncontrollable and unknown way. For example, in the leaning Galton board problem we can imagine that the board oscillates on springs during the entire experimental campaign and thus the result of each experiment depends on the phase at the instance of turning the board over. This phase then can be seen as the latent parameter that defines the initial conditions for the simulation. Another, practical examples, is the impact parameter of the collisions at LHC. We thus can assume that the function of the model is non-deterministic and depends on both θ and an unknown latent parameter $z \in Z$.

Our task is to find such a value of θ that each experimental observation $y_i^{(E)}$ is closely reproduced by $M(\theta, z_i)$ under an appropriate choice of z_i, i.e. $\forall i : \exists z_i$ s.t. $y_i^{(E)} \approx M(\theta, z_i)$. To treat $M(\theta, z)$ in a generic way, we can visualize it by means of a black box shown in Fig. 2.

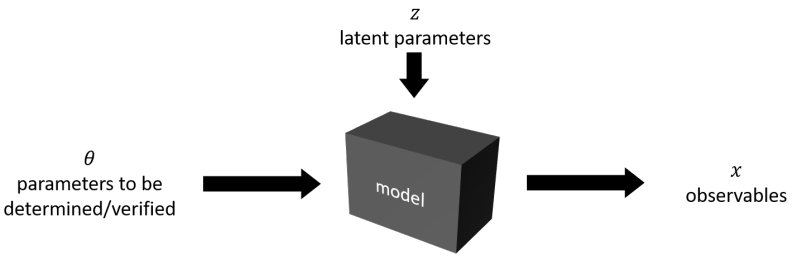


Fig. 2: Schematic representations of the *ABC* problem formulation.

An immediate idea that one can suggest for solving the outlined task is to see the task as a regression problem for the inversion of $M(\theta, z)$. Indeed, we can run our generative model for various θ, z: $((\theta_i, z_i) \to y_i^{(M)} = M(\theta_i, z_i))$ and then train, e.g., an *NN* to determine θ_i, z_i by given $y_i^{(M)}$. We can then take an experimental measurement $y^{(E)}$, send it to the

input of the trained *NN* and determine θ (and also particular z) right away. Although this approach can be sufficient in some cases, there are some severe flows that make it unacceptable in many other cases:

- the problem can be locally (for some y) ill-posed;
- the accuracy is limited and can be insufficient;
- we do not exploit the possibility to learn from many y;
- ehere are no fundamental grounds to assert that the NN did not make a mistake.

The last issue is crucial for scientific inferences, such as hypothesis testing. The cost that we pay for the *NN*s to be flexible (capable) is the fact that we cannot explain the criteria used by *NN*s and thus have no solid arguments to ensure the correctness of their predictions.

Another alternative idea is to solve the problem by brute-force search of appropriate θ, z by comparing each measurement with the outcomes of all the model for all possible θ, z. Although this is clearly impractical for high-dimensional z and y, this can be seen as the core idea behind the *ABC* routine that we develop in the next section. After clarifying the basics of *ABC* we will see that one can employ ML methods, benefiting from their capabilities and still controlling the accuracy of the routine by reliable means.

10.4 The concept of ABC

Learning from multiple observations can be provided by sequential computation of posteriors in a chain of Bayes' theorem applications, where the posterior on a particular step is used as a prior for the next step. To make use of this method, we need to write down the elemental Bayesian inference. Given with any observation y_{obs} and some prior $\pi(\theta)$ we can write the posterior in the following way:

$$p\left(\theta \mid y_{obs}\right) = \frac{p\left(y_{obs} \mid \theta\right)\pi(\theta)}{\int_{\Theta} p\left(y_{obs} \mid \theta'\right)\pi(\theta')d\theta'}. \tag{1}$$

However, under the assumed problem statement, the direct computation of this expression is intractable, because the computation of $p\left(y_{obs} \mid \theta\right)$ cannot be carried out in a reasonable time (the denominator is thus even more problematic). The only we have at our disposal is the generative model that generates cases $y \sim p\left(y \mid \theta\right)$ through, e.g., some demanding simulation. How can we find an approximation of the posterior (Eq. (1))?

Let us try to approximate the posterior by sampling it, i.e. by making a procedure that genrerates values $\theta_i \sim p\left(\theta \mid y_{obs}\right)$ of arbitrary size N such that

$$p\left(\theta \mid y_{obs}\right) = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} \delta\left(\theta_i - \theta\right),$$

where $\delta(\cdot)$ is the Dirac delta function and we assume that $\theta \in \mathbb{R}$ for simplicity. To implement this idea we employ the *rejection sampling algorithm*:

Algorithm 1: Rejection Sampling

Task: Generate samples $\theta_i \sim f(\theta)$, where $f(\theta)$ is not normalized.

Solution:

1. choose a function $g(\theta)$ that is convenient for sampling $\theta \sim g(\theta)$ and $g(\theta) > 0$ for all θ, for which $f(\theta) > 0$
2. generate proposal $\theta^* \sim g(\theta)$
3. accept θ^* with probability $f(\theta)g^{-1}(\theta)M^{-1}$, where constant $M \geq \max_{f(\theta)>0} f(\theta)g^{-1}(\theta)$
4. repeat steps 2 and 3 as many times as needed

Let us take $f(\theta) = p(\theta \mid y_{obs})$, $g(\theta) = \pi(\theta)$ then the acceptance probability is proportional to $p(y_{obs} \mid \theta^*)$ (see Eq. (1)). Even if $p(y_{obs} \mid \theta^*)$ is intractable, everything that we need is to make a decision about acceptance with the probability being proportional to $p(y_{obs} \mid \theta^*)$. We thus can use our model to generate y^* for θ^* and accept the proposal if $y^* = y_{obs}$. Since we bypass the necessity to compute likelihood $p(\theta \mid y_{obs})$ explicitly, this algorithm is sometimes referred to as *likelihood-free sampling algorithm*. Let us summarize this algorithm:

Algorithm 2

Task: Sample posterior by generating $\theta_i \sim p(\theta \mid y_{obs})$, where $p(\theta \mid y_{obs})$ is given by Eq. (1); y_{obs} and prior $\pi(\theta)$ are given.

Solution:

1. generate proposal $\theta^* \sim \pi(\theta)$
2. generate (perform simulation) y^* for θ^*
3. accept θ^* if $y^* = y_{obs}$
4. repeat steps 1 and 3 as many times as needed

Looking at the algorithm we can see an obvious issue: in order to ensure non-zero chance of $y^* = y_{obs}$ we need the set of possible outcomes X of our model to be discrete, and even in this case the chance becomes increasingly small with increasing dimensionality of X. This issue is circumvented by two distinct ideas that constitute the concept of *ABC*:

- kernels
- summary statistic

The use of kernels is the way to account for imperfect matches. For the sake of simplicity, let us assume, for the moment, that $y \in \mathbb{R}$. Then we can describe the idea of sampling by the selection of perfect matches in the following way:

$$p(\theta \mid y_{obs}) \propto p(y_{obs} \mid \theta)\pi(\theta) = \int \delta(y - y_{obs})\, p(y \mid \theta)\, \pi(\theta)\, dy. \tag{2}$$

We want to benefit from accepting $y^* \approx y_{obs}$, which we can quantify by condition $|y^* - y_{obs}| < h$ with constant h controlling the size of the acceptance interval. An obvious improvement is to account for the fact that closer matches should contribute to a larger extent than those provided by y^* from the periphery of the acceptance interval. Such non-uniformity of contributions is quantified by a localized function, referred to as the *kernel* function. To clarify the effect of the kernel function we approximate the likelihood:

$$p(y_{obs} \mid \theta) \approx p_h(y_{obs} \mid \theta) = \int \frac{1}{h} K\left(\frac{y - y_{obs}}{h}\right) p(y \mid \theta)\, dy$$

where we introduced a kernel function $K(\cdot)$ so that it is independent of h. Let us see what do we need to require from the kernel function:

$$\begin{aligned} p_h(y_{obs} \mid \theta) &\overset{u=(y-y_{obs})/h}{=} \int K(u) p(y_{obs} + uh \mid \theta)\, du \\ &= \int K(u) \left(p(y_{obs} \mid \theta) + uhp'_y(y_{obs} \mid \theta) + \frac{u^2h^2}{2} p''_y(y_{obs} \mid \theta) + \ldots \right) du \\ &= p(y_{obs} \mid \theta) \int K(u)du - p'_y(y_{obs} \mid \theta) \int K(u)udu \\ &\quad + \frac{1}{2}h^2 p''_y(y_{obs} \mid \theta) \int K(u)u^2du - \ldots \end{aligned}$$

As we can see the consistency requires $\int K(u)du = 1$. It is also reasonable to require that the kernel function is symmetric, then the second term in the Taylor expansion vanishes. The third and the following terms tend to zero with decreasing h, which is necessary (but generally not sufficient) for the convergence of the approximation, i.e., for

$$p(\theta \mid y_{obs}) = \lim_{h \to 0} p_h(\theta \mid y_{obs}).$$

For high-order terms to tend to zero we certainly need to require that the kernel function tends to zero sufficiently quickly so that the variance and higher order moments are limited, i.e., $\int K(u)u^2du < \infty$ and so on. In this case the error $|p_h(y_{obs} \mid \theta) - p(y_{obs} \mid \theta)|$ tends to zero as $\propto h^2$. Some common examples of kernels are shown in Fig. 3. Note, however, that the kernel function just controls the relative effect of perfect and imperfect matches. That is why commonly the choice of kernel function does not play a decisive role for the efficiency of *ABC* routine.

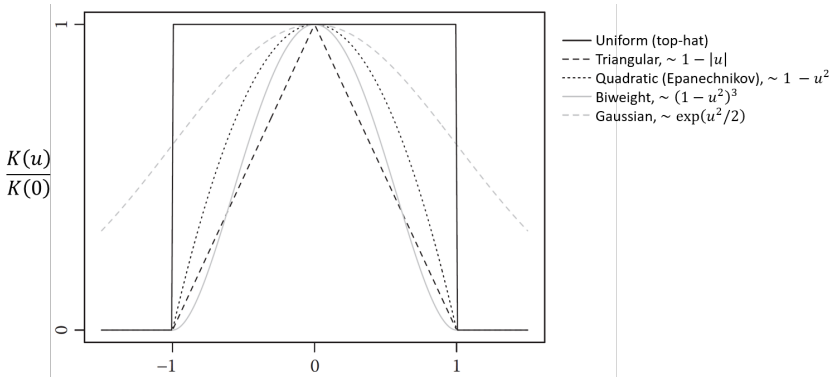


Fig. 3: Commonly used kernel functions. Adapted from Sisson *et al.* (2020).

The size of the kernel h, on the other hand, does play an important role. Overly small h value can lead to an inappropriately low probability of the acceptance, which means that computational resources are mostly wasted. Overly large value of h implies that many cases are accepted, but the sampling of the posterior can become significantly broadened due to kernel-related range $\sim h(\partial\theta/\partial y)$ (see schematic clarification in Fig. 4 around θ^*). That is why the choice of h is a matter of trade-off.

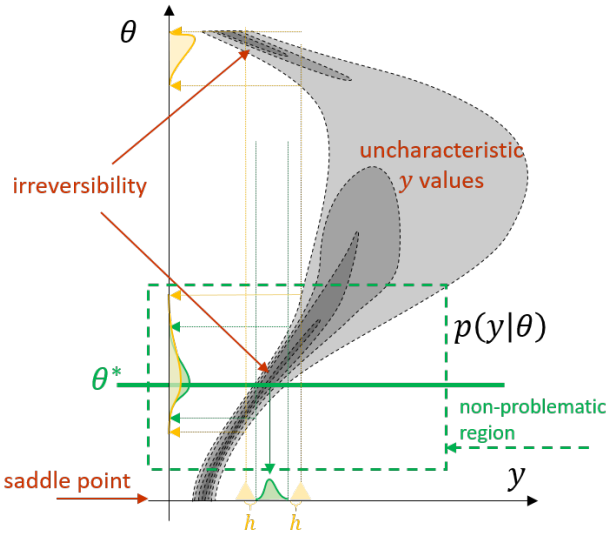


Fig. 4: Schematic illustration of various troubles one can encounter when choosing improper summary statistics.

Another question worth discussing is what to do if y is not one-dimensional. From the first glance, one may think that there is nothing to discuss because this problem can be reduced to the previous one. All we need is to define a metric $\|\cdot\|$

in the space of outcomes Y, i.e., a function that maps $Y \to \mathbb{R}$. This is partially true, but the choice of this metric, referred to as *summary statistics*, can be very delicate and require a good understanding/analysis of the problem. Let us see what kind of difficulties we can encounter in case of improper choice. In fig. Fig. 4 we showed a hypothetical distribution of $p(y \mid \theta)$ that illustrates several unwanted scenarios. First of all, if the model parameter θ is limited to the region outlined by dashed greed rectangle, we can expect convergence to some reasonable posterior shown in yellow next to the vertical axis and, in addition, can further narrow it down to the one shown in green by means of reducing h. However, if the region is not limited to the outlined region, we can encounter several difficulties:

- if the observed value of y are at the marked saddle point ($\partial y/\partial\theta \to \infty$) the accuracy of θ determination reduces significantly
- if the observed values of y are in the region "uncharacteristic y values" we cannot expect to identify the appropriate θ even if it does exist because the metric is indifferent
- even if y appear in the location corresponding to θ^* we cannot differentiate it from the one shown above, that leads to the same values of y

The overall reason in all these cases is the same: the chosen statistics doesn't differentiate the cases in terms of θ values. How can we circumvent this difficulty? One brute-force approach is to increase the dimensionality of y. In doing so, one would need to design and add to y some indicative functions that can disentangle the confused cases. Note that in reality this is further complicated by the fact that plotting $p(y \mid \theta)$ might be overly demanding. The solution is thus likely to be problem-dependent.

Let us highlight the driving principle for choosing summary statistics. The idea of summary statistic is to differentiate outcomes y_i of different values θ_i by the value of some function $s(y_i)$ that maps Y to a space of lower dimensionality. In the outlined example of leaning Galton board, we can note that a reasonable discrimination can be based on the location of the beads' center of mass. Indeed, we can expect that two models with different angles of slant will result in different location of the beads' center of mass.

The use of kernels and summary statistics can be summarized by the following algorithm:

Algorithm 3: standard ABC

Task: Sample posterior by generating $\theta_i \sim p(\theta \mid y_{obs})$, where $p(\theta \mid y_{obs})$ is given by eq.~((1)); y_{obs} and prior $\pi(\theta)$ are given.

Solution:

1. based on the problem determine summary statistics $s(y)$, kernel function $K(\cdot)$ and kernel size h
2. for the given y_{obs} generate proposal $\theta^* \sim \pi(\theta)$
3. generate (perform simulation) y^* for θ^*
4. accept θ^* with probability $K\left(\left(s(y^*) - s(y_{obs})\right)/h\right)/K(0)$
5. repeat steps 2 and 4 as many times as needed

10.5 ABC assisted by ML

As we saw in the previous section, the use of kernels and summary statistics provides a way to learn from matches that are imperfect but yet indicative. Nevertheless, finding such matches can still be unfeasible, especially if θ is multidimensional. One tempting idea is to restrict the plausible values of θ by some heuristic rule. The role of such restricted or, more generally, optimized proposal generator can be played by an *ML* model that is trained by the very same model we need to calibrate. (To not confuse the *ML* model and the model we need to calibrate, we use term *prior* model to denote the latter wherever needed.) This leads to the following algorithm:

Algorithm 4: ML-supported ABC

Task: Sample posterior by generating $\theta_i \sim p\left(\theta \mid y_{obs}\right)$, where $p\left(\theta \mid y_{obs}\right)$ is given by Eq. (1); y_{obs} and prior $\pi(\theta)$ are given.

Solution:

1. based on the problem determine summary statistics $s(y)$, kernel function $K(\cdot)$ and kernel size h
2. using the prior model, compute $y_i = M(\theta_i)$ for various $\theta_i \in \{\theta \mid \pi\left(\theta\right) > \delta\}$ ($\delta > 0$ is some small constant defining hypothetical θ values) and train an *ML* model to determine θ based on y
3. evaluate distribution $g\left(\theta^t \mid \theta^{\text{ML}}\right)$, where θ^t is the true value sent to the input of the prior model and θ^{ML} is the value predicted by the *ML* model for $y = M(\theta^t)$)
4. for the given y_{obs} determine the value θ_{ML} predicted by the *ML* model
5. generate proposal $\theta^* \sim g\left(\theta \mid \theta^{\text{ML}}\right)$
6. generate (perform simulation) y^* for θ^*
7. accept θ^* with probability

$$p_{acceptance} = \frac{K\left(\left(s(y^*) - s(y_{obs})\right)/h\right)\pi(\theta^*)}{g\left(\theta^* \mid \theta^{\text{ML}}\right)m}, \tag{3}$$

   where $m = \max_\theta \left(K(0)\pi(\theta)/g(\theta, \theta^{\text{ML}})\right)$
8. repeat steps 4 and 7 as many times as needed

At first, one may think that this idea is again subject to uncertainly that *ML* models inevitably introduce. However, there are two important differences.

Firstly, we explicitly account for the errors of the *ML* model. In essence, the idea is to perceive *ML* model as another complex system to be subject of our statistical analysis. We can measure the distribution function of errors and/or characterise it by *CI* or model it by some analytical *PDF*. In the algorithm above, we characterize this by function $g\left(\theta^t \mid \theta^{\text{ML}}\right)$. Next we account for the bias of our *ML* model by modifying the acceptance probability given by Eq.(3). In such a way we essentially prioritize the consideration of cases that our *ML* model considers probable. Note that if it happens so that some case θ_1 from peripheral region, i.e., $g(\theta_1, \theta^{\text{ML}}) \ll 1$ appears to be correct according to our summary statistics, it will be accepted with probability boosted by the corresponding factor $1/g(\theta_1, \theta^{\text{ML}}) \gg 1$. In practice, it might be overly expensive to control the importance of cases by the ratio of the probability of acceptance. One way to overcome this is to use weighted samples.

Secondly, under the discussed option, the *ML* model is used only for restricting the space to be searched, whereas the decision of whether to accept proposed case or not is still carried out based on summary statistics. The crucial difference from using trained *ML* models is that summary statistics can be defined, comprehended (by human notions), analysed, verified/disqualified in a competitive way that is crucial for the scientific method. In other words, if one publishes a conclusion made by *ML* model, the assessment of reliability is equivalent to solving long-standing problem of *ML* model explainability. On the other hand, conclusions achieved by using certain summary statistics can be further assessed, e.g., one can try to find cases that are not differentiated or prove that they do not exist.

Looking at Algorithm 4, one may notice that we sample proposals not from the prior but from the distribution governed by the *ML* algorithm and therefore we cannot arrange a sequential routine to make a proper use of several observations. However, using *ML* to accelerate *ABC* in such routine is possible in case we have some latent variable z. In this case, the *ML* model can be trained solely to deal with giving possible proposals of z. The algorithm then takes the form:

Algorithm 5: ABC with ML-based latent variable elimination

Task: Given observation y_{obs} sample the posterior for parameter θ (by generating $\theta_i \sim p\left(\theta \mid y_{obs}\right)$) of a model $y = M\left(\theta \mid z\right)$ that depends on unknown and uncontrollably varied latent parameter $z \sim \pi\left(z\right)$.

Solution:

1. given the specifics of the problem, choose a summary statistic $s(y)$, kernel function $K(\cdot)$ and kernel size h
2. train an ML model I_{ML} for solving an inverse problem: determine z from y independently off θ (generate $\theta_i \sim \pi(\theta)$, $z_i \sim \pi(z)$, compute $y_i = M(\theta_i, z_i)$ and fit $z_i \approx I_{ML}(y_i)$).
3. develop a model for errors: $g(|z - z_i|) \propto \sum_j f_{pdf}\left(\left|I_{ML}\left(M\left(\theta_j, z_j\right)\right) - z_j\right|\right)$
4. for the given y_{obs} determine the value z_{ML} predicted by the *ML* model
5. generate proposal $\theta^* \sim \pi(\theta)$, $z^* \sim g\left(\left|z - z^{\text{ML}}\right|\right)$
6. generate (perform simulation) y^* for (θ^*, z^*)
7. accept θ^* with probability

$$p_{acceptance} = \frac{K\left(\left(s(y^*) - s(y_{obs})\right)/h\right)\pi_z(z^*)}{g\left(|z^* - z^{\text{ML}}|\right)m},$$

   where $m = \max_{\theta,z}\left(K(0)\pi_z(z)/g\left(\left|z - z^{\text{ML}}\right|\right)\right)$
8. repeat steps 4 and 7 as many times as needed

As one can see, the *ML* model can play a role of optimizer or convergence booster, because its function is to prioritize and commit computational resources to the consideration of the cases that are most expected. Note however that reducing chance of considering other cases effectively implies neglecting most rare of them (given limited computational resources) and this constitutes another prior assumption we imply. We neglect the chance that our *ML* model repeatedly rules out the proper cases that could had led us to a different conclusion. Thus the use of *ML* is still subject to the extension of our prior assumption, but it comes in a more controllable way.

10.6 Hypothesis testing

A connected but yet distinct case is the situation when θ can take one of two values θ_0 or θ_1. This can be seen as the problem of *binary hypothesis testing*. To understand the use of *ML* models, summary statistics and kernels, let us recall the standard layout of hypothesis testing.

We denote the case $\theta = \theta_0$ to be the null hypothesis H_0, whereas $\theta = \theta_1$ corresponds to the new hypothesis H_1. To reject H_0 in favor of H_1 we need to achieve a high level of confidence using a sequence of experiments, each or which gives data $y \in Y$ being modeled by our prior models $y^{(0)} = M(\theta_0, z)$ for H_0 or $y^{(1)} = M(\theta_1, z)$ for H_1, z is a non-measured latent variable with a prior distribution $\pi(z)$. To perform testing we introduce a test statistic $T(y) : Y \to \mathbb{R}$ that is ought to discriminate the hypotheses:

- if $T(y) \leq \eta$ then H_0 is true
- if $T(y) > \eta$ then H_1 is true

We then introduce two quantities, probability of true positive detection:

$$\text{power} = p\left(T(y) > \eta \mid H_1\right) = \int_\eta^\infty p\left(T(y) \mid H_1\right),$$

and probability of false positive detection (referred to as p-value or significance):

$$\alpha = p\left(T(y) > \eta \mid H_0\right) = \int_\eta^\infty p\left(T(y) \mid H_0\right).$$

The level of confidence for rejecting H_0 in favor H_1 is quantified by the confidence level (*CL*) being equal to $\alpha/power$. For any given test statistic the value η is supposed to be optimized to give the lowest possible value of *CL* (highest

confidence for the rejection). The value of *CL* is treated differently in different areas of science. In physics, the level of *CL*$= 2.9 \times 10^{-7}$ (5σ) is commonly required to declare a discovery. This is to account for the high cost of mistake associated with the fact that the efforts of research community is predominantly devoted to the analysis of the currently accepted hypothesis rather then to any other hypothesis.

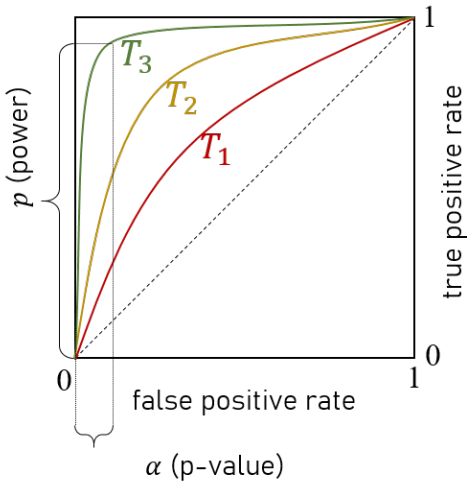


Fig. 5: Clarification of the meaning of values used for hypothesis testing.

The discriminating capability in terms of lowest achievable *CL* depends on the used test statistic. This is illustrated in Fig. 5, where we show three statistics T_1, T_2, T_3, from worst to best. The Neyman-Pearson lemma states that the most powerful test statistics is the likelihood ration:

$$r\left(y \mid H_0, H_1\right) = \frac{p\left(y \mid H_0\right)}{p\left(y \mid H_1\right)}. \tag{4}$$

Just like in the previous discussion, let us consider the case when the straightforward computation of the likelihoods in the enumerator and denominator are intractable. What can we do? We can apply the same ideas: estimate likelihood via sampling, increase acceptance probability by means of summary statistic and kernel and, finally, use *ML* model to accelerate the convergence of computations. This leads to the following algorithm:

Algorithm 6: ML-assisted hypothesis testing

Task: Estimate the likelihood ratio given by Eq. (4), $\pi_z\left(z\right)$ and y_{obs} are given.

Solution:

1. based on the problem determine summary statistics $s(y)$, kernel function $K(\cdot)$ and kernel size h
2. using the prior model, compute $y_i = M(\theta, z_i)$ for $\theta = \theta_0, \theta_1$ and various $z_i \in \{z \mid \pi_z\left(z\right) > \delta_z\}$ and train an *ML* model to determine z based on y
3. evaluate distribution $g\left(z^t \mid z^{\text{ML}}\right)$, where z^t is the true value sent to the input of the prior model and z^{ML} is the value predicted by the *ML* model for $y = M(\theta_{0,1}, z^t)$
4. for the given y_{obs} determine the value z_{ML} predicted by the *ML* model
5. generate proposal $z^* \sim g\left(z \mid z^{\text{ML}}\right)$
6. for the obtained z^* generate (perform simulation) y_0^* using H_0 and y_1^* using H_1
7. increase the cumulative estimate of $p\left(y_{obs} \mid H_i\right)$ by relative weight ($i = 0, 1$):

$$p_i = \frac{K\left(\left(s(y_i^*) - s(y_{obs})\right)/h\right)\pi_z(z^*)}{g\left(z^* \mid z^{\text{ML}}\right)}$$

8. repeat steps 4 and 7 as many times as needed

CHAPTER

ELEVEN

SUPPORT VECTOR MACHINES

Recommended literature

- M. Bishop, *Pattern Recognition and Machine Learning* (2006); Chapters 6.1 + 7.1
- A. J. Smola and B. Scholkopf, *A tutorial on support vector regression*, Statistics and Computing **14**, 199 (2004).

11.1 Introduction

One of the central problems of supervised learning is the lack of clarity with respect to the cause of errors done my ML algorithm. Tackling this problem constitutes a notable research area, referred to as eXplainable Artificial Intelligence (*XAI*). The central question to be addressed in this area is how to learn from the errors in the most efficient way so the ML model will never make the same kind of errors. Certainly, if we are lucky to see that our trained ML model does not make mistakes, this question will not arise on the first place. Looking at the principle of multi-layer NNs, one can see that the capability to learn by generic rules comes at the cost of opaqueness of the decisions made. An interesting alternative is provided by Support Vector Machines (*SVM*), which can give a good vision of the solution, while remaining fairly capable for a large class of problems.

To get some vision of the ideas behind *SVM*s let us consider the problem of binary classification. We denote the training set by pairs $(x_i, t_i)\,, i = 1, \ldots, N$, where $x_i \in \mathbb{R}^M$ are vectors of input and $t_i \in \{0, 1\}$ are outputs (binary labels). From the perspective of a physicist, one can notice that the idea of NNs can be naturally invoked from answering three basic questions:

- Model/rule: What do we take for the system that is ought to give answers?
- Driving principle: How to adjust the system?
- Implementation: How to make the adjustment scalable?

Fairly natural answers that lead to the idea of NNs are:

- Model/rule: multi-layer composition of non-linear functions + weighted sums;
- Driving principle: minimization of a continuous loss function;
- Implementation: variational principle $\rightarrow$ linearization + chain rule $\rightarrow$ gradient descent.

One can argue that the answer to the first question outlines an unnecessary broad class of systems. Indeed, one can construct the solution to arbitrary case using only one hidden layer (Cybenko's theorem) and there are plenty of possibilities to generate valid solutions using one given solution. At the same time, this excessive broadness of the system description naturally limits the way we can answer the second question and this, sequentially, deprives us of the possibility to interpret (understand) the solution. One tempting alternative is to restrict our choice of the system so that we can invoke interpretable principles when answering the two last questions.

Let us consider a simple example shown in Fig. 1a [52]. In this example we have two sets of points in 2D plane (i.e., $M = 2$); let us assume the sets correspond to $t = 1$ (yellow points) and $t = -1$ (red points). We need to find a rule, by which we can determine whether an arbitrary point belongs to one or another class. Such a separation means having some curve that separates the given points of two classes on the plane. Looking at the case shown in Fig. 1a, we can notice that we do not need anything more complicated than a line, i.e., a linear separator. It is also clear that such linear separation is achieved by computing a scalar product, which can be generalized to an arbitrary high dimensionality of x. An important observation is that in this and many other cases there exist more than one solution (line of separation). How can we make best use of this flexibility? We would certainly want our choice to contribute to the generalization capability, i.e., to provide high chance of our model to perform well on a test set (being randomly selected from the available data, but not "shown" to the algorithm). A natural idea is to stay away from bordering cases by maximizing the margin, i.e., to find the thickest possible layer that separates the points and take its middle for the separation rule.

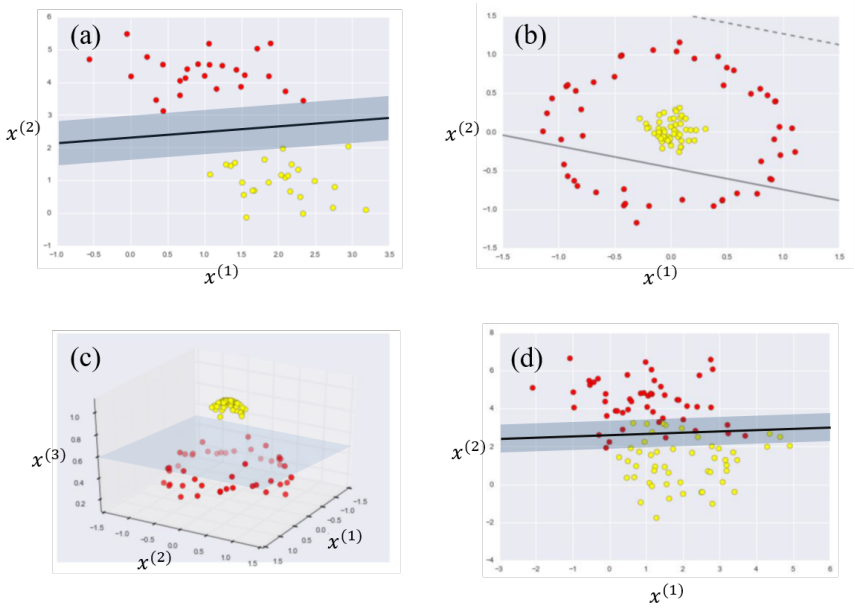


Fig. 1: Examples of linearly separable problems that illustrate the core ideas behind *SVM*s (see description in the text). Adopted from VanderPlas (2016).

Nevertheless, what can we do if the sets are not linearly separable as in the case shown in Fig. 1b? As one can see from Fig. 1c, a remarkably simple idea is to reduce the problem to the previous one (linear separation), but with higher dimensionality of x, composing an extra component of x using nonlinear function of the other components, i.e., $x_i^{(3)} = 1 - \left(x_i^{(1)}\right)^2 - \left(x_i^{(2)}\right)^2$. This is a particular example of the kernel trick, which we discuss in the following sections. For now, note that this example indicates the possibility to solve various problems staying within the principle of linear separation.

Another difficult case is the situation when the points are not separable because of intrinsic errors in labeling (or x determination) rather than because of systematic data complexity (see Fig. 1c). Even if we can increase the dimensionality of input vector x to achieve the separation, this would imply an overly sophisticated rule that we do not need when acknowledging the existence of errors. This is again the problem of overfitting. Can we invoke another geometrical principle to deal with this difficulty? Another natural idea is to minimize the number of misclassified points and some cumulative function of their distances from the line of separation. The outlined ideas constitute the basic principles behind *SVM*s and can be summarized by the following points:

- Model/rule: separation by a hyperplane;
- Driving principle: maximal margin or minimal cumulative distance of misclassified points from the hyperplane;
- Implementation: scalar product + kernel trick.

11.2 Regularization by margin maximization

The idea of margin maximization in case of binary classification of separable sets can be seen as an intuitive choice for increasing the chance of model to perform well on a test set. That is why this idea should effectively set a regularization principle. Let us see how.

Suppose we need to find a hyperplane that separates a set of N points $x_i \in \mathbb{R}^M$ according to labels $y_i \in \{-1, 1\}$, $i = 1, \dots, N$. The separation by a hyperplane can be defined by a vector $w \in \mathbb{R}^M$ and a bias $b \in R$ via scalar product, so that an arbitrary point $x \in \mathbb{R}^M$ is attributed to the class determined by:

$$y(x) = \text{sign}\left(w^T x - b\right). \tag{1}$$

Note that we can incorporate bias b into parameter vector w by extending the feature vector x by an adding component that is equal to -1 for all points. We do not do this due to the reasons that will become clear later. We denote the quantity

$$M_i(w, b) = y_i\left(w^T x_i - b\right) \tag{2}$$

as the *margin* for point x_i. The margin indicates the correctness of classification via sign and also quantifies the distance of x_i from the hyperplane.

Let us see the solution as a result of minimization of some appropriate loss function. As a simple choice for the loss function one can try taking the number of misclassified points:

$$L_{miss} = \sum_{i=1}^{N} H\left(-M_i(w)\right), \tag{3}$$

where $H(\cdot)$ is the Heaviside step function. However, this loss function does not provide the gradient needed for the gradient optimization. To overcome this difficulty we can try to modify the loss function so that its gradient indicates the direction of w adjustment towards $L_{\text{miss}} = 0$. One natural idea is to replace the Heaviside function by its argument $-M_i(w, b)$ under the sum of Eq. (3). In addition, we want our optimization process be governed exclusively by the points that are yet misclassified, while ignoring how far the correctly classified points are from the hyperplane. A simple way to implement this is to restrict the value under the sum to take non-zero values only if $M_i(w, b) < 0$. Finally, in case of separable classes, we want the hyperplane to not "touch" or lie closely to correctly specified points as this presumably decreases the chance of misclassification for test points nearby the bordering/peripheral points of both classes within the training set. To foster this we require some gradient nearby $M_i = 0$ so that the optimization leads to M_i being well positive. This geometrical intuition leads to the following form of the loss function (see Fig. 2):

$$L_m = \sum_{i=1}^{N} \left(1 - M_i(w, b)\right)_+,$$

where we use the positive cutoff function defined by $(\xi)_+ = \xi H(\xi)$. The choice of the function under the sum to deviate from zero at $(0, 1)$ and pass through $(1, 0)$ is arbitrary but not restrictive, because both the function and its argument are determined to within a positive factor. Indeed, multiplying the loss by a positive number does not change the objective of optimization, whereas multiplying w and b by a positive number rescales the absolute values of M_i.

Now we turn to the question of regularization, which concerns the principle for choosing the preferable hyperplane (or w and b) among all possible options that yield no misclassification on the separable training set (see Fig. 3a). Let us consider the set of all points in space (w, b) that yield zero loss $L_m(w, b) = 0$, i.e., the set of valid solutions. This set is limited by N sets defined by conditions $M_i(w, b) \geq 1$, $i = 1, \dots, N$. The case with maximal margin corresponds to the existence of points that yield $M_i = 1$ in both classes (otherwise the margin can be extended further to such a point). From the training set we choose any peripheral ($M_i = 1$) point from each class and denote them by x_- and x_+ for class $y = -1$ and $y = 1$, respectively, i.e.

$$w^T x_- - b = -1, \quad w^T x_+ - b = 1. \tag{4}$$

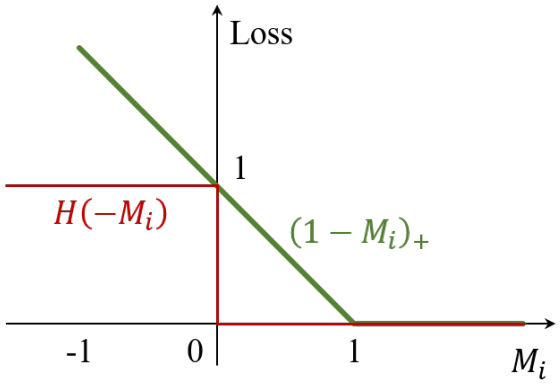


Fig. 2: Contribution to the loss as a function of margin based on simple counting (red) and on geometrically motivated ideas (green).

We compute the thickness of the margin via scalar product (see Fig. 3b) and substitute Eqs. (4):

$$T = \frac{w^T (x_+ - x_-)}{\sqrt{w^T w}} = \frac{w^T (x_+ - x_-)}{\|w\|} = \frac{2}{\|w\|}.$$

As we can see the maximization of the margin thickness corresponds to minimization of $\|w\|$. Thus in case of separable classes adding any monotonous function of $\|w\|$ to the loss function corresponds to the principle of margin maximization, which can thus be seen as the principle of regularization. A conventional (convenient) choice is

$$L_{\text{SVM}} = C \sum_{i=1}^{N} (1 - M_i(w,b))_+ + \frac{1}{2} w^T w, \tag{5}$$

where constant $C > 0$ controls the trade-off between the importance of reducing the number of mistakes and the importance of keeping most of the points far from the hyperplane.

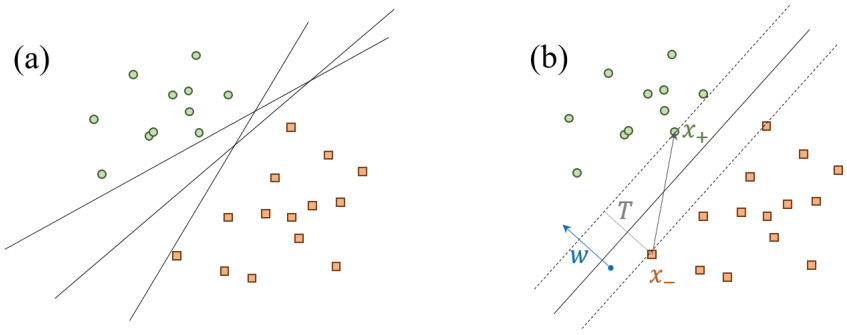


Fig. 3: Illustration of the idea behind the margin maximization principle.

11.3 SVM loss minimization

In this section we analyze the solution that one can obtain by minimizing the loss (or objective) function Eq. (5) obtained in the previous section, i.e.,

$$(w,b) = \arg\min_{w,b} \left(C \sum_{i=1}^{N} (1 - M_i(w,b))_+ + \frac{1}{2} w^T w \right). \tag{6}$$

To enable gradient-based analysis we need to get rid of non-smooth functions $(\cdot)_+$ in the loss function. We therefore formulate a different problem that, as we will see further below, is equivalent to the initial problem Eq. (6):

$$C \sum_{i=1}^{N} \xi_i + \frac{1}{2} w^T w \to \min_{w,b,\xi}, \tag{7}$$

$$M_i(w,b) \geq 1 - \xi_i, \quad i = 1 \ldots N, \tag{8}$$

$$\xi_i \geq 0, \quad i = 1 \ldots N, \tag{9}$$

where we introduced the vector of penalties $\xi \in \mathbb{R}^N$ to be subject of optimization (the components of ξ are enumerated by the subscript). Let us see the equivalence. Eq.~(8) means $\xi_i \geq 1 - M_i(w,b)$ and in combination with Eq. (9) yields $\xi_i \geq (1 - M_i(w,b))_+$. Since $C > 0$ in Eq. (7), among all options for ξ_i we should take the minimal $\xi_i = (1 - M_i(w,b))_+$ and this transforms Eq. (7) into Eq. (6).

The problem given by Eqs. (7) - (9) is the problem of convex function minimization with restrictions. The solution of such problems can be analyzed by considering a dual optimization problem and the Karush-Kuhn-Tucker (*KKT*) conditions. It is possible to handle restrictions given by both inequalities and equalities. The latter type of conditions is not relevant to our problem and we thus will use the following reduced formulation. Suppose we need to find

$$\min_{\{z | g_k(z) \leq 0\}} f(z), \tag{10}$$

where the loss function $f(z)$ and all the constraint functions $g_k(z)$ are continuously differentiable, then the *dual* problem is

$$\max_{\mu_k \geq 0} \left(\min_{z} \left(f(z) + \sum_{i=1}^{K} \mu_i g_i(z) \right) \right). \tag{11}$$

Let us note some intuition behind the duality of the problems. To get rid of the restrictions $g_k(z) \leq 0$ in the original minimization problem, referred to as *primal* problem, we can effectively incorporate them into the loss function by supplementing it with functions $p_k(z)$ that give infinite penalty for violating each of the restrictions. An approximate solution could be to add $H(g_k(z))$ multiplied by some sufficiently large positive constant. To stay within finite differentiable function, we consider a different option. For each restriction $g_k(z)$ we add $\mu_k g_k(z)$ with $\mu_k \in \mathbb{R}$. For each specific value of μ_k we have $p_k(z) \geq \mu_k g_k(z)$, but for each z we can approach the needed penalty $p_k(z)$ by choosing either $\mu_k = 0$ if $g_k(z) \leq 0$ or $\mu_k \to \infty$ otherwise. That is why our goal can be achieved by requiring that the minimum has to be achieved for the worst choice of $\mu_k \in [0, \infty)$ for any specific z:

$$\min_{z} \left(\max_{\mu_k \geq 0} \left(f(z) + \sum_{i=1}^{K} \mu_i g_i(z) \right) \right). \tag{12}$$

By changing the order of maximization and minimization we transform this problem to the one given by expression Eq. (11).

Since the loss function in Eq. (11), referred to as the Lagrange dual function $L(z)$, is continuously differentiable function, its minimum is reached when this function has zero gradient with respect to z. Thus the minimum of $f(z)$ with restrictions Eq. (10) and the point z^* where it is reached can be found by equating this gradient to zero and then considering the worst

possible choice of $\mu_k \geq 0$ (the choice that maximizes the value of the Lagrangian). This leads to the *KKT* conditions, which can be formulated in the following way. If z^* is the solution of the primal problem, then $\exists \mu \in \mathbb{R}^K$ so that

$$\nabla_z L(z)|_{z=z^*} = 0, \text{ where } L(z) = f(z) + \mu^T g(z), \tag{13}$$

$$\begin{aligned} g_k(z^*) \leq 0, \quad & k = 1 \dots K, \\ \mu_k \geq 0, \quad & k = 1 \dots K, \\ \mu_k g_k(z^*) = 0, \quad & k = 1 \dots K. \end{aligned} \tag{14}$$

The *KKT* conditions can be understood as follows. With respect to each restriction there could be two options: either z^* is at the boundary of the restriction region ($g_k(z^*) = 0$) or somewhere insider it ($g_k(z^*) < 0$). In the latter case we can say that the restriction is *inactive* and we take $\mu_k = 0$ (in accordance with Eq. (14)) to not affect the Lagrangian L (if all restriction are inactive, the expression (13) turns into Eq. (11), for which the minimum z^* gives $\nabla_z f(z)|_{z=z^*} = 0$). For active restrictions the statement implies that the gradient $-\nabla_z f(z)|_{z=z^*}$ can be expressed as a linear combination of gradients $\nabla_z g_k(z)|_{z=z^*}$ with positive coefficients. Let us see why by considering the contrary. First, if vector $\nabla_z f(z)|_{z=z^*}$ is not representable by a linear combination of vectors $\nabla_z g_k(z)|_{z=z^*}$ then there exists a direction that is perpendicular to all of them but not to $\nabla_z f(z)|_{z=z^*}$. In the limit of small shift from z^* along this direction, we can decrease $f(z)$ while not changing $g_k(z)$ (to the first order), i.e., not violating the restrictions. Thus z^* cannot be the minimum. Second, the sign $\mu_j > 0$ means that the relative orientation of vectors $\nabla_z f(z)|_{z=z^*}$ and $\nabla_z g_j(z)\big|_{z=z^*}$ does not permit further decrease of $f(z)$ under a small shift from z^*. (If we choose the direction that is perpendicular to all other $\nabla_z g_k(z)|_{z=z^*}$, $\forall k \neq j$, then the shift either decreases $f(z)$ and brings z into prohibited region $g_j(z) > 0$ or vice versa.)

To make use of the *KKT* conditions we write the Lagrangian for the problem Eqs. (7) - (9):

$$\begin{aligned} L(w, b, \xi) &= \frac{1}{2} w^T w + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \lambda_i \left(M_i(w, b) - 1 + \xi_i \right) - \sum_{i=1}^N \eta_i \xi_i \\ &= \frac{1}{2} w^T w - \sum_{i=1}^N \lambda_i \left(y_i \left(w^T x_i - b \right) - 1 \right) - \sum_{i=1}^N \xi_i \left(\lambda_i + \eta_i - C \right), \end{aligned} \tag{15}$$

where $\lambda \in \mathbb{R}^N$ and $\eta \in \mathbb{R}^N$. According to the *KKT* conditions the minimum can be described as follows (from hereafter we consider the values at minimum, omitting the indication by star):

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^N \lambda_i y_i x_i = 0 \;\Rightarrow\; w = \sum_{i=1}^N \lambda_i y_i x_i, \tag{16}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^N \lambda_i y_i = 0 \;\Rightarrow\; \sum_{i=1}^N \lambda_i y_i = 0, \tag{17}$$

$$\frac{\partial L}{\partial \xi_i} = -\lambda_i - \eta_i + C = 0 \;\Rightarrow\; \eta_i + \lambda_i = C, \tag{18}$$

$$\lambda_i \left(M_i(w, b) - 1 + \xi_i \right) = 0, \tag{19}$$

$$\lambda_i \geq 0, \tag{20}$$

$$\eta_i \xi_i = 0, \tag{21}$$

$$\eta_i \geq 0, \tag{22}$$

where expressions (18)-(22) hold for $i = 1, \dots, N$. From Eqs. (18), (20), (22) we deduce $\lambda_i \in [0; C]$. A useful classification of points can be carried out by considering possible values of λ_i:

- $\lambda_i = 0 \to$ Eq. (18) $\to \eta_i = C \to$ Eq. (21) $\to \xi_i = 0 \to$ Eq. (8) $\to M_i(w, b) \geq 1$;
- $0 < \lambda_i < C \to$ Eq. (18) $\to 0 < \eta_i < C \to$ Eq. (21) $\to \xi_i = 0 \to$ Eq. (8) $\to M_i(w, b) = 1$;
- $\lambda_i = C \to$ Eq. (18) $\to \eta_i = 0 \to$ Eq. (21) $\to \xi_i > 0 \to$ Eq. (8) $\to M_i(w, b) < 1$.

The points that correspond to the first class do not affect the resultant w given by Eq. (16). These points are correctly classified and lie far from the separating hyperplane (deeply inside their region of classification). The points that correspond to the second class lie at the boundary of the margin, they are correctly classified and the loss still yields zero penalty for them. These points do enter the resultant expression for w but they contribute to a limited extend given by $\lambda_i < C$. The points that correspond to the third class, contribute to the resultant w at the maximal possible extent $\lambda_i = C$. The loss function gives non-zero penalty for them. Note that these points can be classified either correctly (if $M_i(w, b) \geq 0$) or incorrectly (if $M_i(w, b) < 0$).

The points that belong to the last two classes are the only ones that directly affect the solution. The solution changes under small changes of their location, while small shifts of the other points do not affect the result. In geometrical sense they support the separating hyperplane (with margins) from both sides and are therefore called *support vectors*. This is the origin of the name *support vector machine*.

Now let us return to the dual problem and change the order of minimization and maximization (we regroup terms in Eq. (15)):

$$\begin{gathered}\text{find} \quad \min_{w,b,\xi} \left(\max_{\lambda,\eta} \left(L\left(w, b, \xi, \lambda, \eta\right)\right)\right) \\ \begin{aligned} L\left(w, b, \xi, \lambda, \eta\right) =& \frac{1}{2} w^T w - \sum_{i=1}^{N} \lambda_i y_i w^T x_i + b \sum_{i=1}^{N} \lambda_i y_i \\ &+ \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \xi_i \left(\lambda_i + \eta_i - C\right). \end{aligned}\end{gathered} \tag{23}$$

Looking at the last term and noting that in the dual problem $\xi_i \in (-\infty; \infty)$, we see that unless $\lambda_i + \eta_i - C = 0$ the value of the Lagrangian goes to $-\infty$ at either $\xi_i \to \infty$ or $\xi_i \to -\infty$. Thus the maximum is achieved for η_i is $\eta_i = C - \lambda_i$ (just as Eq. (18) states) and thus $\lambda_i \in [0, C]$ (since $\eta \geq 0$). In a similar way, looking at the third term, we see that we can take any λ but subject to $\lambda^T y = 0$ (just as Eq. (17) states). We substitute Eqs. (17)-(18) into the Lagrangian. Finally, Eq. (16) gives the best choice for w, which we also substitute to the Lagrangian. After this the Lagrangian becomes a function of λ only. In such a way we arrive to the formulation:

$$\text{find} \quad \max_{\lambda \in \Lambda} \left(\hat{L}(\lambda) \right), \tag{24}$$

$$\hat{L}(\lambda) = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_{i=1}^{N} \lambda_i, \tag{25}$$

$$\Lambda = \left\{ \lambda \in [0;\ C]^N \mid \lambda^T y = 0 \right\}. \tag{26}$$

We can see that this problem implies minimization of quadratic function $-\hat{L}(\lambda)$ within the region being the intersection of hypercube $[0;\ C]^N$ with the hyperplane given by $\lambda^T y$, i.e., within a convex region. This problem always has exactly one solution. One can imagine solving it by Newton's method, while problem-tailored efficient methods are a matter of separate consideration (an essential simplification is provided by the fact that commonly most of λ components are zeros, which corresponds to inactive restrictions or x_i being deeply insider its correct region of classification).

The values of λ obtained from the solution of problem (24)-(26) determine the sought-for classifier. To find b we choose any λ_i in the interval $0 < \lambda_i < C$ and, since $M_i = 0$, use Eq. (2):

$$b = \sum_{j=1}^{N} \lambda_j y_j x_j^T x_i - y_i. \tag{27}$$

To find w we can use Eq. (16). However, we can directly express the classifier via λ and b:

$$y(x) = \text{sign} \left(\sum_{i=1}^{N} \lambda_i y_i x_i^T x - b \right). \tag{28}$$

Let us summarize the results of our analysis. We showed that the problem of *SVM* loss minimization has exactly one solution. The parameter $C \geq 0$ can be used to control the strength of regularization: choosing large values favours the correctness of classification for maximal number of points, whereas choosing small values favours the use of a thicker margin so that the classifier can be less affected by small peculiarities or errors (with respect to x_i and/or y_i) in the training set. Finally, the solution provides a limited number of points (support vectors) that are the only used for determining the class of a new point. To some extent, this gives an important property of interpretability, especially if we get only a small number of support vectors.

11.4 Dual representation and kernels

Let us make an important observation: the problem statement for λ (24)-(26) and the resultant classifier (27)-(28) are expressed so that points x_i of the training set and the subject of classifier x enter the expressions via scalar products only. This means that for the use of *SVM* classifier the only we need is the procedure that computes scalar products (not the components themselves) between arbitrary points in the space, where we need to perform classification. Since the scalar product can be seen as a measure of closeness between points, we can see that the *SVM* classifier is navigated by this measure. This gives an important leverage: we can tailor the *SVM* classifier by defining and using problem-relevant scalar products. We can also try different options and check which one gives the best result. Let us extend the scope of this observation by noting this property in the problem of linear regression.

Let us consider a linear regression model with *MSE* loss function and L2 regularization:

$$J(w) = \frac{1}{2}\sum_{i=1}^{N}\left(w^T\phi(x_i) - t_i\right) + \frac{\lambda}{2}w^T w,$$

where vectors x_i and target values t_i form the training set of N pairs, vectors $\phi(x)$ $(M \times 1)$ are basis functions and w is the vector $(M \times 1)$ of weights to be optimized for minimizing the loss $J(w)$. Forming the design matrix $\Phi = (\phi(x_1), \dots, \phi(x_N))^T$ of size $M \times N$ and the target vector t (matrix of size $N \times 1$) we rewrite the loss:

$$J(w) = \frac{1}{2}\left(\Phi w - t\right)^T\left(\Phi w - t\right) + \frac{\lambda}{2}w^T w.$$

Next we equate the gradient $\nabla J_w(w)$ to zero, compute the optimal w and obtain the linear regression model:

$$\begin{aligned} w &= \left(\Phi^T\Phi + \lambda I_M\right)^{-1}\Phi^T t, \\ y(x) &= \phi(x)^T w = \phi(x)^T\left(\Phi^T\Phi + \lambda I_M\right)^{-1}\Phi^T t, \end{aligned} \tag{29}$$

where I_M is the identity matrix of size $M \times M$.

To make some useful transformation let us consider an equation

$$\left(\Phi^T\Phi + \lambda I_M\right)^{-1}\Phi^T = \Phi^T X, \tag{30}$$

and solve it for matrix X. Multiplying both sides by $\left(\Phi^T\Phi + \lambda I_M\right)$ from left, we get

$$\begin{aligned} &\Phi^T = \left(\Phi^T\Phi + \lambda I_M\right)\Phi^T X \;\Rightarrow\; \Phi^T = \left(\Phi^T\Phi\Phi^T + \lambda\Phi^T I_N\right)X \\ &\Rightarrow X = \left(\Phi\Phi^T + \lambda I_N\right)^{-1} \;\Rightarrow\; \left(\Phi^T\Phi + \lambda I_M\right)^{-1}\Phi^T = \Phi^T\left(\Phi\Phi^T + \lambda I_N\right)^{-1}, \end{aligned}$$

where I_N is the identity matrix of size $N \times N$. Substituting the last expression into Eq. (29) we obtain

$$y(x) = \underbrace{\phi(x)^T\Phi^T}_{k(x)^T}\left(\underbrace{\Phi\Phi^T}_{K} + \lambda I_N\right)^{-1}t, \tag{31}$$

where we outlined the Gram matrix K $(N \times N)$ and vector $k(x)$ $(N \times 1)$ with elements:

$$\begin{aligned} K_{ij} &= \phi(x_i)^T\phi(x_j),\ i,j = 1,\dots,N \\ k_i(x) &= \phi(x_i)^T\phi(x),\ i = 1,\dots,N. \end{aligned}$$

As we can now see, the linear regression model (29) can be expressed in the form (31) where x and x_i appear only via scalar products of basis functions. Again, this scalar product acts as the measure of closeness and its definition is a matter of distinct leverage for problem optimization. It is useful to define the *kernel function*:

$$k(x, x') = \phi(x)^T \phi(x').$$

One may think that changing the representation from Eq. (29) to that given by Eq. (31) causes an unnecessary increase of problem complexity because in Eq. (29) we need to invert matrix of size $M \times M$, whereas Eq. (31) implies the inversion of matrix $N \times N$, which is much larger for expected $N \gg M$. However, the actual benefit comes from the fact that via the kernel function computation we can implicitly handle higher or even infinitely high dimensionality of the basis function space, because we do not necessarily need to compute the components of x and x' to compute the kernel function.

This observation leads us to the idea of finding useful kernel functions. How can we check if a given function $\hat{k}(x, x')$ is a valid kernel? To enable the presented trick with linear regression model transformation (i.e., Eq. (29)→Eq. (31)), we need to require the existence of the basis functions $\hat{\phi}(\cdot)$ for which the given function is equal to the scalar product, i.e., $k(x, x') = \hat{\phi}(x)^T \hat{\phi}(x')$. In particular, this means that the kernel function has to be symmetric with respect to the permutation of x and x', i.e., $\forall x, x' : \; k(x, x') = k(x', x)$. This is a necessary but certainly not a sufficient condition.

The existence of basis functions that provide scalar product identical to the kernel function can be demonstrated directly. Consider the following classic example. Let us determine whether function

$$g(x, x') = (x^T x')^2 \tag{32}$$

is a valid kernel for $x, x' \in \mathbb{R}^2$. We bring the expression into the form

$$g(x, x') = x_1^2 {x'_1}^2 + 2x_1 x'_1 x_2 x'_2 + x_2^2 {x'_2}^2 \tag{33}$$

$$g(x, x') = \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2\right)^T \left({x'_1}^2, \sqrt{2} x'_1 x'_2, {x'_2}^2\right), \tag{34}$$

which makes it clear that the choice $\phi_1(x) = x_1^2$, $\phi_2(x) = \sqrt{2} x_1 x_2$ and $\phi_3(x) = x_2^2$ makes the trick, i.e., $g(x, x')$ is a valid kernel function.

The general procedure for kernel verification can be rather intricate. For example, one can use the following criterion [53]. The necessary and sufficient condition for $\hat{k}(x, x')$ be a valid kernel is that the Gram matrix K is positive semidefinite (i.e., $\forall z \in \mathbb{R}^N : \; z^T K z > 0$) for all choices of sets $\{x_i\}$.

As one can see the verification can be non-trivial. A useful alternative is to generate valid kernels out of other valid kernels using certain rules. Some rules for generating valid kernels $k(x, x')$ using valid kernels $k_1(x, x')$ and $k_2(x, x')$ ($x, x' \in X$) are (see also, e.g., [54], [53]):

$$k(x, x') = c k_1(x, x'), \text{ where } c > 0 \tag{35}$$

$$k(x, x') = f(x) k_1(x, x') f(x'), \text{ where } f : X \to \mathbb{R} \tag{36}$$

$$k(x, x') = q(k_1(x, x')),$$

where $q(\cdot)$ is a polynomial function with non-negative coefficients

$$k(x, x') = \exp\left(k1(x, x')\right), \tag{37}$$

$$k(x, x') = k_1(x, x') + k_2(x, x'), k(x, x') = k_1(x, x') k_2(x, x').$$

One can notice that the example Eq. (32) can be generalized to

$$k(x, x') = (x^T x + c)^m.$$

Using the same trick as in Eqs. (33)-(34) we can show that this kernel, referred to as *polynomial kernel*, is the scalar product in the base function space given by all monomials (of x components) of order up to m. In such a way we can

make linear model of regression to be based on large set of nonlinear base function even without the necessity to compute the values of these functions.

Another instructive example is the *Gaussian* kernel, which is given by

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma}\right),$$

where $\sigma > 0$ is a parameter. One can construct this kernel from $x^T x'$ using Eqs.(35), (36), and (37). Since the exponent (37) can be represented by Taylor series with infinite number of terms (powers of the argument), this kernel effectively use basis function space of infinite dimensionality.

It is interesting to note that valid kernels can be defined for objects as diverse as images, graphs, texts, etc (see, e.g., [54]).

11.5 Summary and insights

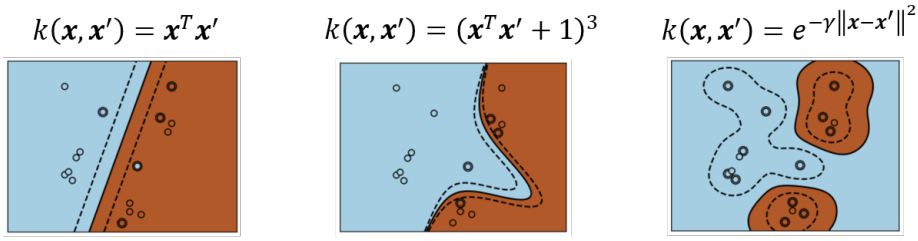


Fig. 4: An example that shows the effect of using different kernels for the *SVM* binary classification. The points of the training set are shown using circles filled with the color that indicates the target label. For the same set of point the *SVM* classifier is determined using different kernel shown above the picture. The pictures are colored according to the resultant classification and the support vectors are highlighted with thicker circles. In each case the solid curve shows the separator and the dashed curves show the boundary of the margin. Reproduced from the scikit-learn documentation.

To see the effect of different kernels, let us return back to the *SVM* binary classification and consider a particular example shown in Fig. 4. If we use scalar product $x^T x$ for the kernel (leftmost picture), the separation (solid line) and the margin boundaries (dashed lines) are straight lines. We see that one of the points is misclassified. Once we replaced the kernel by the third order polynomial kernel (central picture), the shape of the separation boundary becomes non-linear and this helps to separate all the points. The dimensionality of implied basis function space has grown from 2 to 9 and thus we have more degrees of freedom to separate the classes by the separation hyperplane in that space. Note that the *SVM* computational routine remains exactly the same: the only change is a different function used for the kernel computation. In such a way, we effectively made our classifier nonlinear. Moreover, the identified support vectors provide a way to understand the result without considering the 9 dimensional space of basis functions.

In case we use the Gaussian kernel (rightmost picture), the dimensionality of the basis function space is infinite and this means that any finite number of points are linearly separable in this space. Certainly, the parameter C can still be used to control the trade-off between correctness of classification for particular points and the generalization capabilities of the classifier. But how is this possible that our classifier became capable of perfect classification under this replacement of the kernel? If we look at the form of the kernel, we see that it quantifies the L2 closeness between points. Thus the classifiers effectively builds regions of certain classification around all points, while some "internal" points are not selected as support vectors. This is exactly what we can observe on the rightmost picture in Fig. 4.

Let us make another interesting observation. The classification rule (1) can be seen as a neural network with one hidden layer and sign$(\cdot)$ activation function at the output layer, see Fig. 5. The hidden layer is formed by the kernel computation with all the support vectors. What can we learn from this perspective? The NNs concern the adjustment of weights, while the topology (the number of neurons) and the choice of activation functions are commonly a matter of trial and error method. Here, with *SVM*, we have a deterministic, educated approach to determine the number of neurons in the hidden

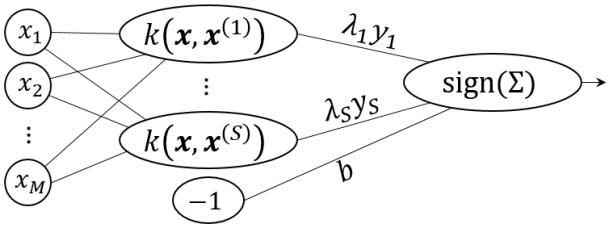


Fig. 5: The *SVM* classifier (1) shown as a neural network with one hidden layer.

layer (the number of support vectors). We can determine the kernel based on our understanding of the problem, namely by suggesting a rational measure of closeness. Moreover, we can control the trade-off between the generalization and the accuracy via the adjustment of parameter C and we can interpret the result by considering the points selected to be the support vectors. In such a way, the *SVM* appears a useful methodology for structuring topology, choosing activation functions (in the hidden layer) and interpreting the mechanism of making decision by this type of neural networks. Note that we here do not need to train the network, instead we compute the only solution via the optimization problem solution.

We can now summarize the strong sides of *SVM*s:

- The problem is well-posed and always has the only solution that can be found.
- The non-linearity is explicitly disentangled from the solver itself via the use of kernels.
- The solution can be interpreted and explained in terms of the chosen set of support vectors.
- We can vary the kernel and the regularization to control the properties of the classifier.

To balance this discussion we should outline some known weak sides of *SVM*s:

- Sensitive to noise (the noise in the labels and/or x locations can dramatically affect the location of the decision hyperplane; in practice clean data or filtering preprocessing is needed).
- There is no clear theory of how to build the optimal kernel (long-standing unsolved problem), the trial and error method is use in practice.
- The method selects support vectors but says nothing about the important features.

The materials presented in this part concern the core principles of *SVM*. We briefly outline some of the most notable generalizations. The *SVM* approach can be generalized to deal with regression. This can be achieved by choosing a specific loss function $(|\varepsilon| - \delta)_+$, where ε is the error and δ quantifies the margin thickness. This loss gives penalty when the predicted value deviates from the correct value for more than the margin half thickness δ but gives no penalty for the predictions within the margin. A similar analysis leads to the convex minimization problem and the only solution can be found by specialized methods. This method is referred to as *Relevance Vector Machines* (*RVM*) (see [54]). Another direction of improvements is the choice of regularization function. Instead of L2 regularization one can apply L1 [55] or their sum [56] to enable feature selection and other useful properties.

CHAPTER

TWELVE

UNSUPERVISED LEARNING

Recommended literature

- M. Bishop, *Pattern Recognition and Machine Learning* (2006); Chapter 12
- Carl Doersch, *Tutorial on Variational Autoencoders*, arXiv:1606.05908
- Diederik P. Kingma and Max Welling, *An Introduction to Variational Autoencoders*, arXiv:1906.02691

12.1 Introduction

So far the course primarily concerned supervised learning being one of three main *ML* branches, along with unsupervised learning and reinforcement learning. These branches are different in the formulation of the problem statement Supervised learning commonly aims at finding (fitting with respect to given data) models to map a high-dimensional input into a low-dimensional output (e.g., pictures into labels). Unsupervised learning concerns finding hidden structures of the data, which practically means finding low-dimensional representation of high-dimensional data points. Finally, reinforcement learning aims at finding useful strategies of interacting with complex systems. In Fig. 1 the problem statements, indicative examples and key problems of each branch are summarized.

Note that these branches are highly interdependent: the progress in each of them can be dramatically boosted by achievements in the others. In this part of the course we will try to grasp the essence of some key approaches in unsupervised learning, namely:

- Principal Component Analysis (*PCA*)
- Variational AutoEncoders (*VAE*s)

Our aim is not learning them in detail, but rather building up the understanding of core ideas.

12.2 Motivation

Before moving to the discussion of unsupervised learning models, let us focus on one of the central stumbling blocks of supervised learning: the capability to generalize. More parameters make ML models more flexible, but the price we pay is that these parameters can be used to "memorize" rather than "understand" the data. Although regularization and the principle of holdout test set alleviate the problem, indicative examples of "One Pixel Attack" [57] (see also [58]) show that *ML* models exploit superficial clues, which makes them unreliable in real-world applications. As a result, *ML* models can fail on examples that are rare or not present in the training set. This is also known as the problem of invariance, i.e., the problem of making *ML* models invariant with respect to the changes between training (limited in unknown way) and real-world data. One way to overcome this is to invoke some notions of common sense, but the formalization of these is intricate.

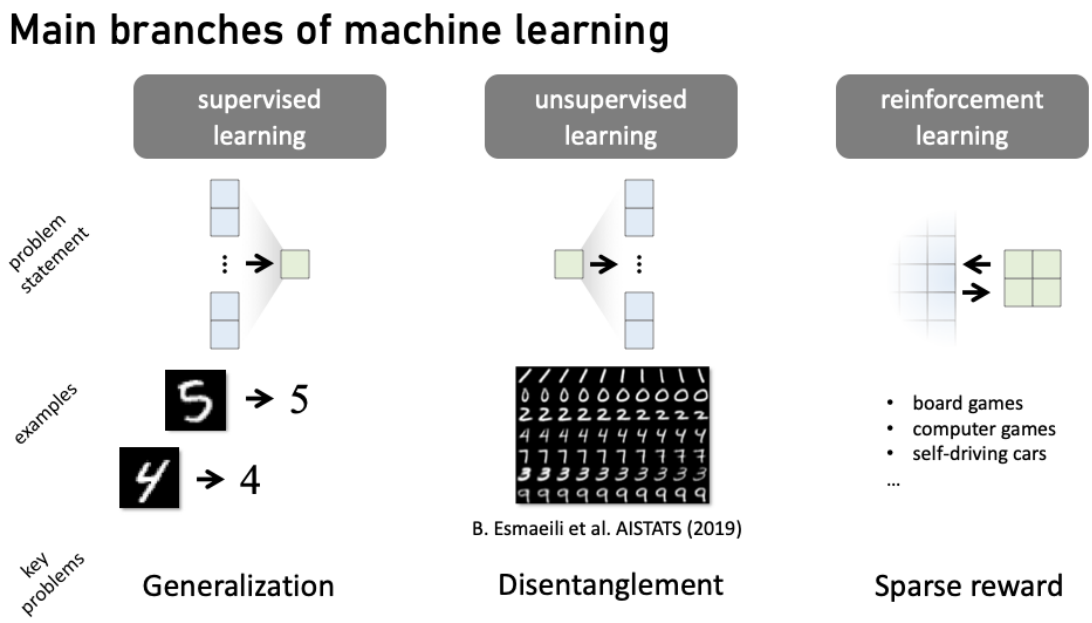


Fig. 1: Main branches of machine learning.

Unsupervised learning provides a useful framework for this problem. Consider images of digits in the *MNIST* dataset. Apart from the digit itself, the pictures are different in style, inclination, stroke width, etc. These characteristics can span the space of possibilities and can be seen as basic notions for understanding rather than memorizing images. The possibility to vary characteristics independently (although they might be correlated) is known as *marginal independence*. The problem of finding such low-dimensional representations is known as the problem of *disentanglement* and is considered one of the milestones on the way to reliable *ML* models (like the one needed for self-driving cars).

Given the problem statement of unsupervised learning, one can imagine a system that can learn how to convert low-dimensional vectors into high-dimensional data points, both given and new ones. Why would one want to do this? There are two main ways to use such a system. The first way is to generate *synthetic* samples to enlarge the training sets. This is known as *data augmentation*. The second way exploits the reverse function of mapping high-dimensional samples into low-dimensional representation (also known as *latent* representation) that is sufficient to "encode" the hidden structure of all given samples. This function is naturally built in many approaches and provide a way to reduce the dimensionality of the input vectors in the supervised learning: instead of original vectors one can assign *ML* model to learn vectors of latent representation. We start from the discussion of linear approach to the dimensionality reduction problem, which is known as Principal Component Analysis (*PCA*).

12.3 Principal Component Analysis

Principal Component Analysis (*PCA*) is a basic method for making linear conversion of data samples into a low-dimensional representation that is optimal in terms of L2 metric for a given set of samples. Although the *PCA* routine doesn't depend on labels at all, it provides a way to reduce the dimensionality of input vectors, which makes possible using *ML* models with smaller number of parameters, meaning that such ML models can be trained by smaller training sets. In other words *PCA* provides a basic approach to controlling the balance between the amount of data and the model complexity, the balance that is crucially restricted by the problem of overfitting. That is why *PCA* is considered as a "must-know" subject for *ML* practitioners.

12.3.1 Recap: representations in the regression problem

A good starting point for the discussion of *PCA* is the problem of linear regression. Consider the task of finding a function $\mathbb{R}^m \to \mathbb{R}$ that approximates the mapping defined by N pairs (x_i, y_i) forming the training set T (i.e., $x_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}$, $i = 1, \ldots, N$). We assume that the pairs of the training set are randomly chosen from some given data set and the remaining pairs (x_i', y_i') form test set T'. For the *ML* model to be trained, we consider $f(x, \alpha) \in \mathbb{R}$ that is a function (can be nonlinear) of $x \in \mathbb{R}^m$ and a parameter vector $\alpha \in \mathbb{R}^M$ (we assume $M \leq N$). We can formulate the problem by setting some loss function $L(\alpha, T)$ as a metric for comparing alternative choices of α. For L2 metric we have:

$$\alpha^* = \arg\min_{\alpha} L(\alpha, T) = \arg\min_{\alpha} \sum_{i=1}^{N} \left(f\left(x_i, \alpha\right) - y_i\right)^2,$$

where we use residual sum of squares (*RSS*) as the loss.

It is instructive to note that the use of *RSS* makes the problem equivalent to maximum likelihood estimator with respect to the pairs (x_i, y_i) being the results of measurements with normally distributed errors:

$$\begin{aligned}
&y(x_i) = f(x_i, \alpha) + \varepsilon_i; \quad \varepsilon_i \sim N(0, \sigma_i) \\
&\alpha^{**} = \arg\max_{\alpha} L\left(\varepsilon_1, \ldots, \varepsilon_N \mid \alpha\right) = \arg\max_{\alpha} \prod_{i=1}^{N} \frac{1}{\sigma_i^2 \sqrt{2\pi}} \exp\left(-\frac{\varepsilon_i}{2\sigma_i^2}\right) \\
&\alpha^{**} = \arg\min_{\alpha} \left(-\ln\left(L\right)\right) = \arg\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left(f\left(x_i, \alpha\right) - y_i\right)^2 \\
&\alpha^{**} = \alpha^* \text{ with } \sigma_i^{-2} = 1.
\end{aligned}$$

Let us now restrict the choice of f to functions that are linear with respect to α, i.e.,

$$f(x, \alpha) = \sum_{i=1}^{M} \alpha_i \phi_i(x) = \alpha \phi(x),$$

where $\phi_i(x)$ are functions (can be nonlinear) that have no parameters to be adjusted (the rightmost expression stands for the scalar product of vectors α and $\phi(x)$). The problem has the following explicit solution:

$$\alpha^* = \arg\min_{\alpha} \sum_{i=1}^{N} \left(\alpha \phi(x_i) - y_i\right)^2 \arg\min_{\alpha} \left\|\Phi\alpha - y\right\|^2, \tag{1}$$

$$\nabla_\alpha L = 2\Phi^T \left(\Phi\alpha - y\right) = 0 \quad \Rightarrow \quad \alpha^* = \underbrace{\left(\Phi^T\Phi\right)^{-1}\Phi^T}_{\Phi^+} y, \tag{2}$$

where $y \in \mathbb{R}^N$ is the vector of labels, Φ stands for the design matrix (of size $N \times M$) and in the last expression we introduced Φ^+ that stands for the Moore-Penrose inverse of Φ (to understand the reason for this naming consider $M = N$). To interpret the outlined solution we define projection matrix

$$P = \Phi\Phi^+ \tag{3}$$

so that $L = \left\|Py - y\right\|^2$. This form of the loss function indicates that the solution implies the approximation of $y \in \mathbb{R}^N$ by a linear combination of $M \leq N$ columns of matrix Φ and this approximation is given by Py. Note that Py belongs to the linear span of these columns and thus lays on an M dimensional subspace of $\mathbb{R}^N$. That is why the solution (2) provides the projection of y on the subspace spanned by columns of the design matrix.

We can now notice that the choice of functions $\phi_i(\cdot)$ provides a way to embed elements of our knowledge or understanding into the *ML* model. In essence we get an opportunity to suggest those nonlinear functions that give essential characteristics that should span the space of all possibilities in the space of x. It is commonly claimed that an educated choice of these functions makes linear *ML* models sufficient in many practical tasks within supervised learning.

This observation leads to a natural idea of finding some simple but yet practical choice of functions $\phi_i(\cdot)$. In the next section we consider *PCA*, which provides such a practical choice.

12.3.2 Use of Singular Value Decomposition for regularization

Let us start from applying the reduced Singular Value Decomposition (*SVD*), referred to as *thin* SVD, which enables the following possibility. Matrix Φ (as any other matrix) can be represented in the form

$$\Phi = VDU^T, \tag{4}$$

where $V = (v_1, \dots, v_M)$ is the $N \times M$ orthonormal matrix (i.e., $V^T V = I_M$) composed of the $\Phi\Phi^T$ eigenvectors $v_i \in \mathbb{R}^N$ (used as columns) that correspond to non-zero eigenvalues; $U = (u_1, \dots, u_M)$ is the $M \times M$ orthonormal matrix (i.e., $U^T U = I_M$) composed of the $\Phi^T\Phi$ eigenvectors $u_i \in \mathbb{R}^M$ used as columns; $D = \text{diag}\left(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_M}\right)$ is the $M \times M$ diagonal matrix with λ_i being the positive eigenvalues of $\Phi\Phi^T$ and $\Phi^T\Phi$.

Let us substitute such a decomposition into the definition of Φ^+,

$$\Phi^+ = \left(\Phi^T\Phi\right)^{-1}\Phi^T = \left(UD^TV^TVDU^T\right)^{-1}UD^TV^T.$$

Using $V^TV = I_M$, $D^T = D$, $U^TU = I_M$, we obtain:

$$\Phi^+ = \left(UDDU^T\right)^{-1}UD^TV^T = UD^{-1}V^T = \sum_{i=1}^{M}\frac{1}{\sqrt{\lambda_i}}u_i v_i^T,$$

where we exploit the fact that $D^{-1} = \text{diag}\left(\lambda_1^{-1/2}, \dots, \lambda_M^{-1/2}\right)$ and use the representation of matrices U and V^T in terms of columns u_i and rows v_i^T, respectively. With help of the obtained expression we can represent the sought-for parameter vector α^* and its norm in the following form (see (2)):

$$\begin{aligned}\alpha^* &= \Phi^+ y = \sum_{i=1}^{M}\frac{1}{\sqrt{\lambda_i}}u_i\left(v_i^T y\right);\\ \left\|\alpha^*\right\|^2 &= \sum_{i=1}^{M}\frac{1}{\lambda_i}\left(v_i^T y\right)^2.\end{aligned} \tag{5}$$

Note that $\left(v_i^T y\right)$ are scalars and thus α^* is represented by a linear combination of u_i. In a similar way we express projection Py (see (3)):

$$Py = \Phi\alpha^* = VDU^TUD^{-1}V^Ty = VV^Ty = \sum_{i=1}^{M}v_i\left(v_i^T y\right). \tag{6}$$

The obtained expressions allow us to make some useful observations. From (5) one can see that the solution and thus our model can become divergent if one or several eigenvalues tend to zero ($\lambda_i \to 0$). This can happen if two (or several) columns of the design matrix (i.e., one or several features) appear to be almost or exactly collinear (i.e., linearly dependent). Large absolute values of α^* components make the *ML* model unstable, because small variations of labels y (e.g., due to measurement errors) are greatly enhanced by the divergent components of α^*. It is instructive to note that the divergence does not show up for the samples within the training set because the prediction of our model for them (6) does not include inverse dependency on the small eigenvalues. This is how the overfitting manifests itself: the model works well for the elements of the training set but becomes divergent for new predictions.

The emergence of collinearity of features (columns of Φ) can be related to the inclusion of too many features (components of the input vector x). In the most radical case of $M > N$ this makes the problem under-determined, i.e., suitable options of α appear to be not restricted. That is why one can relate the problem of overfitting with the problem of under-restricting the choice of α. In such a way we arrive to the idea of regularization, i.e., introducing a penalty for large components of α into the loss function. Another, straightforward alternative is to reduce the number of features by selecting or determining new features that are not linearly dependent. We start from the analysis of the first alternative.

Let us consider the idea of introducing the L2 penalty into the loss function:

$$L = \left\|\Phi\alpha - y\right\|^2 + \tau\left\|\alpha\right\|^2,$$

where τ is the coefficient of L2 regularization. In this case the solution can be also found in an explicit form:

$$\alpha^* = \left(\Phi^T\Phi + \tau I_n\right)^{-1}\Phi^T y. \tag{7}$$

Since τ controls the strength of the L2 regularization term, we can consider it as a parameter to be optimized via the following procedure. We vary τ and for each value of τ we compute α^* via (7) and determine the performance of the resultant model on the test set. Next we determine the value of τ that gave the best performance. Nevertheless, the expression (7) implies the computation of inverse matrix, which can be rather demanding procedure. Let us see how we can make use of the *SVD* for reducing the complexity of this procedure.

We again substitute the *SVD* of Φ into the expression for α^*, which now has the form given by (7):

$$\begin{aligned}\alpha^* &= (UDV^TVDU^T + \tau \overbrace{\hat{I}}^{=UU^{-1}})^{-1}UDV^Ty \\ &= \left(U\left(D^2+\tau I_M\right)U^{-1}\right)^{-1}UDV^Ty = U\left(D^2+\tau I_M\right)^{-1}DV^Ty \\ &= \sum_{i=1}^{M}\frac{\sqrt{\lambda_i}}{\lambda_i+\tau}u_i\left(v_i^Ty\right).\end{aligned}$$

In a similar way we compute the projection:

$$\begin{aligned}Py = \Phi\alpha^* &= VDU^TU\left(D^2+\tau I_M\right)^{-1}DV^Ty \\ &= V\,\mathrm{diag}\left(\frac{\lambda_i}{\lambda_i+\tau}\right)V^Ty = \sum_{i=1}^{M}\frac{\lambda_i}{\lambda_i+\tau}v_i\left(v_i^Ty\right).\end{aligned}$$

We can see that once the *SVD* is computed, we can obtain solutions for various values of τ without repeated matrix inversion. The performance on the test set can be computed in the following way. Suppose within the test set T' we have K pairs (x'_i, y'_i). We introduce matrix Φ' of size $K \times M$ and vector $y' \in \mathbb{R}^K$:

$$\Phi' = \begin{pmatrix}\phi_1(x'_1) & \ldots & \phi_M(x'_1)\\ \ldots & \ldots & \ldots \\ \phi_1(x'_K) & \ldots & \phi_M(x'_K)\end{pmatrix}, \quad y' = \begin{pmatrix}y'_1\\ \ldots \\ y'_K\end{pmatrix}.$$

Using these we compute the loss on the test set as a function of τ:

$$L'(\tau) = \left\|\Phi'\alpha^* - y'\right\|^2 = \| \underbrace{\Phi'U}_{K\times M}\underbrace{\left(D^2+\tau I_M\right)^{-1}D}_{\mathrm{diag}(\sqrt{\lambda_i}/(\lambda_i+\tau))}\underbrace{V^Ty}_{M\times 1} - y'\|^2,$$

where we explicitly indicate the size of the matrices entering the expression. As one can see the *SVD* provides a way to significantly reduce the amount of computations for each value of τ by replacing $M \times M$ matrix inversion with multiplication of pre-computed $K \times M$ matrix $\Phi'U$ by a vector with coefficients varied with τ.

Note that obtained expression allow us to interpret the role of τ. The appearance of τ in the denominator prevents the singularity caused by $\lambda_i \to 0$. Nevertheless this happens at the cost of deviation from the exact optimum for the test set. The balance between these two unwanted factors determines the optimum for τ.

Let us now consider briefly the use of L1 regularization:

$$L(\alpha) = \|\Phi\alpha - y\|^2 + \xi\|\alpha\|.$$

An important difference from the case of L2 regularization is that the penalty grows linearly from zero and thus the gradient doesn't decay at zero. While in case of L2 regularization reaching sufficiently small α components makes their further decrease inessential, L1 regularization implies benefits from decreasing them right to zeros. With increase of ξ we can expect more and more components of α to become exactly zero. (One can see this by considering the *KKT* conditions for variables $\alpha_i^+ \geq 0$, $\alpha_i^- \geq 0$ so that $\alpha_i = \alpha_i^+ - \alpha_i^-$, and $|\alpha_i| = \alpha_i^+ + \alpha_i^-$.)

A good demonstration of this observation from [59] is shown in Fig. 2. One can see that in the limit of weak regularization (the left most points of the horizontal axes) the solutions are the same (as it should be because in this limit the problems

turn into identical form (1). In the limit of strong regularization (the left most points) both solutions yield zero α^* vectors because the penalty for deviation is higher than the reward for learning the dependency. When departing from this point to the right, L2 regularization causes immediate and simultaneous activation (i.e., deviation from zero) of all α^* components, whereas L1 activates them one-by-one. Note that zero α^* components mean that corresponding features are not taken into account. This is known as the effect of feature selection provided by L1 (*LASSO*) regularizarion.

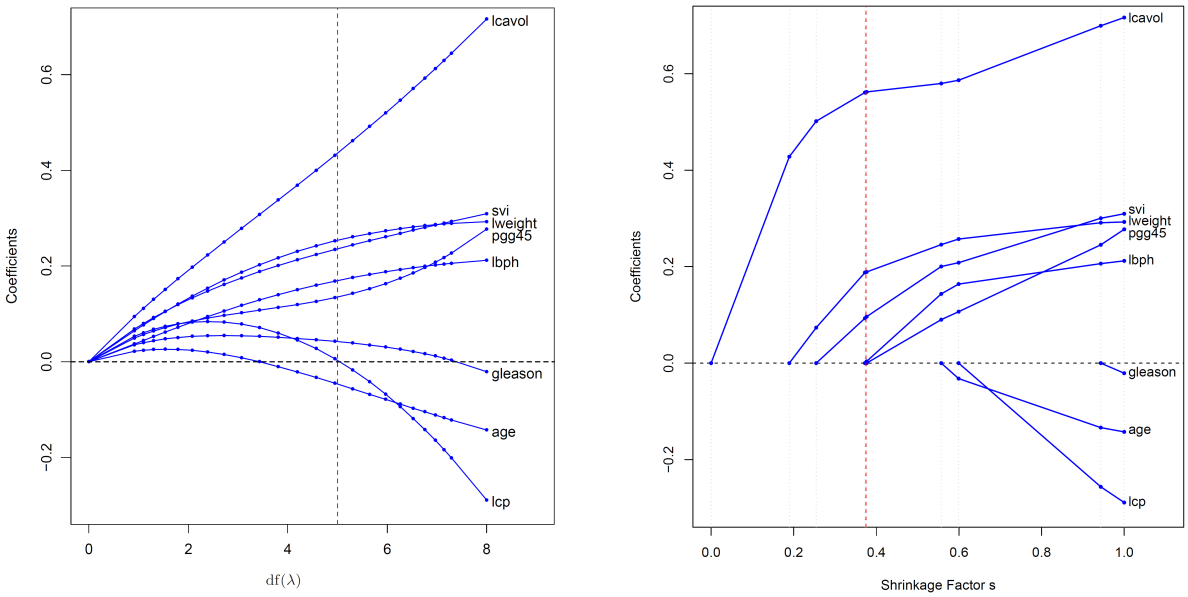


Fig. 2: An example that demonstrates the effect of L2 (left) and L1 (right) regularization. The coefficients of the model parameters are shown as functions of the regularization strength which decreases from left to right. Reproduced from Hastie *et al.* (2009).

12.3.3 Dimensionality reduction with PCA

In the previous section we saw that L1 regularization can effectively select features. A possible interpretation is that L1 regularization neglects the features that are least important. This picture makes it possible to see an interesting alternative: instead of just selecting useful features we can define a smaller number of new, even more useful (than existing) features being expressed by some functions of the original features. However, appealing to the usefulness makes the problem formulation difficult. A useful trick is to require something that is more straightforward for analysis. Let us require that the new features capture the essential information so that the original features can be restored to a highest possible accuracy. The key element here is that the number of new features is smaller than the number of original features. This is known as the problem of dimensionality reduction, the corner-stone problem of unsupervised learning. Let us highlight that this problem formulation doesn't concern labels at all, this is indicated by the term *unsupervised*.

A particularly useful solution can be obtained within the class of linear transformations. To formulate the problem we denote new features by $g_1(x), \dots, g_K(x)$ and we want their number to be much smaller than the number of original features, i.e., $K \ll M$. At the same time we want to be able to reconstruct the original features to a highest possible accuracy. Using L2 metric as a measure of accuracy we can write:

$$\hat{\phi}_j(x) = \sum_{k=1}^{K} g_k(x) u_{jk}, \quad j = 1, \dots, M$$

$$L = \sum_{i=1}^{N} \sum_{j=1}^{K} \left(\hat{\phi}_j(x_i) - \phi_j(x_i) \right)^2 \to \min,$$

Although $g(x)$ can be arbitrary (nonlinear) functions, we are only concerned about their value at x_i and thus they can be

treated as sets of numbers. It is convenient to arrange all the unknowns in matrices:

$$G = \begin{pmatrix} g_1(x_1) & ... & g_K(x_1) \\ ... & ... & ... \\ g_1(x_N) & ... & g_K(x_N) \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & ... & u_{1K} \\ ... & ... & ... \\ u_{M1} & ... & u_{MK} \end{pmatrix}.$$

Here we redefined matrix U, but as we will see further this definition will be consistent with the previous for a particular case. The problem statement takes the form:

$$(G, U) = \arg\min_{G,U} \left\| GU^T - \Phi \right\|^2.$$

This is known as the problem of low-rank matrix approximation (for Frobenius norm) and the solution is given by the Eckart-Young-Mirsky theorem, which states the following. For any $K \leq M = \text{rank}\,(\Phi)$ the minimum of $\left\| GU^T - \Phi \right\|^2$ is achieved by $U = (u_1, ..., u_K)$ composed by the columns $u_1, ..., u_K$ being the eigenvectors of $\Phi^T\Phi$ that correspond to K largest eigenvalues $\lambda_1, ..., \lambda_K$ and:

$$G = \Phi U \text{ is orthogonal, i.e., } G^T G = \Lambda = \text{diag}(\lambda_1, ..., \lambda_K), \tag{8}$$

$$U \text{ is orthonormal, i.e., } U^T U = I_K, \tag{9}$$

$$U\Lambda = \Phi^T\Phi U \text{ and } G\Lambda = \Phi\Phi^T G, \tag{10}$$

$$\left\| GU^T - \Phi \right\|^2 = \left\| \Phi \right\|^2 - \text{tr}(\Lambda) = \sum_{i=K+1}^{M} \lambda_i. \tag{11}$$

This is the central theorem for *PCA*. The term *principal components* in *PCA* stands for the idea of taking components that provide the best possible approximation of the design matrix according to Frobenius norm.

As one can see the statement of the theorem has a lot in common with the *SVD*. To make clear the connection let us consider $K = M$. In this case the theorem states that we get zero error, i.e., $\left\| GU^T - \Phi \right\|^2 = 0$ (see Eq. (11)). This means that $\Phi = GU^T$ exactly. If we now compare this expression (with redefined U) with the *SVD* for Φ (see Eq. (4)), we can notice that they look similar and include matrices U defined in an identical way (being composed of eigenvectors of $\Phi^T\Phi$). The expression are identical if $G = VD$. Given that the *SVD* does exist, let us consider if taking G to be equal to VD from (4) would be consistent with the statements of the theorem. Substituting $G = VD$ into Eq. (8) yields $G^T G = D^T V^T V D = D^T D = \text{diag}(\lambda_1, ..., \lambda_K) = \Lambda$. Equation (9) is also provided by the choice of U from the *SVD* of Φ. Finally, Eq. (10) states nothing else then the fact that u_i are the eigenvectors of $\Phi^T\Phi$ that correspond to $\lambda_i > 0$ for $i = 1, ..., K$. As we see, for $K = M$ the approximation provided by the Eckart-Young-Mirsky theorem converges to the *SVD*. In other words, this theorem generalizes the *SVD* to the case of having not perfect (for $K = M$), but the best possible approximation within any given dimensionality $K < M$. One can also say that both decompositions provide a way to compress data, but *SVD* concerns lossless compression, while the Eckart-Young-Mirsky theorem ensures the least possible corruption according to the Frobenius norm.

The theorem provides a way to transform the design matrix to a mew design matrix that has any given number of features K (the dimensionality of the input vector):

$$G = \Phi U. \tag{12}$$

(The inverse transformation $\hat{\Phi} = GU^T$ is exact only if $K = M$.) Note that the transformation (12) makes our new design matrix orthogonal ($G^T G = \Lambda$ from Eq. (8)). This means that the new features are uncorrelated. Looking at the solutions (2) and (7) we can see that this is very fortunate because the needed inversion of matrix becomes trivial.

The next question worth discussion is how to chose K in practice. We can start from finding eigenvalues λ_i of $\Phi^T\Phi$. As we saw, the most problematic issue is the presence of small eigenvalues, which should be compared to the largest eigenvalues. If we order the eigenvalues $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_M \geq 0$ we can explicitly decide on the trade-off between accuracy of the approximation (12) and the stability of solution (absence of overfitting). Essentially we can decide how many smallest eigenvalues to toss aside to make the model more stable at the cost of neglecting a part of information. To make this choice more apparent we can plot ordered eigenvalues λ_i as a function of index i; this is also referred to as the

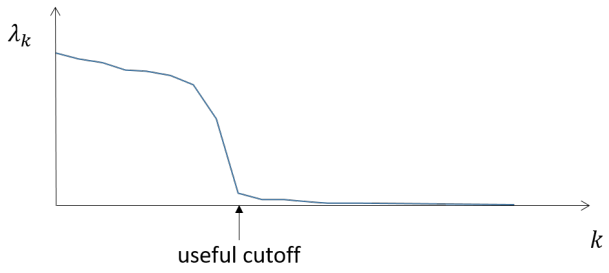


Fig. 3: Possible spectrum of $\Phi^T\Phi$ that indicates a prominent cutoff in the dependency of ordered eigenvalues on the index.

spectrum of matrix. One possible (and fortunate) situation, shown in Fig. 3, is the presence of a prominent and sharp drop of values at certain index, which can be considered as a good option for K. However, unfortunately for many real problems the spectrum doesn't show any prominent option. In this case one can try and compare various choices.

Let us now observe the ideas behind the considered methods of combating the problem of overfitting. In case of L2 regularization, we try to avoid divergence caused by small eigenvalues by adding some positive number to the denominators. Essentially we "detach" lambdas from zero making the model more stable at the cost of becoming less accurate. In case of *PCA*, we toss those problematic small lambdas aside.

We conclude this section by discussing the geometrical meaning of *PCA*, which gives a perspective on the ways to generalize this approach. We already noticed that *PCA* provides a presentation that has uncorrelated features. A good way to see this is to consider a set of points on a 2D plane shown in Fig. 4a. The eigenvectors scaled proportionally to corresponding eigenvalues are shown in Fig. 4b. We see that the principal component (longest vector) denotes the direction over which the data is varied to a greatest extent, i.e., the projection on this vector provides the most accurate approximation of each point. This approximation (with the only principal component) is shown in Fig. 4d. Finally, the transformed representation, shown in Fig. 4c, demonstrates that the new features are indeed uncorrelated.

The presented case can be interpreted in terms of physics. The problem of finding the line that has the minimal sum of squared distances from each point in a given set is essentially equivalent to the problem of finding the lowest energy of the mechanical system shown in Fig. 5a. From this point of view we can imagine several ways to generalize the approach of *PCA*. For examples, Fig. 5b shows the concept of clustering, i.e., representing the data by attributing all points to a small number of centers. One can design a more sophisticating systems using positive and negative springs (for regularization, or flattening the shapes), as well as by adding rules for splitting branches into several manifolds in case of reaching critical tension during gradient adaptations. It is interesting to note that in such problems, we can try to design useful rules appealing to simple intuition and then make use of classical mechanics to find the solution.

12.4 Variational Autoencoders

12.4.1 Autoencoders

Before the discussion of variational autoencoders (*VAE*s) we need to introduce the concept of autoencoders. An autoencoder is a composition of two stacked *ML* models that are assigned to reproduce the input at the output. The first model, called *encoder*, converts input vector x into vector z being the input for the second model called *decoder*. The decoder converts z into the output x' that is "pulled" towards x, i.e., the loss function is the distance (of some form) between x and x'. If training converges to an acceptable reconstruction (for a given set of x vectors) while using z of lower dimensionality than x, we can see this process as finding the latent representation that conveys (via z) the most essential information according to the chosen metric of distance. Note, that if both encoder and decoder are linear models and we use Euclidian norm for the distance between x and x' the problem formulation turns into that of linear *PCA*. Nevertheless, we see that nothing prevents us from considering nonlinear *ML* models for both encoder and decoder. A

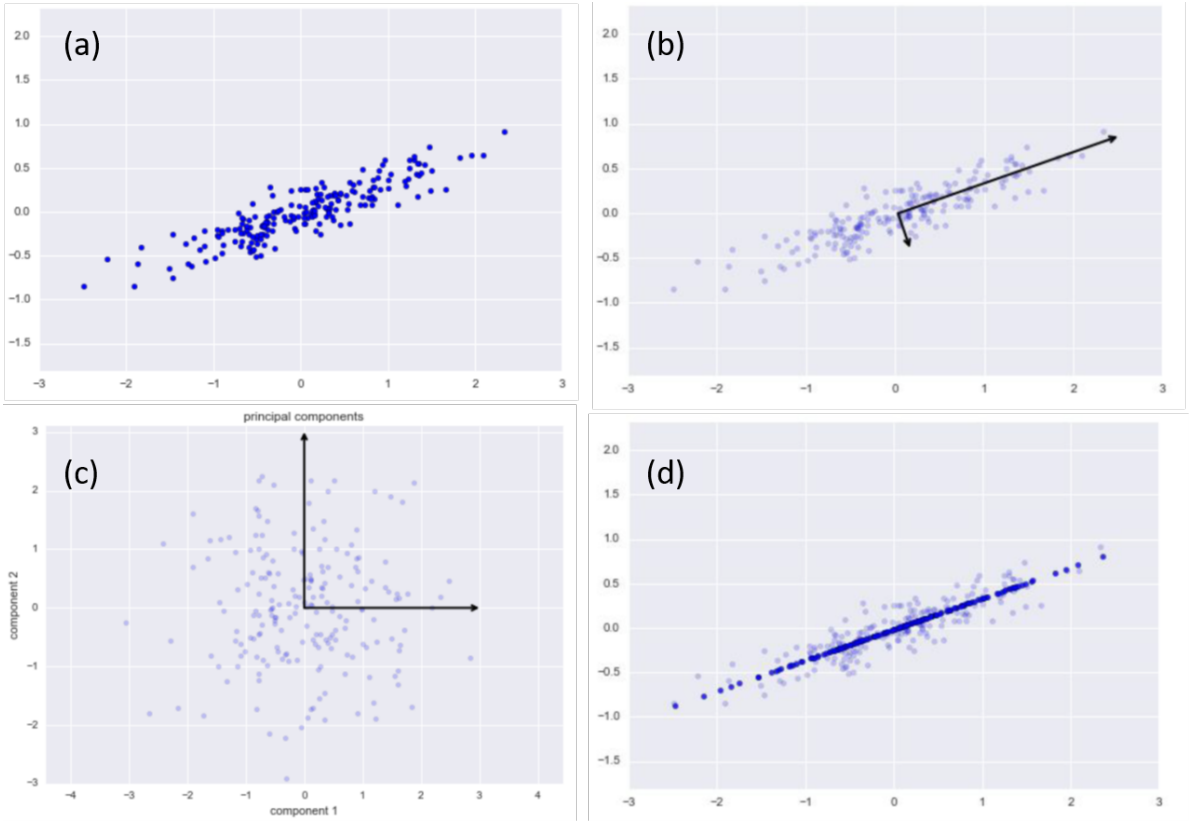


Fig. 4: Geometrical meaning of *PCA*. Reproduced from VanderPlas (2016).

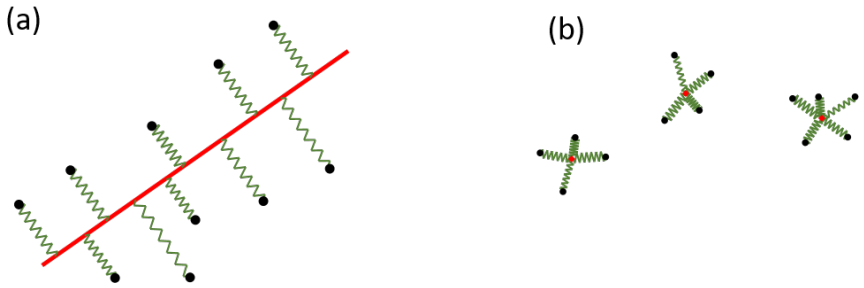


Fig. 5: Interpretation of ideas behind single component *PCA* (a) and clustering (b) in terms of mechanical systems.

particularly interesting results are achieved by using multi-layer (deep) neural networks (*NN*s). Although the discussion can be abstracted from this choice, hereafter we will assume that both encoder and decoder are *NN*s (see Fig. 6).

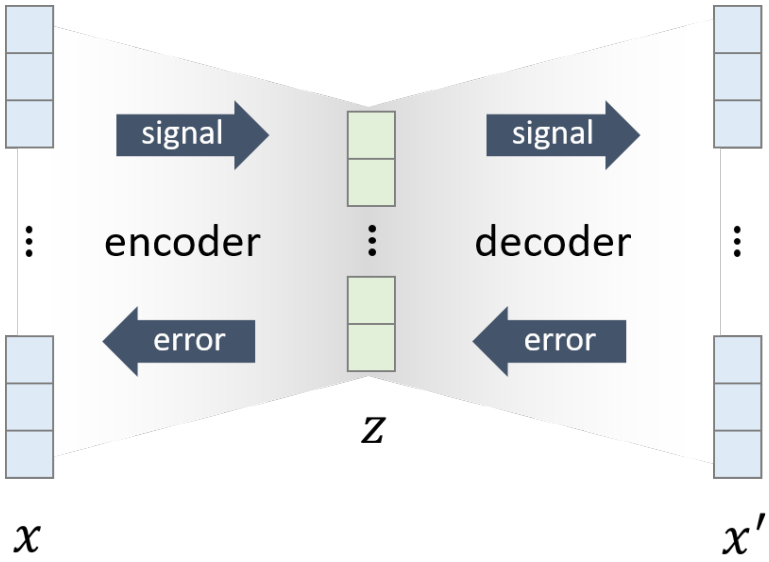


Fig. 6: Schematic representation of the autoencoder concept.

Note that the training process applies to the entire composition, i.e., the errors first back-propagate across the decoder from x' to the latent layer z and then further across the encoder to the input. In such a way we avoid the necessity to decide on the content of vector z, while effectively forcing finding most useful way of information "compression" by setting the dimensionality of z being lower than that of x. That is why autoencoders provide a flexible (and nonlinear!) framework for finding the functions that convert high-dimensional samples to low-dimensional latent representations. The dimensionality of vector z at the junction (also called the "bottleneck") can be varied to identify that the population of high-dimensional samples effectively lie on a low-dimensional manifold spanned by z via the nonlinear functions of decoder. The meaning of the term *effectively* here is defined by the chosen form of distance in the space of x. It is also possible to reduce the dimensionality of the junction sequentially, e.g., after reaching satisfactory reconstruction one can add one or few layers at the output of encoder and input of decoder to "tighten" the junction further. Alternatively, one can determine the optimal dimensionality of z with help of various methods, known as *intrinsic dimension estimators*, that are ought to indicate the dimensionality of the manifold, which the samples effectively lie on (see, e.g., [60]).

It is instructive to accentuate one more point. From the first glance, one may think that the problem is ill-posed because the encoder can potentially learn to convert the x into a single number that enumerates all the samples in the population (given set of x vectors), while the decoder can "memorize" all these samples and learn to "show" each by its number. Although this scenario is rather hypothetical, it indicates that autoencoders can also be subject to the problem of overfitting and thus one should use holdout sets for testing. Using holdout sets excludes the possibility for the outlined hypothetical scenario and enforce the autoencoder to search for a solution, which appears to be practical in many cases.

To conclude this brief introduction to the idea of autoencoders let us outline their use. As was already mentioned, they can be used to simplify the task of supervised learning by reducing the dimensionality of the input vector. In this case, just as with *PCA*, the pairs (x, y) (feature vectors and label/target vectors) are converted into pairs (z, y) that become subject of supervised learning. The reduced size of the input vector makes it possible to reduce the number of parameters in the *ML* model and this consequentially facilitates reaching higher accuracy for the given size of the training set. A notable difference from *PCA* is that autoencoders are not limited to linear functions and thus can be more capable in this task. However, there are other, rather creative ways of exploiting autoencoders. One can train autoencoders to remove noise from pictures. For this one takes original (clean) pictures, superimpose some level of random noise and trains an autoencoder to reconstruct the original pictures (not the noisy ones). In this case we are not interested in the latent representation, but we do expect that its reduced dimensionality favors the selection of essential information, while the information related to the noise is filtered out by the bottleneck. In the same way, one can train autoencoders to complete

missing elements on given images, remove watermarks or increase image resolution (image up-sampling).

One may think that autoencoders can also be used for the data augmentation, since we can vary the latent vector state and generate outputs by the decoder. In addition, this seemingly should provide some clue about the meaning of the z vector components. However the training process of autoencoders doesn't imply that we restrict them to generate possible vectors in the population of x vectors for *any* values of z; the training concerns exclusively the z states that are generated by encoders from the given training set. Certainly, the continuity of *ML* models makes this possible in some close vicinity of such z states, but in practice this is hardly useful. The idea of generating probable states of x and understanding the latent space leads us to the concept of variational autoencoders that we discuss in the following section.

12.4.2 The concept of variational autoencoders

The design of variational autoencoders can be seen as a modification of autoencoders that makes them capable of generating probable samples out of various z, not only the ones obtained by the encoder from the given training set. We will first arrive to this modification from the perspective of variational Bayesian optimization and then we will discuss some geometrical insights into the reasons of why and how this modification works.

Let us for the moment forget about autoencoders and focus on the problem we want to solve. We have a population of vectors x_i in a high-dimensional space $X = \mathbb{R}^M$ and want to find their low-dimensional representation, i.e., a nonlinear function $x = f(z)$ that goes through all x_i under continuous variation of its parameter $z \in Z = \mathbb{R}^K$ ($K \ll M$). We can imagine replacing $f(z)$ by a neural network and adjusting its weights. The use of holdout test set invalidates the strategy of simple enumeration of given samples x_i and thus forces the neural network to find some unification, so that the variation of z can be used to generate other sensible samples. We can again think of the generation of images of digits. In this case we want Z to span the space of various characteristics, such as the digit itself, inclination, stroke thickness etc., while $f(z)$ should generate various options completing the population x_i in a sensible way.

Let us now change the gears from deterministic to probabilistic notions and reformulate the problem in the following way. We want to be able to sample z from some simple distribution in low-dimensional space and transform it into space X by a deterministic function so that x is "sensible" with respect to given population x_i. From the above mentioned argument, we can expect that the notion "sensible" can be achieved by requiring that x_i are probable, i.e., are generated under certain probable choice of z_i. Without loss of generality we can assume that $z \sim P(z)$ is sampled from normal distribution $z \sim N(0, I)$. Since z is a random variable so is $x = f(z)$ (although $f(\cdot)$ is deterministic). We can now formulate our problem in the following way. We want to maximize likelihood of getting our given x_i from the sampling process that we described:

$$P(x) = \int P(x \mid z)\, P(z) dz. \tag{13}$$

If $f(z)$ is a deterministic function then $P(x \mid z)$ is given by shifted Dirac delta function. Nevertheless, for the reason explained below, we keep this general notation that works also for non-deterministic functions $f(z)$.

Looking at the problem formulation we can imagine a straightforward way of adjusting the parameters of $f(z)$: (1) take a random sample x_i out of the given training set; (2) generate some number of samples $z_k \sim N(0, I)$; (3) computer $x_k = f(z_k)$ and (4) modify the parameters of $f(z)$ so that the number of cases when $x_k = x_i$ is increased. Then we can repeat these steps for other x_i. We can immediately see difficulties with this naive plan. Accounting for only perfect matches ($x_k = x_i$) doesn't allow us to use the gradient adjustment. Moreover, for the matches to occur we need x components being discrete, and even in this case the chance of having such perfect matches becomes increasingly small with increase of dimensionality of X. What can we do? Just as in Approximate Bayesian Computations, a useful idea is to learn from imperfect matches by introducing some sort of kernel. The kernel can work here as a loss function: if x_k is close to x_i the loss should grow with increase of the difference so that the gradient can indicate the direction for parameter adjustment that makes x_k closer to x_i. Can the loss grow quadratically with the distance? Not always, because we would want to ignore the cases when x_k is far from x_i, assuming that they might correspond to another sample from the training set (in other words each x_i should be produced by certain z, not by any). A good function that can serve our needs is a negative Gaussian. Returning back to the probabilistic formulation, this means that we need to modify the problem by taking non-deterministic function $f(z)$. We can assume that mapping from Z to X is provided by normal

sampling from the vicinity of the point given by a deterministic function $g : Z \to X$. We therefore have

$$P(x \mid z) \sim \exp\left(-0.5 \left\|x - g(z)\right\|_2^2 / \sigma^2\right), \tag{14}$$

where $\|\cdot\|_2$ is L2 norm and σ is a hyperparameter that controls the width of our effective kernel.

Although the introduced modification is useful, the outlined approach to the problem still remains hardly feasible. In essence, the approach suggests that for each x_i we will be able to generate sufficiently many samples $z_k \sim N(0, I)$ to ensure one or several close matches. The chance of such matches becomes increasingly small with increase of the dimensionality of Z. One can again think about generation of images of digits: our naive method requires that we have a reasonable chance to get all the characteristics of a given image by random selection. If the dimensionality of Z is high, this is hardly realistic and clearly impractical. We can expect that for any given x_i the overwhelming majority of samples $z_k \sim N(0, I)$ results in cases x_k that are far from x_i and thus are useless for the optimization. How can we combat this difficulty? We can do the same trick as we did for ABC computations: we can assign another adjustable function $q(x)$ (we can again think of *NN*) to propose an approximate area in Z where we can likely get $x_k = x$. Since the function should propose an area rather than a single point, it can be seen as a probabilistic function that is quantified by *PDF* $Q(z \mid x)$. In the simplest case (commonly used in practice), the probabilistic function $q(x)$ is represented by an *NN* that returns the mean point $\mu_q(x)$ and the diagonal covariance matrix $\Sigma_q(x) = \sigma_q(x) * I$ for further normal sampling of z:

$$Q(z \mid x) \sim N\left(\mu_q(x), \Sigma_q(x)\right)). \tag{15}$$

It is now tempting to replace the maximization of likelihood Eq. (13) by the requirement to maximize the likelihood

$$P_q(x \mid x_i) = \int P(x \mid z)\, Q(z \mid x_i)\, dz \tag{16}$$

for all given x_i. Nevertheless, in this case we will lose the key requirement that sensible cases can be generated from any $z \sim N(0, I)$. To embed this requirement into the optimization procedure we require the consistency of Eqs. (16) and (13). We focus on the fact that $Q(z \mid x)$ should be close to $P(z \mid x)$ that we can express via $P(z)$ (being $N(0, I)$) from Eq. (13)

$$P(z \mid x) = \frac{P(x \mid z) P(z)}{P(x)}. \tag{17}$$

It turns out useful to employ the Kullback-Leibler (*KL*) divergence as a metric of distance between $Q(z \mid x)$ and $P(z \mid x)$

$$\mathcal{D}\left[Q(z \mid x) \| P(z \mid x)\right] \stackrel{\text{def}}{=} E_{z \sim Q}[\log\left(Q(z \mid x)\right) - \log\left(P(z \mid x)\right)], \tag{18}$$

where $\mathcal{D}[\cdot\|\cdot]$ denotes the *KL* divergence and $E_{z \sim Q}[(\cdot)] = \int (\cdot)\, Q(z \mid x) dz$ stands for the expectation of a function $(\cdot)$ with respect to $z \sim Q(z \mid x)$. We now substitute Eq. (17) into Eq. (18) and move $\log P(x)$ out of the expectation (square brackets) because $\log P(x)$ doesn't depend on z:

$$\mathcal{D}\left[Q(z \mid x) \| P(z \mid x)\right] = E_{z \sim Q}[\log\left(Q(z \mid x)\right) - \log\left(P(x \mid z)\right) - \log\left(P(z)\right)] + \log P(x).$$

Rearranging and noticing the presence of another term of *KL* divergence, we obtain:

$$\log P(x) - \mathcal{D}\left[Q(z \mid x) \| P(z \mid x)\right] = E_{z \sim Q}\left[\log\left(P(x \mid z)\right)\right] - \mathcal{D}\left[Q(z \mid x) \| P(z)\right]. \tag{19}$$

On the left hand side of (19) we have two terms that we want tom maximize: for any given x_i we want to maximize the likelihood of getting it (i.e., $P(x)$) from the sampling procedure Eq. (13) and minimize the divergence of $Q(z \mid x_i)$ from $P(z \mid x_i)$. Recall that here *PDF* $Q(z \mid x_i)$ is defined by a composition of *NN* followed by normal sampling (see Eq. (15)), whereas *PDF* $P(z \mid x_i)$ is defined via Bayes rule Eq. (17) from a composition of another *NN* also followed by normal sampling (see Eq. (14)). In essence, the input of the former *NN* and the output of the latter are required to be close for all x_i and this resembles the traditional autoencoder. An important difference of *VAE* is that for each x_i these *NN*s are connected by a *PDF* in the latent space Z rather than by a single state z.

Let us note that $P(z \mid x)$ on the left hand side of Eq. (19) is defined "implicitly" via Eq. (17) and thus the maximization of the left hand size appears to be problematic. As we will see further, it is the right hand size that can be maximized. This is the key idea of *VAE*s and the reason why we derived equality Eq. (19).

Let us now focus on the maximization of the right hand side of Eq. (19). The second term is the *KL* divergence between two normal distributions:

$$\begin{aligned}\mathcal{D}\left[Q\left(z \mid x\right) \| P(z)\right] &= \mathcal{D}\left[N\left(\mu, \sigma * I\right) \| N(0, I)\right] \\ &= \frac{1}{2}\left(\mathrm{tr}\Sigma_q(x) + \mu_q^T(x)\mu_q(x) - K - \log\det\Sigma_q(x)\right).\end{aligned} \tag{20}$$

The gradient of this expression can be computed analytically. This makes the gradient optimization for this term straightforward. Writing the expectation given by the first term at the right hand side of Eq. (19) makes it clear that its maximization goes in line with the maximization of the livelihood given by Eq. (15), i.e., of the one that we optimized by introducing the proposal-giving non-deterministic function $q(x)$. Moreover, if we recall that the probabilistic part of $P(x \mid z)$ is given by the normal distribution around the state $\tilde{x}$ given by the *NN* of decoder (see Eq. (14)), we see that this term quantifies the difference between $\tilde{x}$ and x in terms of L2 metric:

$$\begin{aligned}E_{z\sim Q}\left[\log\left(P\left(x \mid z\right)\right)\right] &= \int \log P(\tilde{x} \mid z) Q(z \mid x) dz \\ &= -\int \frac{1}{2}\frac{\|x - \tilde{x}\|_2^2}{\sigma^2} Q(z \mid x) dz.\end{aligned} \tag{21}$$

We can now see how the gradient optimization for *VAE* can look like:

- sample x_i from the given training set;
- compute $\mu_q(x_i)$ and $\Sigma_q(x_i)$ using the *NN* of encoder;
- sample one (or many) $z \sim N(\mu_q(x_i), \Sigma_q(x_i))$;
- for the obtained z compute $\tilde{x}_i$ using the *NN* of decoder;
- back-propagate the error seeded by the derivatives of both ((21)) and (20));
- adjust the weights of both *NN*s.

It is worth mentioning that in order to back-propagate the error through the sampling at stage (3) we need to express the components of sampled z in the way that disentangles the parameters to be modified from the random variable:

$$\begin{aligned}z^{(j)} &= \mu_q^{(j)} + \sigma_q^{(j)} r^{(j)}, \\ r^{(j)} &\sim N(0, 1).\end{aligned}$$

In other words we take the derivative with respect to parameters that can be modified and disregard the effect of the random variable since it cannot be modified anyways. The training procedure is then composed of steps (1-6) that can be performed for various x_i independently or batched in groups. The procedure is summarized in Fig. 7. Once trained, a *VAE* can be used to generate a manifold of possible samples x. These samples are to be generated by the decoder out of various $z \sim N(0, I)$. An interesting alternative is to vary z continuously in between the states obtained by the decoder from specific samples of x. We review some indicative examples in the next section.

12.4.3 Examples of VAEs

In 2013 Kingma and Welling introduced and demonstrated the use of the *VAE* concept [61]. They showed two indicative examples. The first one concerns the *MNIST* dataset, the set of monochrome 28×28 images of hand-written digits. The trained *VAE* was used to generate new images of possible digits (for details see [61]). The results for different dimensionality of the latent space (the "width" of the bottleneck) is shown in Fig. 8. One can see that even two-dimensional space is sufficient for capturing the idea of digits, but the pictures appear a bit blurred (an unfortunate property of *VAE*s that was a subject of improvement in further studies). In case of higher dimensional latent space, the pictures become sharper. Note, that the pictures appear coherent: the stroke thickness and the style overall appears the same on all parts of each image.

It is now interesting to see how the image of a digit changes under continuous variation of the latent variable z. Fig. 9 shows the corresponding visualization of the latent space (transformed to quadratic region that spans the pictures) for the *VAE* trained on *MNIST* and also for the *VAE* trained on the pictures of face that were used for the second example.

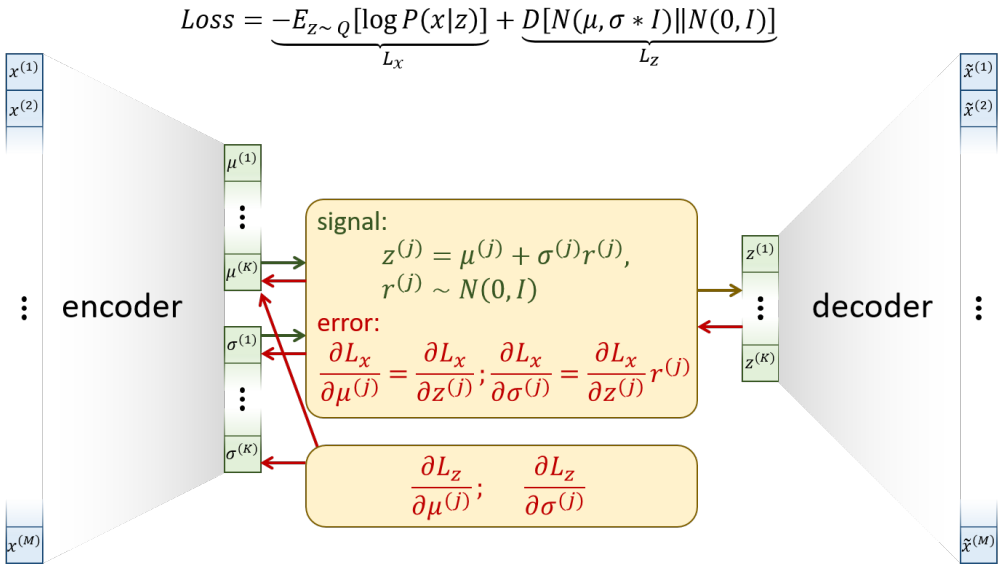
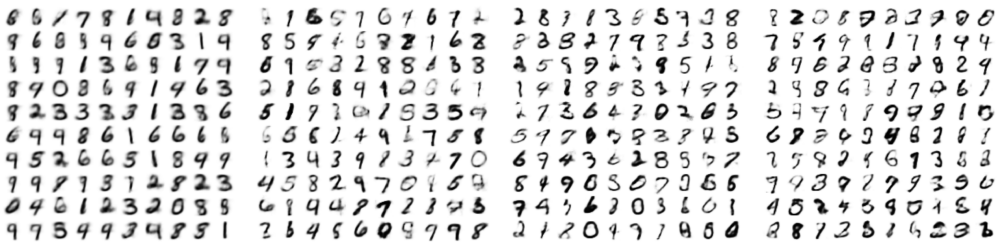
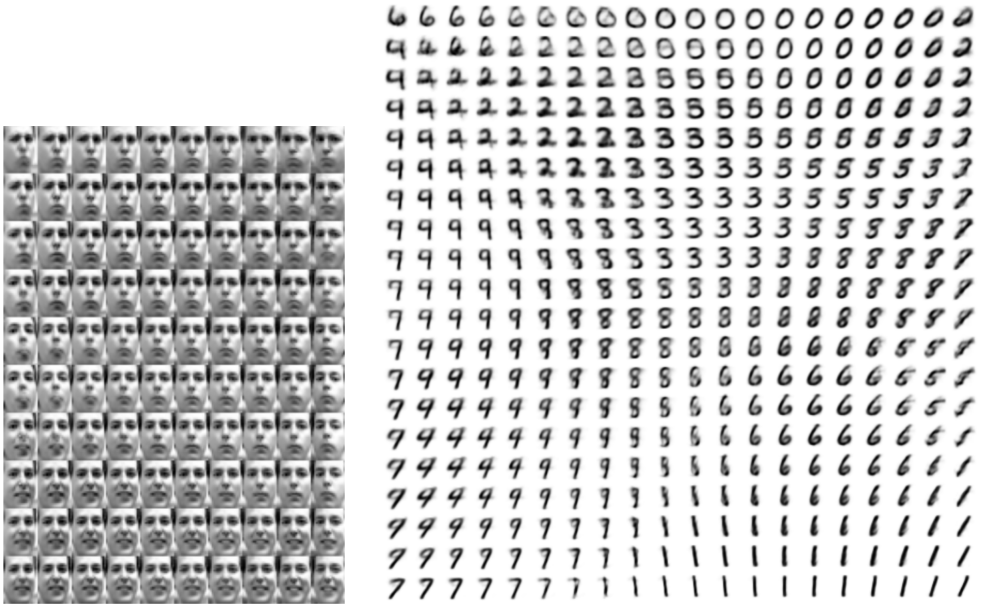


Fig. 7: Training algorithm of variational autoencoder.



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Fig. 8: The generation of new possible images by a *VAE* trained using the *MNIST* dataset. Sets of images generated from random samples of z are shown for different dimensionality of the latent space. Reproduced from Kingma and Welling (2013).



(a) Learned Frey Face manifold (b) Learned MNIST manifold

Fig. 9: The results of continuous variation of two-dimensional latent variable z in case of *VAE*s trained on the pictures of faces (left) and on the *MNIST* dataset (right). Reproduced from Kingma and Welling (2013).

12.4.4 Geometrical insights into the concept of VAEs

A common way to interpret the concept of *VAE*s is to discuss the effect of both terms in their loss function Eq. (19) on the latent space. The first term is the reconstruction loss, which is the only criterion used in training of traditional autoencoders. That is why its role is associated with the formation of regions (in the latent space), where z states yield x_i from the training set. However, and this is the key point, nothing "motivates" the traditional encoder to provide possible states x in between these regions. It is the second term, also referred as the *KL*-divergence term, that is ought to regularize learning so that the regions are "compressed" towards each other, enabling the possibility of continuous generation of probable x out of $z \sim N(0, I)$. This can be illustrated by the comparison of the distribution of z states in the 2D latent space in case the training on the *MNIST* dataset is carried out using the first, the second or both terms in the loss function [62]. Such a comparison is shown in Fig. 10.

Although this interpretation provides an intuition about the derived loss function of *VAE*s, the idea of the principles used for the problem formulation in the very beginning may remain unclear. Why extending the notion of single points in latent space (used in autoencoders) to *PDF*s (used in *VAE*s) is necessary and sufficient? Why under the suggested regularization via *KL*-divergence the regions in z must stick together rather than become compressed together with remaining voids in between? Let us discuss some geometrical insights that not only clarify such points but also lead to the understanding of further fundamental possibilities.

Let us imagine we trained an ordinary autoencoder with two-dimensional latent space using samples of four categories. We can again think of *MNIST* dataset, from which we select, say, sevens and ones written in thick or thin stroke. In Fig. 11 on the left, we schematically show some possible regions (in the latent space) that correspond to our four categories and are colored respectively. We intentionally draw sophisticated shapes of these regions to emphasize the difficulty of generating new samples from any random point in the latent space. The fact that the regions are separated by voids indicates that we cannot generate continuous transitions between samples. This is because we have not restricted our model to achieve this property. (The only property that we can probably expect is that each region is a connected set due to continuous nature of *NN*s.) How can we modify the training rules to make the regions stick together and form a joined distribution that is close to $N(0, I)$?

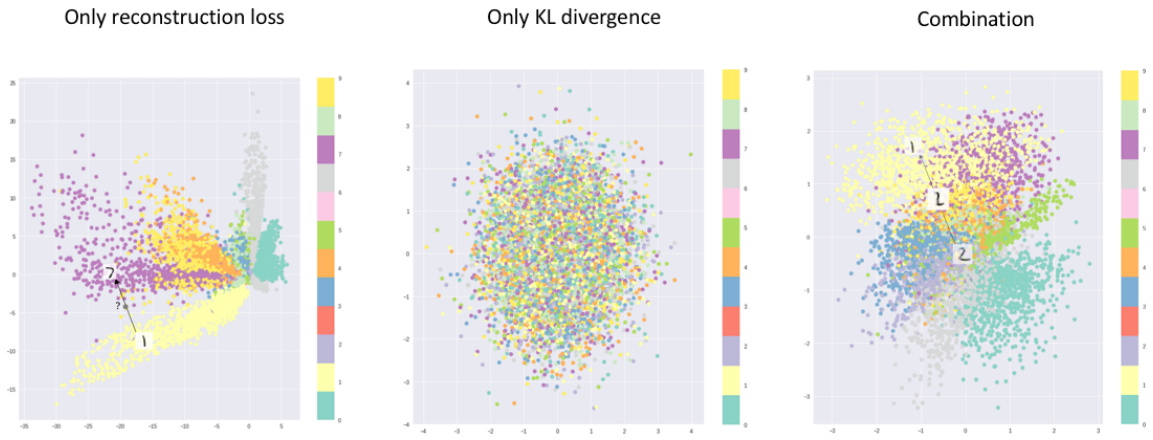


Fig. 10: Comparison of the distribution of digit-labeled regions in two-dimensional latent space of a *VAE* trained on the *MNIST* dataset using only the first (left), only the second (central), and both (right) terms of the loss function Eq. (19). Reproduced from Shafkat (2018).

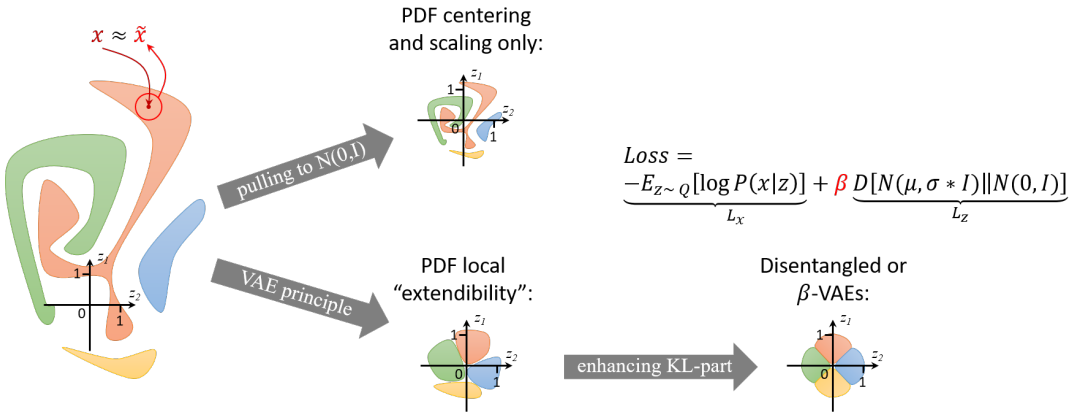


Fig. 11: Geometrical principles behind the mechanism of *VAE*s and the extension to β-*VAE*s.

Our initial idea was to require that the distribution of points produced by encoder should be close to $N(0, I)$. In principle, we can generate z points from many (or all) given x samples, compute their distribution and form a loss function that quantifies the difference from $N(0, I)$. For instance, if we quantify this difference by the deviations of mean and variance, our loss function would correspond to "pulling" the *PDF* of z to $N(0, I)$ by centering and rescaling (see Fig. 11 (at the upper center)). However, nothing seems to force our system to close the gaps between the regions. We can probably use higher moments of *PDF*, but then will need to use more and more samples and this becomes impractical. It is clear that *VAE*s provide a more efficient and scalable mechanism.

Let us recall how the training of *VAE*s is different from that of ordinary autoencoders. We assign the encoder to propose (from a given x) not a point but a region, which consists of a vicinity of some computed point $\mu_q(x)$. Next, we require that all points from this vicinity yield samples $\tilde{x}$ that are close to the original x. In essence, we require that the transformation is *extendible*: if we make a shift from the determined point $\mu_q(x)$ in any direction the result of the decoder should still look probable. Assessing whether $\tilde{x}$ is probable or not is a difficult task. A simple assessment that we can still easily do is to compare it with x according to L2 metric. Given that the shift is small, this assessment is reasonable and can be seen as a clever choice. Now, let us think when the generated samples $\tilde{x}$ do not look like x (or any other probable sample). This can happen at the boundaries of the colored regions, whereas the points in the inner parts are likely to be similar in this or that way (i.e., $x \approx \tilde{x}$). Since more roundish shapes correspond to lower chance for inner points to be close to the boundary, this procedure would favor round shapes of the regions for each category. However, if this procedure only concerns reconstruction accuracy, nothing prevents the system from collapsing the proposed regions into points, in addition the rounded regions can likely appear far from the center leaving large voids in between each other. This is why we need to "pull" all the regions proposed by encoder to $N(0, I)$. When we combine all these finding, we can expect that the regions of different categories take round shapes and stick together near the center (see Fig. 11, at the lower center). This is what explains the mechanism of *VAE*s.

In addition, we can now see that the *KL*-divergence term in the loss function is responsible for making roundish shapes of regions that correspond to different categories. What happens if we further strengthen this effect by introducing a factor $\beta > 1$ for this term? More close packing of more roundish shapes can favor a more isotropic placement that can be interpreted as factorization or disentanglement of the latent representation. The effect of disentanglement has been shown in 2016 by Higgins *et al.* in Ref. [63] (see Fig. 12).

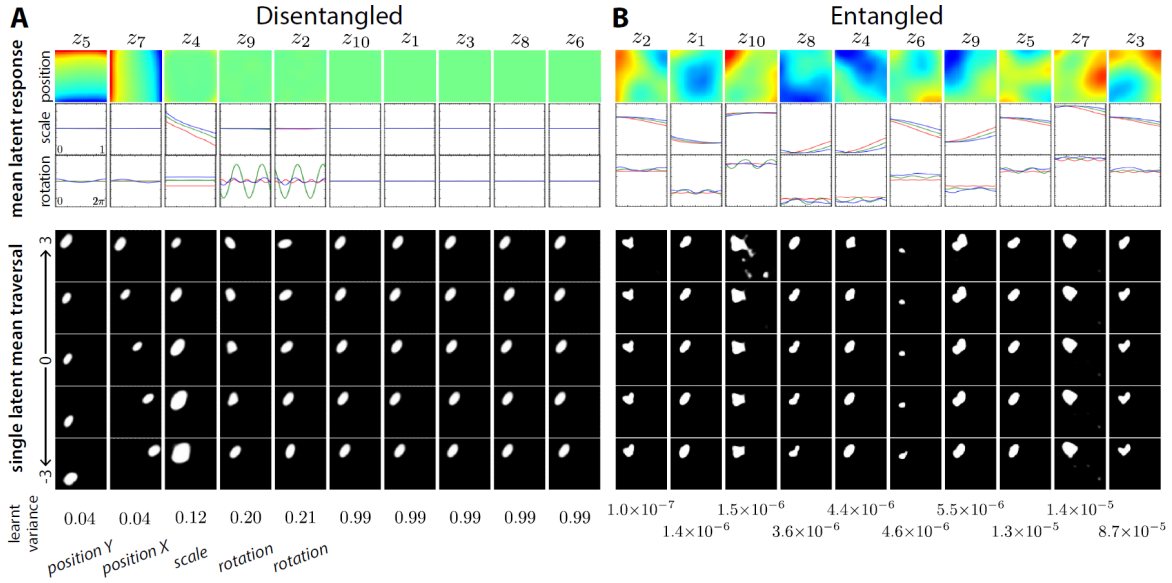


Fig. 12: The effect of the latent space disentanglement achieved by β-*VAE*s. The pictures show the results of training *VAE*s using pictures of objects that vary in terms of simple properties, such as locations, size and shape. Ordinary *VAE* generates latent space (right) with components that has no any simple specific meaning, whereas β-*VAE* (left) generate latent space with first components being correlated with certain properties. Reproduced from Higgins *et al.* (2016).

Part II

Backmatter

CHAPTER

THIRTEEN

BIBLIOGRAPHY

CHAPTER

FOURTEEN

GLOSSARY

ABC approximate Bayesian computation; see Section 10

ACO ant colony optimization; see Section 9.2.2

AIC Akaike information criterion

ARD automatic relevance detection

BCI Bayesian credible interval

BIC Bayesian information criterion

BMA Bayesian model averaging

CE cluster expansion

CI confidence interval

CL confidence level

CS compressive sampling

CV cross-validation

DIC deviance information criterion

DFT density functional theory

DOB degree of belief

DOF

DOFs degree(s) of freedom

ECI effective cluster interactions

ESS effective sample size

FC force constant

FDTD finite difference time domain

FFT fast Fourier transformation

GA

GAs genetic algorithm

GLM generalized linear model

GP

GPs Gaussian process

Heteroscedastic

heteroscedastic a sequence of random variables is heteroskedastic if the variability of the random disturbance is different across elements of the sequence; see here for more information

homoscedastic a sequence of random variables is homoscedastic if all its random variables have the same finite variance; see here for more information

i.i.d. independently and identically distributed

IC information criteria

IG inverse-gamma

KDE kernel density estimate

KKT Karush-Kuhn-Tucker conditions; see Section 11.3

KL Kullback-Leibler

LASSO least-absolute shrinkage and selection operator

LOOCV leave-one-out cross-validation

MC Monte Carlo (simulations)

MCMC Markov-chain Monte Carlo

ML machine learning

MNIST Modified National Institute of Standards and Technology database; large database of handwritten digits; see here for more information

MSE mean square error

NIG normal-inverse-gamma

NN neural network

OLS ordinary least squares

OMP orthogonal matching pursuit

PCA principal component analysis; see Section 12

PDF probability density function

PES potential energy surface

Precision

precision A term sometimes used to refer to the inverse variance.

PI prediction interval

PPD posterior predictive distribution

PSO particle swarm optimization; see Section 9.2.1

RFE recursive feature elimination

RMSE root mean square error

RSS residual sum of squares

RVM relevance vector machines; see Section 11.5

SMC sequential Monte Carlo

SQE squared exponential

SVD singular value decomposition

SVM support vector machine

VAE variational autoencoders

WAIC Watanabe-Akaike IC

WLS weighted least squares

XAI eXplainable Artificial Intelligence

BIBLIOGRAPHY

[1] Sébastien Rivat and Alexei Grinbaum. Philosophical foundations of effective field theories. *The European Physical Journal A*, 56(3):90, 2020. doi:10.1140/epja/s10050-020-00089-w.

[2] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Review of Modern Physics*, 91:045002, Dec 2019. doi:10.1103/RevModPhys.91.045002.

[3] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1 – 124, 2019. A high-bias, low-variance introduction to Machine Learning for physicists. doi:10.1016/j.physrep.2019.03.001.

[4] Nick Bostrom. Are We Living in a Computer Simulation? *The Philosophical Quarterly*, 53(211):243–255, April 2003. doi:10.1111/1467-9213.00309.

[5] Silas R Beane, Zohreh Davoudi, and Martin J Savage. Constraints on the universe as a numerical simulation. *The European Physical Journal A*, 50(9):148, 2014. doi:10.1140/epja/i2014-14148-0.

[6] Jenný Brynjarsdóttir and Anthony O'Hagan. Learning about physical parameters: the importance of model discrepancy. *Inverse Problems*, 30(11):114007, oct 2014. doi:10.1088/0266-5611/30/11/114007.

[7] Phil Gregory. *Bayesian Logical Data Analysis for the Physical Sciences: A Comparative Approach with Mathematica® Support*. Cambridge University Press, 2005. doi:10.1017/CBO9780511791277.

[8] Daniel Foreman-Mackey, David W. Hogg, Dustin Lang, and Jonathan Goodman. Emcee: the mcmc hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306–312, Mar 2013. doi:10.1086/670067.

[9] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

[10] Jaakko Riihimäki and Aki Vehtari. Gaussian processes with monotonicity information. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of Machine Learning Research*, volume 9 of Proceedings of Machine Learning Research, 645–652. Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings. URL: http://proceedings.mlr.press/v9/riihimaki10a.html.

[11] D V Lindley. A Sstatistical Paradox. *Biometrika*, 44(1-2):187–192, June 1957. doi:10.1093/biomet/44.1-2.187.

[12] Robert E. Kass and Adrian E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1995.10476572, doi:10.1080/01621459.1995.10476572.

[13] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461 – 464, 1978. URL: https://doi.org/10.1214/aos/1176344136, doi:10.1214/aos/1176344136.

[14] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 1974. doi:10.1109/TAC.1974.1100705.

[15] David J. Spiegelhalter, Nicola G. Best, Bradley P. Carlin, and Angelika Van Der Linde. Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(4):583–639, 2002. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00353, doi:https://doi.org/10.1111/1467-9868.00353.

[16] Andrew Gelman, Jessica Hwang, and Aki Vehtari. Understanding predictive information criteria for Bayesian models. *Statistics and Computing*, 24(6):997–1016, 2014.

[17] John Skilling. Nested sampling for general bayesian computation. *Bayesian Anal.*, 1(4):833–859, 12 2006. URL: https://doi.org/10.1214/06-BA127, doi:10.1214/06-BA127.

[18] F Feroz, M P Hobson, and M Bridges. MultiNest: an efficient and robust Bayesian inference tool for cosmology and particle physics. *Monthly Notices of the Royal Astronomical Society*, 398(4):1601–1614, September 2009. URL: https://doi.org/10.1111/j.1365-2966.2009.14548.x, doi:10.1111/j.1365-2966.2009.14548.x.

[19] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2006.00553.x, arXiv:https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2006.00553.x, doi:10.1111/j.1467-9868.2006.00553.x.

[20] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging: a tutorial (with comments by m. clyde, david draper and e. i. george, and a rejoinder by the authors. *Statist. Sci.*, 14(4):382–417, 11 1999. URL: https://doi.org/10.1214/ss/1009212519, doi:10.1214/ss/1009212519.

[21] Yuling Yao, Aki Vehtari, Daniel Simpson, and Andrew Gelman. Using stacking to average bayesian predictive distributions (with discussion). *Bayesian Anal.*, 13(3):917–1007, 09 2018. URL: https://doi.org/10.1214/17-BA1091, doi:10.1214/17-BA1091.

[22] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2013. ISBN 9780262018029.

[23] Erik Fransson, Fredrik Eriksson, and Paul Erhart. Efficient construction of linear models in materials modeling and applications to force constant expansions. *npj Comp. Mater.*, 6:135, 7 2020. doi:10.1038/s41524-020-00404-5.

[24] Hui Zou. The adaptive Lasso and its oracle properties. *J. Amer. Stat. Assoc.*, 101(476):1418–1429, 2006. doi:10.1198/016214506000000735.

[25] Mattias Ångqvist, William A. Muñoz, J. Magnus Rahm, Erik Fransson, Céline Durniak, Piotr Rozyczko, Thomas H. Rod, and Paul Erhart. Icet – a python library for constructing and sampling alloy cluster expansions. *Adv. Theo. Simul.*, 2(7):1900015, 2019. doi:10.1002/adts.201900015.

[26] Richard M. Martin. *Electronic Structure*. Cambridge University Press, Cambridge, 2004.

[27] J. M. Sanchez, F. Ducastelle, and D. Gratias. Generalized cluster description of multicomponent systems. *Physica*, 128(1-2):334, 1984.

[28] A. van de Walle and G. Ceder. Automating first-principles phase diagram calculations. *J. Phase Equil.*, 23(4):348–359, 2002. doi:10.1361/105497102770331596.

[29] Gus L. W. Hart, Chris Sutton, Santiago Rigamonti, and Maria Troppenz. Hands-on tutorial on cluster expansion: modeling of configurational energetics. URL: https://th.fhi-berlin.mpg.de/sitesub/meetings/DFT-workshop-2016/uploads/Meeting/Tutorial_7_2016.pdf.

[30] Tim Mueller and Gerbrand Ceder. Bayesian approach to cluster expansions. *Phys. Rev. B*, 80:024103, Jul 2009. doi:10.1103/PhysRevB.80.024103.

[31] Brent Fultz. Vibrational thermodynamics of materials. *Progress in Materials Science*, 55(4):247 – 352, 2010. doi:10.1016/j.pmatsci.2009.05.002.

[32] Michael P. Marder. *Condensed matter physics*. Wiley-Blackwell, 2010. An online version is available via the Chalmers Library (log in via OpenAthens): https://www.vlebooks.com/Vleweb/Product/Index/12176.

[33] N. W. Ashcroft and N. D. Mermin. *Solid State Physics*. Saunders, 1976.

[34] Max Born and Kun Huang. *Dynamical Theory of Crystal Lattices*. Oxford University Press, Oxford, 1954. ISBN 0-19-850369-5.

[35] Duane C. Wallace. *Thermodynamics of Crystals*. Dover, Mineola, New York, 1998.

[36] Keivan Esfarjani and Harold T. Stokes. Method to extract anharmonic force constants from first principles calculations. *Physical Review B*, 77(14):144112, April 2008. doi:10.1103/PhysRevB.77.144112.

[37] Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna. Geometry of nonlinear least squares with applications to sloppy models and optimization. *Physical Review E*, 83:036701, 2011. doi:10.1103/PhysRevE.83.036701.

[38] James P. Sethna. Sloppy models. URL: http://www.lassp.cornell.edu/sethna/Sloppy/.

[39] E. J. Candes and M. B. Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30, March 2008. doi:10.1109/MSP.2007.914731.

[40] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit. *CS Technion*, 2008.

[41] Kenneth P. Burnham and David R. Anderson. Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods & Research*, 33(2):261–304, 2004. doi:10.1177/0049124104268644.

[42] Robert H. Swendsen and Jian-Sheng Wang. Replica monte carlo simulation of spin-glasses. *Phys. Rev. Lett.*, 57:2607–2609, Nov 1986. doi:10.1103/PhysRevLett.57.2607.

[43] Cristian V. Ciobanu and Cristian Predescu. Reconstruction of silicon surfaces: a stochastic optimization problem. *Phys. Rev. B*, 70:085321, Aug 2004. doi:10.1103/PhysRevB.70.085321.

[44] Daan Frenkel. Simulations: the dark side. *arXiv:1211.4440*, 2012. URL: https://arxiv.org/abs/1211.4440.

[45] Maurice Clerc. Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem. In Godfrey C. Onwubolu and B. V. Babu, editors, *New Optimization Techniques in Engineering*, Studies in Fuzziness and Soft Computing, pages 219–239. Springer, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-39930-8_8.

[46] Bruno Seixas Gomes de Almeida and Victor Coppo Leite. Particle swarm optimization: a powerful technique for solving engineering problems. *Swarm Intelligence - Recent Advances, New Perspectives and Applications*, December 2019. doi:10.5772/intechopen.89633.

[47] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, February 1996. doi:10.1109/3477.484436.

[48] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, 2951–2959. Red Hook, NY, USA, December 2012. Curran Associates Inc.

[49] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, December 2019. URL: https://doi.org/10.1103/revmodphys.91.045002, doi:10.1103/revmodphys.91.045002.

[50] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124, May 2019. URL: https://doi.org/10.1016/j.physrep.2019.03.001, doi:10.1016/j.physrep.2019.03.001.

[51] Johann Brehmer, Gilles Louppe, Juan Pavez, and Kyle Cranmer. Mining gold from implicit models to improve likelihood-free inference. *Proceedings of the National Academy of Sciences*, 117(10):5242–5249, February 2020. URL: https://doi.org/10.1073/pnas.1915980117, doi:10.1073/pnas.1915980117.

[52] Jake VanderPlas. *Python data science handbook : essential tools for working with data*. O'Reilly Media, Inc, Sebastopol, CA, 2016. ISBN 978-1491912058.

[53] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, June 2004. URL: https://doi.org/10.1017/cbo9780511809682, doi:10.1017/cbo9780511809682.

[54] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Volume 4 of Information science and statistics. Springer, 2006. ISBN 9780387310732. URL: http://www.library.wisc.edu/selectedtocs/bg0137.pdf, arXiv:0-387-31073-8, doi:10.1117/1.2819119.

[55] Paul S. Bradley and Olvi L. Mangasarian. Feature selection via concave minimization and support vector machines. In *ICML*. 1998.

[56] Li Wang, Ji Zhu, and Hui Zou. The doubly regularized support vector machine. *STATISTICA SINICA*, pages 589–616, 2006.

[57] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, October 2019. doi:10.1109/TEVC.2019.2890858.

[58] Jason Jo and Yoshua Bengio. Measuring the tendency of cnns to learn surface statistical regularities. *arXiv*, 2017. arXiv:arXiv:1711.11561, doi:10.48550/arXiv.1711.11561.

[59] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN 9780387848846. URL: https://books.google.se/books?id=eBSgoAEACAAJ.

[60] Jonathan Bac, Evgeny M. Mirkes, Alexander N. Gorban, Ivan Tyukin, and Andrei Zinovyev. Scikit-Dimension: A Python Package for Intrinsic Dimension Estimation. *Entropy*, 23(10):1368, October 2021. doi:10.3390/e23101368.

[61] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 2013. arXiv:arXiv:1312.6114.

[62] Irhum Shafkat. Intuitively understanding variational autoencoders. 2018. URL: https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf.

[63] Irina Higgins, Loic Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. Early visual concept learning with unsupervised deep learning. 2016. arXiv:arXiv:1606.05579.

INDEX

R

S

V

W

X