AUTHOR: PHILIPP HENNIG
SCRIBE: FREDERIK KUNSTNER

PROBABILISTIC INFERENCE AND LEARNING

LECTURES NOTES

Questions about this document (typos, structure, etc.) should be directed to
Frederik Kunstner at frederik.kunstner@epfl.ch

Questions regarding the content should be directed to
Philipp Hennig at philipp.hennig@uni-tuebingen.de

Lecture notes prepared for the course
Probabilitstic Inference and Learning
given by Prof. Philipp Hennig at the University of Tübingen.

Last edited January 9, 2019.
Contributors: Philipp Hennig, Frederik Künstner.

Introduction

Probabilistic inference is a foundation of scientific reasoning, statistics, and machine learning. The goal of this course is to establish a formal framework for probabilistic reasoning, show how to use it to build powerful inference mechanisms for real-world problems and develop the technical tools necessary to implement inference in practice.

The lecture begins with a general introduction to basic principles of the rules of probability theory and graphical models, then covers the probabilistic view on many standard settings, like supervised regression and classification, unsupervised dimensionality reduction and clustering.

In a parallel thread through the lecture, we will also encounter a number of popular algorithms for inference in probabilistic models, including exact inference in Gaussian models, sampling, and free-energy methods.

Some of the points we hope to cover are

- Connections between probabilistic inference and Boolean logic
- Learning functional relationships between variables.
- Code examples, such as how to build a model for your body or how to rate a human's credit worthiness
- The world's fastest learning algorithms
- A generalization from "shallow" and "deep" to "structured" learning
- Formal and symbolic languages for artificial intelligence
- A general toolbox for encoding structured domain knowledge in a learning agent,transferring it into a concrete algorithm

And much more to give a joint, connected, holistic view on reasoning, inference, learning and intelligence.

Contents

Reasoning under Uncertainty 7

Probabilistic Reasoning 13

Probabilities over Continuous Variables 19

Gaussian probability distributions 23

Parametric Gaussian Regression 27

Hierarchical Inference: learning the features 35

Gaussian Processes 39

Gaussian Process Classification 47

Practical Examples 53

Understanding Kernels 55

Markov Chains: Models for Time Series 61

Exponential Family 65

Graphical Models 69

Factor graph 75

The Sum-Product Algorithm 77

Bibliography 81

Reasoning under Uncertainty

Probability theory is nothing but common sense reduced to calculation.

— Pierre-Simon de Laplace

An inference problem requires statements about the value of an *unobserved* variable x based on observations y which are *related* to x, but may not be sufficient to fully determine x. This requires a notion of *uncertainty*.

We hope in this chapter to give an intuition on reasoning under incomplete information and the formalization of probabilities from both a a philosophical point of view and purely mathematical construction.

Examples

A Card Trick

Three cards with colored faces are placed into a bag. One card is red on both faces, one is white on both faces and the last is red on one face and white on the other - see Figure [1]. After mixing the bag, we pick a card and see that one face is white. What is the color of its other face?

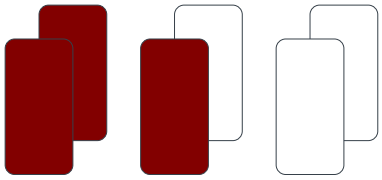


Figure 1: **A card trick**

While we do not have direct information about the back of the card, we can use what we know about the setup to make an educated guess about the *probability* that it is also white. Make a guess; is it $1/2$? $2/3$? Something else? We will revisit the problem at a later stage.

Reasoning about the weather

Consider the two following events,

$$R := \text{It is raining at 6pm},$$
$$C := \text{It is cloudy before 6pm}.$$

Assume, for the moment, that C is a necessary condition for R; it can not rain at 6pm if there were not clouds before. In classical logic, $R \Rightarrow C$ and we can use *deductive reasoning* to reach the conclusions

$$\text{If } R \text{ is true, then } C \text{ is true},$$
$$\text{If } C \text{ is false, then } R \text{ is false}.$$

However, deductive reasoning does not help in determining if it will rain if we see clouds, or if there were clouds if we don't

see rain. To reason about those relationships, we turn to *plausible reasoning*;

If R is false, then C is less plausible,

If C is true, then R is more plausible,

and we write $p(C|R) > p(C)$ "if R is true, then C becomes more plausible". The conclusions reached by deductive reasoning still hold in probable reasoning;

If R is true, then C is more plausible,

If C is false, then R is less plausible.

Formalization

The goal of the this section is to establish a formal framework for probable reasoning. We will present Cox's theorem as an intuitive way to define *plausibility*, appealing to *common sense*, and the more modern axioms of Kolmogorov's probability theory.

Cox's Axioms are an attempt to formalize reasoning in a way that is consistent with human "common sense".

Definition 1 (Cox's axioms)**.**

1. The plausibility of B assuming A is true is a real number, denoted by $p(B|A)$. Larger numbers correspond to higher plausibility.
2. Plausibility complies with common sense:

$$\text{If } p(A|C) > p(A|C') \text{ and } p(B|A \wedge C) = p(B|A \wedge C'), \text{ then}$$

$$p(A \wedge B|C) \geq p(A \wedge B|C) \text{ and } p(\neg A|C) \leq p(\neg A|C').$$

3. Plausibility is consistent:
   (a) If a conclusion can be reached in several ways, then every possible way must lead to the same result.
   (b) All available evidence must be taken into account, none can be excluded.
   (c) Equivalent (isomorphic) states of knowledge must be represented by the same plausibility

Cox's axioms imply that, up to monotonic transformations, the plausibility of A can be represented by a real number $p(A) \in [0,1]$ where $p(A) = 1$ indicates the certainty that A is true and $p(A) = 0$ the certainty that $\neg A$ is true, as well as the *Sum rule* and the *Product rule*;

Definition 2 (Sum rule)**.**

$$p(A|C) + p(\neg A|C) = 1$$

Definition 3 (Product rule)**.**

$$p(A, B|C) = p(A|B, C)p(B|C) = p(B|A, C)p(A|C)$$

For a thorough treatment, see Chapters 1 and 2 in *Probability Theory - the Logic of Science*[1].

[1] Jaynes. *Probability theory: The logic of science*. Cambridge University Press, 2003. URL bayes.wustl.edu/etj/prob/book.pdf

Kolmogorov's Axioms[2] are, in contrast to Cox's philosophical definition, a pure mathematical construction. A helpful mental image is to think of *truth* as a finite amount of "mass" to be spread, or distributed, over a space of mutually exclusive "elementary" events. More events can then be constructed from unions and intersections of sets of elementary events. An example of this construction is *roulette* – see Figure [2] – where numbers 0-36 constitute the set of elementary events and Red/Black, Odd/Even, Low/High are constructed from elementary events.

[2] Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. 1933

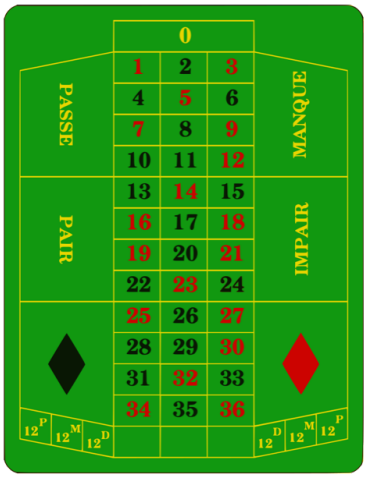


Figure 2: **Events for a Roulette**

We first present a simplified form of the axioms;

Definition 4 (Kolmogorov's Axiom (Simplified))**.** Let Ω be a space of possible "elementary" events, such as samples or proposition, and let F be the set of all possible subsets of Ω. The probability p of an event $A \in F$ is a real map $p : F \to \mathbb{R}$ that has the following three properties:

1. **Non-Negativity:** For all $A \in F$, $0 \leq p(A) \leq 1$.
2. **Normalization:** $p(\Omega) = 1$.
3. **Additivity:** If A and B are mutually exclusive, then

$$p(A \cup B) = p(A) + p(B).$$

It is easy that Kolmogorov's axioms lead to some of the same properties as Cox's axioms; for example as an event and its complement are disjoint,

$$p(A) + p(\bar{A}) = p(A \cup \neg A) = p(\Omega) = 1$$

And as $A \cap B$ is disjoint from $A \cap \bar{B}$ and $B \cup \bar{B} = \Omega$,

$$p(A) = p(A \cap \Omega) = p(A \cap (B \cup \bar{B})) = p(A \cap B) + p(A \cap \bar{B})$$

With the additional definition of conditionning, Kolmogorov's axioms also satisfy the Sum rule and the Product rule.

Definition 5 (Conditional Probability)**.** Let A, B be events such that $p(A) > 0$. The conditional probability $p(B|A)$ is such that

$$p(B|A) := \frac{p(A \cap B)}{p(A)}.$$

Both Cox's and Kolmogorov's axioms lead to the Sum and Product rules - the first by stating basic *desiderata* plausibilities should hold, the latter by defining measures on sets, and taking together they provide both a philosophical and mathematical argument for representing plausibilities as *probabilities*. We will see in the next section how the Sum and Product rules imply Bayes' Theorem, the basis of probabilistic inference.

Contemporary formal form

For completeness, we give the full formulation of Kolmogorov's axioms. Don't get put off by the technical termy, the intuition from the simpler statement still holds.

Definition 6 (Kolgomorov's axioms, contemporary formal form)**.** Let (Σ, F, p) be a *measure space*, also known as a Borel space, that is

1. F is a *σ-algebra* on Ω. That is, F is a collection of subsets of Ω that is closed under countable unions and intersections and includes the empty set: $\emptyset \in F$ and $(A, B \in F) \Rightarrow (A \cup B \in F), (A \cap B \in F)$.
2. p is a *measure* on (Ω, F). That is, $p : F \to \mathbb{R}$ with $p(A) \geq 0$ for all $A \in F$, $p(\emptyset) = 0$ and (*σ-additivity*) if $A \cap B = \emptyset$, then $p(A \cup B) = p(A) + p(B)$.
3. The measure p is called a *probability measure* if the *normalization* property holds; $p(\Omega) = 1$.

A note on notation

The notation differs depending on the point of view taken; Cox considers the plausibility of true/false statements and the notation of logic is used, $A \wedge B, A \vee B, \neg A$, whereas Kolgomorov defines measures on sets using $A \cap B, A \cup B, \bar{A}$. Inference is most often concerned with multiple joint distribution and (ab)uses the notation $p(A, B) = p(A \wedge B)$.

So far, inputs to the probability where propositional variables; $p(A)$ = probability that A is true. In the remaining of this document, $p(A)$ is a function over the possible values that A can take, with a slightly unusual notation. Given two binary variables A, B, writing, for example, that

$$p(A, B) = p(A)p(B)$$

means *all* of the following;

$$\begin{aligned} p(A = 0, B = 0) &= p(A = 0)p(B = 0), \\ p(A = 0, B = 1) &= p(A = 0)p(B = 1), \\ p(A = 1, B = 0) &= p(A = 1)p(B = 0), \\ p(A = 1, B = 1) &= p(A = 1)p(B = 1). \end{aligned}$$

Bayes' Theorem

Bayes' Theorem, a corollary to the Product rule, states how to compute the probability of an event X when observing data D.

Corollary 7 (Bayes' Theorem)**.**

$$p(X|D) = \frac{p(D|X)p(X)}{p(D)}$$

If X is some hypothesis and D some observations, Baye's theorem tells how to update the plausibility of the hypothesis based on seen data.

THE LANGUAGE OF INFERENCE, commonly used in Bayesian inference, assuming that X is an hypothesis and D an observation, is

$$\underbrace{p(X|D)}_{\text{posterior}} = \frac{\overbrace{p(X)}^{\text{prior}} \times \overbrace{p(D|X)}^{\text{likelihood}}}{\underbrace{p(D)}_{\text{evidence}}}$$

- The *posterior* is the probability that the hypothesis X is true after observing D.
- The *prior* is the probability that the hypothesis X is true, before any observation.
- The *likelihood* is the conditional probability of observing D given that the hypothesis is true.
- The *evidence* is the probability of observing D, regardless of the truth of hypothesis X.

A *marginal* distribution is the distribution of a subset of variables, where some variables have been *averaged out*. Given a joint distribution $p(A, B)$, the marginal $p(A)$ can be found as

$$p(A) = \sum_{b \in \mathcal{B}} p(A|B = b)p(B = b),$$

where $\mathcal{B}$ is the space of possible values for B.

See wikipedia.org/wiki/Posterior_probability#Example and wikipedia.org/wiki/Marginal_distribution#Real-world_example for more examples.

Revisiting the card trick

There are multiple ways to define events to approach the problem; here is one possible solution Let C be the card taken out of the bag, with possible values $\{RR, RW, WW\}$ – for Red-Red, Red-White and White-White – and let W be the event "the observed face is white". The other side of the card is also white iff. we have picked $C = WW$, so we are interested in the probability $p(C = WW|W)$. Applying Bayes' Theorem, we have that

$$p(C = WW|W) = \frac{p(W|C = WW)p(C = WW)}{p(W)}.$$

Filling in numbers,

- the prior probability of picking $C = WW$ is $1/3$,
- the probability that the observed face is white given WW is 1,
- as half of the faces are white, $p(W) = 1/2$,

leading to $p(C = WW|W) = 1/3/1/2 = 2/3$. Try to apply the same strategy to solve the famous Monty Hall problem[3]!

[3] wikipedia.org/wiki/Monty_Hall_problem

Summary of plausible reasoning

We can cast deductive reasoning in terms of plausible reasoning, and using the Sum and Product rules and Bayes' theorem we can show that $A \Rightarrow B$ implies that

$A \Rightarrow B$	is equivalent to $p(B\|A) = 1$
$\neg B \Rightarrow \neg A$	is equivalent to $p(\neg A\|\neg B) = 1$
$p(B\|\neg A) \leq p(B)$	A is false implies B becomes less plausible
$p(A\|B) \geq p(A)$	B is true implies A becomes more plausible

but plausible reasoning is more general, and $p(B|A) \geq p(B)$ implies that

$p(B\|A) \geq p(B)$	A is true implies B becomes more plausible
$p(B\|\neg A) \leq p(B)$	A is false implies B becomes less plausible
$p(A\|B) \geq p(A)$	B is true implies A becomes more plausible
$p(\neg A\|\neg B) \geq p(\neg A)$	B is false implies A becomes less plausible

Probabilistic Reasoning

Probability Theory can stumble onto computational difficulties, as the number of parameters required to describe a system grows exponentially with the number of variables considered. The joint distribution of $n = 26$ binary variables $A, B, ..., Z$ has 2^n free parameters, $q_1, q_2, ..., q_{2^n-1}$,

$$\begin{aligned} p(A, B, ..., Z) &= p_1 \\ p(\neg A, B, ..., Z) &= p_2 \\ &\vdots \\ p(\neg A, \neg B, ..., Z) &= p_{67\,108\,863} \\ p(\neg A, \neg B, ..., \neg Z) &= 1 - \sum_{i=1}^{2^n-1} p_i \end{aligned}$$

Storing the parameters alone would already require $\approx$ 500Mb of RAM. In addition to a large memory requirement, computing marginal probabilities such as $p(A)$ is also very time consuming. Thankfully, under some assumption, we can express the joint distribution in fewer numbers.

The earthquake and the burglar

Consider the following example, which we will use as an example for probabilistic reasoning. Assume that you have a home alarm system that can detect burglars, but can also be triggered by earthquakes or other elements. Being away from home, you receive a text message from the alarm system and you want to assess the probability that your home is currently being robbed. To get more information, you can turn the radio on, which will reliably broadcast a message if an earthquake happened.

Let's define the following observable variables,

A : The alarm was triggered,
R : The radio announced an earthquake,

and the following latent variable, which we will need to infer,

E : There was an earthquake,
B : There is a burglar in your home.

The joint probability distribution over those 4 binary variables would typically need $2^4 - 1 = 15$ parameters to be fully repre-

sented,

$$p(A, R, E, B) = p(A|R, E, B)p(R|E, B)p(E|B)p(B).$$

However, we can use domain knowledge to remove irrelevant conditions;

- We can assume that the probability of an earthquake occurring is independent of being robbed, such that $p(E|B) = p(E)$.
- Similarly, we can assume that the radio broadcast does not depend on your house being robbed, such that $p(R|E, B) = p(R|E)$.
- Lastly, we can assume that your home alarm is independent of the radio broadcast *when conditioned on the occurrence of an earthquake*, that is $p(A|R, E, B) = p(A|E, B)$.

Note that this last point does not imply that the alarm system is independent of the radio broadcast, $p(A|R) \neq p(A)$. If an earthquake increases the probability of false alarms and the probability of radio broadcast, knowing that there was a radio broadcast increases the probability that the alarm will go off. Those simplifications leads to a system with $8 = 4 + 2 + 1 + 1$ parameters,

$$p(A, R, E, B) = p(A|E, B)p(R|E)p(E)p(B).$$

To start reasoning about the problem, we'll need to plug a few numbers in. We will start by assuming that both earthquakes and burglars are quite rare, and that each day has a $1/1000$ chance of seeing any of them occurring, translating to a frequency of roughly one earthquake/robbery every three years.

$$p(E) = 10^{-3}, \qquad p(B) = 10^{-3}.$$

We will assume that the radio is perfectly reliable, such that

$$p(R = 1|E = 1) = 1, \qquad p(R = 1|E = 0) = 0.$$

For the alarm, we will assume that it can send false alarms, with a rate $f = 1/1000$, that a burglar has a $\alpha_B = 99/100$ chance of triggering it while an earthquake only has a $\alpha_E = 1/100$ chance of triggering it. This yields the following table of probabilities,

$$\begin{aligned}
&p(A = 1|B = 0, E = 0) = f = 0.001,\\
&p(A = 1|B = 0, E = 1) = 1 - (1 - f)(1 - \alpha_E) = 0.01099,\\
&p(A = 1|B = 1, E = 0) = 1 - (1 - f)(1 - \alpha_B) = 0.99001,\\
&p(A = 1|B = 1, E = 1) = 1 - (1 - f)(1 - \alpha_B)(1 - \alpha_E) = 0.9901099.
\end{aligned}$$

Using Bayes' Theorem, we can now reason about various scenarios. Given the information that our alarm went off, but without knowledge of the radio broadcast, we can compute the probability that there was a break-in. Plugging the numbers above in the following equation yields

$$\begin{aligned}
p(B = 1|A = 1) &= \frac{p(A = 1, B = 1)}{p(A = 1)},\\
&= \frac{\sum_{R,E} p(A = 1, B = 1, R, E)}{\sum_{B,R,E} p(A = 1, B, R, E)} \quad = 0.495.
\end{aligned}$$

If, in addition, we know that the radio broadcast an announcement about an earthquake,

$$p(B=1|A=1,R=1) = \frac{p(A=1,B=1,R=1)}{p(A=1)}, \\ = \frac{\sum_E p(A=1,B=1,R=1,E)}{\sum_{B,R,E} p(A=1,B,R,E)} = 0.08.$$

The phenomenon of reducing the probability of an event by adding more observation is often referred to as *explaining away* the break-in as a reason for the alarm.

THE GENERAL RECIPE for probabilistic reasoning can be summarized as identifying the relevant variables, here A, R, E, B, defining the *joint probability* distribution, $p(A, R, E, B)$, also known as the generative model, fix some variables through *observations*, $A = 1$, and finally perform *inference* through Bayes' Theorem, which requires *marginalizing* the latent variables not being inferred.

Graphical representation of (in)dependence

A visual summary of our probabilistic model, shown in Figure [3], displays the relationship between variables as a directed graphs. Observable variables are shown in dark nodes, while latent variables are shown in white nodes and directed edges indicate dependencies.

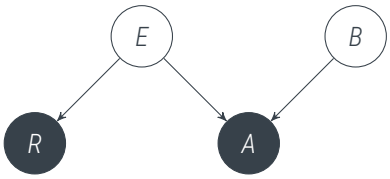


Figure 3: **Graphical model** for the earthquake burglar example.

Definition 8 (Bayesian Network)**.** A *Directed Graphical Model* (DGM), aka. Bayesian Network is a probability distribution over variables $X_1, ..., X_D$ that can be written as

$$p(X_1, ..., X_D) = \prod_{i=1}^{D} p(X_i|pa(X_i)),$$

where $pa(Xi)$ are the parental variables of X_i, that is, $X_i \notin pa(X_j) \forall X_j \in pa(X_i)$. A DGM can be represented by a *Directed Acyclic Graph* (DAG) with the propositional variables as nodes, and arrows from parents to children.

By the product rule, every joint probability distribution can be factorized into a dense DAG. The following factorization,

$$p(A, E, B, R) = p(A|E, B, R)p(R|E, B)p(E|B)p(B),$$

leads to one DAG, but this other factorization leads to another graphical representation where the direction of each edge is reversed,

$$p(A, E, B, R) = p(B|A, E, R)p(E|A, R)p(R|A)p(A).$$

The direction of the arrows is not a causal statement, but an indication of dependence. Representing the probabilistic model as a DAG might thus not always be useful, but it is when the structure reveals

independence, as in Figure 3, where the factorization leading to the graphical model is

$$p(A, E, B, R) = p(A|E, B)p(R|E)p(E)p(B).$$

Definition 9 (Independence)**.** Two variables A and B are independent if and only if their joint distributions factorizes into so-called marginal distributions, i.e.

$$p(A, B) = p(A)p(B).$$

In that case $p(A|B) = p(A)$ and we use the notation $A \perp\!\!\!\perp B$. Information about B does not give information about A and vice versa.

Note that $p(A|B) = p(A)$ is an equivalent statement to $p(A, B) = p(A)p(B)$ due to the definition of conditional probability, as $p(A, B) = p(A|B)p(B)$.

Definition 10 (Conditional independence)**.** Two variables A and B are conditionally independent given variable C if and only if their conditional distribution factorizes,

$$p(A, B|C) = p(A|C)p(B|C).$$

In that case we have $p(A|B, C) = p(A|C)$, i.e. in light of information C, B provides no further information about A. Notation: $A \perp\!\!\!\perp B|C$.

Independence and conditional independence are related but do not imply each other. Consider the following example; given two coins, let A be the event that the first coin shows head, let B be the event that the second coin shows head and C be the event that both coins show the same result. $A \perp\!\!\!\perp B$ should be intuitive, as the result of a coin toss does not give information on another coin toss, but we also have that $A \perp\!\!\!\perp C$ and $B \perp\!\!\!\perp C$. To see why, fix the value of a coin, say A is true. Then, we have that $p(C|A = 1) = p(B) = 1/2$, which is equal to $p(C)$. However, we have that $A \not\perp\!\!\!\perp B|C$, as knowing the output of the second coin and whether both coins show the same face gives full information on the result of the first coin.

READING INDEPENDENCE from DAGs can be made easier if we consider subsets of variables. For subsets of one and two variables, the independence structure is easy to see, but tri-variate structures get interesting. Note, however, that it is not possible to deduce

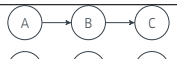
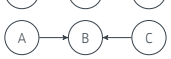
$p(A, B, C)$	DAG	Independence	but!
$p(C\|B)p(B\|A)p(A)$		$A \perp\!\!\!\perp C\|B$	$A \not\perp\!\!\!\perp C$
$p(A\|B)p(C\|B)p(B)$		$A \perp\!\!\!\perp C\|B$	$A \not\perp\!\!\!\perp C$
$p(B\|A, C)p(A)p(C)$		$A \perp\!\!\!\perp C$	$A \not\perp\!\!\!\perp C\|B$

Figure 4: **Independence structure for tri-variate subgraphs.**

more complex relations by looking at those simple subgraphs - it is possible, for example, that $A \perp\!\!\!\perp C|B$ but bringing in a new variable D, we could have that $A \not\perp\!\!\!\perp C|B, D$. Also, a single DAG might not reveal all the independence properties of a probabilistic model.

THE DAG FOR FOR THE TWO COINS EXAMPLE is not unique, for example. Computing the probabilities involved,

$$p(A=1)=\frac{1}{2}, \qquad p(B=1)=\frac{1}{2},$$

$$p(C=1|A=1,B=1)=1, \qquad p(C=1|A=0,B=1)=0,$$

$$p(C=1|A=1,B=0)=0, \qquad p(C=1|A=0,B=0)=1,$$

we have that the conditional probabilities imply that

$$p(A|B)=p(A),\ p(B|C)=p(B),\ p(C|A)=p(C),\ p(C|B)=p(C),$$

leading to the following three possible factorizations,

$$p(A,B,C)=p(C|A,B)p(A)p(B),$$

$$p(A,B,C)=p(A|B,C)p(B)p(C),$$

$$p(A,B,C)=p(B|A,C)p(A)p(C),$$

each mathing a DAG in Figure 5.

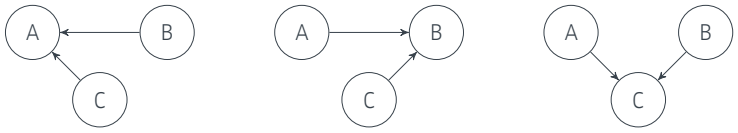


Figure 5: **Three possible DAG for the two coin example.**

Summary of independence and graphical representation

Multivariate distributions can have exponentially many degrees of freedom, but (conditional) independence can help reduce this complexity to make things tractable. Directed graphical models provide a notation from which conditional independence can be read using simple rules. However, every probability distribution can be represented as a DAG, and not every independence structure of a distribution is captured by a single DAG. Conditional independence is a tool, and may be required, to keep inference tractable in multivariate problems.

Probabilities over Continuous Variables

Probability theory extends propositional logic with propositional variables $A, ..., Z \in \{0, 1\}$ ranging over the space of all possible boolean assignments Ω, with a normalized probability measure $p : \Omega \to [0, 1]$, such that $\sum_{w \in \Omega} p(w) = 1$. Discrete probability theory can also handle variables in a discrete set $\Omega = \{0, 1, ...\}$ using a similar probability measure, while continuous probability theory uses the *probability density function* $f : \Omega \to \mathbb{R}_+$ to handle continuous sample spaces, such as $\Omega = \mathbb{R}$, with the property that $\int_{w \in \Omega} f(w)\, \mathrm{d}w = 1$.

Let X be a variable taking real values, $X \in \mathbb{R}$, and define the following events: $A = (X \leq a), B = (X \leq b), W = (a < X \leq b)$. As A and W are mutually exclusive, by the sum rule we have that

$$p(B) = p(A) + p(W), \qquad p(W) = p(B) - p(A).$$

Thinking of the events as functions of the limits they are checking, $F(x) = p(X \leq x)$, we can use their derivative $f(x) = \frac{\partial}{\partial x} F(x)$, to express this problem using integrands,

$$p(a < X \leq b) = F(b) - F(a) = \int_a^b f(x)\, \mathrm{d}x.$$

F is called the *cumulative distribution function* (CDF) and f is the *probability density function* (PDF).

The Product and the Sum rules apply to the probability density function, and taken together imply Bayes' rule.

$$f(x, y) = f(x|y) f(y) \qquad \text{product rule,}$$
$$f(x) = \int f(x, y)\, \mathrm{d}y \qquad \text{sum rule,}$$

Those rules, however, do not apply to the cumulative distribution function $F(x)$. Figure 6 illustrates the joint, marginal and conditional densities on a two-dimensional example.

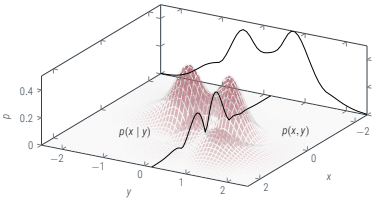


Figure 6: **Joint probability density function for two variables**, highlighting the marginal and conditional probability densities.

The base measure

Probability density function are only defined relatively to a *base measure*, and changes of variables need additional care.

Theorem 11 (Change of variables for PDFs)**.** Consider a continuous random variables X with PDF $f_X(x)$ over $c_1 < x < c_2$, and a monotonic, invertible, differentiable function u, with inverse v. The random variable $Y = u(X)$ follows a PDF f_Y that can be defined w.r.t. f_X as

$$f_Y(y) = f_X(v(y)) \left| \frac{\partial v(y)}{\partial y} \right| = f_X(v(y)) \left| \frac{\partial u(x)}{\partial x} \right|^{-1}.$$

To see why, assume that u is monotically increasing, $u'(X) > 0$, and let $d_1, d_2 = u(c_1), u(c_2)$. f_Y is defined on $d_1 < y < d_2$ and we have that the CDF F_y is defined w.r.t. the PDF f_X as

$$F_y(y) = P(Y \leq y) = P(u(X) \leq y) = P(X \leq v(y)) = \int_{c_1}^{v(y)} f(x)\,\mathrm{d}x,$$

and the PDF f_Y follows from the CDF F_y as

$$f_Y(y) = \frac{\partial F_Y(y)}{\partial y} = f_X(v(y)) \frac{\partial v(y)}{\partial y}.$$

To obtain the absolute value, repeat the previous steps with a monotically decreasing change of variable such that $u'(x) < 0$.

Formal definition

We now give the formal definition of densities and probabilities on continuous spaces. This is mostly useful to understand other reference material on the subject - don't worry if the definitions sound too convoluted at first.

Definition 12 (probability measure)**.** Let $(\Omega, \mathcal{F}, p)$ be a measurable space (i.e. a sample space Σ measurable by a σ-algebra F) with a positive measure p on (Ω, F). If the measure is normalized, $p(\Omega) = 1$, it is called a probability measure.

Definition 13 (probability distribution)**.** Let X be a measurable function from (Ω, F, p) to $(\mathcal{X}, \mathcal{A})$, i.e., a function between two measurable spaces such that the pre-image of every measurable set is measurable. Those functions provide the formal definition for *random variables*. The probability distribution of X is the pushforward $X_* p$ of p. That is, it is the measure satisfying $X_* p = pX^{-1}$.

Definition 14 (probability density)**.** Let X be a random variable with distribution X_*P of $(\mathcal{X}, \mathcal{A})$, and μ be a reference measure on $\mathcal{X}$, for example the Lebesgue measure on $\mathbb{R}^N$. The *probability density function* (PDF, or simply *density*) of X is a measurable function f on $(\mathcal{F}, \mathcal{A})$ such that, for any measurable set $A \in \mathcal{A}$,

$$X_*P(A) = \int_{X^{-1}A} \mathrm{d}X_*P = \int_A f \,\mathrm{d}\mu.$$

This property is also written, short-hand, as

$$f = \frac{\partial X_*P}{\partial \mu}$$

and f is also called the *Radon-Nikodym derivative* of X_*P with respect to μ.

For our purposes, consider the following, hopefully simpler description.

A *probability measure* $p : \Omega \to \mathbb{R}_+$ is the function giving the *probability* of an event $E \in \Omega$, $p(E)$.

The probability measure is defined with respect to a sample space Ω equipped with a σ-Field $\mathcal{F}$ – one way of thinking about $\mathcal{F}$ is that it is the set of all possible subsets of Ω on which probabilities can be defined – giving a *base space* $(\Omega, \mathcal{F}, p)$.

A *random variable* is a variable obtained by mapping from the base space $(\Omega, \mathcal{F}, p)$ to another measure space, e.g. for a derived variable X, $(\mathcal{X}, \mathcal{A}, p_X)$. The probability measure of the derived variable X, p_X, is called a *distribution*. We will use *measure* and *distribution* interchangeably, but formally a measure induces a distribution.

A *probability density* is the function $p_X : \mathcal{X} \to \mathbb{R}_+$ such that if P is the measure on X,

$$P(X \in A \subset \mathcal{X}) = \int_A p_X(x) \,\mathrm{d}x.$$

We write $p(x) = {}^{\partial P(X)}/_{\partial x}$, but it might not be true for some cases when the traditional notion of differentiability does not hold for P - this is where the additional formality in the previous definitions helps in making the theory mathematically correct.

Example: inference over binoculars with the binomial distribution

What is the probability - π - for a person to be wearing glasses? As we do not know this probability, we can model our uncertainty about it as using a random variable π ranging in $[0, 1]$. To answer the question, we can collect some observations X and use inference;

$$p(\pi|X) = \frac{p(X|\pi)p(\pi)}{p(X)} = \frac{p(X|\pi)p(\pi)}{\int p(X|\pi)p(\pi)\,\mathrm{d}\pi}.$$

To define the *prior* distribution, we can start with a uniform distribution, $p(\pi) = 1$ if $\pi \in [0, 1]$, 0 elsewhere. Assuming we sample observations independently, the *likelihood* of a positive or negative sample, given knowledge of π, is simply

$$p(X = 1|\pi) = \pi, \qquad p(X = 0|\pi) = 1 - \pi.$$

For multiple observations, this process gives rise to a *binomial distribution*, illustrated in Figure 7, traditionally defined in terms of coin tosses; what is the probability distribution over the number of times a coin will show head over N tosses given a probability of landing head of f. The probability of sampling n positive and m negative observations if the probability of an independent positive observation is given by π is

$$p(n, m|\pi) = \binom{n}{n+m} \pi^n (1 - \pi)^m.$$

Plugging this back in the computation of the posterior yields

$$p(\pi|n, m) = \frac{\binom{n}{n+m} \pi^n (1 - \pi)^m \cdot 1}{\int \binom{n}{n+m} \pi^n (1 - \pi)^m \cdot 1 \, \mathrm{d}\pi}. = \frac{\pi^n (1 - \pi)^m \cdot 1}{\int \pi^n (1 - \pi)^m \cdot 1 \, \mathrm{d}\pi}.$$

A nice choice of prior $p(\pi)$ to make the computation easy is the *Beta distribution*, with parameters $a, b > 0$,

$$p(\pi) = \frac{\pi^{a-1}(1 - \pi)^{b-1}}{Z},$$

where Z is a normalization constant to ensure that $\int_0^1 p(\pi) \, \mathrm{d}\pi = 1$ and is given by the *Beta function*,

$$Z = B(a, b) = \int_0^1 \pi^{a-1}(1 - \pi)^{b-1} \, \mathrm{d}\pi.$$

The uniform distribution can be represented as a Beta distribution[4]with parameters $a = b = 1$. Given a Beta(a, b) prior, n positive and m negative additional observations, the posterior is then given by

$$p(\pi|n, m) = \frac{\pi^{n+a-1}(1 - \pi)^{m+b-1}}{B(a + n, b + m)}.$$

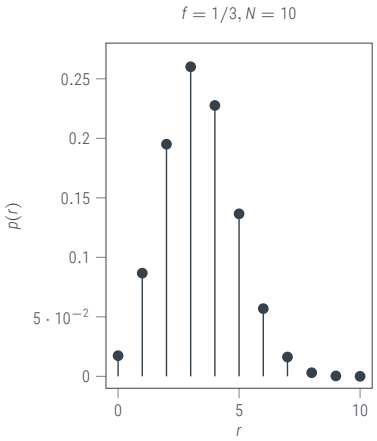


Figure 7: **Probability distribution for the number of heads** in $N = 10$ coin flips with a probility of landing head of $f = 1/3$.

[4] wikipedia.org/wiki/Beta_distribution

Gaussian probability distributions

Now that we have introduced the basics of continuous probabilities, let us delve in more details into the Gaussian distribution.

Definition 15 (Univariate Gaussian distribution)**.** Parametrized by two scalars, the mean μ and variance σ^2, the density function of the univariate Gaussian distribution is

$$\mathcal{N}\left(x;\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

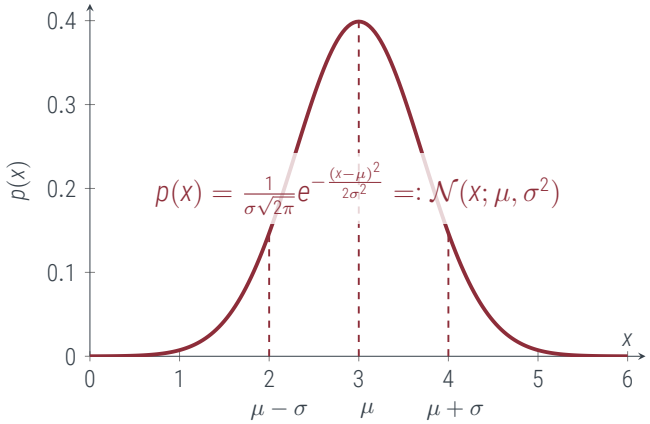


Figure 8: **Univariate Gaussian probability distribution,** also called here shown with $\mu = 3$, $\sigma^2 = 1$.

The more interesting, but more difficult to visualize on a 2D piece of paper, is the multivariate extension;

Definition 16 (Multivariate Gaussian distribution)**.** Parametrized my a mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$ and a positive definite[5] covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$, the density of the n-dimensional Gaussian is given by

$$\mathcal{N}\left(\boldsymbol{x},\boldsymbol{\mu},\boldsymbol{\Sigma}\right) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

As $\int \mathcal{N}\left(x;\mu,\sigma^2\right) \mathrm{d}x = 1$ and $\mathcal{N}\left(x;\mu,\sigma^2\right) > 0 \forall x$, $\mathcal{N}$ is a well defined probability measure. Gaussians have many useful properties, mostly stemming from being exponentiation of quadratic terms, making equi-probability lines be elipsoids (See Figure 9), which we will now go into. Those properties make it very convenient to perform inference using Gaussian random variables using the simple tools of linear algebra.

[5] A symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is positive definite if, for all vectors $\boldsymbol{v} \in \mathbb{R}^n$, $\boldsymbol{v}^\top \boldsymbol{A}\boldsymbol{v} > 0$ (or positive semi-definite if $\geq$).

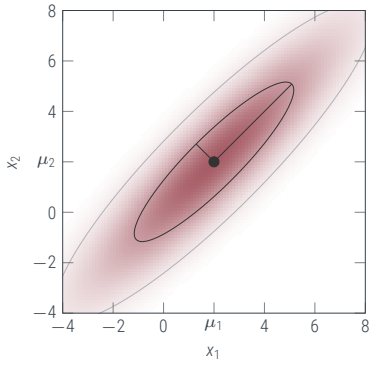


Figure 9: **Two-dimensional Gaussian distribution.**

THE GAUSSIAN IS ITS OWN CONJUGATE PRIOR, meaning that given a Gaussian prior $p(x)$ and a Gaussian Likelihood $p(y|x)$, the posterior $p(x|y)$ also is a Gaussian (see Figure 10). For

$$p(x) = \mathcal{N}\left(x; \mu, \sigma^2\right) \quad \text{and} \quad p(y|x) = \mathcal{N}\left(y; x, \nu^2\right),$$

the posterior is given by

$$p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x)p(x)\,\mathrm{d}x} = \mathcal{N}\left(x; m, s^2\right),$$

$$\text{where } m = \frac{\sigma^{-2}\mu + \nu^{-2}y}{\sigma^{-2} + \nu^{-2}} \quad \text{and} \quad s^2 = \frac{1}{\sigma^{-2} + \nu^{-2}}.$$

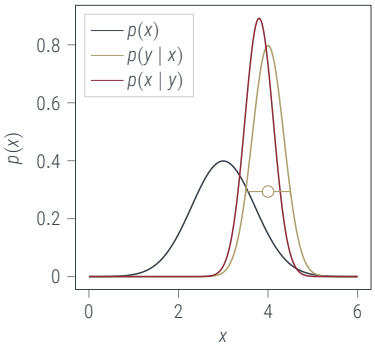


Figure 10: **Gaussian Prior, Likelihood and Posterior.**

GAUSSIANS ARE CLOSED UNDER MULTIPLICATION.

$$\mathcal{N}(x; \boldsymbol{a}, \boldsymbol{A})\,\mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{B}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{c}, \boldsymbol{C})\,Z$$

$$\text{where } \boldsymbol{C} = (\boldsymbol{A}^{-1} + \boldsymbol{B}^{-1})^{-1}, \quad \boldsymbol{c} = \boldsymbol{C}(\boldsymbol{A}^{-1}\boldsymbol{a} + \boldsymbol{B}^{-1}\boldsymbol{b}),$$

$$\text{and } Z = \mathcal{N}(\boldsymbol{a}; \boldsymbol{b}, \boldsymbol{A} + \boldsymbol{B}).$$

GAUSSIANS ARE CLOSED UNDER LINEAR PROJECTIONS. If $\boldsymbol{x}$ is a random variable distributed according to $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{A}$ is an arbitrary matrix (of according size), then $\boldsymbol{A}\boldsymbol{x}$ is distributed according to

$$\mathcal{N}\left(\boldsymbol{A}\boldsymbol{x}; \boldsymbol{A}\boldsymbol{\mu}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top\right).$$

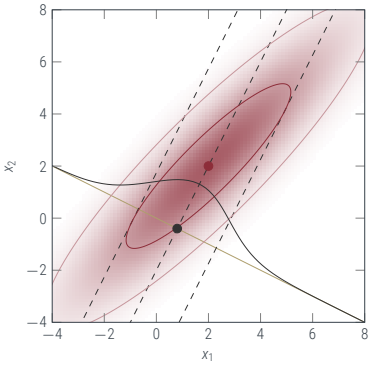


Figure 11: **Linear projection of a Gaussian distributed random variable.**

GAUSSIANS ARE CLOSED UNDER MARGINALIZATION. Assuming that $\boldsymbol{x}, \boldsymbol{y}$ are distributed according to

$$\mathcal{N}\left(\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{pmatrix}; \begin{pmatrix}\boldsymbol{\mu}_x\\ \boldsymbol{\mu}_y\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy}\\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy}\end{pmatrix}\right),$$

marginalization is a special case of a linear projection, projecting with a matrix

$$\boldsymbol{A} = \begin{pmatrix}\boldsymbol{I} & 0\\ 0 & 0\end{pmatrix}.$$

This can also be seen by using the sum rule,

$$\int p(\boldsymbol{x}, \boldsymbol{y})\,\mathrm{d}\boldsymbol{y} = \int p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{y} = p(\boldsymbol{x})\int p(\boldsymbol{y}|\boldsymbol{x})\,\mathrm{d}\boldsymbol{y} = p(\boldsymbol{x}),$$

GAUSSIANS ARE CLOSED UNDER CONDITIONING of scaled vectors, i.e.,

$$p(\boldsymbol{x}|\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}) = \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{y})},$$

$$= \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top)^{-1}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\mu}), \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top)^{-1}\boldsymbol{A}\boldsymbol{\Sigma}\right)$$

BAYES' THEOREM ALSO LEADS TO GAUSSIAN VARIABLES, thanks to conditioning and marginalization.

$$\begin{aligned}
\text{If } p(\boldsymbol{x}) &= \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \\
\text{and } p(\boldsymbol{y}|\boldsymbol{x}) &= \mathcal{N}\left(\boldsymbol{y}, \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}, \boldsymbol{\Lambda}\right), \\
\text{then } p(\boldsymbol{y}) &= \mathcal{N}\left(\boldsymbol{y}; \boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{\Lambda} + \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top\right)
\end{aligned}$$

$$\begin{aligned}
\text{and } p(\boldsymbol{x}|\boldsymbol{y}) &= \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu} + \underbrace{\boldsymbol{\Sigma}\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top + \boldsymbol{\Lambda})^{-1}}_{\text{Gain}}\underbrace{(\boldsymbol{y} - (\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}))}_{\text{Residual vector}}, \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{A}^\top\underbrace{(\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top + \boldsymbol{\Lambda})^{-1}}_{\text{Gram matrix}}\boldsymbol{A}\boldsymbol{\Sigma}\right) \\
&= \mathcal{N}\left(\boldsymbol{x}; \underbrace{(\boldsymbol{\Sigma}^{-1} + \boldsymbol{A}^\top\boldsymbol{\Lambda}^{-1}\boldsymbol{A})}_{\text{Posterior Precision matrix}}^{-1}(\boldsymbol{A}^\top\boldsymbol{\Lambda}^{-1}(\boldsymbol{y} - \boldsymbol{b}) + \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}), \underbrace{(\boldsymbol{\Sigma}^{-1} + \boldsymbol{A}^\top\boldsymbol{\Lambda}^{-1}\boldsymbol{A})}_{\text{Posterior Precision matrix}}^{-1}\right)
\end{aligned}$$

GAUSSIAN DISTRIBUTIONS PROVIDE THE LINEAR ALGEBRA OF INFERENCE. If all variables in a generative model are linearly related, and the distributions of the parent variables are Gaussian, then all conditionals, joints and marginals are Gaussian, with means and covariances computable by linear algebra operations.

Parametric Gaussian Regression

This chapter will show how we can use Gaussian distributions to learn/infer functions in the base case of supervised learning.

Assume we have observations X, y, representing input-output pairs, and we would like to learn the relation between them: $f(x) \approx y$. To learn f, we can assume that the outputs are distributed according to

$$p(y|f(x)) = \mathcal{N}\left(y; f(x), \sigma^2 I\right).$$

Here, $y \in \mathbb{R}^N$, indicating we have N independent samples, and $X \in \mathbb{R}^{N \times D}$, indicating a D-dimensional observation for each sample. For a starting example, assume that the inputs are 1-dimensional, $D = 1$, as in the example in Figure 12.

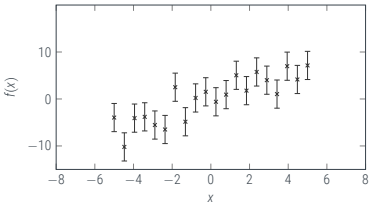


Figure 12: **Small dataset example**

Assume f is a linear function, with weights w_1, w_2;

$$f(x) = w_1 + w_2 x.$$

We introduce an abstraction, the feature mapping ϕ giving the feature vector ϕ_x;

$$\phi(x) = \phi_x = \begin{pmatrix} 1 \\ x \end{pmatrix},$$

such that we can rewrite the function as $f(x) = \phi_x^\top w$. Why this is useful will become clearer when we will expand on this example, but for the moment you can just think about it as a nicer notation.

The last piece we need to perform inference is a prior on the weights $[w_1, w_2]$. Assuming that $w \sim \mathcal{N}(\mu, \Sigma)$, everything is Gaussian, which makes our life easier as the whole inference process will be computable using linear algebra. Figure 13 shows an example of such a prior on w and the resulting prior on the function f,

$$p(f) = \mathcal{N}\left(f; \phi_x^\top \mu, \phi_x^\top \Sigma \phi_x\right)$$

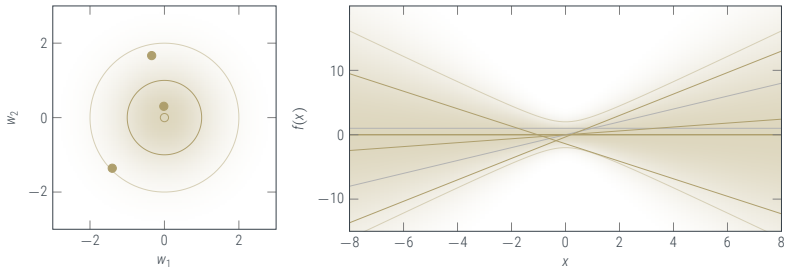


Figure 13: **Gaussian prior on the weights,** and matching prior on the function space, using $w \sim \mathcal{N}(0, I)$.

To RECAP THE NOTATION, we have a dataset on inputs $X \in \mathbb{R}^N$ and outputs $y \in \mathbb{R}^N$, a feature vector $\phi_x = [1, x]^\top$ for each data point and feature matrix

$$\phi_X = \begin{pmatrix} \phi_{x_1} & \dots & \phi_{x_N} \end{pmatrix} = \begin{pmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \end{pmatrix}.$$

We can also think of the function f, as applied to the data X, as a vector;

$$f_X = f(X) = \phi_X^\top w = \begin{pmatrix} \phi_{x_1}^\top w \\ \dots \\ \phi_{x_N}^\top w \end{pmatrix} = \begin{pmatrix} f(x_1) \\ \dots \\ f(x_N) \end{pmatrix}$$

Assuming that the likelihood is given by

$$p(y|w, \phi_X) = \mathcal{N}\left(y; \phi_X^\top w, \sigma^2 I\right) = \mathcal{N}\left(y; f_X, \sigma^2 I\right)$$

and with a prior on the weights $p(w)$ giving rise to a prior on the function f_X,

$$p(w) = \mathcal{N}\left(w; \mu, \Sigma\right), \qquad p(f_X) = \mathcal{N}\left(\phi_X^\top w; \phi_X^\top \mu, \phi_X \top \Sigma \phi_X\right)$$

we can compute the posterior on the weights using the linear algebra rules of the previous chapter;

$$p(w|y, \phi_X) = \mathcal{N}\Big(w; \underbrace{\mu + \Sigma\phi_X \left(\phi_X^\top \Sigma \phi_X + \sigma^2 I\right)^{-1} (y - \phi_X^\top \mu)}_{\text{posterior mean}}, \underbrace{\Sigma - \Sigma\phi_X \left(\phi_X^\top \Sigma \phi_X + \sigma^2 I\right)^{-1} \phi_X^\top \Sigma}_{\text{posterior covariance}}\Big).$$

In the same way the prior on the weights $p(w)$ gives rise to a prior on the functions $p(f) = \mathcal{N}\left(f_X; \phi_X^\top \mu, \phi_X^\top \Sigma \phi_X\right)$, the posterior on the weights $p(w|y, \phi_X)$ gives rive to a posterior on the functions;

$$p(f_X|y, \phi_X) = \mathcal{N}\Big(\phi_X^\top w; \phi_X^\top \mu + \phi_X^\top \Sigma \phi_X \left(\phi_X^\top \Sigma \phi_X + \sigma^2 I\right)^{-1} (y - \phi_X^\top \mu), \\ \phi_X^\top \Sigma \phi_X - \phi_X^\top \Sigma \phi_X \left(\phi_X^\top \Sigma \phi_X + \sigma^2 I\right)^{-1} \phi_X^\top \Sigma \phi_X\Big).$$

Those equations can be difficult to digest at first glance, but don't be frightened just yet. We will first see some results those equations produce, but will return to them at the end of chapter for more details.

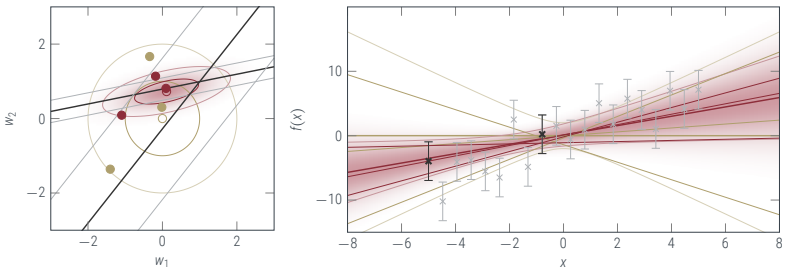


Figure 14: **Posterior on the weights,** and matching posterior on the function, after seeing two datapoints.

Feature functions

So far, we have shown how to perform linear regression using a feature vector $\phi_x = [1 \quad x]^\top$. However, the feature vector is not only a nice notation tool; it can also be redefined to capture more complex structure.

Instead of limiting ourselves to linear relationships, we can define a new feature function to capture second and third order polynomials,

$$\phi(x) = \phi_x = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix}^\top,$$

or even up to order 7 polynomials of x,

$$\phi(x) = \phi_x = \begin{bmatrix} 1 & x & x^2 & x^3 & \dots & x^7 \end{bmatrix}^\top,$$

or using sines and cosines to get a Fourier regression.

$$\phi(x) = \phi_x = \begin{bmatrix} \cos(x) & \cos(2x) & \cos(3x) & \sin(x) & \sin(2x) & \sin(3x) \end{bmatrix}^\top,$$

or any combination of step functions, Legendre or Laguerre polynomials, bell curves, or any function one can think of.

Each of those feature functions gives rise to a different prior and the characteristics of the posterior on f will be very different, but the inference can still be performed using the same linear algebra as before, as the uncertainty is over the weights w. The choice of features is essentially unconstrained.

This poses the question of how best to choose the feature functions. We will see in the next chapters that we can define parametrized family of features, where the parameters controlling the features used are called hyperparameters and optimize the hyperparameters to find a good feature space using *Hierarchical Bayesian Inference*, and even perform inference directly on *infinite* feature functions using *Gaussian Processes*.

The following page illustrates different choices of feature functions, with resulting priors and posteriors on f_X, on the following, slightly more complex dataset;

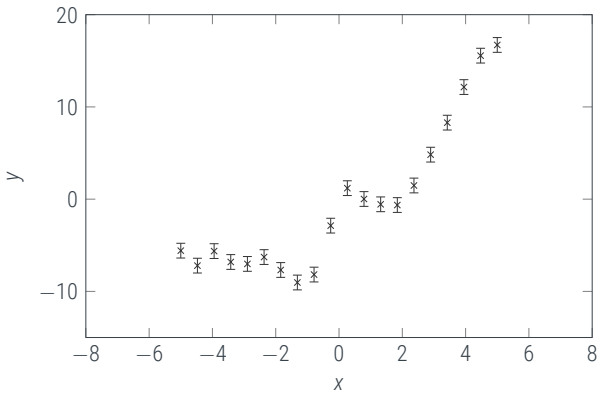
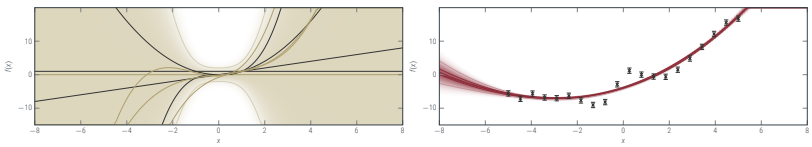
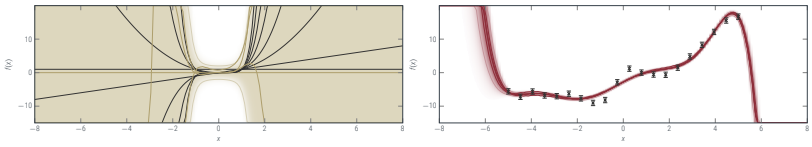


Figure 15: **Non-linear, one-dimensional dataset.**



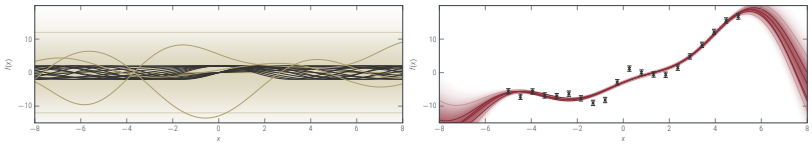
Cubic regression, using polynomials up to the third order

$$\phi_x = [1 \; x \; x^2 \; x^3]^\top$$



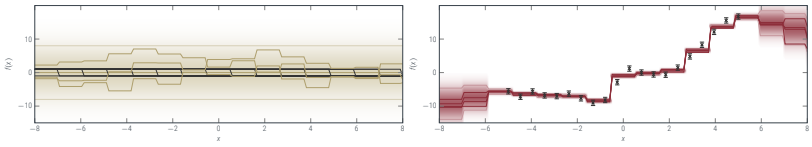
Septic(?) regression, using polynomials up to the seventh order

$$\phi_x = [1 \; x \; x^2 \; x^3 \; x^4 \; x^5 \; x^6 \; x^7]^\top$$



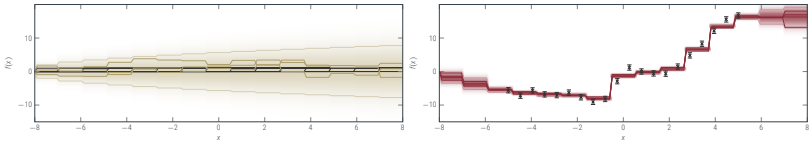
Fourier regression, using sines and cosines with different frequencies, e.g.

$$\phi_\alpha(x) = [\cos(\alpha x) \qquad \sin(\alpha x)]$$



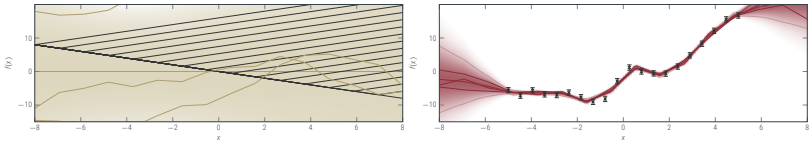
Pixel regression, using functions of the form

$$\phi_\alpha(x) = 1 \text{ if } x \in [\alpha, \alpha+1], -1 \text{ otherwise}$$



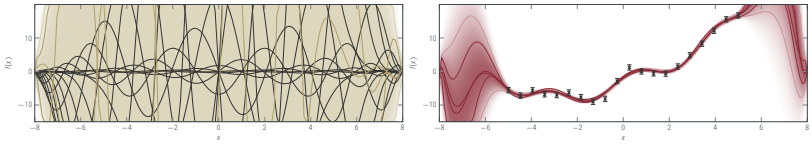
Switch regression, using functions of the form

$$\phi_\alpha(x) = 1 \text{ if } x < \alpha, 0 \text{ otherwise}$$



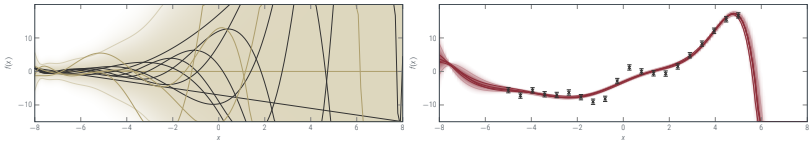
V regression, using differently shifted absolute values of the form

$$\phi_\alpha(x) = |x - \alpha| - \alpha.$$



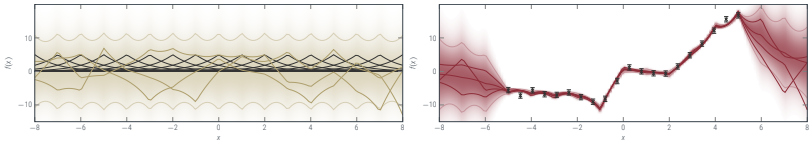
Legendre regression, using Legendre polynomials $b^k P_k(x)$,

$$P_k(x) = 1/2^k k! \; \partial^k/\partial x^k \; (x^2 - 1)^k$$



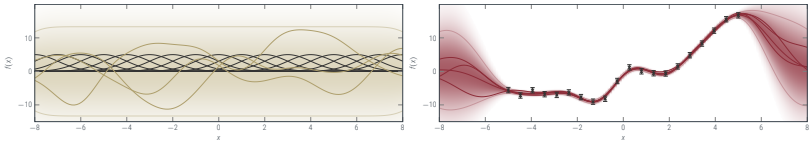
Laguerre regression, using Laguerre polynomials $L_k(x)$,

$$L_k(x) = 1/k! (\partial/\partial x - 1)^k x^k$$



Eiffel Tower regression, using Laplace distributions with different locations,

$$\phi_\alpha(x) = e^{-|x-\alpha|}$$



Bell curve regression, using Gaussian distributions with different locations,

$$\phi_\alpha(x) = e^{-(x-\alpha)^2}$$

More details on the posterior equation

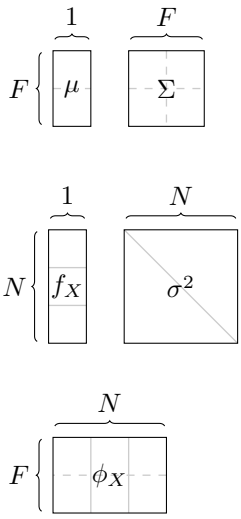
In the notation we introduced, the posterior can be a bit of a mess to write, especially the posterior on function values. We give more details here, in the hope of making the structure more visible.

Dimensionality

It is useful to look at the dimensionality of each variable to get a sense of the computations involved. Assume we have a dataset with N data points and choose a set of F features to fit.

- The quantities involved in the prior are the prior mean μ and prior covariance matrix Σ of the weights w; $p(w) = \mathcal{N}\left(w; \mu, \Sigma\right)$, where μ is a F-dimensional vector and Σ a $[F \times F]$ matrix.
- The quantities involved in the likelihood are the function vector f_X and noise matrix $\sigma^2 I$; $p(y) = \mathcal{N}\left(y; f_X, \sigma^2 I\right)$, where f_X is a N-dimensional vector and $\sigma^2 I$ a $[N \times N]$ (diagonal) matrix for N data points.
- The feature function generates a $[N \times F]$ matrix that links the $[N \times N]$ data space and the $[F \times F]$ feature space.

Figures 16 and 17 illustrate the dimensionality of the computations required for the posterior of the weights and function values.



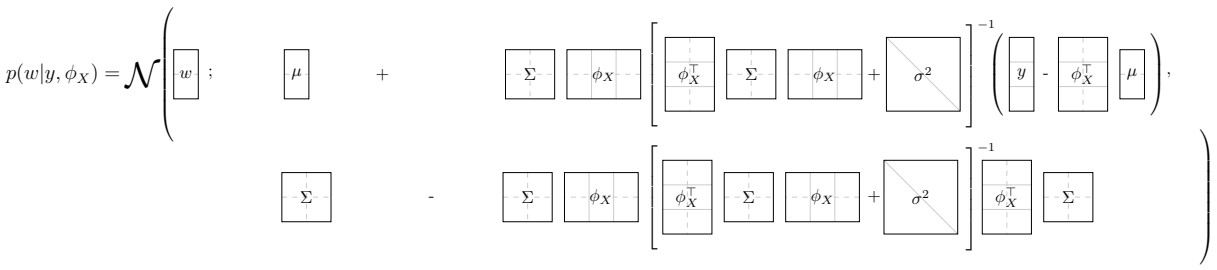


Figure 16: **Weight posterior,** with dimensional representation

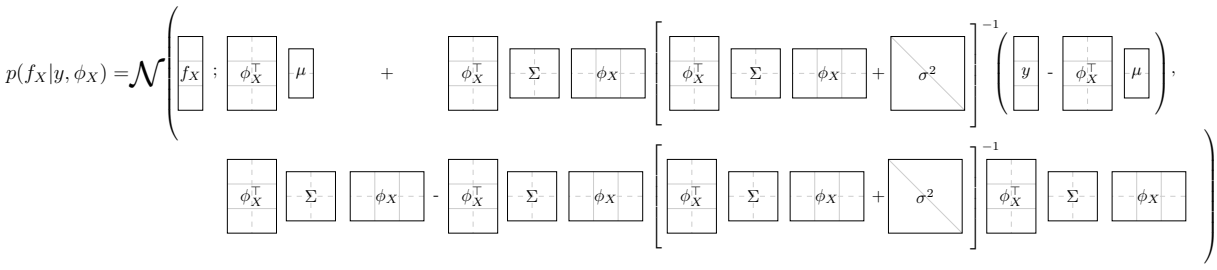


Figure 17: **Function posterior,** with dimensional representation.

The Matrix Inversion Lemma

The computations outlined above require the inversion of $[N \times N]$ matrices. If $F \ll N$, it can be beneficial to perform those inversion if the $[F \times F]$ space, which is possible using the *Matrix Inversion Lemma*, also known as the *Woodbury Matrix Identity*[6]. In its general form, it states that

[6] wikipedia.org/wiki/Woodbury_matrix_identity

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1},$$

where A, U, C, V are matrices of size $[N \times N]$, $[N \times F]$, $[F \times F]$ and $[F \times N]$. Applying those identities to the computation of the posterior of the weights, $p(w|y, \phi_X)$, we get that

$$\text{posterior mean:} \quad \underbrace{\left(\phi_X^\top \Sigma \phi_X + \sigma^2 I\right)^{-1}}_{[N\times N]}(y - \phi_X^\top \mu) = \underbrace{\left(\Sigma^{-1} + \sigma^{-2}\phi_X \phi_X^\top\right)^{-1}}_{[F\times F]}\left(\Sigma^{-1}\mu + \sigma^{-2}\phi_X y\right)$$

$$\text{posterior covariance:} \quad \Sigma - \Sigma\phi_X \underbrace{\left(\phi_X^\top \Sigma \phi_X + \sigma^2 I\right)^{-1}}_{[N\times N]} \phi_X^\top \Sigma = \underbrace{\left(\Sigma^{-1} + \sigma^{-2}\phi_X^\top \phi_X\right)^{-1}}_{[F\times F]}$$

This identity gives the following ways to compute the posterior, both giving the same solution but of different cost depending on the number of data points N or number of features F, see Figures 18 and 19.

$$\begin{aligned} p(w|y,\phi_X) &= \mathcal{N}\Bigg(w; \quad \mu + \Sigma\phi_X\left(\phi_X^\top\Sigma\phi_X + \sigma^2 I\right)^{-1}(y - \phi_X^\top\mu), \\ &\qquad \Sigma - \Sigma\phi_X\left(\phi_X^\top\Sigma\phi_X + \sigma^2 I\right)^{-1}\phi_X^\top\Sigma\Bigg), \\ &= \mathcal{N}\Bigg(w; \quad \left(\Sigma^{-1} + \sigma^{-2}\phi_X^\top\phi_X\right)^{-1}\left(\Sigma^{-1}\mu + \sigma^{-2}\phi_X y\right), \\ &\qquad \left(\Sigma^{-1} + \sigma^{-2}\phi_X^\top\phi_X\right)^{-1}\Bigg). \end{aligned}$$

Figure 18: **Matrix Inversion Lemma** for the posterior on the weights

$$\begin{aligned} p(f_X|y,\phi_X) &= \mathcal{N}\Bigg(\phi_X^\top w; \quad \phi_X^\top\mu + \phi_X^\top\Sigma\phi_X\left(\phi_X^\top\Sigma\phi_X + \sigma^2 I\right)^{-1}(y - \phi_X^\top\mu), \\ &\qquad \phi_X^\top\Sigma\phi_X - \phi_X^\top\Sigma\phi_X\left(\phi_X^\top\Sigma\phi_X + \sigma^2 I\right)^{-1}\phi_X^\top\Sigma\phi_X\Bigg), \\ &= \mathcal{N}\Bigg(\phi_X^\top w; \quad \phi_X\left(\Sigma^{-1} + \sigma^{-2}\phi_X^\top\phi_X\right)^{-1}\left(\Sigma^{-1}\mu + \sigma^{-2}\phi_X y\right), \\ &\qquad \phi_X\left(\Sigma + \sigma^{-2}\phi_X^\top\phi_X\right)^{-1}\phi_X^\top\Bigg). \end{aligned}$$

Figure 19: **Matrix Inversion Lemma** for the posterior on the function values.

A primer on Kernels

Writing the posterior for the function value can be made less tedious by introducing the shortcuts $m = \phi_X^\top \mu$ and $K = \phi_X^\top \Sigma \phi_X$;

$$p(f_X|y, \phi_X) = \mathcal{N}\Big(w; \underbrace{m + K\left(K + \sigma^2 I\right)^{-1}(y - m)}_{\text{posterior mean}}, \underbrace{K - K\left(K + \sigma^2 I\right)^{-1} K}_{\text{posterior covariance}}\Big).$$

In the Gaussian Process literature, $m_X = \phi_X^\top \mu$ is called the *mean function* and $K_{XX} = \phi_X^\top \Sigma \phi_X$ the *Kernel function*, defining the prior mean and covariance of the function f_X. We will return to those in more details when discussing Gaussian Processes in a later chapter.

Numerical Stability

Inverting matrices is very often subject to numerical stability, which happens when the matrices are close to singular[7]. If you want to compute $x = A^{-1}b$, Numpy will happily invert your matrix if it technically is non-singular, using

```
x = numpy.linalg.inv[8](A) @ b,
```

but the results might be nonsensical if it is *close* to singular. A better option is to ask Numpy to solve the system $Ax = b$ with

```
x = numpy.linalg.solve[9](A, b),
```

which is more stable. If you need to compute the multiplication of A^{-1} with multiple vectors or matrices, and the matrix A is positive definite[10], you can precompute the Cholesky decomposition[11]of A with

```
L = numpy.linalg.cholesky[12](A)
```

and use Scipy's routine to solve $Ax = b$ systems using the Cholesky decomposition as a starting point,

```
x = scipy.linalg.cho_solve[13](L, b).
```

For added stability, if your matrix A is positive definite but close to positive semi-definite, you can try to compute the Cholesky of the slightly modified matrix $A' = A + \epsilon I$, where ϵ is a small constant, say 10^{-6}, to ensure that the eigenvalues of A' are above 0.

[7] wikipedia.org/wiki/Invertible_matrix

[8] Docs for numpy.linalg.inv

[9] Docs for numpy.linalg.inv

[10] wikipedia.org/wiki/Positive-definite_matrix

[11] wikipedia.org/wiki/Cholesky_decomposition

[12] Docs for numpy.linalg.cholesky

[13] Docs for scipy.linalg.cho_solve

Sampling from Gaussians

So far, we have seen how to compute the parameters of a Gaussian distribution, but not how to take sample from such distributions. The Box-Muller Transform, and possible improvement including Marsaglia's polar form and Marsaglia's Ziggurat algorithm, are algorithms to transform a sample from a uniform distribution on $(0, 1)$ to samples distributed according to $\mathcal{N}(0, 1)$. You're encouraged to look them up, especially the Box-Muller Transform[14]which has a nice geometrical interpretation, but we will assume we already have a method to sample from a $\mathcal{N}(0, 1)$ distribution.

[14] wikipedia.org/wiki/Box-Muller_transform

Remember that scaling a Gaussian still gives a Gaussian; taking $p(x) = \mathcal{N}(x; \mu, \Sigma)$ and multiplying it by a matrix A gives the distribution

$$p(Ax) = \mathcal{N}\left(Ax; A\mu, A\Sigma A^\top\right)$$

Given that we have access to samples for $p(x) = \mathcal{N}(x; \mathbf{0}, I)$, using vector addition and matrix multiplication, we can compute

$$p(\mu + Ax) = \mathcal{N}\left(\mu + Ax; \mu, AA^\top\right).$$

We just need to find a matrix A such that $AA^\top = \Sigma$, the covariance matrix we want.

The *eigendecomposition* of $\Sigma = UDU^\top$ gives one option. Remember that U is an orthogonal matrix containing the eigenvectors and D is a diagonal matrix containing the eigenvalues. Using $A = UD^{1/2}$, where the square-root in this case is simply applied elementwise to the elements of the diagonal, gives $AA^\top = \Sigma$.

In practice, though, the *Cholesky decomposition* is easier to compute than the eigendecomposition, roughly $4\times$ faster. For positive definite matrix, which Σ is, the Cholesky decomposition of Σ gives a (upper- or) lower-triangular matrix L such that $LL^\top = \Sigma$.

Note that, for a given data point x sampled from $\mathcal{N}(0, I)$, applying the eigendecomposition scaling $\mu + UD^{1/2}x$ or the Cholesky decomposition scaling $\mu + Lx$ will *not* give the same answer. However, both will follow the same $\mathcal{N}(\mu, \Sigma)$ distribution.

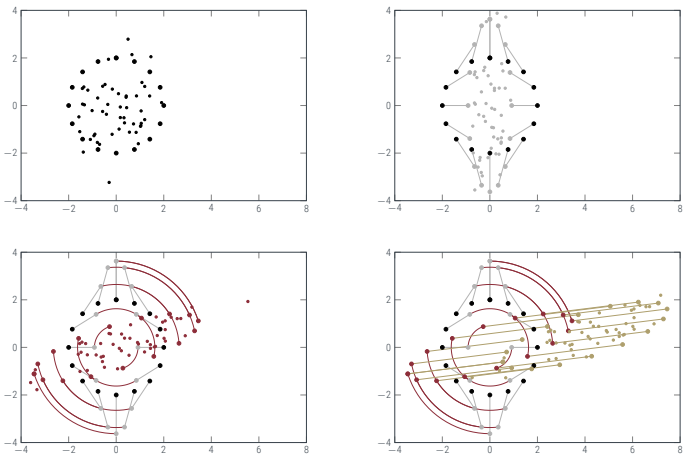


Figure 20: **Sampling from $\mathcal{N}(\mu, \Sigma)$ using an eigendecomposition scaling,**

$$\mu + UD^{1/2}X, \quad X \sim \mathcal{N}(0, 1).$$

From left-to-right, top-to-bottom;

Starting with data from $\mathcal{N}(0, I)$: X
Scaling by eigenvalues: $D^{1/2}X$
Rotating by eigenvectors: $UD^{1/2}X$
Adding of the mean: $\mu + UD^{1/2}X$

Hierarchical Inference: learning the features

We saw how to apply Bayesian inference to infer the parameters of a linear function and that we could change the function being inferred by specifying different features. However, the choice of features can be a daunting task, as any transformation could be picked. We will now see that it is possible to also learn which features to use.

Hierarchical Bayesian Inference

Searching an infinite-dimensional space of feature function is difficult, but we can restrict ourselves to a finite-dimensional sub-space, characterized by some parameters, and search this subspace instead. Consider the family of feature functions parametrized by θ,

$$\phi(x;\theta) = \frac{1}{1+\exp\left(-\frac{x-\theta_1}{\theta_2}\right)},$$

illustrated in Figure 21. The number of feature functions is still infinite as there is an infinite array of choice for (θ_1, θ_2), but the dimensionality of the parametrization is fixed.

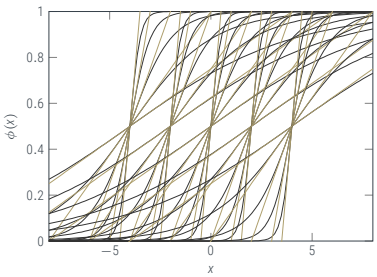


Figure 21: **Parametrized Family of Functions,**

$$\phi(x;\theta) = \left(1+\exp\left(-\frac{x-\theta_1}{\theta_2}\right)\right)^{-1}.$$

The parameter θ_1 controls the intercept - where on the x-axis the function value crosses the $1/2$ point - and the parameter θ_2 controls the slope.

The parameters θ_1 and θ_2 can be treated as unknown parameters, just as the weights w, but it is more difficult to infer them as the likelihood function

$$p(x|w,\theta) = \mathcal{N}\left(y;\phi(x;\theta)^\top w, \sigma^2 I\right)$$

now contains a *non-linear* map of θ. It is still *technically possible* to do inference over θ, but the computational cost is prohibitive due to the non-linearity. However, if θ is known, the distributions are still linear combinations of Gaussian and we can us linear algebra to infer the weights. It would be nice to have an *approximate* solution for θ, but still do full inference on the weights w. This is where *Maximum Likelihood* (ML) and *Maximum A-Posteriori* (MAP) estimation comes into play; instead of *integrating* over θ, we can *fit* it; select the values that are most likely using

$$\underbrace{\theta^\star = \arg\max_\theta p(D|\theta),}_{\text{Maximum Likelihood}} \qquad \underbrace{\theta^\star = \arg\max_\theta p(\theta|D)\,.}_{\text{Maximum A-Posteriori}}$$

Maximum A-Posteriori weights the likelihood term by a prior on θ,

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \propto p(D|\theta)p(\theta),$$

and maximize the posterior w.r.t. θ, hence "A-Posteriori".

The parameters of the linear model w will still get a full probabilistic treatment and be *integrated out* in the inference of the pos-

terior, but the parameters that select the features, θ, called *hyper-parameters*, are too costly to properly infer and will thus get *fit*.

Maximum Likelihood in practice

To find the "best fit" $\theta^\star$, we have to solve $\arg\max_\theta p(y|w, X, \theta)$. To make the problem easier, we can use a few tricks to transform $p(y|w, X, \theta)$ to make it easier to compute, based on the observation that a transformation $g(p(y|w, X, \theta))$ does not change the maximization problem as long as $g(x) > g(y) \Leftrightarrow x > y$, and so

1. Solving $\arg\max_\theta f(x)$ is equivalent to solving $\arg\max_\theta \log f(x)$.
2. Solving $\arg\max_\theta f(x)$ is equivalent to solving $\arg\min_\theta -f(x)$.
3. Solving $\arg\max_\theta f(x) + c$ for some c independent of θ is equivalent to solving $\arg\max_\theta f(x)$.

This gives

$$
\begin{aligned}
\theta^\star &= \arg\max_\theta p(y|w, X, \theta), \\
&\overset{(1)}{=} \arg\max_\theta \log p(y|w, X, \theta), \\
&\overset{(2)}{=} \arg\min_\theta -\log p(y|w, X, \theta), \\
&= \arg\min_\theta \frac{1}{2}\left((y - {\phi_X^\theta}^\top \mu)^\top \left({\phi_X^\theta}^\top \Sigma \phi_X^\theta + \Lambda \right)^{-1} (y - {\phi_X^\theta}^\top \mu) + \log\det\left({\phi_X^\theta}^\top \Sigma \phi_X^\theta + \Lambda \right) \right) + \frac{N}{2}\log(2\pi), \\
&\overset{(3)}{=} \arg\min_\theta \frac{1}{2}\left(\underbrace{(y - {\phi_X^\theta}^\top \mu)^\top \left({\phi_X^\theta}^\top \Sigma \phi_X^\theta + \Lambda \right)^{-1} (y - {\phi_X^\theta}^\top \mu)}_{\text{Square Error}} + \underbrace{\log\det\left({\phi_X^\theta}^\top \Sigma \phi_X^\theta + \Lambda \right)}_{\text{Model complexity}} \right).
\end{aligned}
$$

To better see that the first term indeed is a square error, it might be useful to rewrite it as $\left\| \left({\phi_X^\theta}^\top \Sigma \phi_X^\theta + \Lambda \right)^{-1/2} (y - {\phi_X^\theta}^\top \mu) \right\|^2$, which is the squared error of the distance between ${\phi_X^\theta}^\top \mu$ and y, scaled by (the square-root of) the precision matrix. The Model Complexity term, $\log\det\left({\phi_X^\theta}^\top \Sigma \phi_X^\theta + \Lambda \right)$, measures the "volume" of the hypotheses covered by the joint Gaussian distribution.

The Model Complexity term is also called Occam's factor, and adds a penalty for features that lead to a large hypothesis space – features that can adapt to more structure – based on the principle that, everything kept equal, simpler explanations should be favored over more complex ones. The minimization procedure tries to minimize both; try to explain the observed data well - by making the resulting ${\phi_X^\theta}^\top \mu$ close to y - while keeping the model complexity low.

The same techniques and software used for fitting neural networks can be used to find the best fit $\theta^\star$, such as (Stochastic) Gradient Descent[15]. The gradient of the the square error and model complexity terms must be provided, but thankfully we can also use automatic differentiation[16]. By using the Chain Rule[17], we do not need to write code for the whole gradient every time we change the feature function ϕ; only the code for gradient of $\phi(x; \theta)$ with respect to θ needs to be provided.

Were we previously used

$$p(y|f_X) = \mathcal{N}\left(y; f_X, \sigma^2 I\right)$$

for the likelihood, we now use Λ for the covariance matrix, as in

$$p(y|f_X) = \mathcal{N}\left(y; f_X, \Lambda\right)$$

for more generality.

Numquam ponenda est pluralitas sine necessitate.
Plurality must never be posited without necessity.

— William of Occam

[15] wikipedia.org/wiki/Gradient_descent

[16] wikipedia.org/wiki/Automatic_differentiation

[17] wikipedia.org/wiki/Chain_rule

By using Maximum Likelihood (or Maximum A-Posteriori) solutions, we do not capture uncertainty on hyper-parameters. However, they make it possible to get *some* solution about *which features to use* in a reasonable time, which would be intractable otherwise..

If you are worried about fitting or hand-picking features for Bayesian regression, note that those concerns also apply to deep learning. By highlighting assumptions and priors, the probabilistic view forces us to address many problems directly, rather than obscuring them with notation and intuitions.

A linear Gaussian regression can be viewed as a single hidden layer neural network, with quadratic output loss and fixed input layer, where hyper-parameter fitting corresponds to training the input layer - see Figure 22 for an illustration. Automatic Differentiation (AD), Gradient Descent and data sub-sampling (Stochastic Gradient Descent) are algorithmic tools that are just as helpful for Bayesian inference as they are for deep learning. The two domains, deep learning and bayesian inference, are not separate; they are just different perspectives. It is possible to construct a point estimate for a Bayesian model, and to construct full posteriors for deep networks. The different viewpoints (probabilistic, statistical or empirical ("deep")) on Machine Learning often overlap and inform each other. Understanding of Bayesian linear (Gaussian) regression can help us build a better intuition for deep learning, too.

The usual way to train such network, however, does not include the Occam factor. The method used here is often referred to as *Type-II* Maximum Likelihood, whereas neural networks typically use *Type-I*. The following reference might be of interest for more details on the application of those ideas to neural networks;

MacKay. *The Evidence Framework Applied to Classification Networks*. Neural Computation, 1992

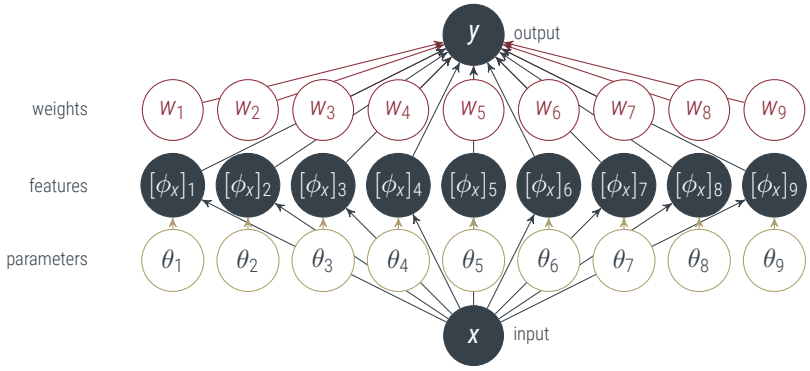


Figure 22: **Graphical representation of Hierarchical Bayesian Linear Regression.** The parameters controlling the features, θ, are learned from the data using Maximum Likelihood, in a similar fashion as a Neural Network, and full inference is carried over the weights of those features, w.

Gaussian Processes

In the previous chapter, we have seen that it is possible to learn a which features to use from a parametric family. We will now delve into Gaussian Processes which, instead of learning a few features, sometimes can get away with using *infinitely many features* (in finite time!). This does not mean that the model is infinitely complex; using N data points, the final posterior will still be described using a N-dimensional vector for the mean and a $[N \times N]$ covariance matrix. However, the number of features *considered* for the of the posterior covariance will be infinite, and the number of features *selected* will grow with the number of data points. This is an example of a *non-parametric* model, where the complexity grows when more data is added.

Mean function and Kernel

Consider the posterior function value for a single data point x', $f_{x'}$, where the posterior is conditioned on a bigger dataset of inputs X and output y;

$$p(f_{x'}|y,\phi_X) = \mathcal{N}\Big(f_{x'}; \phi_{x'}^\top\mu + \phi_{x'}^\top\Sigma\phi_X\left(\phi_X^\top\Sigma\phi_X + \sigma^2 I\right)^{-1}(y - \phi_X^\top\mu),$$
$$\phi_{x'}^\top\Sigma\phi_X - \phi_{x'}^\top\Sigma\phi_X\left(\phi_X^\top\Sigma\phi_X + \sigma^2 I\right)^{-1}\phi_X^\top\Sigma\phi_x\Big).$$

We can use the following abstraction,

$$\begin{aligned} \text{mean function:} \quad & m(x) = \phi_x^\top\mu, \qquad m : \mathbb{X} \to \mathbb{R}, \\ \text{covariance function (}\textit{Kernel}\text{):} \quad & K(a,b) = \phi_a^\top\Sigma\phi_b, \quad K : \mathbb{X}\times\mathbb{X} \to \mathbb{R}, \end{aligned}$$

to rewrite the posterior as

$$p(f_{x'}|y,\phi_X) = \mathcal{N}\Big(f_{x'}; m_{x'} + K_{x'X}(K_{XX} + \sigma^2 I)^{-1}(y - m_X),$$
$$K_{x'X} - K_{x'X}(K_{XX} + \sigma^2 I)^{-1}K_{Xx'}\Big),$$

where $m_a = \phi_a^\top\mu$ and $K_{ab} = \phi_a^\top\Sigma\phi_b$. The feature vectors $\phi_X, \phi_{x'}$ are hidden in the computation of the mean function and the kernel, and it might not be necessary to actually *form* the feature vectors to compute the posterior.

More features might mean cheaper to compute

For simplicity, fix the covariance matrix to $\Sigma = \sigma^2 \frac{c_{\max} - c_{\min}}{F} I$, where F is the number of features are $c_{\max} > c_{\min}$ constants, and assume that we are trying to learn features of the form

$$\phi(x, c_\ell) = \exp\left(-\frac{(x - c_\ell)^2}{2\lambda^2}\right),$$

with parameters $c_1 < ... < c_F$ in $[c_{\min}, c_{\max}]$. The kernel can then be written as

$$\begin{aligned}\phi_a^\top \Sigma \phi_b &= \sigma^2 \frac{c_{\max} - c_{\min}}{F} \sum_{\ell=1}^{F} \exp\left(-\frac{(a - c_\ell)^2}{2\lambda^2}\right) \exp\left(-\frac{(b - c_\ell)^2}{2\lambda^2}\right), \\ &= \sigma^2 \frac{c_{\max} - c_{\min}}{F} \exp\left(-\frac{(a-b)^2}{4\lambda^2}\right) \sum_{\ell=1}^{F} \exp\left(-\frac{(c_\ell - \frac{1}{2}(a+b))^2}{\lambda^2}\right),\end{aligned}$$

If we increase the number of feature towards ∞, the number of features per unit of $\mathrm{d}c$ approaches $\frac{F}{c_{\max} - c_{\min}} \mathrm{d}c$ and

$$\lim_{F\to\infty} \phi_a^\top \Sigma \phi_b = \sigma^2 \exp\left(-\frac{(a-b)^2}{4\lambda^2}\right) \int_{c_{\min}}^{c_{\max}} \exp\left(-\frac{(c - \frac{1}{2}(a+b)^2}{\lambda^2}\right) \mathrm{d}c.$$

The part inside the integral is an unnormalized Gaussian probability distribution function. Further taking the limit as $c_{\min} \to -\infty$ and $c_{\max} \to \infty$, the integral converges to the normalization factor, $\sqrt{2\pi}\lambda$, and the result is

$$K_{ab} := \lim_{\substack{F \to \infty, \\ c_{\max} \to \infty, \\ c_{\min} \to -\infty}} \phi_a^\top \Sigma \phi_b = \sqrt{2\pi}\lambda\sigma^2 \exp\left(-\frac{(a-b)^2}{4\lambda^2}\right).$$

This specific kernel is known as a *Radial Basis Function*, or *Square(d)-exponential Kernel*. The general definition is given by

Definition 17 (Positive Definite/Mercer Kernels)**.** $K : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is a positive definite Kernel, or *Mercer* Kernel, if for any finite collection $X = [x_1, ..., x_n]$ the matrix $K_{XX} \in \mathbb{R}^{N \times N}$ constructed from

$$[K_{XX}]_{ij} = K(x_i, x_j)$$

is positive semi-definite[18].

[18] wikipedia.org/wiki/Positive-definite_matrix

Definition 18 (Positive Definite)**.** A matrix $A \in \mathbb{R}^{N \times N}$ is positive (semi-)definite if, for any $x \in \mathbb{R}^N$,

$$\underbrace{x^\top A x > 0}_{\text{positive definite}}, \qquad \underbrace{x^\top A x \geq 0}_{\text{positive semi-definite}}.$$

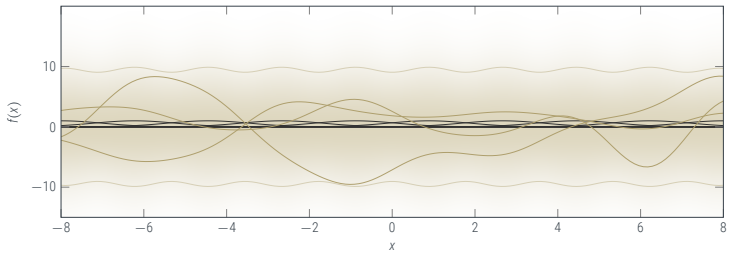
Equivalently, A is positive (semi-)definite if

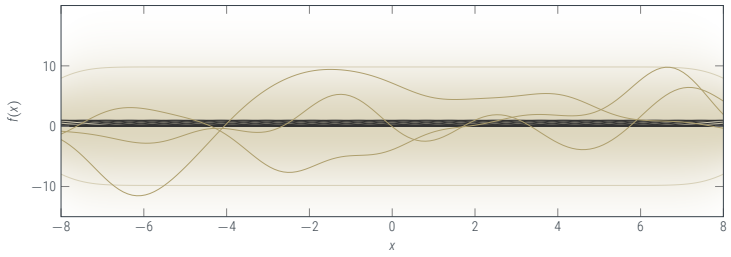
- All its eigenvalues are positive (non-negative for semi-definite).
- It is a Gram matrix - the sum of the outer product vectors - and full rank (not necessary for semi-definite).

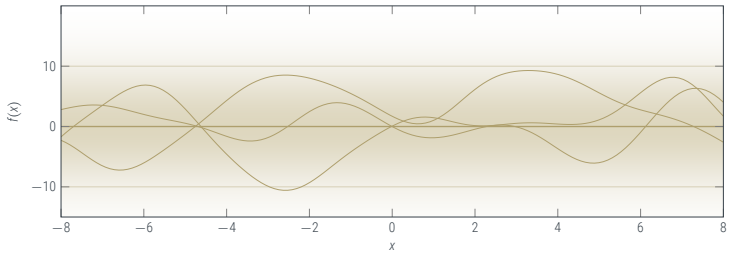
Visualizing kernels

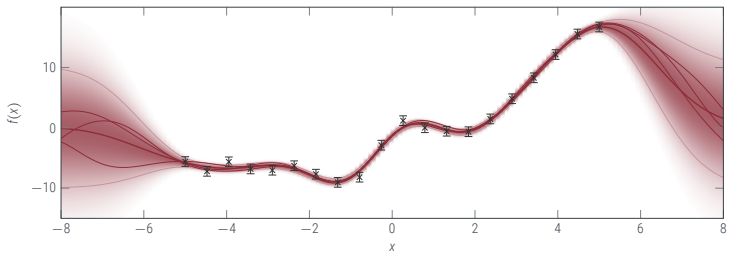
The following figures shows the prior for different feature functions and how increasing the number of feature, taking the limit towards infinity, leads to a Kernel. The final posteriors are inferred from the dataset introduced in Figure 15, Page 29

Visualizing: Radial Basis Functions









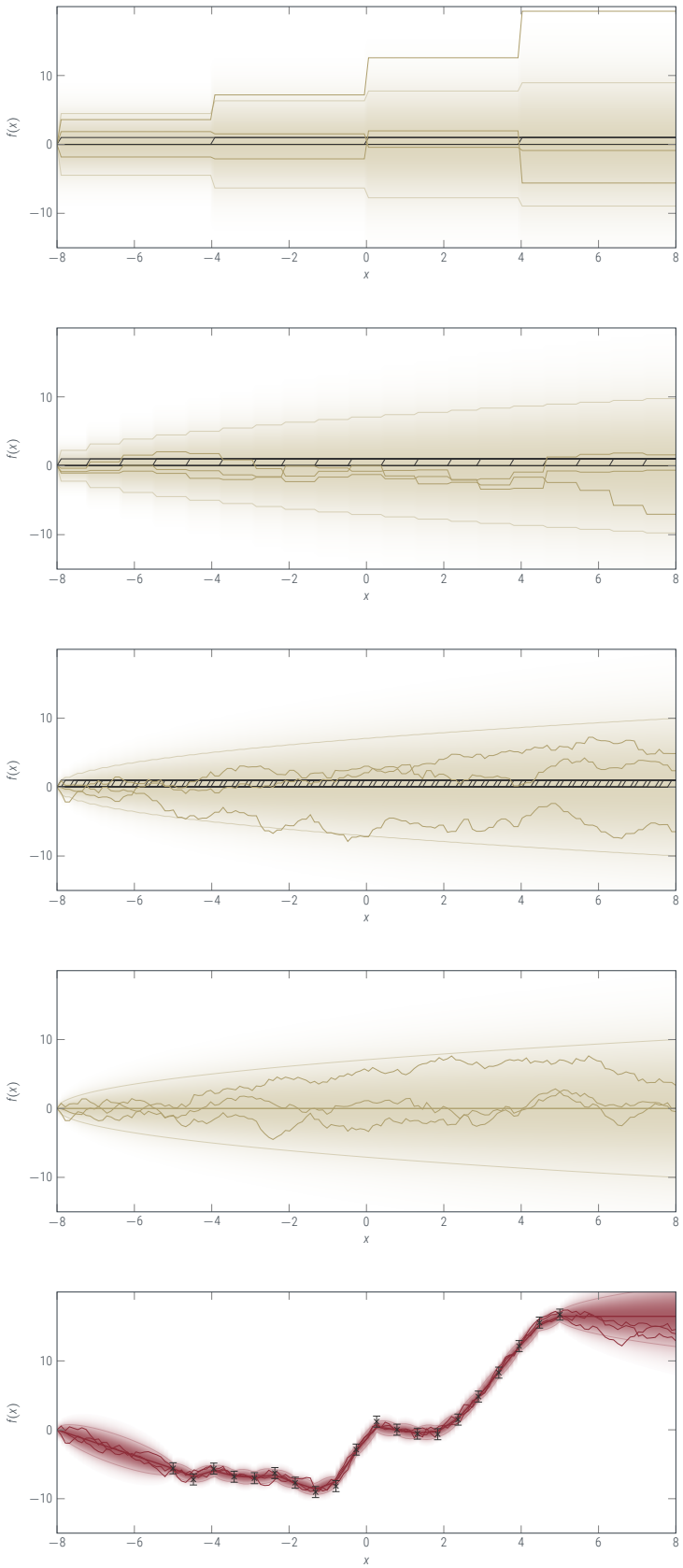
The Radial Basis Function, or square(d)-exponential Kernel, is generated from functions of the form

$$\phi_\ell(x) = \exp\left(-\frac{(x-c_\ell)^2}{2\lambda^2}\right),$$

and in the limit leads to

$$K(a,b) = \sqrt{2\pi}\lambda\sigma^2 \exp\left(-\frac{(a-b)^2}{4\lambda^2}\right).$$

Visualizing: Wiener process



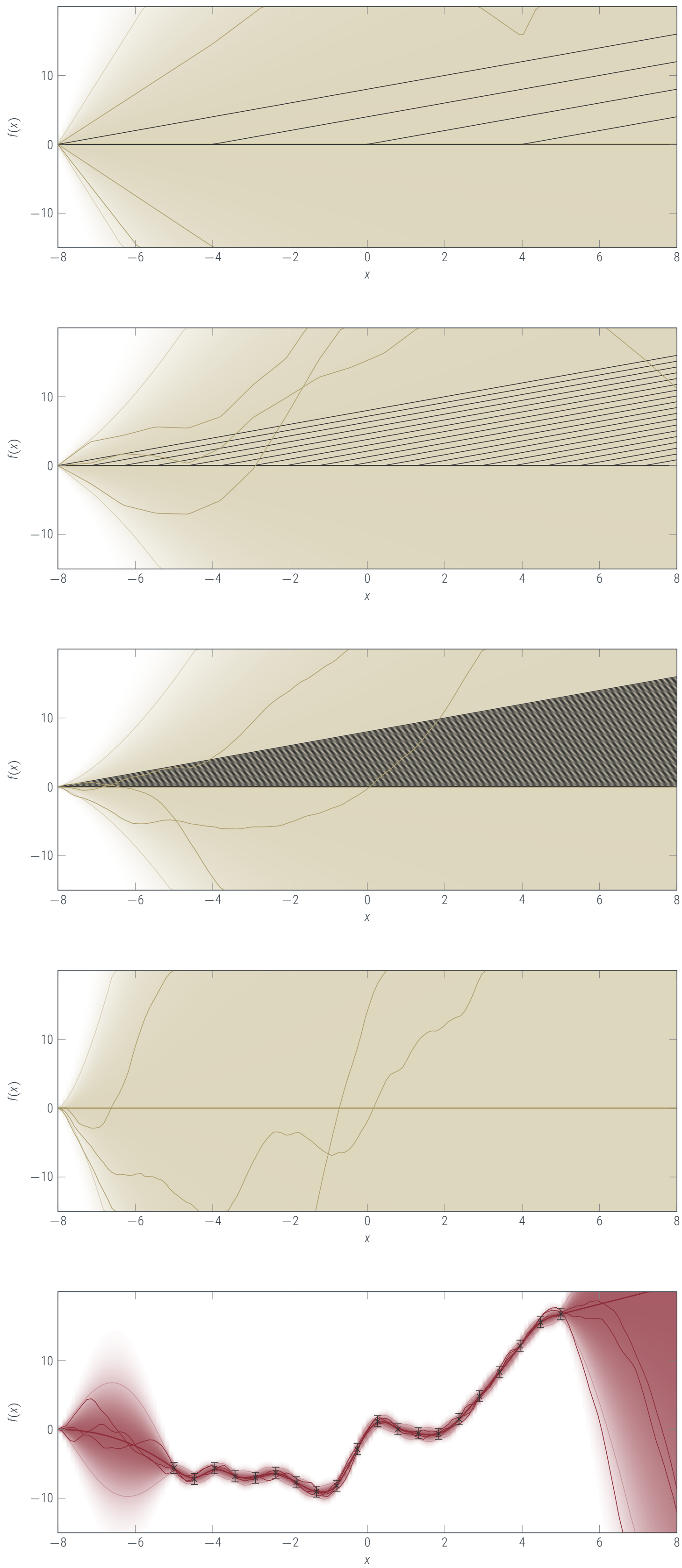
The Wiener Process is defined from step functions, that start from some c_0 at 0 and switch to 1 at some threshold,

$$\phi_\ell(x) = \begin{cases} 1 & \text{if } x \geq c_\ell, \\ 0 & \text{otherwise,} \end{cases}$$

and converges to

$$K(a,b) = \sigma^2(\min(a,b) - c_0).$$

Cubic Splines



The Cubic Splines start similarly to the Wiener process' threshold functions, but they keep increasing linearly after being activated, as RELU activations functions,

$$\phi_\ell(x) = \begin{cases} x - c_\ell & \text{if } x \geq c_\ell, \\ 0 & \text{otherwise,} \end{cases}$$

and converge to

$$K(a,b) = \sigma^2 \left(\frac{1}{3} \left(\min(a - c_0, b - c_0)\right)^3 + \frac{1}{2} |a - b| \left(\min(a - c_0, b - c_0)\right)^2 \right).$$

KERNELS CAN ALSO BE COMBINED to form new kernels.

Theorem 19. If K_1, K_2 are Mercer kernels from $\mathbb{X} \times \mathbb{X} \to \mathbb{R}$ and ϕ is a mapping from $\mathbb{Y} \to \mathbb{X}$, then the following functions are also Mercer kernels (up to minor regularity assumption);

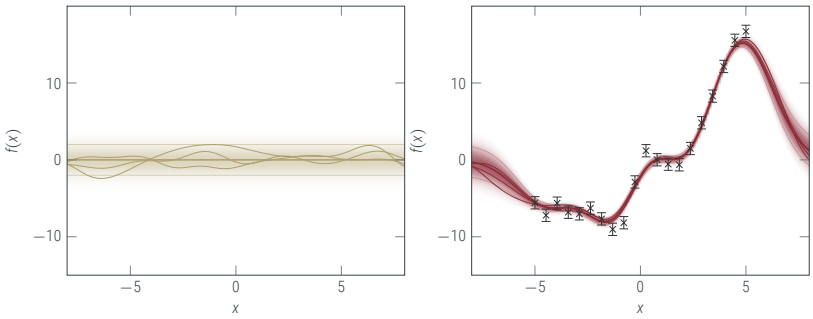
- **Scaling:** $\alpha K_1(a, b)$ for $\alpha \in \mathbb{R}_+, a, b \in \mathbb{X}$.
- **Kernel Addition:** $K_1(a, b) + K_2(a, b)$, for $a, b \in \mathbb{X}$.
- **Change of representation:** $K_1(\phi(a), \phi(b))$, for $a, b \in \mathbb{Y}$.
- **Kernel Multiplication:** $K_1(a, b)K_2(a, b)$, for $a, b \in \mathbb{X}$.

The first two properties should be easy to prove from the properties of PSD matrices. The third property is the result of Mercer's Theorem[19], which we will cover in the next chapter, and the last property is the result of the Shur Product Theorem[20]. Its proof is involved and is the result of the fact that the Hadamard Product[21] of two positive semi-definite matrices give positive semi-definite. The following figures show some examples of transformed kernels.

[19] wikipedia.org/wiki/Mercer's_theorem

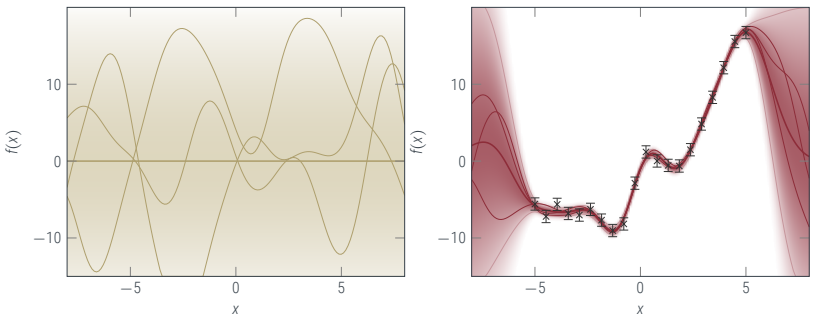
[20] wikipedia.org/wiki/Schur_product_theorem

[21] wikipedia.org/wiki/Hadamard_product_(matrices)



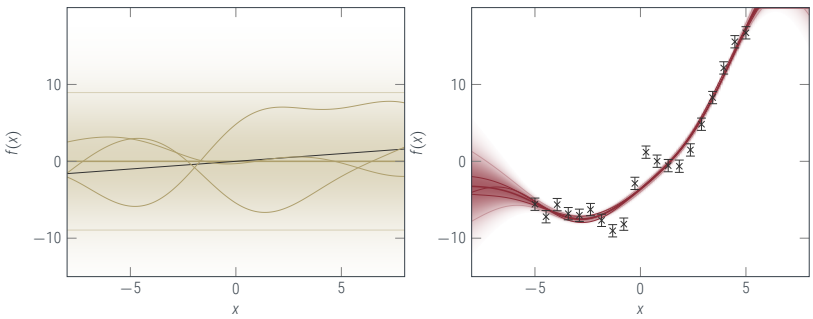
A simple RBF Kernel, as a reference point,

$$K(a, b) = \exp\left(-\frac{(a-b)^2}{2}\right).$$



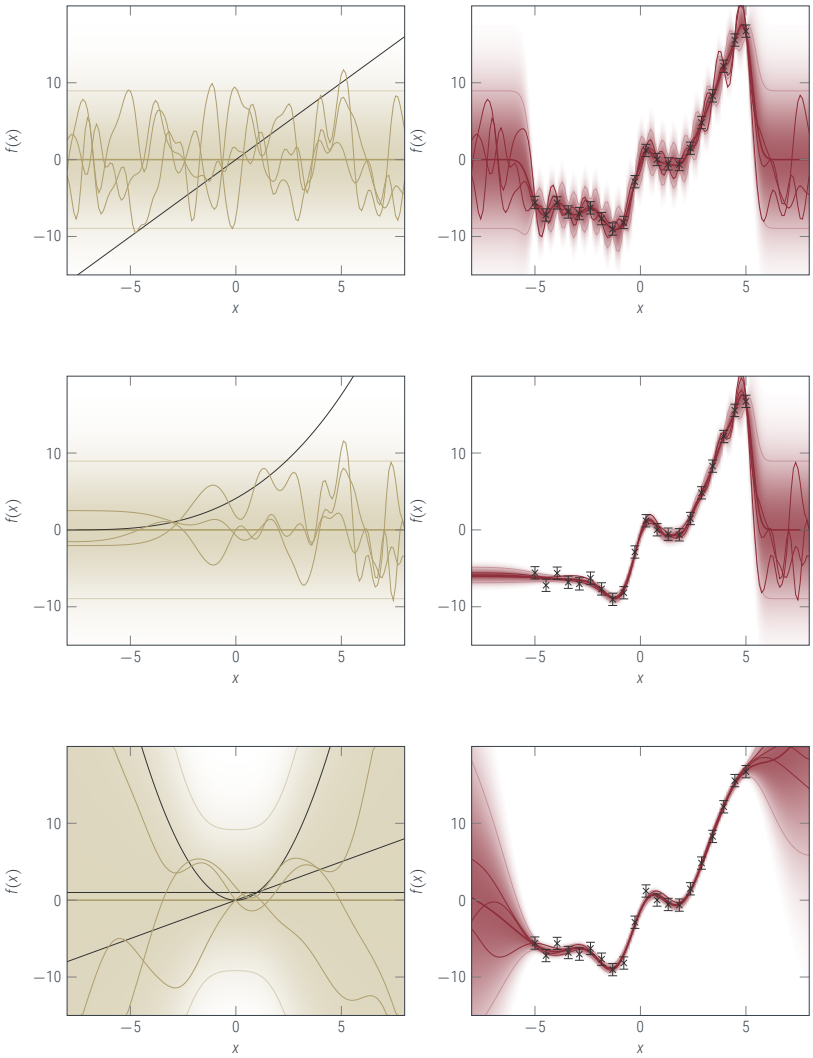
Different scaling for the RBF Kernel, to affect the effect of noise,

$$K(a, b) = 1^2 \cdot \exp\left(-\frac{(a-b)^2}{2}\right).$$



Transforming the inputs of a RBF Kernel, to change the width of the bumps, here

$$K(a, b) = 20 \cdot \exp\left(-\frac{(a-b)^2}{2 \cdot 5^2}\right).$$



Another transformation of the inputs of a RBF Kernel, here to reduce the width of the bumps with

$$K(a,b) = 20 \cdot \exp\left(-\frac{(a-b)^2}{2 \cdot 0.5^2}\right).$$

A non-linear transformation of the inputs of a RBF Kernel, here using

$$K(a,b) = 20 \cdot \exp\left(-\frac{(\phi(a)-\phi(b))^2}{2}\right),$$

$$\phi(x) = \left(\frac{x+8}{5}\right)^3.$$

A sum of Kernels, here using a simple RBF kernel and a quadratic feature function,

$$K(a,b) = 20 \cdot \exp\left(-\frac{a-b)^2}{2}\right) + \phi(a)^\top \phi(b),$$

$$\phi(x) = [1\ x\ x^2]^\top.$$

We can also learn Kernels, using similar techniques as we have seen before, by parametrizing the mean and/or the Kernels;

$$m_\theta(x) = \phi(x)^\top \theta, \qquad K_\theta(a,b) = \theta_1 \exp\left(-\frac{(a-b)^2}{2\theta_2^2}\right).$$

We can use Maximum Likelihood, as we saw for learning features, but the number of parameters we need to fit to learn the kernel is typically less than the number of parameters we needed to learn the features. To learn the features, we needed to put some parameters for *each* feature to be learned. The Kernel already takes care of selecting individual features, and we only need a few parameters to specify the family of features to pick from. As the problem is smaller in terms of dimensionality, we can use *numerical quadrature* to marginalize over θ, by computing $p(f) = \int p(f|\theta)\,\mathrm{d}\theta \approx \sum_i \alpha_i p(f|\theta_i)$. Quadrature is expensive for high-dimensional spaces, but if precision is needed, it is the best option.

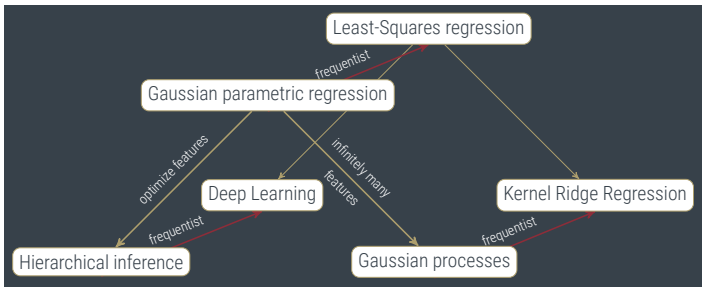
Gaussian Processes are thus defined as follow

Definition 20 (Gaussian Process)**.** Let $\mu : \mathbb{X} \to \mathbb{R}$ be any function and $K : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ be a Mercer Kernel. A *Gaussian Process* $p(f) = \mathcal{GP}\left(f; \mu, K\right)$ is a probability distribution over the function $f : \mathbb{X} \to \mathbb{R}$ such that every finite restriction to function values $f_X = [f_{x_1}, ..., f_{x_n}]$ is a Gaussian distribution $p(f_X) = \mathcal{N}\left(f_X, \mu_X, K_{XX}\right)$.

A Gaussian Process is a *prior over function values*, to be used in Bayesian inference with a likelihood to get a posterior. Given a prior $p(f) = \mathcal{GP}\left(f; m, K\right)$ and a likelihood $p(y|f) = \mathcal{N}\left(y; f_X, \Lambda\right)$, the posterior is given by

$$p(f|y) = \mathcal{GP}\left(f_{x'}; m_{x'} + K_{x'X}(K_{XX} + \Lambda)^{-1}(y - m_X), K_{x'x'} - K_{x'X}(K_{XX} + \Lambda)^{-1}K_{Xx'}\right).$$

The term *Gaussian* does not refer to the use of a specific kernel, but to the probability distribution over the function values. There are many kernels, and new ones can be constructed from combinations of known one, and they can be learned just like features using either MAP or numerical integration. The following figure illustrate the relations between the methods we have seen so far and their frequentist counterparts.



Gaussian Process Classification

So far, we have seen how to model continuous outputs; learn the relation between an output y given some features x as $y \sim p(f(x))$. We will now see how to adapt this framework to do *classification*; given data that can belong in two classes, say A and B, we want to learn the probability that a sample with features x belongs to class A.

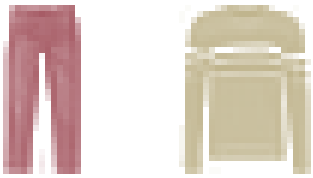


Figure 23: Two samples from `FashionMNIST` (`github.com/zalandoresearch/fashion-mnist`)

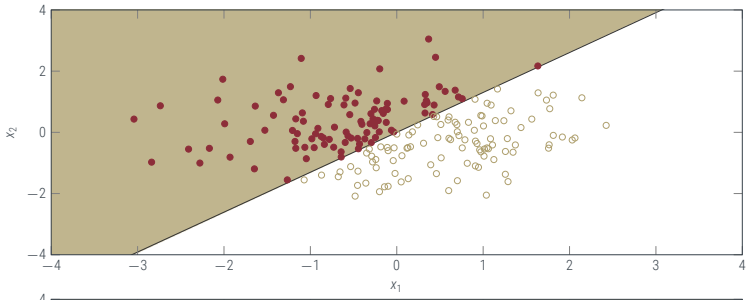


Figure 24: Simple classification problem: it is posible to draw a line to perfectly separate the samples

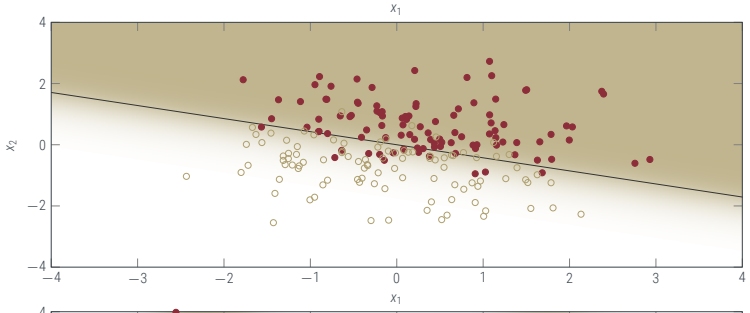


Figure 25: Imperfect linear classification problem: It is not possible to perfectly separate the two classes with a simple line and the boundary needs to be fuzzy.

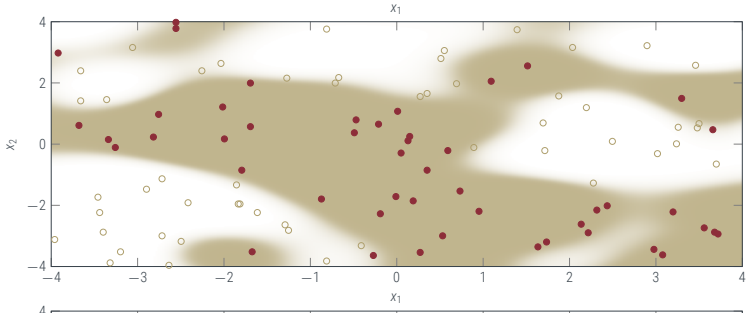


Figure 26: Nonlinear classification problem: while it is not possible to find a single line to separate the two classes, a more complex boundary can be found

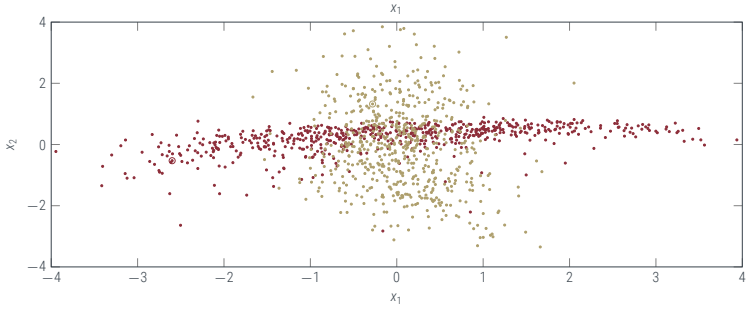


Figure 27: Non-separable classification problem: the overlap between the two classes is too strong to be separable, but some structure exists

To clarify the difference between regression and classification, consider the two definitions;

Definition 21 (Regression)**.** Given input-output pairs $(x_i, y_i)_{i=1,...,n}$ with $x_i \in \mathbb{X}$ and $y_i \in \mathbb{R}^d$, we want to find a function $f : \mathcal{X} \to \mathbb{R}^d$ such that f *models* $y_i \approx f(x_i)$

Definition 22 (Classification)**.** Given input-output pairs $(x_i, c_i)_{i=1,...,n}$ with $x_i \in \mathbb{X}$ and $c_i \in \{0, 1, ..., d-1\}$, we want to find a probability $\pi : \mathcal{X} \to U^d$, where $U^d = \{p \in [0,1]^d : \sum_{i=1}^d p_i = 1\}$, such that π *models* $c_i \approx \pi_{x_i}$

For a first approach, we will consider only *binary* classification, with $y \in \{-1, 1\}$, where we would like to learn the function

$$p(y|x) = \begin{cases} \pi(x) & \text{if } y = 1, \\ 1 - \pi(x) & \text{if } y = -1. \end{cases}$$

This is very similar to regression, where we learned $p(y|x) = \mathcal{N}\left(y; f_x, \sigma^2\right)$, but the *domain* is wrong; here, $y \in \{-1, 1\}$ instead of $y \in \mathbb{R}$.

THE SIGMOID. To fix this issue, we will introduce a *link function*; a mapping from a real value $f \in \mathbb{R}$ to a probability $\pi \in [0, 1]$, using the *sigmoid* or *logistic link function*;

$$\pi_f = \sigma(f) = \frac{1}{1 + e^{-f}}.$$

We can use the logistic function on top of a Gaussian Process regression model to adapt it for classification, creating a *logistic regression*; take a Gaussian Process prior over f, $p(f) = \mathcal{GP}\left(f; m, k\right)$ and use the likelihood

$$p(y|f_x) = \sigma(y f_x) = \begin{cases} \sigma(f) & \text{if } y = 1, \\ 1 - \sigma(f) & \text{if } y = -1. \end{cases}$$

The sigmoid as some useful properties; it is symmetric,

$$\sigma(f) = 1 - \sigma(-f),$$

its inverse is easy to compute,

$$f(\pi) = \log \pi_f - \log(1 - \pi_f),$$

as is its derivative,

$$\frac{\partial \pi_f}{\partial f} = \pi_f(1 - \pi_f).$$

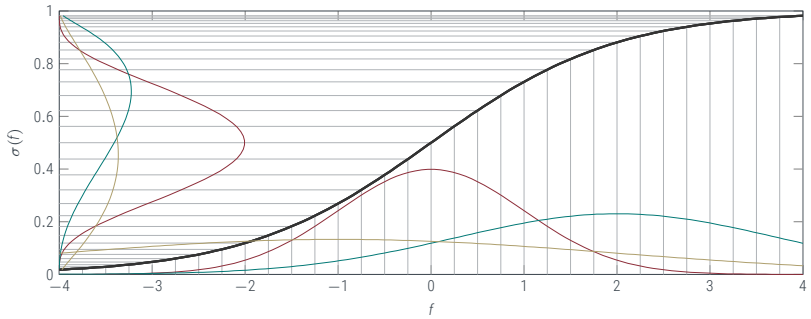


Figure 28: The shape of the sigmoid, along with the shape of Gaussian probability density functions, when passed through a sigmoid.

A SLIGHT ISSUE with this formulation is that the posterior is no longer Gaussian. For a matrix X and vector Y, we have

$$p(f_X|Y) = \frac{p(Y|f_X)p(f_X)}{p(Y)} = \frac{\sigma(Y \odot f_X)\mathcal{N}\left(f_X; m, k\right)}{\int \sigma(Y \odot f_X)\mathcal{N}\left(f_X; m, k\right) \mathrm{d}f_X}.$$

$$\log p(f_X|Y) = -\frac{1}{2} f_X^\top K_X X^{-1} f_X + \sum_{i=1}^{n} \log \sigma(y_i f_{x_i}) + \text{const.}$$

This makes the computation of the posterior more complex, and in most cases intractable. The following figures show the prior, likelihood and posterior of a 2D classification model.

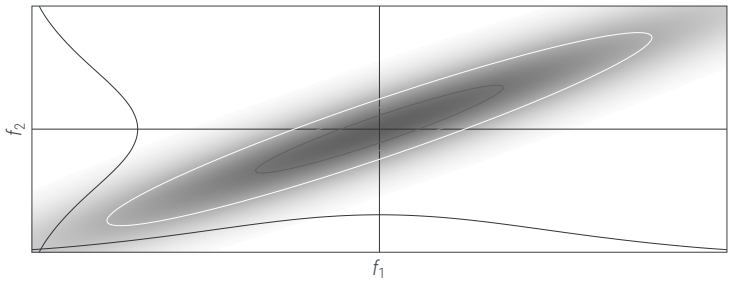


Figure 29: Prior on f

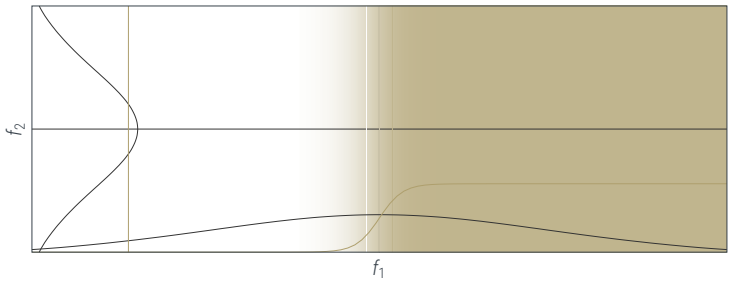


Figure 30: Likelihood

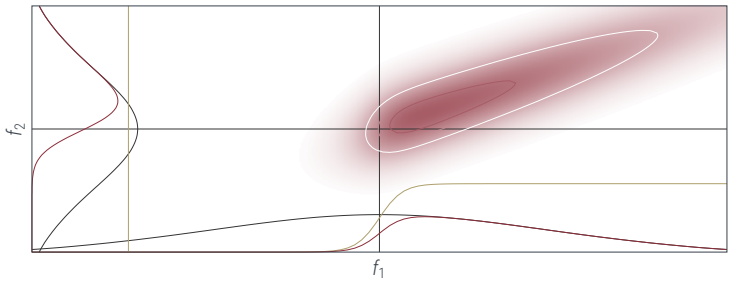


Figure 31: Posterior

We do not always need to compute the full posterior, however, and an approximation giving "key aspects" of the posterior can be sufficient. We might be interested in the *moments* of $p(f, y) = p(y|f)p(f)$ rather than in the full posterior;

$$\begin{aligned}
\text{The evidence:} \quad & \mathbb{E}_{p(y,f)}\left[1\right] = \int 1 \cdot p(y,f)\,\mathrm{d}f && = \int p(y,f)\,\mathrm{d}f && = Z, \\
\text{The mean:} \quad & \mathbb{E}_{p(y|f)}\left[f\right] = \int f \cdot p(f|y)\,\mathrm{d}f && = \frac{1}{Z}\int f \cdot p(f,y)\,\mathrm{d}f && = \bar{f}, \\
\text{The variance:} \quad & \mathbb{E}_{p(f|y)}\left[f^2 - \bar{f}^2\right] = \int f^2 \cdot p(f|y)\,\mathrm{d}f - \bar{f}^2 && = \frac{1}{Z}\int f^2 \cdot p(f,y)\,\mathrm{d}f - \bar{f}^2 && = \mathrm{var}(f).
\end{aligned}$$

Z is useful for parameter tuning, $\bar{f}$ provides a point estimate and $\mathrm{var}(f)$ an estimate of the error around $\bar{f}$.

The Laplace Approximation

The Laplace approximation is a *rough* and *local* approximation to the posterior—it can be arbitrarily wrong—but it is computationally efficient and often works relatively well for logistic regression as the posterior is concave and the link function yields "almost" a Gaussian posterior; see Figure 32

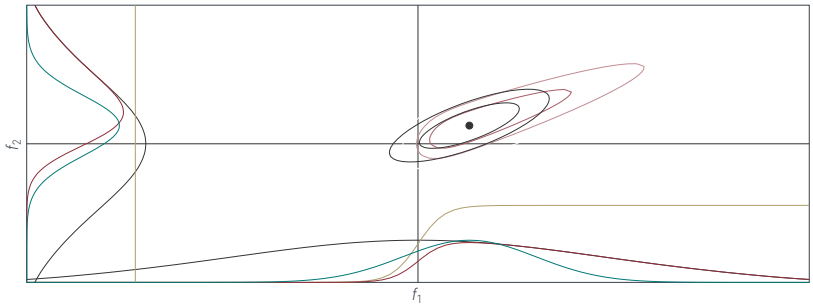


Figure 32: Laplace approximation of the Posterior.

The Laplace approximation builds a Gaussian approximation $q(\theta) = \mathcal{N}\left(\theta; \hat{\mu}, \hat{\Sigma}\right)$ to a non-Gaussian probability distribution $p(\theta)$, which in our context can be a likelihood or a posterior. To construct the $q(\theta)$, we first find a (local) maximum to $\log p(\theta)$ (or, equivalently, $p(\theta)$) to find the mean of $q(\theta)$,

$$\hat{\mu} = \theta^\star = \arg\max_\theta \log p(\theta),$$

and the covariance is given by the negative of the inverse-Hessian at $\theta^\star$ by approximation p by a second-order Taylor expansion of $\log p(\theta)$ around $\theta^\star$,

$$\log p(\theta) \approx \log p(\theta^\star) + \frac{1}{2}(\theta^\star - \theta)^\top \underbrace{\nabla^2 \log p(\theta^\star)}_{=-\hat{\Sigma}^{-1}} (\theta^\star - \theta),$$

leading to the approximation

$$q(\theta) = \mathcal{N}\left(\theta; \theta^\star, -\left(\nabla^2 \log p(\theta^\star)\right)^{-1}\right) \approx p(\theta).$$

If $p(\theta)$ is Gaussian, the approximation is exact and $q(\theta) = p(\theta)$.

We found a maximum of $\log p$ and used the negative-inverse-Hessian of $\log p$as the covariance, but the equivalent formulation of finding a minimum of $-\log p$ and using the inverse-Hessian of $-\log p$ is also common.

Application to Gaussian Process Logistic Regression

We find the maximum-a-posteriori for the latent f on the training set,

$$\hat{f} = \arg\max_{f_X} \log p(f_X|y)$$

and assign a Gaussian posterior at the training points,

$$q(f_X) = \mathcal{N}\left(f_X; \hat{f}, -\left(\nabla^2_{f_X} \log p(f_X|y)\Big|_{f_x=\hat{f}}\right)^{-1}\right),$$

and we can now approximate the posterior *predictions* at f_x;

$$
\begin{aligned}
q(f_x|y) &= \int p(f_x|f_X)q(f_X)\,\mathrm{d}f_X, \\
&= \int \mathcal{N}\left(f_x; m_x + K_{xX}K_{XX}^{-1}(f_X - m_X), K_{xx} - K_{xX}K_{XX}^{-1}K_{Xx}\right) q(f_X)\,\mathrm{d}f_X, \\
&= \mathcal{N}\left(f_x; m_x + K_{xX}K_{XX}^{-1}(\hat{f} - m_X), K_{xx} - K_{xX}K_{XX}^{-1}K_{Xx} + K_{xX}K_{XX}^{-1}\Sigma K_{XX}^{-1}K_{Xx}\right).
\end{aligned}
$$

Prediction probabilities can then be computed as

$$
\hat{\pi}_x = \mathbb{E}_{p(f_x|y)}\left[\pi_x\right] \approx \mathbb{E}_{q(f_x|y)}\left[\pi_x\right] = \int \sigma(f_x)q(f_x|y)\,\mathrm{d}f_x,
$$

or using (not the same!) $\hat{\pi}_x \approx \sigma\left(\mathbb{E}_q\left[f_x\right]\right)$.

For practical examples and implementations, look at the references in the next chapter.

Beyond binary classification

Other non-linearities / link-functions can be used, too, to model other output types (bounded, strictly positive, structured, etc.)

Practical Examples

Practical examples with an implementation in Python were presented in class in lecture 8 and 11, covering Gaussian Process Regression and Gaussian Process Classification, respectively. Python notebooks containing the code some guidelines are available on ILIAS, in the `code` section;

- `WeightTrackModel.ipynb`: Gaussian Process Regression
- `GPC_german_credit.ipynb`: Gaussian Process Classification

Understanding Kernels

The goal of this chapter is to show the connections between Gaussian Processes and Kernel methods and look at their expressive power and limitations. Many different methods are equivalent or closely related to Gaussian process regression, for example Kriging (in geosciences), Kernel ridge regression, Wiener–Kolmogorov prediction, or even Linear Least-Squares.

Warning: The following are simplified expositions! Some regularity assumptions have been dropped for easier readability. For the full story of the relationship on GPs and kernel methods, check out

Kanagawa, Hennig, Sejdinovic, and Sriperumbudur. *Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences*. CoRR, 2018. URL arxiv.org/abs/1807.02582

For a deeper treatment, you can check the following

Schölkopf and Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. MIT Press, 2002

Rasmussen and Williams. *Gaussian processes for machine learning*. MIT Press, 2006

The Gaussian Posterior Mean is a Least-Squares estimate. Recall the expression of the posterior for the weights of a linear Gaussian regression with feature vectors ϕ_X;

$$p(w|y) = \mathcal{N}\Big(w;\, \mu + \Sigma\phi_X(\phi_X^\top\Sigma\phi_X + \sigma^2 I)^{-1}(y - \phi_X^\top\mu),\\ \Sigma - \Sigma\phi_X(\phi_X^\top\Sigma\phi_X + \sigma^2 I)^{-1}\phi_X^\top\Sigma\Big).$$

As the Gaussian distribution is symmetric, the mean of the weights correspond to the maximum a posteriori;

$$\begin{aligned}\mathbb{E}_{p(w|y)}\left[w\right] &= \arg\max_w p(w|y) = \arg\min_w -\log p(w|y),\\ &= \arg\min_w \frac{1}{2\sigma^2}\left\|y - \phi_X^\top w\right\|^2 + \frac{1}{2}(w-\mu)^\top\Sigma(w-\mu),\\ &= (\Sigma^{-1} + \sigma^{-2}\phi_X\phi_X)^{-1}(\Sigma^{-1}\mu + \sigma^{-2}\phi_X y),\\ &\overset{\sigma^{-1}\to 0}{\longrightarrow} (\phi_X\phi_X^\top)^{-1}\phi_X y.\end{aligned}$$

As the precision Σ^{-1} goes to 0, converging to a uniform prior on w, the solution converges to the traditional least-squares estimate $(\phi_X\phi_X^\top)^{-1}\phi_X y$. With a well-defined Gaussian prior on the weights, the expected value of the posterior mean of a Gaussian Regression is equal to the *regularized least-squares* (or Ridge Regression) estimate with the μ-centered regularizer $\left\|w-\mu\right\|^2$ (or Tikhonov regularizer[22]). With typical value encountered in a Machine Learning course on linear regression when teaching L_2 regularization, $\Sigma^{-1} = \lambda I$, $\sigma^2 = 1$ and $\mu = 0$, the expected posterior mean is the solution of the MAP problem

$$\arg\min_w \left\|\phi_X^\top w - y\right\|^2 + \frac{\lambda}{2}\|w\|^2 = (\lambda I + \phi_X\phi_X)^{-1}(\phi_X y).$$

[22] wikipedia.org/wiki/Tikhonov_regularization

THE GAUSSIAN PROCESS POSTERIOR MEAN IS A KERNEL RIDGE ESTIMATE. Using a similar analysis on the expected value of the posterior mean of a Gaussian Process,

$$p(f_x|X,y) = \mathcal{GP}\left(f_x; m_x + K_{xX}(K_{XX}+\sigma^2 I)^{-1}(y - m_X), K_{xx} - K_{xX}(K_{XX}+\sigma^2 I)^{-1}K_{Xx}\right),$$

we have that the posterior mean is equal to the *regularized least-squares* estimate with regularizer $\|f\|_K^2$, or the *kernel ridge estimate*,

$$\begin{aligned}\mathbb{E}_{p(f_x|X,y)}\left[f_x\right] &= \arg\max_{f_x} p(f_x|X,y) = \arg\min_{f_x} -\log p(f_x|X,y),\\ &= \arg\min_{f_x} \frac{1}{2\sigma^2}\|y - f_X\|^2 + \frac{1}{2}\|f_{x,X} - m_{x,X}\|_K^2.\end{aligned}$$

Connection with kernel methods

To take a closer look at what kind of functions a Gaussian process produces and what is the sample-space, we will look at the connection with Kernel methods.

A quick reminder on Linear Algebra;

Theorem 23 (Eigenvalues, Eigenvectors and the Spectral Theorem)**.** Let $A \in \mathbb{R}^{N\times N}$ be a matrix. A scalar $\lambda \in \mathbb{C}$ and vector $v \in \mathbb{C}^N$ are called *eigenvalue* and corresponding *eigenvector* if

$$Av = \lambda v.$$

The eigenvectors of symmetric matrices $A = A^\top$ are real and form the basis of the image of A. A symmetric positive definite matrix A can be written as a gramian (outer product) of the eigenvectors;

$$A = \sum_{n=1}^{N} \lambda_n v_n v_n^\top.$$

The eigenvalues-vectors of a matrix give its full characterization and its image.

Kernels give rise to finite matrices when applied to data, but are infinite sums of functions. Properties and restrictions that are relatively straightforward for the finite-dimensional case (matrices / Gaussian inference) can be surprising and subtle in the infinite-dimensional case (kernels / Gaussian processes), but a similar decomposition exists.

Theorem 24 (Eigenfunctions and Mercer's theorem)**.** A function $\phi : \mathbb{X} \to \mathbb{R}$ and scalar $\lambda \in \mathbb{C}$ that obeys

$$\int K(x,\tilde{x})\phi(\tilde{x})\,\mathrm{d}\nu(\tilde{x}) = \lambda\phi(x)$$

are called the *eigenfunctions* and *eigenvalue* of K with respect to a measure ν. Let $(\mathbb{X},\nu)$ be a finite measure space and $K : \mathbb{X}\times\mathbb{X}\to\mathbb{R}$ a continuous Mercer Kernel. Then, there exists eigenvalues and eigenfunctions $(\lambda_i,\phi_i), i \in \mathcal{I}$ with respect to ν such that $\mathcal{I}$ is countable, all λ_i are real and non-negative, the eigenfunctions can

be made orthonormal and the following serie converges absolutely and uniformly ν^2-almost-everywhere;

$$K(a,b) = \sum_{i\in\mathcal{I}} \lambda_i \phi_i(a)\phi_i(b), \qquad \forall a,b \in \mathbb{X}.$$

In the sense of Mercer's theorem, one may think vaguely of a kernel $K : \mathbb{X}\times\mathbb{X} \to \mathbb{R}$ evaluated at $K(a,b)$, for $a,b \in \mathbb{X}$, as the "element" of an "infinitely large" matrix K_{ab}. However, this interpretation is only relative to the measure $\nu : \mathbb{X} \to \mathbb{R}$ and, in general, it is not straightforward to find the eigenfunctions.

Using this approach, we can also define a concept similar to the image of a matrix for kernels; the Reproducing kernel Hilbert space (RKHS)

Definition 25 (Reproducing kernel Hilbert space (RKHS))**.** Let $\mathcal{H} = (X, \langle\cdot,\cdot\rangle)$ be a Hilbert space of functions $f : \mathbb{X} \to \mathbb{R}$. Then $\mathcal{H}$ is called a reproducing kernel Hilbert space (RKHS) if there exists a kernel $k : \mathbb{X}\times\mathbb{X} \to \mathbb{R}$ such that

1. $\forall x \in \mathbb{X} :\ k(\cdot,x) \in \mathcal{H}$
2. $\forall f \in \mathcal{H} :\ \langle f(\cdot), k(\cdot,x)\rangle_{\mathcal{H}} = f(x)$ - K reproduces $\mathcal{H}$.

Theorem 26 (Aronszajin, 1950)**.** For every positive definite K on $\mathbb{X}$, there exists a unique RKHS.

This definition might not be very helpful, but as it is possible to define the image of a matrix A using a weighted combination of eigenvalues, $\sum_{n=1}^{N} \alpha_n\lambda_n v_n v_n^\top$ for any set of $[\alpha_1,...,\alpha_N]$, it is possible to find a similar representation using Mercer's theorem.

Theorem 27 (Mercer Representation)**.** Let $\mathbb{X}$ be a compact metric space, K be a continuous Mercer kernel on $\mathbb{X}$, ν be a finite Borel measure on $\mathbb{X}$ and $(\phi_i,\lambda_i)_{i\in\mathcal{I}}$ be the eigenfunctions and eigenvalues of K with respect to ν. Then, the RKHS $\mathcal{H}_K$ is given by

$$\mathcal{H}_K = \left\{ f(x) := \sum_{i\in\mathcal{I}} \alpha_i \lambda_i^{1/2}\phi_i(x) \text{ such that } \|f\|^2_{\mathcal{H}_K} := \sum_{i\in\mathcal{I}} \alpha_i^2 < \infty \right\}.$$

and

$$\langle f, g\rangle := \sum_{i\in\mathcal{I}} \beta_i\gamma_i,$$

where α_i, γ_i are the weights of f and g,

$$f = \sum_{i\in\mathcal{I}} \beta_i\lambda_i^{1/2}\phi_i \qquad\qquad g = \sum_{i\in\mathcal{I}} \gamma_i\lambda_i^{1/2}\phi_i.$$

A perhaps more useful representation is in terms of related points;

Corollary 28 (Reproducing kernel map representation)**.** Let $\mathbb{X}, \nu, (\phi_i,\lambda_i)_{I\in\mathcal{I}}$ be defined as before and let $(x_i)_{i\in\mathcal{I}}$ be a countable collection of

points in $\mathbb{X}$. Then, the RKHS can also be written as the space of linear combinations of kernel functions:

$$\mathcal{H}_K = \left\{ f(x) := \sum_{i \in \mathcal{I}} \tilde{\alpha}_i k(x_i, x) \right\},$$

and

$$\langle f, g \rangle_{\mathcal{H}_K} := \sum_{i \in \mathcal{I}} \frac{\tilde{\beta}_i \tilde{\gamma}_i}{K(x_i, x_j)}.$$

For a Gaussian Process $p(f) = \mathcal{GP}\left(f; 0, K\right)$ with likelihood $p(y|f, X) = \mathcal{N}\left(y; f, \sigma^2 I\right)$, the RKHS is the space of all *possible* posterior mean functions

$$\mu(x) = K_{xX} \underbrace{(K_{XX} + \sigma^2 I)^{-1} y}_{:=w} = \sum_{n=1}^{N} w_n K(x, x_n),$$

and given a particular dataset, we can use the reproducing kernel map representation to express the posterior mean as a sum of functions, see Figure 33.

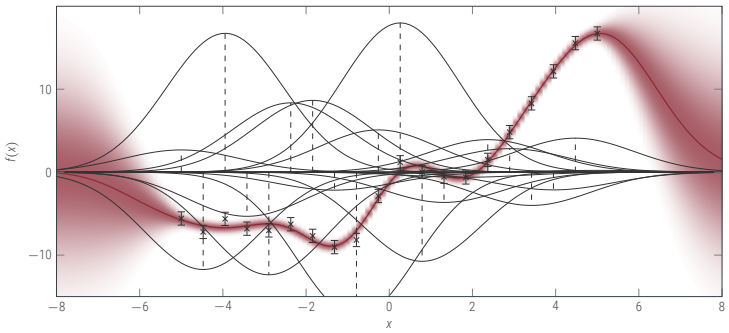


Figure 33: **Posterior mean of a GP with RBF kernel,** represented as a sum of individual Gaussians, centered at the observed points, scaled by their posterior weights.

Note, however, that Gaussian Process *draws* are not part of the RKHS; only the posterior mean is. While the distribution of the GP can be *characterized* in terms of the eigenfunctions and eigenvalues (Thm 29), the individual samples are *not* part of the RKHS (Thm 30) - however, GP samples are *close* to be in the RKHS (Thm 31). In practice, we can usually ignore the distinction and think of GP samples as RKHS functions.

For a more thorough treatment, see Theorems 4.3 and 4.9 in

Kanagawa, Hennig, Sejdinovic, and Sriperumbudur. *Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences*. CoRR, 2018. URL arxiv.org/abs/1807.02582

Theorem 29 (Karhunen-Loève expansion)**.** Let $\mathbb{X}, K, \nu, (\phi_i, \lambda_i)_{i \in \mathcal{I}}$ as above, and let $(z_i)_{i \in \mathcal{I}}$ be a collection of i.i.d. standard Gaussian random variables,

$$z_i \sim \mathcal{N}(0, 1) \qquad \text{and } \mathbb{E}\left[z_i, z_j\right] = \delta_{ij},\ for i, j \in \mathcal{I}.$$

Then,

$$f(x) = \sum_{i \in \mathcal{I}} z_i \lambda_i^{1/2} \phi_i(x) \sim \mathcal{GP}(0, K).$$

Theorem 30 (Wahba, 1990. See also Thm 4.9 in Kanagawa et al.)**.** If I is infinite, $f \sim \mathcal{GP}(0, K)$ implies almost surely that $f \notin \mathcal{H}_K$. To see this, note that

$$\mathbb{E}\left[\|f\|_{\mathcal{H}_K}^2\right] = \mathbb{E}\left[\sum_{i \in \mathcal{I}} z_i^2\right] = \sum_{i \in \mathcal{I}} 1 \not< \infty.$$

Theorem 31 (Kanagawa, 2018. Restricted from Steinwart, 2017, itself generalized from Driscoll, 1973)**.** Let $\mathcal{H}_K$ be a RKHS and let $\theta \in (0, 1]$. Consider the θ-power of $\mathcal{H}_K$, given by

$$\mathcal{H}_K^\theta = \left\{ f(x) := \sum_{i \in \mathcal{I}} \alpha_i \lambda_i^{\theta/2} \phi_i(x) \text{ such that } \|f\|_{\mathcal{H}_K^\theta}^2 \right\},$$

with

$$\langle f, g \rangle_{\mathcal{H}_K^\theta} := \sum_{i \in \mathcal{I}} \beta_i \gamma_i.$$

Then,

$$\sum_{i \in \mathcal{I}} \lambda_i^{1-\theta} < \infty \Rightarrow f \sim \mathcal{GP}(0, K) \in \mathcal{H}_K^\theta \text{ with probability 1.}$$

Markov Chains: Models for Time Series

Definition 32 (Time series)**.** A time series is a sequence $[y(t_i)]_{i\in\mathbb{N}}$ of observations $y_i := x(t_i) \in \mathbb{Y}$, indexed by a scalar variable $t \in \mathbb{R}$. In many applications, the time points t_i are equally spaced: $t_i = t_0 + i\delta_t$. Models that account for all values $t \in \mathbb{R}$ are called continuous time, while models that only consider $[t_i]_{i\in\mathbb{N}}$ are called discrete time.

Examples include climate & weather observations, sensor readings in cars, EEG, ECG, patch clamp signals, just about any sensing of a dynamical process in Physics stock prices, supply & demand data, polling numbers and even body weight measurements.

Inference in time series often has to happen in real-time, and scale to an unbounded set of data, typically on small-scale or embedded systems, so the computational and memory complexity has to be (low) constant. Without further assumptions, evaluating and inverting the whole kernel — in $\mathcal{O}\left(N^3\right)$, where N is the number of samples observed so far — every time a new sample is observed is unfeasible. Ideally, we would like to have a constant time and memory cost for each now sample. We will see that using the *Markov Property*, it is possible to treat each additional data point in constant time.

A joint distribution $p(X)$ over a sequence of random variables $X = [x_1, ..., x_N]$ is said to have the *Markov property* if

$$p(x_i|x_1, ..., x_{i-1}) = p(x_i|x_{i-1}),$$

that is, the observation at time i only depends on what happened at $i-1$. If we know x_i, no more information is gained on x_{i+1} by taking $x_1, ..., x_{i-1}$ into account. Such sequences are called *Markov Chains*, and they give rise to much simpler models when compared to full Gaussian Processes models, as shown with graphical models in Figures 34, 35, 36. To work with Time Series, we will assume that the observations $y_1, ..., y_N$, made at times $[t_1, ..., t_N]$, are generated from a transformation H of some latent space $x_1, ..., x_N$. Such models are known as *state-space models*

Graphical View of Time Series using Gaussian processes

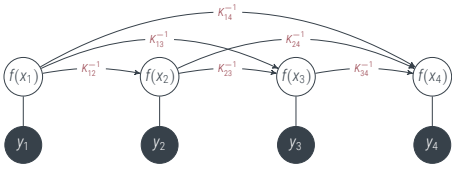


Figure 34: A Full GP Model

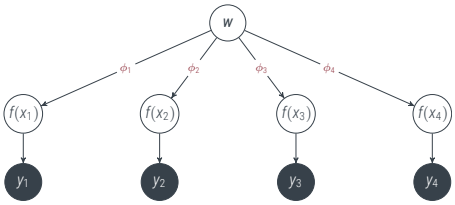


Figure 35: A parametric model

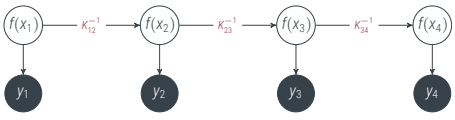


Figure 36: A Markov Chain model

MARKOV CHAINS formalize the notion of a stochastic process with a local finite memory, represented as the latent space x. To be able to build a probabilistic model, we need to define a transition probability over the latent space, $p(x_t|x_{t-1})$ and a likelihood function $p(y_t|x_t)$. From the Markov property, we have that

$$p(x_t|x_1, ..., x_{t-1}) = p(x_t|x_{t-1}) \text{ and } p(y_t|x_1, ..., x_t) = p(y_t|x_t).$$

Inference over Markov Chains separates into three operations, two of them can be performed in constant time at each observation and are collectively called *Filtering*[23],

[23] Also known as the Chapman-Kolmogorov equation.

- **Predict:** Given $Y_{1:t} = [y_1, ..., y_{t-1}]$, we can predict the value of x_t;

$$p(x_t|Y_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|Y_{1:t-1})\,\mathrm{d}x_{t-1}.$$

- **Update:** We can refine the probability distribution over the latent space at x_t after observing y_t;

$$p(x_t|Y_{1:t}) = \frac{p(y_t|x_t)p(x_t|Y_{1:t-1})}{p(y_t)}.$$

And a **smoothing** operation, after having observed the whole dataset Y (i.e., not in streaming), which can be performed in linear time,

$$p(x_t|Y) = p(x_t|Y_{1:t}) \int p(x_{t+1}|x_t) \frac{p(x_{t+1}|Y)}{p(x_{t+1}|Y_{1:t})}\,\mathrm{d}x_{t+1}$$

Gauss-Markov Models

If all relationships are linear and Gaussian, such that

$$p(x_t|x_{t-1}) = \mathcal{N}\left(x_t; Ax_{t-1}, Q\right) \quad \text{and} \quad p(y_t|x_t) = \mathcal{N}\left(y_t; Hx_t, R\right),$$

then the inference procedure can be carried out analyticaly and is given by the *Kalman Filter* and the *Rauch-Tung-Striebel Smoother*. The following figure illustrate a 1-dimensional example of such a model:

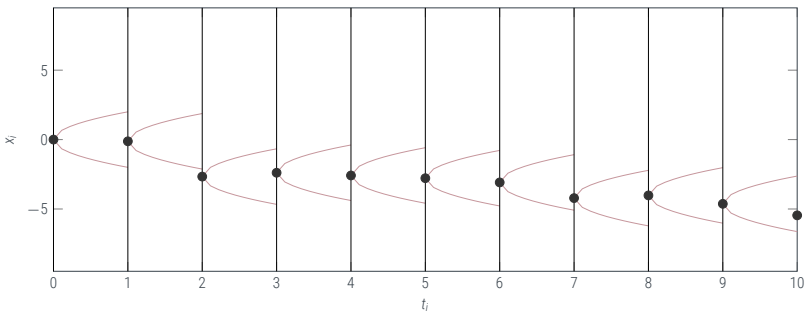


Figure 37: **Simple example of a univariate Gauss-Markov model,** using $A = Q = 1$, $y_t = x_t$.

THE EQUATIONS to solve this system are as follow:

- **Prediction:**

$$\begin{aligned} p(x_t|Y_{1:t-1}) &= \int p(x_t|x_{t-1})p(x_{t-1}|Y_{1:t-1})\,\mathrm{d}x_{t-1}, \\ &= \mathcal{N}\left(x_t; Am_t, AP_tA^\top + Q\right), \\ &= \mathcal{N}\left(x_t, m_t^-, P_t^-\right). \end{aligned}$$

Lexicon of variable names

Filtering: Prediction

Variable	Name
$m_t^- = Am_{t-1}$	predictive mean
$P_t^- = AP_{t-1}A^\top + Q$	predictive covariance

- **Update:**

$$\begin{aligned} p(x_t|Y_{1:t}) &= \frac{p(y_t|x_t)p(x_t|Y_{1:t-1})}{p(y_t)}, \\ &= \mathcal{N}\left(x_t, m_t^- + Kz, (I-KH)P_t^-\right), \\ &= \mathcal{N}\left(x_t, m_t, P_t\right), \end{aligned}$$

where

$$\begin{aligned} K &:= P_t^- H^\top (HP_tH^\top + R)^{-1}, \\ z &:= y_t - Hm_t^-. \end{aligned}$$

Filtering: Update

Variable	Name
$z_t = y - Hm_t^-$	innovation residual
$S_t = HP_t^-H^\top + R$	innovation covariance
$K_t = P_t^-H^\top S^{-1}$	Kalman gain
$m_t = m_t^- + Kz_t$	estimation mean
$P_t = (I-KH)P_t^-$	estimation covariance

- **Smoothing:**

$$\begin{aligned} p(x_t|Y) &= p(x_t|Y_{1:t}) \int p(x_{t+1}|x_t) \frac{p(x_{t+1}|Y)}{p(x_{t+1}|Y_{1:t})}\,\mathrm{d}x_{t+1}, \\ &= \mathcal{N}\left(x_t, m_t + G_t(m_{t+1}^s - m_{t+1}^-), P_t + G_t(P_{t+1}^s - P_{t+1}^-)G_t^\top\right), \\ &= \mathcal{N}\left(x_t, m_t^s, P_t^s\right), \end{aligned}$$

where

$$G_t := P_tA^\top (P_t^-)^{-1}$$

Smoothing

Variable	Name
$G_t = P_tA^\top (P_{t+1}^-)^{-1}$	RTS gain
$m_t^s = m_t + G_t(m_{t+1}^s - m_{t+1}^-)$	smooth mean
$P_t^s = P_t + G_t(P_{t+1}^s - P_{t+1}^-)G^\top$	smooth cov.

Hidden Markov Models

Fully Gaussian Markov models are easy to use, but many combination of transition probabilities and likelihoods are possible. Because streaming data is a common data type, time series are an entire sub-field of their own, studied in a diverse range of domains. There is no time to cover them all, but the following Table gives some references

Name	Distribution $p(y, x)$	Algorithm
Markovian System (general)	$\prod_{i=1}^N p(x_i\|x_{i-1})p(y_i\|x_i)$	Filter
Linear Gaussian System	$\prod_{i=1}^N \mathcal{N}\left(x_i; A_ix_{i-1}, Q_i\right)\mathcal{N}\left(y_i; Hx_i, R\right)$	Kalman filter
Nonlinear Gaussian System	$\prod_{i=1}^N \mathcal{N}\left(x_i; a(x_{i-1}), Q_i\right)\mathcal{N}\left(y_i; h(x_i), R\right)$	e.g. Extended/Particle filter
Non-Gaussian observations	$\prod_{i=1}^N \mathcal{N}\left(x_i; a(x_{i-1}), Q_i\right)p(y_i\|h(x_i))$	e.g. Classification
Hidden Markov Model (e.g.)	$\prod_{i=1}^N p(x_i = \prod x_{i-1}\mathcal{N}\left(y_i; h(x_i), R\right)$	Markov Chain Monte Carlo

Table 1: **Other Markov models**

Exponential Family

The hardest part of computing the posterior in Bayes' rule,

$$p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x)p(x)\,\mathrm{d}x},$$

comes from the integral for the evidence $Z = \int p(y|x)p(x)\,\mathrm{d}x$. Computing expectations and the optimization problem to obtain the (Type-2) Maximum-A-Posteriori solution,

$$\mathbb{E}_{p(x|y)}\left[f(x)\right] = \int f(x)p(x|y)\,\mathrm{d}x \quad \text{and} \quad x^\star = \arg\max_x p(x|y),$$

also inherit from this difficulty. Using Gaussian distributions, we have seen that we can sidestep the problem: if we have a Gaussian prior and a Gaussian Likelihood, the posterior is also Gaussian and the solutions can be computed analytically.

Conjugate Priors

We will extend this idea with the concept of *conjugate priors*. A prior is said to be *conjugate to* a likelihood if the posterior arising from the combination of the likelihood and the conjugate prior has the *same form* as the prior. The Gaussian prior is the *conjugate prior* to the Gaussian Likelihood.

For binary distributions, such as the probability π that a coin flip ends up on head, we have that the likelihood of seeing a heads and b tails in a sequence $x_1, ..., x_{a+b}$ is

$$p(x|\pi) = \pi^a(1-\pi)^b.$$

In an earlier chapter, we have seen that the Beta distribution[24] was a sensible choice of prior, with hyper-parameters α, β,

[24] wikipedia.org/wiki/Beta_distribution

$$p(\pi) = \mathcal{B}\left(\pi; \alpha, \beta\right) = \frac{\pi^{\alpha-1}(1-\pi)^{\beta-1}}{B(\alpha,\beta)},$$

where the beta function, $B(\alpha, \beta)$, is the normalization constant, as it leads to a posterior that is also a Beta distribution,

$$p(\pi|x) = \mathcal{B}\left(\alpha + a, \beta + b\right) = \frac{\pi^{\alpha+a-1}(1-\pi)^{\beta+b-1}}{B(\alpha+a,\beta+b)}.$$

FOR CATEGORICAL DISTRIBUTIONS, where an observation x can be one of K classes, writing n_k the number of observations of class k and π_k the probability of seeing an observation of class k, the likelihood is given by

$$p(x|\pi) = \prod_k \pi_k^{n_k}.$$

Taking a Dirichlet[25]distribution with hyperparameters $\alpha_1, ..., \alpha_K$ as the prior,

[25] wikipedia.org/wiki/Dirichlet_distribution

$$p(\pi) = \mathcal{D}(\alpha_1, ..., \alpha_K) = \frac{1}{B(\alpha_1, ..., \alpha_K)} \prod_k \pi_k^{\alpha_k - 1},$$

leads to a Dirichlet posterior, $p(\pi_1, ..., \pi_k|x) = \mathcal{D}(\alpha_1 + n_1, ..., \alpha_K + n_K)$

THE FORMAL DEFINITION of a conjugate prior is as follow; for more details, see [26,27].

[26] E. Pitman. *Sufficient statistics and intrinsic accuracy*. Mathematics Proceedings of the Cambridge Philosophical Society, 1936

[27] P. Diaconis and D. Ylvisaker. *Conjugate priors for exponential families*. Annals of Statistics, 1979

Definition 33 (Conjugate prior)**.** Let D and x be a data-set and a variable to be inferred, connected by the likelihood $p(D|x) = \ell(D; x)$. A *conjugate prior to ℓ for x* is a probability measure with pdf $p(x) = \pi(x, \theta)$ of functional form π, such that

$$p(x|D) = \frac{\ell(D; x)\pi(x; \theta)}{\int \ell(D; x)\pi(x; \theta)\, \mathrm{d}x} = \pi(x; \theta').$$

That is, the posterior arising from ℓ is of the same functional form as the prior, with updated parameters.

Conjugate priors allow for analytic Bayesian inference, and we will now see a nice family of distributions where conjugate priors can easily be found.

Exponential Family

The exponential family is the family of probability distributions that are formed by taking the exponential of a linear form.

Definition 34 (Exponential Family)**.** A probability distribution over a variable $x \in \mathbb{X} \subset \mathbb{R}^n$ with the functional form

$$p_w(x) = \exp\left(\phi(x)^\top w - \log Z(w)\right) = \frac{1}{Z(w)} \exp\left(\phi(x)^\top w\right),$$

is called an *exponential family* of probability measures. The function $\phi : \mathbb{X} \to \mathbb{R}^d$ is called the *sufficient statistics*, the parameters $w \in \mathbb{R}^d$ are called the *natural parameters* of p_w and the normalization constant $Z : \mathbb{R}^d \to \mathbb{R}$ is the *partition function*.

Some examples of Exponential Family distributions, with sufficient statistics and domain, are shown in Table 2.

Distribution	$\phi(x)$	$\mathbb{X}$
Bernoulli	$[x]$	$\{0,1\}$
Poisson	$[x]$	$\mathbb{R}_+$
Laplace	$[1,x]$	$\mathbb{R}$
χ^2	$[x,-\log x]$	$\mathbb{R}$
Dirichlet	$[\log x]$	$\mathbb{R}_+$
Euler (Γ)	$[x,\log x]$	$\mathbb{R}_+$
Wishart	$[X,\log\det X]$	$\{X\in\mathbb{R}^{N\times N}: v^\top Xv\geq 0\forall v\in\mathbb{R}^N\}$
Gaussian	$[x,xx^\top]$	$\mathbb{R}^N$
Boltzmann	$[X,\text{triag}(XX^\top)]$	$\{0,1\}^N$

Table 2: (Incomplete) List of Exponential Family distributions.

See wikipedia.org/wiki/Exponential_family#Table_of_distributions for a more exhaustive list.

EXPONENTIAL FAMILY HAVE CONJUGATE PRIORS. Taking the Exponential Family likelihood

$$p_w(x|w)=\exp\left(\phi(x)^\top w-\log Z(w)\right),$$

the prior parametrized with α and ν,

$$p_\alpha(w)=\exp\left[\begin{pmatrix}w\\-\log Z(w)\end{pmatrix}^\top\begin{pmatrix}\alpha\\\nu\end{pmatrix}-\log F(\alpha,\nu)\right],$$

where the normalization constant $F(\alpha,\nu)$ is given by

$$F(\alpha,\nu)=\int\exp\left(\alpha^\top w-\nu^\top\log Z(w)\right)\mathrm{d}w,$$

gives rise to the posterior

$$p_\alpha(w|\alpha,\nu)\prod_{i=1}^n p_w(x_i|w)\propto p_\alpha(w|\alpha+\sum_i\phi(x_i),\nu+n).$$

The predictive distribution $p(x)$ can similarly be computed,

$$\begin{aligned}p(x)&=\int p_w(x|w)p_\alpha(w|\alpha,\nu)\,\mathrm{d}w,\\&=\int\exp\left((\phi(x)+\alpha)^\top w+(\nu+1)\log Z(w)+\log F(\alpha,\nu)\right)\mathrm{d}w,\\&=\frac{F(\phi(x)+\alpha,\nu+1)}{F(\alpha,\nu)}.\end{aligned}$$

Computing $F(\alpha,\nu)$ can be tricky, and in general is the main challenge when constructing an Exponential Family distribution.

FOR THE GAUSSIAN DISTRIBUTION, the natural parameters are the precision σ^{-2} and precision-adjusted mean $\mu\sigma^{-2}$ and the sufficient statistics are the first and $(1/2)$ second moment,

$$\begin{aligned}\mathcal{N}\left(x;\mu,\sigma^2\right)&=\exp\left(\begin{pmatrix}x\\-\frac{1}{2}x^2\end{pmatrix}^\top\begin{pmatrix}\mu/\sigma^2\\1/\sigma^2\end{pmatrix}+\left(\frac{\mu^2}{2\sigma^2}+\log\sqrt{2\pi\sigma^2}\right)\right),\\&=\exp\left(\begin{pmatrix}\phi_1(x)\\\phi_2(x)\end{pmatrix}^\top\begin{pmatrix}w_1\\w_2\end{pmatrix}+\left(\frac{w_1^2}{2w_2}-\frac{1}{2}\log w_2+\log\sqrt{2\pi}\right)\right).\end{aligned}$$

The conjugate prior is the *Normal-Gamma*[28](the conjugate prior for the mean is a Normal distribution, the conjugate prior for the covariance is a Gamma distribution) and the predictive distribution is the *Student-t*[29].

[28] wikipedia.org/wiki/Normal-gamma_distribution

[29] wikipedia.org/wiki/Student's_t-distribution

SUFFICIENT STATISTICS owe their name to the fact that they are sufficient for maximum likelihood estimation; no other information about the data is necessary to find the best fit. To see why, take the likelihood of n iid samples for the exponential family,

$$p(x_1, ..., x_n|w) = \prod_{i=1}^{n} p(x_i|w) = \exp\left(\sum_{i=1}^{n} \phi(x_i)^\top w - n \log Z(w)\right).$$

The maximum likelihood estimate for w is found at

$$\nabla_w \log p(x_1, ..., x_n|w) = 0 \quad \Rightarrow \quad \nabla_w \log Z(w) = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i).$$

Hence, it suffice to collect the statistics $\phi(x_i)$, compute $\nabla_w \log Z(w)$ and solve for $w^\star$.

Graphical Models

In the early chapters, we have seen that to represent a full joint probability distribution over 4 variables would require $2^4 - 1 = 15$ parameters,

$$p(a, b, c, d) = p(a|b, c, d)p(b|c, d)p(c|d)p(d),$$

but removing irrelevant conditions (based on domain knowledge) can reduce the number of required parameters. If, for example,

$$p(a, b, c, d) = p(a|b, c)p(b|d)p(c)p(d),$$

we only need 8 parameters to represent the distribution. Graphical models provide a nice language to convey this independence information, and we will go into more details in this chapter.

Directed Graphical Models

Remember the construction of directed graphical models (or Bayesian networks) to graphically represent conditional independence between variables;

1. For each variable in the joint distribution, draw a circle
2. For each term $p(x_1, ..., |y_1, ...)$ in the factorized joint distribution, draw an arrow from every parent (right side, y_i) to every child (left side, x_i)
3. Fill in all observed variables (variables we want to condition on)

leading to such a graphical model such as the one shown in Fig. 38.

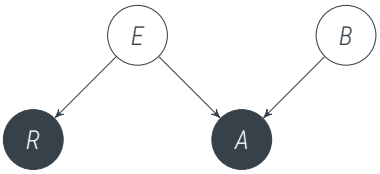


Figure 38: Directed Graphical Model for the factorization

$p(A, E, B, R) = p(A|E, B)p(R|E)p(E)p(B)$.

Repeated observations and hyperparameters have some syntactic sugar to make it easier to draw complex graphical models. A box with sharp edges drawn around a set of nodes and labeled with a number n is called a *plate* and denotes n copies of the content of the box. A small filled circle denotes a (hyper-)parameter that is set or optimized, and which is not part of the generative model.

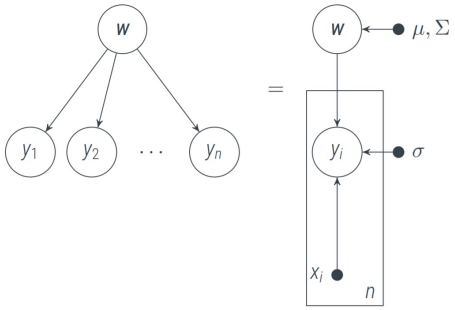


Figure 39: Plates and Hyperparameters

Independence and Directed Graphs

By the product rule, every joint probability distribution can be factorized, but not every factorization is useful. Directed graphs are also an imperfect representation, as a joint probability distribution can have multiple factorizations, each leading to a different graphs expressing some of the independencies, but not all. Remember the atomic independence structure we surveyed in the early chapters:

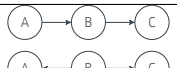
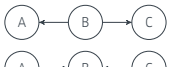
$p(A,B,C)$	DAG	Independence	but!
$p(C\|B)p(B\|A)p(A)$		$A \perp\!\!\!\perp C\|B$	$A \not\perp\!\!\!\perp C$
$p(A\|B)p(C\|B)p(B)$		$A \perp\!\!\!\perp C\|B$	$A \not\perp\!\!\!\perp C$
$p(B\|A,C)p(A)p(C)$		$A \perp\!\!\!\perp C$	$A \not\perp\!\!\!\perp C\|B$

Figure 40: **Independence structure for tri-variate subgraphs.**

A more general statement about independence, d-separation, comes from Pearl[30], and the following presentation is from Bishop[31].

[30] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988

[31] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. URL https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-pdf

Theorem 35 (d-separation)**.** Consider a general directed acyclic graph, in which A, B, C are nonintersecting sets of nodes whose union may be smaller than the complete graph. To ascertain whether $A \perp\!\!\!\perp B|C$, consider all possible paths, regardless of the direction, from any node in A to any node in B. Any such path is considered blocked if it includes a node such that either

- the arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in C, or
- the arrows meet head-to-head at the node, and neither the node, nor any of its descendants is in C.

If all paths are blocked, then A is said to be d-separated from B by C, and $A \perp\!\!\!\perp B|C$.

Thus, all further considerations about computations on the graph can be made in a local fashion based on *Markov Blankets*

Definition 36 (Markov Blanket – for directed graphs)**.** The Markov Blanket of node x_i is the set of all parents, children, and co-parents[32] of x_i. Conditioned on the blanket, x_i is independent of the rest of the graph.

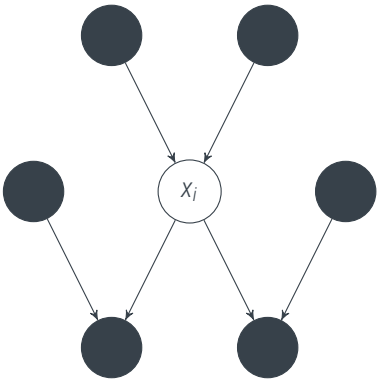


Figure 41: Example of a Markov Blanket for a directed graph

[32] The co-parents of x are the (other) parents of the children of x

The directed nature of connections in Bayesian belief networks reflects the fact that a conditional probability has a left- and right-hand side, $p(x|a)$. This is convenient since it allows writing down the graph directly from the factorization, but conditional independence statements (d-separation) is tricky. Blocking a path requires notions of parents and co-parents, and different rules depending on whether arrows meet head-to-head or head-to-tail. There are joint distributions whose set of conditional independences can not be represented by a single directed graph.

Undirected Graphical Models

Undirected Graphical Models, or Markov Random Fields (MRF), are another notation in which conditional independence can be more simply stated as "two nodes are independent if all paths connecting them are blocked".

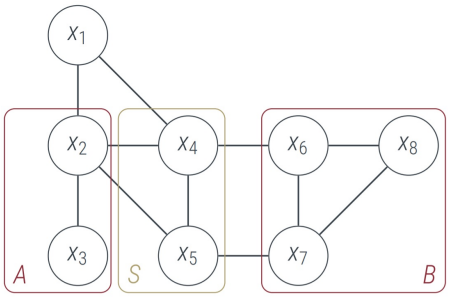


Figure 42: Markov Random Field with Separating set

Definition 37 (Markov Random Field)**.** An undirected Graph $G = (V, E)$ is a set V of nodes and edges E. G and a set of random variables mapping to the nodes $X = \{X_v\}_{v \in V}$ form a *Markov Random Field* if, for any subsets $A, B \subset V$ and a separating set S (a set such that every path from A to B passes through S), $X_A \perp\!\!\!\perp X_B | X_S$.

The above definition is known as the *global Markov property*. It implies the weaker *pairwise Markov property*: Any two nodes u, v that do not share an edge are conditionally independent given all other variables: $X_u \perp\!\!\!\perp X_v | X_{V \setminus \{u,v\}}$.

Markov Blankets are simpler for Markov Random Fields;

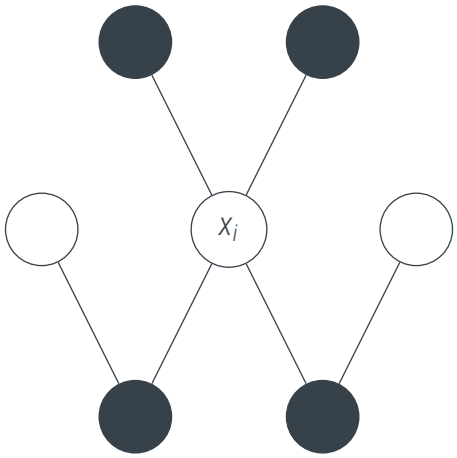


Figure 43: Markov Blanket for a Markov Random Field

Definition 38 (Markov Blanket – for undirected graphs)**.** For a Markov Random Field, the Markov Blanket of node x_i is the set of all direct neighbors of x_i. Conditioned on the blanket, x_i is independent of the rest of the graph.

Essentially by definition, MRFs allow for a more compact definition of conditional independence than directed graphs, but the associated joint probability distribution is not that easily read from the graph. By the pairwise Markov property, any two nodes x_i, x_j *not* connected by an edge have to be conditionally independent given the rest of the graph. Thus, the joint has to factorize as

$$p(x_i, x_j | x_{\setminus \{i,j\}}) = p(x_i | x_{\setminus \{i,j\}}) p(x_j | x_{\setminus \{i,j\}})$$

Hence, for the factorization to hold, nodes that do not share an edge must not be in the same factor. This leads to the use of *cliques* to define the factorization.

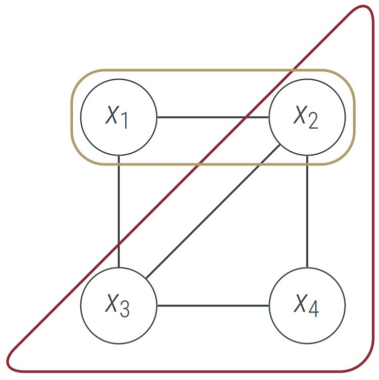


Figure 44: A clique (in gold) and maximal clique (in red)

Definition 39 (Clique)**.** Given a graph $G = (V, E)$, a clique is a subset $c \subset V$ such that there exists an edge between all pairs of nodes in c. A *maximal clique* is a clique such that it is impossible to include any other nodes from V, without it ceasing to be a clique.

Any distribution $p(x)$ that satisfies the conditional independence structures of the graph G can be written as a factorization over all cliques, and thus also just over all maximal cliques since any clique is part of at least one maximal clique. Writing C the set of all maximal cliques,

$$p(x_1, ..., x_n) = \frac{1}{Z} \prod_{c \in C} \psi_c(\{x_i \in c\}).$$

In directed graphs, each factor $p(x_{\text{ch}} | x_{\text{pa}})$ had to be a probability distribution of the children, and not of the parents. In MRFs, there

is no distinction between parents and children, so we only know that each potential function $\psi_c(\{x_i \in c\}) \geq 0$. The normalization constant Z is the partition function

$$Z := \int \prod_{c \in C} \psi_c(\{x_i \in c\}) \, \mathrm{d}x_1, ..., x_n.$$

Because of the loss of structure from directed to undirected graphs, we have to explicitly compute Z. This can be NP-hard, and is the primary downside of MRFs; in the case of n discrete variables with k states each, computing Z may require summing k^n terms.

The Boltzmann distribution

Markov Random Fields with Positive Potentials ($\psi_c(\{x_i \in c\}) > 0$) are Exponential Families as we can write

$$\psi_c(\{x_i \in c\}) = \exp(-E_c(\{x_i \in c\}))$$

for some function E_c, and introduce the scaling factors w_c to get

$$p(x_1, ..., x_n) = \exp\left(- \sum_{c \in C} w_c E_c(\{x_i \in C\}) + \log Z \right).$$

This gives rise to a Boltzmann distribution (or Gibbs measure),

Definition 40 (Boltzmann distribution)**.** A probability distribution with pdf of the form

$$p(x) = \exp(-E(x))$$

is called a Boltzmann or Gibbs distribution, and $E(x)$ is known as the energy function.

Any Gibbs measure, and thus any MRF, is an exponential family; it may not necessarily the helpful kind because $Z(w_c)$ is intractable.

The Gaussian case

For a set of variables $x_1, ..., x_n$ that are jointly Gaussian distributed,

$$p(\boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{x}; \mu, \Sigma\right),$$

the MRF can be constructed directly from the inverse covariance; If the inverse covariance (aka. precision) matrix contains a zero at element, $[\Sigma^{-1}]_{ij}$, then $x_i \perp\!\!\!\perp x_j | x_{\setminus i,j}$, and thus an edge exists between each node that has $[\Sigma^{-1}]_{ij} \neq 0$.

From Directed to Undirected graphs

Given a directed graph, it is possible to find an equivalent undirected graph, sometime very easily. For Markov Chains, the mapping is direct as

$$\begin{aligned} p(x) &= p(x_1)p(x_2|x_1)...p(x_n|x_{n-1}), \\ &= \frac{1}{Z}\psi_{1,2}(x_1, x_2)...\psi_{n-1,n}(x_{n-1}, x_n) \end{aligned}$$

Figure 45: Directed and Undirected graph for a Markov chain

In general, we need to ensure that each conditional term in the directed graph is captured in at least one clique of the undirected graph. For nodes with only one parent, we can simply drop the arrow, and get $p(x_c|x_p) = \psi_{c,p}(x_c, x_p)$. For nodes with several parents, we have to connect all the parents. This process is known as *moralization*, and frequently leads to densely connected graphs, losing all value of the graph.

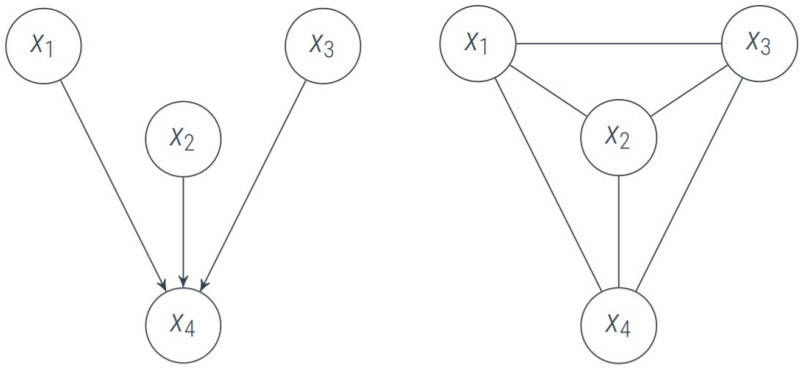
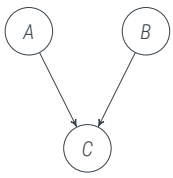


Figure 46: Directed to Undirected, after moralization

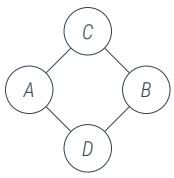
Directed vs. Undirected graphs

Directed and undirected graphs offer tools to graphically represent and inspect properties of joint probability distributions. Both are primarily a design tool, and each framework has its strengths and weaknesses.

In the following figure, the conditional independence properties of the directed graph on the left can not be represented by any MRF over the same three variables; and the conditional independence properties of the MRF on the right can not be represented by any directed graph on the same four variables.



$A \perp\!\!\!\perp B \mid \varnothing$ and $A \not\perp\!\!\!\perp B \mid C$



$x \not\perp\!\!\!\perp y \mid \varnothing \; \forall x, y$ and $C \perp\!\!\!\perp D \mid A \cup B$ and $A \perp\!\!\!\perp B \mid C \cup D$

DIRECTED GRAPHICAL MODELS, or Bayesian networks directly encode a factorization of the joint – it can be read off by parsing the graph from the children to the parents. However, reading off conditional independence structure is tricky as it requires considering d-separation.

Directed graphs are a direct mapping of generative processes; they tend to be useful in highly structured problems with mixed data types; physical, biological, chemical or social processes; where causal structure is known. When you want to model a process for which you have a "scientific" theory or some *generative knowledge*, you might want to start by writing down the directed model.

UNDIRECTED GRAPHICAL MODELS, or Markov Random Fields (MRFs), directly encode conditional independence structure – by definition. However, reading off the joint from the graph is tricky as it requires finding all maximal cliques, and the normalization constant is (usually) intractable.

MRFs tend to be useful in highly regular but high-dimensional problems with unclear generative model, such as those encountered in computer vision and statistical physics. When your model has millions of parameters and you are more worried about computational complexity than interpretability, the conditional independence structure of MRFs, encoding *computational constraints*, can help keep things tractable.

Factor graph

In the last chapter, we saw directed and undirected graphs as tools to graphically represent and inspect properties of joint probability distributions. Both are primarily a design tool, each with its strengths and weaknesses. In this chapter, we will see a third type of graphical model, particularly well-suited for automated inference, along with a general-purpose algorithm for automated inference and efficient Maximum-A-Posteriori computations.

Factor Graphs

Factor Graphs are an explicit representation of functional relationships;

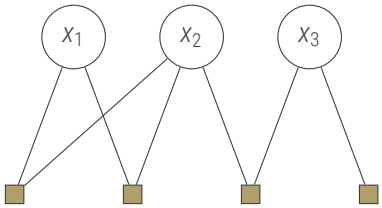


Figure 47: Example of a factor graph, with variables nodes x_1, x_2, x_3 and factors in boxes.

Definition 41 (Factor graph)**.** A factor graph is a bipartite graph $G = (V, F, E)$ of variables $v \in V$, factors $f \in F$ and edges E, such that each edge connects a factor to a variable.

To construct a factor graph from a directed graph

$$p(x) = \prod_{\text{ch}} p(x_{\text{ch}} | x_{\text{pa(ch)}}),$$

draw a circle for each variable x_i, a box for each conditional in the factorization and connect each x_i to the factorizations it appears in.

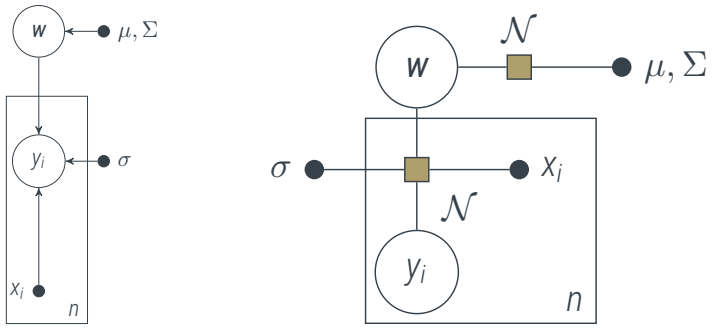


Figure 48: Conversion of a directed graph for a parametric regression to a factor graph

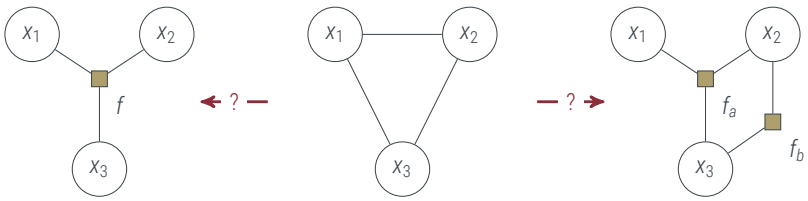
To construct a factor graph from a MRF,

$$p(x) = \frac{1}{Z} \prod_{c \in C} \psi_c(\{x_i \in c\}),$$

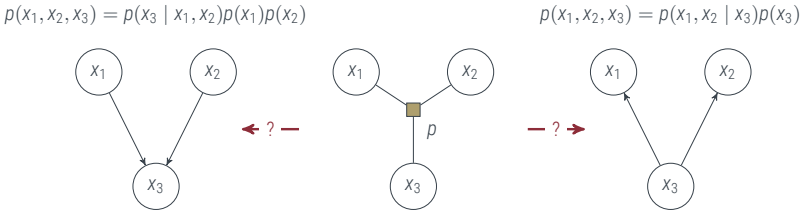
draw a circle for each variable xi, draw a box for each factor (clique) ψ_c and connect each ψ_c to the variables used in the factor.

Some properties of Factor Graphs

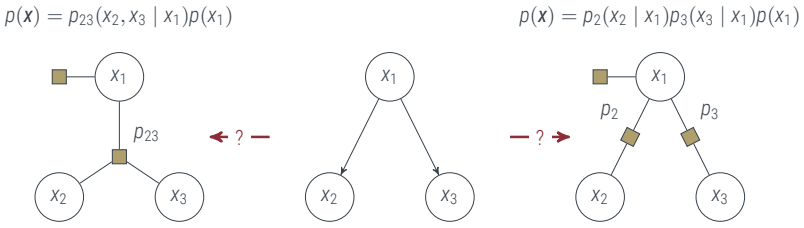
Factor Graphs can express structure not visible in MRFs:



Factor graphs can mask conditional independence:



Factor graphs can also reveal functional relationships:



The graphical view itself does not always capture all structure, but when factor graphs are encoded with the explicit functional form, some interesting structure can be automatically deduced and used for inference, leading to the Sum-Product algorithm.

The Sum-Product Algorithm

The Sum-Product, message passing or Belief propagation algorithm ([33],[34],[35]) leverages the structure of factor graphs to perform inference, i.e., compute a marginal distribution,

$$p(x_i) = \int p(x_1, ..., x_i, ..., x_n)\, \mathrm{d}x_{j \neq i},$$

if the joint $p(x_1, ..., x_n)$ is given by a factor graph.

[33] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988

[34] S.L. Lauritzen and D.J. Spiegelhalter. *Local computations with probabilities on graphical structures and their application to expert systems*. Journal of the Royal Statistical Society, 1988

[35] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. *Factor graphs and the sum-product algorithm*. IEEE Transactions on Information Theory, 2001

Base case: Markov chain

Filtering and Smoothing are special cases of the sum-product algorithm on chains. For simplicity, consider the Markov Chain with discrete $x_i \in [1, ..., k]$, such that

$$p(x_1, ..., x_n) = \frac{1}{Z} \psi_{0,1}(x_0, x_1) ... \psi_{n-1,n}(x_{n-1}, x_n)$$

Figure 49: Factor Graph of a Markov Chain

The marginal $p(x_i)$ is then given by

$$\begin{aligned}
p(x_i) &= \sum_{x_{\neq i}} p(x_1, .., x_n), \\
&= \frac{1}{Z} \underbrace{\left(\sum_{x_{i-1}} \psi_{i-1,i}(x_{i-1}, x_i) \left(... \left(\sum_{x_0} \psi_{0,1}(x_0, x_1) \right) \right) \right)}_{:= \mu_{\rightarrow}(x_i)} \underbrace{\left(\sum_{x_{i+1}} \psi_{i,i+1}(x_i, x_{i+1}) \left(... \left(\sum_{x_n} \psi_{n-1,n}(x_{n-1}, x_n) \right) \right) \right)}_{:= \mu_{\leftarrow}(x_i)}, \\
&= \frac{1}{Z} \mu_{\rightarrow}(x_i) \mu_{\leftarrow}(x_i),
\end{aligned}$$

with $Z = \sum_{x_i} \mu_{\rightarrow}(x_i) \mu_{\leftarrow}(x_i)$. $\mu_{\rightarrow}(x_i)$ and $\mu_{\leftarrow}(x_i)$ are called *message*, and are defined recursively;

$$\begin{aligned}
\mu_{\rightarrow}(x_i) &= \sum_{x_{i-1}} \psi_{i-1,i}(x_{i-1,i}) \mu_{\rightarrow}(x_{i-1}), \\
\mu_{\leftarrow}(x_i) &= \sum_{x_{i+1}} \psi_{i,i+1}(x_{i,i+1}) \mu_{\leftarrow}(x_{i+1}).
\end{aligned}$$

By storing local messages, all marginals can be computed in $\mathcal{O}\left(nk^2\right)$, as in filtering and smoothing. Computing a message from the preceding one can be done by taking the sum of the product of the local factor and incoming message, and the local marginal can be computed by taking the sum of the product of incoming messages, hence the name of the algorithm.

The sum-product algorithm splits the inference into local messages being sent forwards and backwards along the factor graph, and both the local marginals and the most-probable state can be inferred in this way.

Sum-Product on trees

The efficiency of the sum-product algorithm is preserved when, instead of chains, the graph is a *tree*.

Definition 42 (Tree)**.** An undirected graph is a tree if there is one, and only one, path between any pair of nodes (such graphs have no loops). A directed graph is a tree if there is only one node which has no parent (the root), and all other nodes have only one parent. When such graphs are transformed into undirected graphs by moralization, they remain a tree. A directed graph such that every pair of nodes is connected by one and only one path is called a polytree. When transformed into an undirected graph, such graphs, in general, acquire loops, but the corresponding factor graph is still a tree.

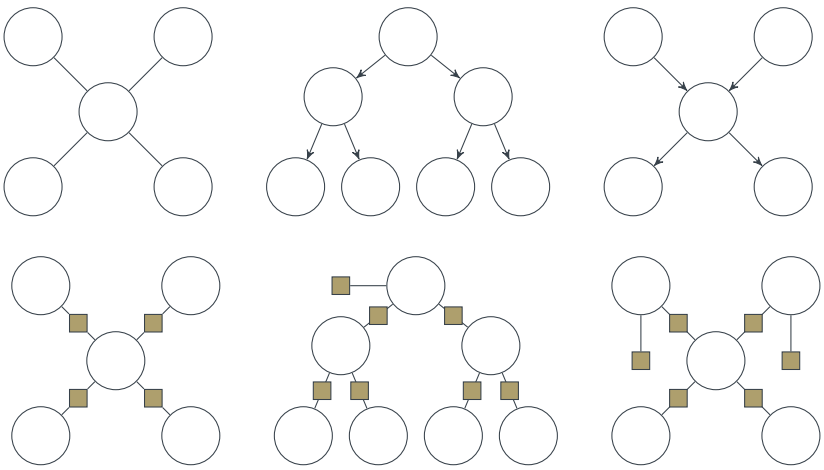


Figure 50: Directed and Undirected trees, and their factor-graph correspondance.

STARTING POINT: MESSAGES FROM FACTORS TO VARIABLES. For any variable x in the tree, the likelihood factorizes as

$$p(x) = \prod_{s \in \text{ne}(x)} F_s(x, \{x_j \in X_s\}),$$

where $\text{ne}(x)$ are the neighbors of x and F_S is the sub-graph of nodes X_s, other than x itself, that are connected to the neighbor s. The marginal distribution can then be written as

$$p(x) = \sum_{x_{j \neq i}} \prod_{s \in \text{ne}(x)} F_s(x, X_s) = \prod_{s \in \text{ne}(x)} \underbrace{\left(\sum_{x_j \in X_s} F_s(x, \{x_j \in X_s\}) \right)}_{:= \mu_{f_s \to x}(x)},$$

$$= \prod_{s \in \text{ne}(x)} \mu_{f_s \to x(x)},$$

the product of incoming messages $\mu_{f_s \to x}$ from the factors connected to the variable x.

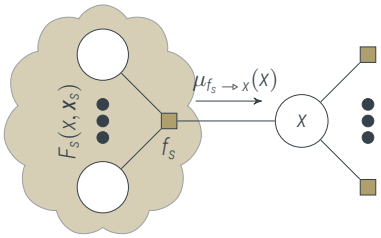


Figure 51: Message from factor to variables

FURTHER FACTORIZATION: MESSAGE FROM FACTORS TO VARIABLES AS A FUNCTION OF MESSAGES FROM VARIABLE TO FACTOR. The subgraphs $F_s(x, X_s)$ itself factorize further into tree-structure subgraphs,

$$F_s(x, X_s) = f_s(x, x_1, .., x_m) G_1(x_1, X_{s1}) ... G_m(x_m, X_{sm}),$$

where $\{x_1, ..., x_m\}$ are the nodes in X_s and X_{si} are the neighbors of x_i. The computation of the factor-to-variable message therefore factorize into the computation of variable-to-factor messages $\mu_{x \to f_s(x_i)}$;

$$\mu_{f_s \to x}(x) = \sum_{x_1, ..., x_m} f_s(x, x_1, ..., x_m) \prod_{i \in \text{ne}(f_s) \setminus x} \underbrace{\left(\sum_{X_{si}} G_i(x_i, X_{si}) \right)}_{\mu_{x_i \to f_s(x_i)}},$$

$$= \sum_{x_1, ..., x_m} f_s(x, x_1, ..., x_m) \prod_{i \in \text{ne}(f_s) \setminus x} \mu_{x_i \to f_s(x_i)}.$$

To compute the factor-to-variable messages $\mu_{f_s \to x}(x)$, *sum* over the *product* of the factor and remaining sub-graph-sums, which themselves are messages from the variables connected to the factor.

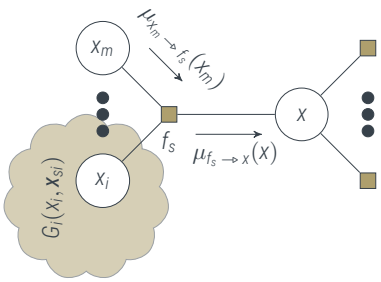


Figure 52: Further factorization of the subgraph F_s

THE VARIABLE TO FACTOR MESSAGES also factorize as

$$G_i(x_i, X_{si}) = \prod_{\ell \in \text{ne}(x_i) \setminus f_s} F_\ell(x_i, X_{i\ell}),$$

variable-to-factor message $\mu_{x_i \to f_s}$ as a product of the incoming factor-to-variables messages $\mu_{f_\ell \to x_i}$,

$$\mu_{x_i \to f_s(x_i)} = \sum_{X_{si}} G_i(x_i, X_{si}) = \sum_{X_{si}} \left(\prod_{\ell \in \text{ne}(x_i) \setminus f_s} F_\ell(x_i, X_{i\ell}) \right),$$

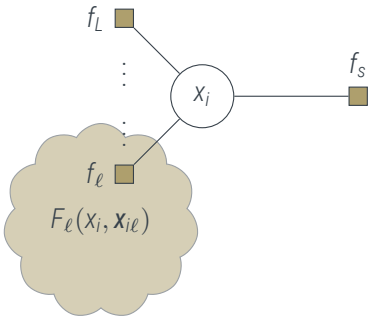


Figure 53: Messsages from the variables to the factors

THE SUM-PRODUCT ALGORITHM then repeats those steps until reaching a *leaf* node. To initiate the messages at leaves of the graph, which have no neighbor left, we define them to be unit for variable leaves and identities for factor leaves;

$$\mu_{x\to f}(x) = \prod_{\emptyset} \sum_{\emptyset} := 1$$
$$\mu_{f\to x}(x) = \sum_{\emptyset} f(x,\emptyset) \prod_{\emptyset} := f(x).$$

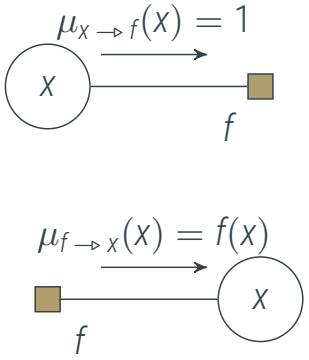


Figure 54: Messages in the sum-product algorithm

THE FULL ALGORITHM to compute the marginal of $p(x)$ treats x as the root of the tree and;

- Starting at the leaf nodes, initialize the messages as

$$\mu_{f\to x}(x) = f(x)$$
$$\mu_{x\to f}(x) = 1$$

- Pass messages from the leaves towards the roots using

$$\mu_{f_\ell ll x_j} = \sum_{X_{\ell j}} f_\ell(x_j, X_{\ell j}) \prod_{i\in\{\}=\mathrm{ne}(f_\ell)\setminus x_j} \mu_{x_i\to f_\ell}(x_i)$$
$$\mu_{x_j\to f_\ell}(x_j) = \prod_{i\in\mathrm{ne}(x_j)\setminus f_\ell} \mu_{f_i\to x_j}(x_j).$$

- At the root x, take the product of all incoming messages (and normalize).

To get the marginal of each node, once the root has received all the messages, pass messages *from* the root *to* the leaves again. Once all nodes ave received the messages from all their neighbors, take the product of all incoming messages at each variable (and normalize).

Generalization to any graph

There is a generalization from trees to general graphs, known as the junction tree algorithm. The principal idea is to join sets of variables in the graph into larger maximal cliques until the resulting graph is a tree. The exact process, however, requires care to ensure that every clique that is a sub-set of another clique ends up in that clique. The resulting algorithm (like the sum-product algorithm) has complexity exponential in the dimensionality of the largest variable in the graph, and linear in the size of tree.

The computational cost of probabilistic inference on the marginal of a variable in a joint distribution is exponential in the dimensionality of the maximal clique of the junction tree, and linear in the size of the junction tree. The junction tree algorithm is exact for any graph (it produces correct marginals), and efficient in the sense that, given a graph, there does not in general (i.e. without using properties of the functions instead of the graph) exist a more efficient algorithm.

Bibliography

Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. URL https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

P. Diaconis and D. Ylvisaker. *Conjugate priors for exponential families*. Annals of Statistics, 1979.

Jaynes. *Probability theory: The logic of science*. Cambridge University Press, 2003. URL bayes.wustl.edu/etj/prob/book.pdf.

Kanagawa, Hennig, Sejdinovic, and Sriperumbudur. *Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences*. CoRR, 2018. URL arxiv.org/abs/1807.02582.

Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. 1933.

F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. *Factor graphs and the sum-product algorithm*. IEEE Transactions on Information Theory, 2001.

S.L. Lauritzen and D.J. Spiegelhalter. *Local computations with probabilities on graphical structures and their application to expert systems*. Journal of the Royal Statistical Society, 1988.

MacKay. *The Evidence Framework Applied to Classification Networks*. Neural Computation, 1992.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

E. Pitman. *Sufficient statistics and intrinsic accuracy*. Mathematics Proceedings of the Cambridge Philosophical Society, 1936.

Rasmussen and Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

Schölkopf and Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.