Machine Learning and Pattern Recognition - Week 5 - Model Selection in Bayesian Frameworks & Gaussian Processes

Antonio León Villares

October 2022

Contents

1 Bayesian Model Choice **2**

1.1 Bayesian Models and Overfitting . . . 2

1.2 Picking Models in Simple Cases . . . 4

1.3 Hyperparameter Selection in Linear Regression . . . 6

2 Gaussian Processes **8**

2.1 Defining Gaussian Processes: Gaussians to Represent Functions . . . 9

2.2 Joint Distributions of Gaussians . . . 11

2.3 Gaussian Processes for Regression . . . 11

2.4 Connection Between Gaussian Processes and Bayesian Linear Regression . . . 15

3 Question **16**

3.1 Notes Questions . . . 16

4 Tutorial **16**

Based on the online notes here.

1 Bayesian Model Choice

1.1 Bayesian Models and Overfitting

- **Can Bayesian methods overfit or underfit?**
  - with **Bayesian models**, we generate models by **integrating/summing** over parameters
  - at no point do we **fit** our model parameters to the data, so Bayesian methods can't overfit/underfit
  - however, problems still arise, based on the **complexity** of the model
- **When is a Bayesian framework too simple?**
  - consider fitting a Bayesian linear regression model on a **large** sample of data which isn't distributed **linearly** (i.e a parabola, oscillatory, etc...)
  - our model will predict a **linear** fit to the data, and have **high certainty** about it (i.e high posterior)
  - this is because the posterior is **proportional** to prior times likelihood
  - even if a model is a bad fit, it'll still be better than models which are **terrible** fits
  - hence, when we normalise to get the posterior, the bad fit models will still obtain a very high probability (simply because the terrible models get an infinitesimal probability)

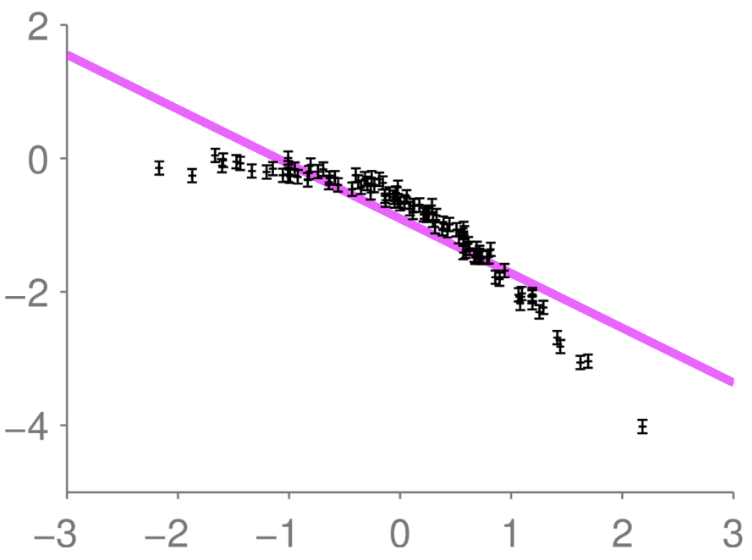


Figure 1: If a model is too simple, a linear model will have high posterior, even if the fit is bad.

- **How could simple models be detected?**
  - we can sample many times from the posterior
  - if we see that the **residuals** are **highly correlated** (i.e all are of the same size and about the same magnitude at the same points), this can be indicative of a **simple model**
  - this is because simple models with high precision will center about the mean function of the distribution, so we expect that the residuals behave similarly across most samples (in the image above, a dozen samples almost look like the same line)

- **When is a Bayesian framework too complicated?**
  - consider using 10^6 RBFs evenly spaced on the interval $[0, 1]$ - critically, the **bandwidth** will be very small
  - if we sample from the **prior**, we obtain a **random linear combination** of these RBFs, which due to the small bandwidth produces very squiggly plots:

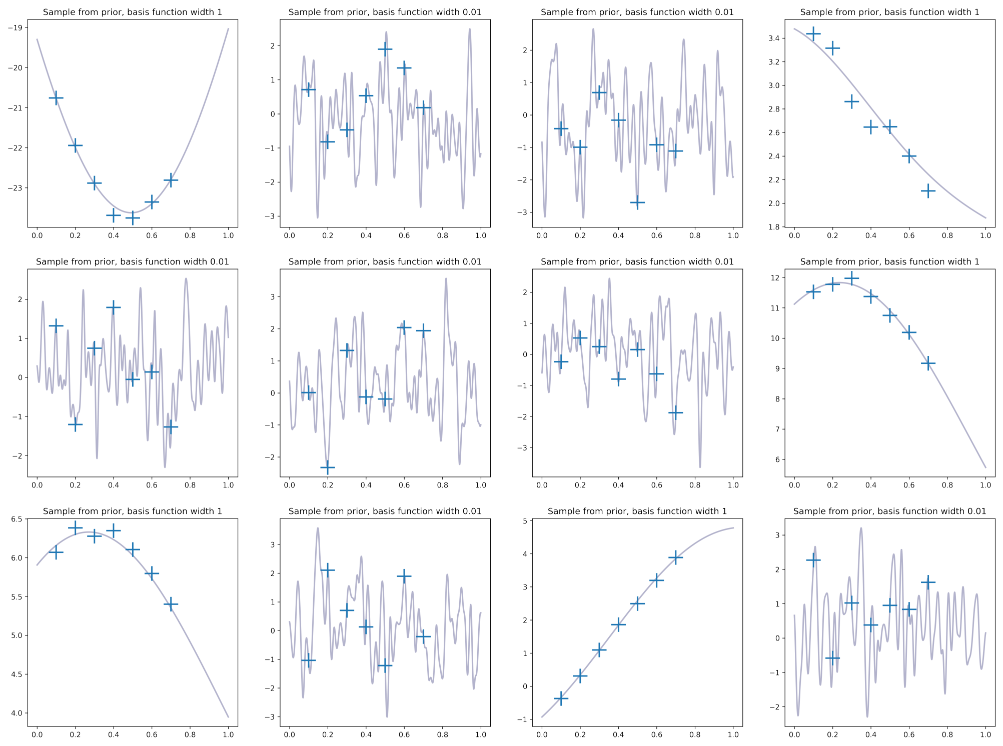


Figure 2: Here, the markers represent points from the distribution with some noise. The curves sampled from the prior are in grey. Since RBFs can approximate continuous functions, a low bandwidth ($h = 0.01$) leads to noisy priors; however, with $h = 1$, we get smoother curves.

  - now, if we want to fit a **posterior**, only a small proportion of the RBFs will actually be used to approximate the training data
  - this is because most of the RBFs will be very far away from most of the training points (as the bandwidth is tiny)
  - this means that globally, the posterior will be **very similar** to the prior, except possibly at the training points:

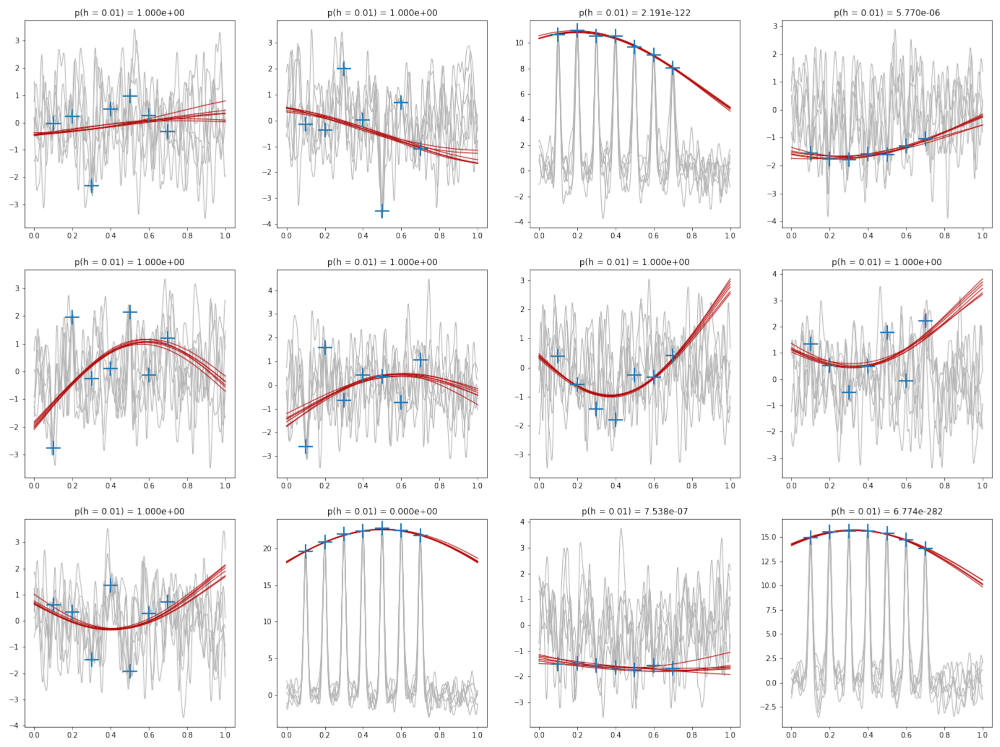


Figure 3: In blue we have the training data (sometimes generated using the large bandwidth, and other times using the small bandwidth). In grey we have the posteriors generated using the **small** bandwidth $h = 0.01$. In red we have posteriors generated using the **large** bandwidth $h = 1$. The small bandwidth posteriors look like the priors, except at the training points generated by the large bandwidth distribution; this is due to the fact that the RBFs barely overlap, so only certain RBFs get extremely large weights. On the other hand, the posteriors for large bandwidths seem to all be bunched up together (indicating high confidence), even though the fit isn't always good for the data generated with small bandwidths.

1.2 Picking Models in Simple Cases

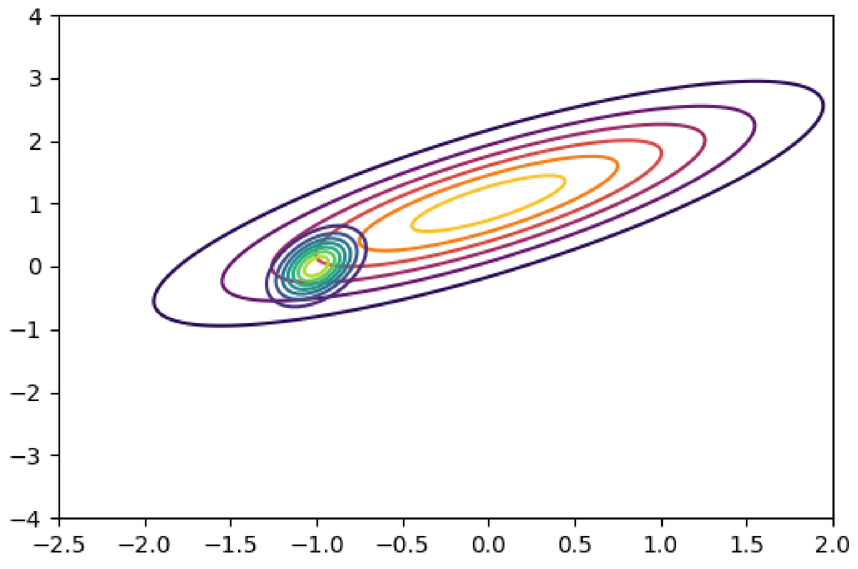
- **How does Naive Bayes select the class of an input?**
  - it learns a **distribution** of **features** given the **class**:

$$P(\underline{x} \mid y = k)$$

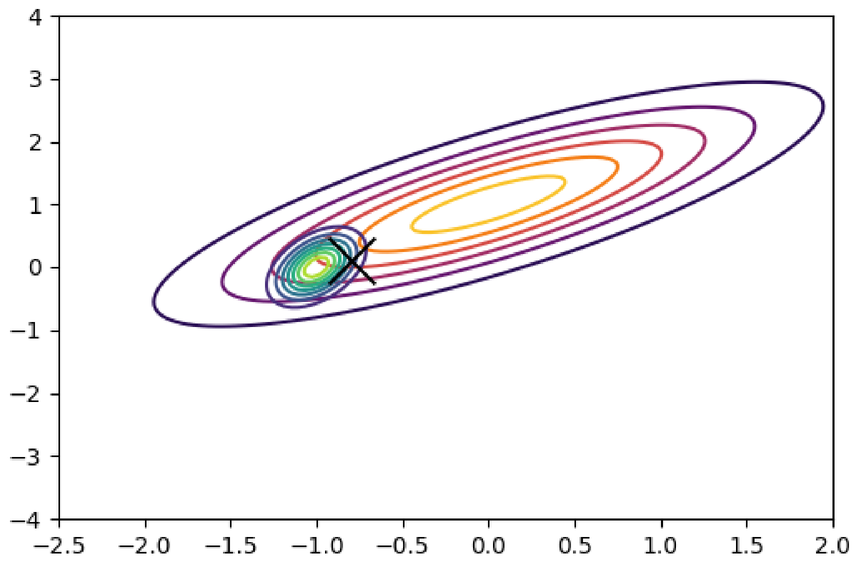
  - then it uses **Bayes' Rule** to select the **most likely class**:

$$P(y = k \mid \underline{x}) \propto P(y = k) \times P(\underline{x} \mid y = k)$$

- **How can we determine the most likely Gaussian which could've generated a datapoint?**
  - consider 2 Gaussians: one is broad (high variance), whilst the other one is narrow

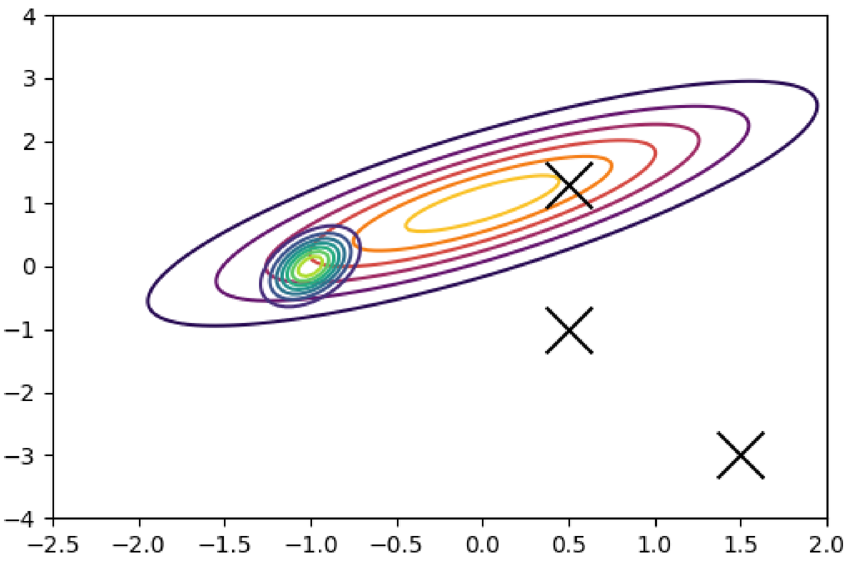


– if we make the following observation:



Then we will be more confident that the narrow Gaussian generated the point (the observation is close to its mean, and its variance is incredibly small)

– however, if we make the following observation:



Then we will be more confident that the broad Gaussian generated the points (since it has a higher variance, and its simply more unlikely that the narrow Gaussian could've generated the point)

- a simple way to see this is that if we have 2 die, one with 10 sides, and one with 10^6 sides, if we roll a 5 it will most likely have been generated by the D10, since small numbers are more easily generated by a D10
- **priors** are also important to consider (it is more likely for someone to have a D10 in the first place)

1.3 Hyperparameter Selection in Linear Regression

- **What is a model's marginal likelihood?**
  - to select a model, we can **condition** over the different **model hyperparameters**:

$$P(\underline{y} \mid X, \mathcal{M})$$

  - this is the model's **marginal likelihood**, which we can use to score the different models $\mathcal{M}$
  - if we condition on weights and use the **product rule**:

$$\begin{aligned} P(\underline{y} \mid X, \mathcal{M}) &= \int P(\underline{y}, \underline{w} \mid X, \mathcal{M}) d\underline{w} \\ &= \int P(\underline{y} \mid X, \underline{w}, \mathcal{M}) P(\underline{w} \mid \mathcal{M}) d\underline{w} \end{aligned}$$

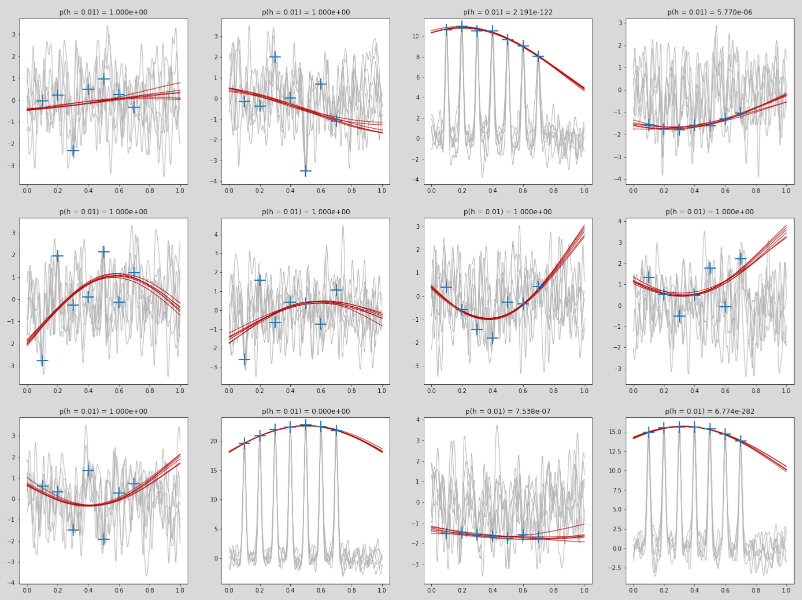
  - here $\underline{y}, X$ represent the actual data on which we train on
  - we can use **Bayes' Rule** to compute the **marginal likelihood** more easily:

$$P(\underline{w} \mid \underline{y}, X, \mathcal{M}) = \frac{P(\underline{y} \mid X, \underline{w}, \mathcal{M}) P(\underline{w} \mid \mathcal{M})}{P(\underline{y} \mid X, \mathcal{M})} \implies P(\underline{y} \mid X, \mathcal{M}) = \frac{P(\underline{y} \mid X, \underline{w}, \mathcal{M}) P(\underline{w} \mid \mathcal{M})}{P(\underline{w} \mid \underline{y}, X, \mathcal{M})}$$

where:

* $P(\underline{y} \mid X, \underline{w}, \mathcal{M})$ is the **likelihood** of our linear regression model
* $P(\underline{w} \mid \mathcal{M})$ is the **prior** distribution of our linear regression model
* $P(\underline{w} \mid \underline{y}, X, \mathcal{M})$ is the **posterior** distribution of our linear regression model

which are all known quantities (given $\mathcal{M}$) and is valid for **all** $\underline{w}$



*We can rephrase our analysis of the above in terms of **marginal likelihoods**.*
*When the data is generated by the **small bandwidth** distribution:*

$$P(\underline{y} \mid X, \underline{w}, h = 0.01) <<< P(\underline{y} \mid X, \underline{w}, h = 1)$$

since the posteriors with $h = 1$ osciallate wildly.
*On the other hand, if we use data generated by the **large bandwidth** distribution:*

$$P(\underline{y} \mid X, \underline{w}, h = 1) <<< P(\underline{y} \mid X, \underline{w}, h = 0.01)$$

since the small bandwidth can't adapt to the large oscillations.
*In other words, just with the **marginal likelihood** we can select the best model.*

* **Why should we optimise over the marginal likelihood and not the standard likelihood?**
  - **marginal likelihood** accounts for all possible settings of $\underline{w}$

- however, the **standard likelihood** is defined for a **specific** weight:

$$P(\underline{y} \mid X, \underline{w}, \mathcal{M})$$

- we can always find a $\underline{w}$ which fits to the data perfectly (or nearly so), thus maximising standard likelihood
- however, this would be akin to choosing hyperparmeters based on training performance
- **moreover**, **marginal likelihood** is computed by just using the **training data -** no need for a **validation set**

- **What type of hyperparameters can be included in $\mathcal{M}$?**
  - this can be all the things we assumed as given:
    * σ_y^2 (error for outputs y)
    * $V_0 = \sigma_w^2$ (covariance for prior)
    * $\underline{w}_0$ (mean for prior)
    * Φ (basis functions) (i.e the bandwidth for RBFs)
  - all these hyperparameters ultimately define our model

- **Can we choose to optimise over only a few hyperparameters?**
  - applying **cross-validation** is unfeasible for selecting hyperparameters (there are a lot of them, and they are often continuous)
  - instead we can choose to optimise the **marginal likelihood** over a few of the hyperparameters

*For example, say we want to optimise for σ_w, σ_y, which define the **prior**:*

$$P(\underline{w} \mid \sigma_w) = \mathcal{N}(\underline{w}; \underline{0}, \sigma_w^2 \mathbb{I})$$

*and the (standard) **likelihood**:*

$$P(y \mid \underline{x}, \underline{w}, \sigma_y) = \mathcal{N}(y; \underline{w}^T \underline{x}, \sigma_y^2)$$

*We could maximise the **integral** form of **marginal likelihood**, but it's simpler if we use the **Bayesian Expression**:*

$$P(y \mid \underline{x}, \sigma_w, \sigma_y) = \frac{P(y \mid \underline{x}, \underline{w}, \sigma_y) P(\underline{w} \mid \sigma_w)}{P(\underline{w} \mid \underline{y}, X, \sigma_w, \sigma_y)} = \frac{\mathcal{N}(\underline{w}; \underline{0}, \sigma_w^2 \mathbb{I}) \mathcal{N}(y; \underline{w}^T \underline{x}, \sigma_y^2)}{\mathcal{N}(\underline{w}; \underline{w}_N, V_N)}$$

where:

$$V_n = \sigma_y^2 (\sigma_y^2 V_0^{-1} + \Phi^T \Phi)^{-1}$$

$$\underline{w}_N = V_N V_0^{-1} \underline{w}_0 + \frac{1}{\sigma_y^2} V_n \Phi^T \underline{y}$$

and we use $\sigma_w^2 \mathbb{I} = V_0$ and $\underline{w}_0 = \underline{0}$.

2 Gaussian Processes

- This is a great, 3 part blog post on Gaussian Processes (including code)
- This videos gives a great overview of GPs, with great visualisations.

2.1 Defining Gaussian Processes: Gaussians to Represent Functions

- **Why can we think of functions as infinite-dimensional vectors?**
  - a **function** is defined by the values it takes on its **domain**
  - these values can be stored in a **vector**, but it must have infinite dimensions
- **How can we specify a function with finite dimensions?**
  - we can consider the values of f on a **finite** subdomain
  - for example, for values x which can be represented as **floating point numbers** in memory (due to precision errors, and memory restrictions, there are finitely many of these)
  - then, we could define a vector:

$$\tilde{f} = \begin{pmatrix} f(\underline{x}_1) \\ f(\underline{x}_2) \\ \vdots \\ f(\underline{x}_n) \end{pmatrix} = \begin{pmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \vdots \\ \tilde{f}_n \end{pmatrix} \in \mathbb{R}^n$$

  - we can think of $\underline{\tilde{f}}$ as representing the whole function f (up to an arbitrary finite discretisation)
- **How can Gaussians be used to represent a function?**
  - since we can represent a **function** as a **finite vector**, we can think of this vector as being **sampled** from a **multivariate gaussian**
  - for this we need to define a **mean vector** $\underline{\mu}$ and a **covariance matrix** Σ
- **Why do we use $\underline{\mu} = \underline{0}$ to represent a function?**
  - if we generate random functions, at a given point x_i, they will evaluate to something **positive** as often as they'll evaluate to something **negative**
  - setting $\underline{\mu} = \underline{0}$ encompasses this intuition
- **Why shouldn't we set the covariance matrix to a diagonal matrix?**
  - function values are typically **correlated**
  - if a function is **continuous**, shifting $\underline{x}$ to $\underline{x} + \underline{\delta}$ slightly should shift $f(\underline{x})$ slightly to $f(\underline{x}) + \varepsilon \approx f(\underline{x} + \underline{\delta})$
  - if Σ were diagonal, then we'd be assuming that all function values are independent, but intuitively nearby values should be **similar**
- **What is a kernel function?**
  - a "function" which we can use to define our covariance matrix Σ:

$$\Sigma_{ij} = k(\underline{x}_i, \underline{x}_j)$$

  - there are families of **positive definite kernel functions** (Mercer Kernels), which produce **positive definite matrices** if they are used to define the entries of a matrix

- for example a **Gaussian Kernel**:

$$k(\underline{x}_i, \underline{x}_j) = \exp(-\|\underline{x}_i - \underline{x}_j\|^2)$$

  or:

$$k(\underline{x}_i, \underline{x}_j) = (1 + \|\underline{x}_i + \underline{x}_j\|) \exp(-\|\underline{x}_i - \underline{x}_j\|)$$

  or the **Exponentiated Quadratic Kernel**:

$$k(\underline{x}_i, \underline{x}_j) = \sigma^2 \exp\left(-\frac{\|\underline{x}_i - \underline{x}_j\|^2}{2\ell^2}\right)$$

- **What is a Gaussian Process?**
  - a **multivariate distribution** for defining a function f
  - if $\underline{f}$ is a **vector** of the function values, then:

$$P(\underline{f}) = \mathcal{N}(\underline{f}; \underline{0}, K)$$

    where K is a **covariance matrix** defined by:

$$K_{ij} = k(\underline{x}_i, \underline{x}_j)$$

    and:

$$f_i = f(\underline{x}_i)$$

  - we sample f from a **Gaussian Process** via:

$$f \sim \mathcal{GP}$$

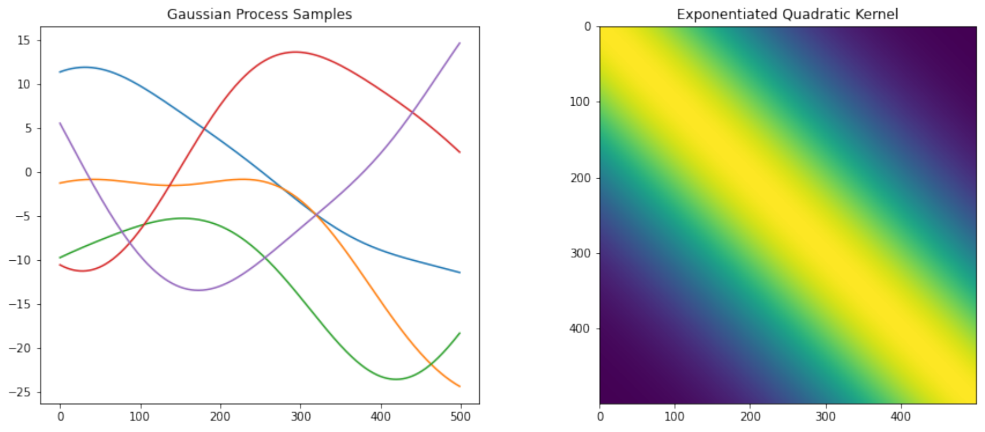


Figure 4: 5 samples from a Gaussian process, generated by using discrete samples on the domain $[0, 10]$, with increments of 0.02. The covariance matrix (shown above) was computed by using the **exponentiated quadratic kernel**.

- **How does the kernel affect the shape of the Gaussian Process sample?**
  - depending on the kernel, we prioritise certain types of functions
  - for example, exponential kernels will lead to smooth, continuous functions
  - other kernels might generate straight lines, or periodic lines, etc ...
  - examples of these can be seen in the link at the beginnning of the section

2.2 Joint Distributions of Gaussians

Before applying Gaussian Processes to Bayesian Linear Regression, we first consider how joint Gaussian distributions behave.

Consider a joint Gaussian Distribution:

$$P(\underline{f}, \underline{g}) = \mathcal{N}\left(\begin{pmatrix}\underline{f} \\ \underline{g}\end{pmatrix}; \begin{pmatrix}\underline{a} \\ \underline{b}\end{pmatrix}, \begin{pmatrix} A & C \\ C^T & B \end{pmatrix}\right)$$

*Then, the **marginal probability** gives:*

$$P(\underline{f}) = \int P(\underline{f}, \underline{g}) d\underline{g}$$

*such that $P(\underline{f})$ is a **Gaussian**:*

$$P(\underline{f}) = \mathcal{N}(\underline{f}; \underline{a}, C)$$

*Moreover, the **conditional distribution** will also be Gaussian:*

$$P(\underline{f} \mid \underline{g}) = \mathcal{N}(\underline{f}; \underline{a} + CB^{-1}(\underline{g} - \underline{b}), A - CB^{-1}C^T)$$

2.3 Gaussian Processes for Regression

- **How are Gaussian Processes used in Bayesian Linear Regression?**
  - say we have a set of **training** observations $\underline{y}$ generated at some training locations:
    $$X = \{\underline{x}_i\}$$
  - we assume that $\underline{y}$ is actually the graph of a function with noise added at each point:
    $$y_i \sim \mathcal{N}(f_i, \sigma_y^2)$$
    where σ_y^2 is some known (and fixed) noise
  - we assume that $f_i = f(\underline{x}_i)$, where f is actually **sampled** from a **Gaussian Process**:
    $$f \sim \mathcal{GP} \qquad P(\underline{f}) = \mathcal{N}(\underline{f}; \underline{0}, K)$$
    where $\underline{f}$ represents the values of f at the training locations X
  - $P(\underline{f})$ represents our **prior belief**: that the data is generated by a gaussian process
- **How can we predict new values using a GP prior?**
  - say now that we have a bunch of **test locations** whose value we want to predict given f:
    $$X_* = \{\underline{x}_i^*\}$$
  - let $\underline{f}_*$ denote the **vector** of values taken by f on X_*

- we can consider a **joint distribution** for our **observations** $\underline{y}$, and what should be predicted by the model at X_*, $\underline{f}_*$:

$$P\left(\begin{pmatrix}\underline{y}\\ \underline{f}_*\end{pmatrix}\right) = \mathcal{N}\left(\begin{pmatrix}\underline{y}\\ \underline{f}_*\end{pmatrix}; \underline{0}, \begin{pmatrix}K(X,X)+\sigma_y^2\mathbb{I} & K(X,X_*)\\ K(X_*,X) & K(X_*,X_*)\end{pmatrix}\right)$$

  where we abuse notation by setting:

$$K(X,Y)_{ij} = k(\underline{x}_i, \underline{y}_j)$$

  (where $\underline{x}_i, \underline{y}_i$ are row vectors of X, Y respectively, such that if $X \in \mathbb{R}^{N\times D}$ and $Y \in \mathbb{R}^{M\times D}$, then $K(X,Y) \in \mathbb{R}^{N\times M}$)
- using the rule for **conditional Gaussians** given a joint distribution, we get that the parameters of $P(\underline{f}_* \mid \underline{y})$ (the **posterior over function values** $\underline{y}$) are:

$$\underline{\mu}_* = K(X_*,X)(K(X,X)+\sigma_y^2\mathbb{I})^{-1}\underline{y}$$

$$\Sigma_* = K(X_*,X_*) - K(X_*,X)(K(X;X)+\sigma_y^2\mathbb{I})^{-1}K(X,X_*)$$

- if we sample from $P(\underline{f}_* \mid \underline{y})$ we get vectors for **discretised predicted values** of f at the testing points X_*

*The **covariances** for test values $\underline{f}_*$ and function values $\underline{y}$ are relatively straightforward to understand.*
However, why should the cross-covariances for X, X_* be given by $K(X,X_*)$? We compute:

$$\begin{aligned} cov(y_i, f_j^*) &= \mathbb{E}[y_i f_j^*] - \mathbb{E}[y_i]\mathbb{E}[f_j^*] \\ &= \mathbb{E}[(f_i+\nu)f_j^*], \qquad \textit{(since } \mathbb{E}[y_i]=0 \textit{ and where } \nu \sim \mathcal{N}(0,\sigma_y^2)\textit{)} \\ &= \mathbb{E}[f_i f_j^*] + \mathbb{E}[\nu]\mathbb{E}[f_j^*] \\ &= \mathbb{E}[f_i f_j^*] \\ &= \mathbb{E}[f_i f_j^*] - \mathbb{E}[f_i]\mathbb{E}[f_j^*] \\ &= cov(f_i, f_j^*) \\ &= k(x_i, x_j^*) \end{aligned}$$

- **How can we visualise the posterior of our GP linear regression?**
  - we can generate a grid of **test points**
  - then, we can compute the **posterior** at each **test point**:

$$P(\underline{f}_* \mid \underline{y})$$

  - this gives us a **mean function** $\underline{\mu}_*$ and a **covariance**, which tells us the **variance** (uncertainty) at **each** test point

– we can plot the mean function, with the variance acting as error bounds:

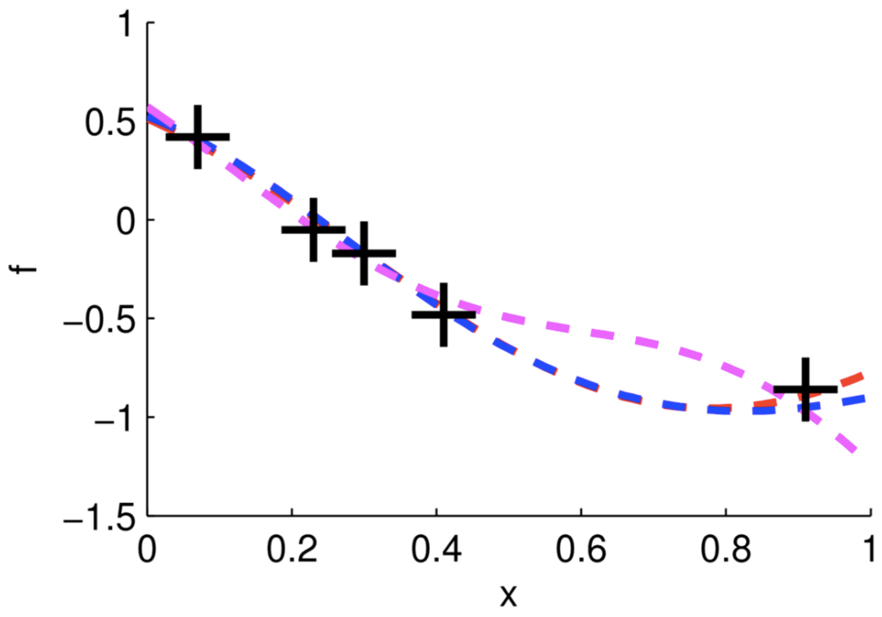


Figure 5: Given 4 training data points, we sample 3 times from the posterior at 100 test locations.

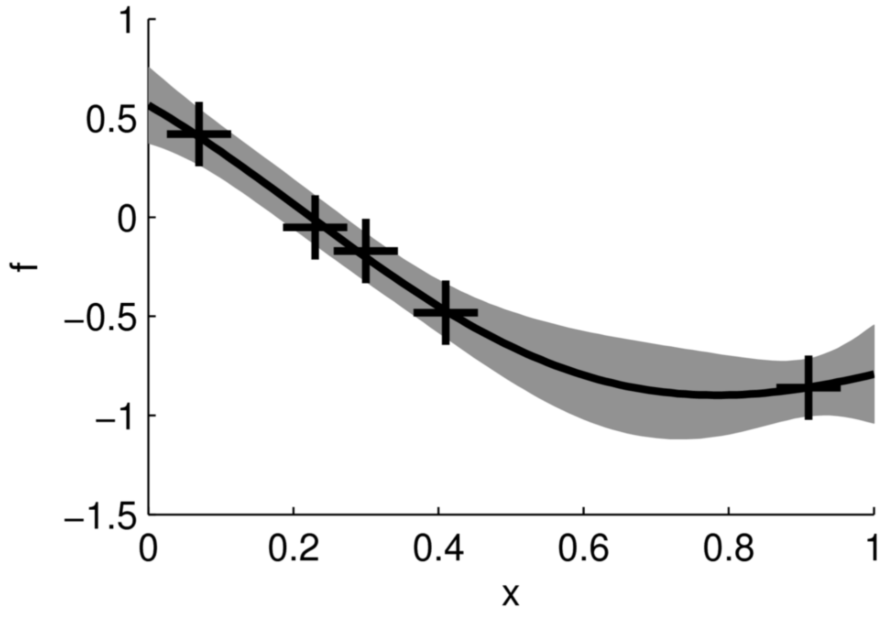


Figure 6: It is more informative if instead we plot the **mean** curve, alongside the variance at each test location. This provides us with "bounds" on the possible functions which fit the data, without necessarily explaining how the functions might look. In this case, the error bounds are twice the standard deviation at each test point.

- **How do the predicted curves improve given that we observe new data?**

- let M be the covariance matrix at the **training points**:

$$M = K + \sigma_y^2$$

where K is the matrix obtained by using a kernel on the training points:

$$K_{ij} = k(\underline{x}_i, \underline{x}_j)$$

- we want to compute the posterior of a **single** test location, given the observed data:

$$P(f(\underline{x}_*) \mid data) = \mathcal{N}(\tilde{f}; m, s^2)$$

- if we define a "covariance vector" for our test point $\underline{x}_*$:

$$(\underline{k}_*)_i = k(\underline{x}_*, \underline{x}_i)$$

(that is, the entries are obtained by applying the kernel to the test point, alongside each training point)

- then, we get that the parameters of the **posterior** at the test point $\underline{x}_*$ are given by:

$$\mu_* = \underline{k}_*^T M^{-1} \underline{y}$$

$$\sigma_*^2 = k(\underline{x}_*, \underline{x}_*) - \underline{k}_*^T M^{-1} \underline{k}_*$$

(these are just the formulae above, but applied at a single point)

- notice:
  1. The **mean prediction** is just a **linear combination** of $\underline{y}$, our observations
  2. The **variance** decreases the more data that we observe, since M is **positive definite**, so:

  $$\underline{k}_*^T M^{-1} \underline{k}_* > 0$$

  Moreover, our uncertainty decreases (from the value at the prior) by an amount which is **independent** of the observed values $\underline{y}$. Thus, we will know how much more certain we will be, even before updating our model.

- we can adjust our **uncertainty** by modifying the **kernel**, alongside our **noise model** σ_y^2

2.4 Connection Between Gaussian Processes and Bayesian Linear Regression

In ***Bayesian Linear Regression****, at test points $\underline{x}_i$, we made the prediction:*

$$\tilde{f}_i = \underline{w}^T \underline{x}_i + b$$

where we had a ***prior*** *distribution on model parameters:*

$$\underline{w} = \mathcal{N}(\underline{0}, \sigma_w \mathbb{I}) \qquad b \sim \mathcal{N}(0, \sigma_b^2)$$

From linearity of the expected value, we can see that the ***mean*** *of $\tilde{f}_i$ will be 0.*
But what about the covariance between predictions $\tilde{f}_i, \tilde{f}_j$? We compute:

$$\begin{aligned}
Cov(\tilde{f}_i, \tilde{f}_j) &= \mathbb{E}[\tilde{f}_i \tilde{f}_j] - \mathbb{E}[\tilde{f}_i]\mathbb{E}[\tilde{f}_j] \\
&= \mathbb{E}[(\underline{w}^T \underline{x}_i + b)(\underline{w}^T \underline{x}_j + b)] - 0 \\
&= \mathbb{E}[(\underline{w}^T \underline{x}_i)(\underline{w}^T \underline{x}_j) + b(\underline{w}^T \underline{x}_i + \underline{w}^T \underline{x}_j) + b^2] \\
&= \mathbb{E}[(\underline{x}_i^T \underline{w})(\underline{w}^T \underline{x}_j) + b(\underline{w}^T \underline{x}_i + \underline{w}^T \underline{x}_j) + b^2] \\
&= \underline{x}_i^T \mathbb{E}[\underline{ww}^T]\underline{x}_j + \mathbb{E}[b(\underline{w}^T \underline{x}_i + \underline{w}^T \underline{x}_j)] + \mathbb{E}[b^2]
\end{aligned}$$

Now, since b is independent of $\underline{w}$, we can split the expectation:

$$\mathbb{E}[b(\underline{w}^T \underline{x}_i + \underline{w}^T \underline{x}_j)] = \mathbb{E}[b]\mathbb{E}[\underline{w}^T \underline{x}_i + \underline{w}^T \underline{x}_j] = 0$$

since $\mathbb{E}[b] = 0$.
We also recognise:

$$\mathbb{E}[\underline{ww}^T] = \sigma_w \mathbb{I}$$

from definition of the covariance matrix.
Lastly:

$$\mathbb{E}[b^2] = \mathbb{E}[b^2] - (\mathbb{E}[b])^2 = Var[b] = \sigma_b^2$$

Hence, we get:

$$Cov(\tilde{f}_i, \tilde{f}_j) = \underline{x}_i^T (\sigma_w \mathbb{I}) \underline{x}_j + \sigma_b^2$$

Thus, we can define a "linear regression" kernel:

$$k(\underline{x}_i, \underline{x}_j) = \underline{x}_i^T (\sigma_w \mathbb{I}) \underline{x}_j + \sigma_b^2$$

(we can also use basis functions $\phi(\underline{x}_i)$ in this expression).
Hence, ***Bayesian Linear Regression*** *is nothing but a* ***Gaussian Process****.*
In practice, we wouldn't use GPs for Bayesian Linear Regression, since inference requires matrix inversion, which is expensive if we have a lot of data points. However if there are more input features or basis functions than data-points, using GPs might be cheaper.

3 Question

3.1 Notes Questions

1. **We have that the marginal likelihood is:**

$$P(\underline{y} \mid X, \mathcal{M}) = \int P(\underline{y} \mid X, \underline{w}, \mathcal{M}) P(\underline{w} \mid \mathcal{M}) d\underline{w}$$

$\underline{w}$ doesn't appear in $P(\underline{y} \mid X, \mathcal{M})$ because we integrate them out. However, $\mathcal{M}$ might include hyperparameters which define the distribution of $\underline{w}$ (i.e its prior mean, $\underline{w}_0$). Is the marginal likelihood affected by this prior mean for $\underline{w}$, even if $\underline{w}$ doesn't appear explicitly?

Yes. This is because we express the marginal likelihood as an integral over a weighted product of the (standard) likelihood $P(\underline{y} \mid X, \underline{w}, \mathcal{M})$. Hence, if the prior $P(\underline{w} \mid \mathcal{M})$ is high (as will be the case when $\underline{w}$ is similar to the prior mean), then the standard likelihood will be "upweighted", so the marginal likelihood will be pushed towards these standard likelihoods.

4 Tutorial

1. **Say we have 2 possible models M_1, M_2 for some features, and we define:**

$$a_1 = \log P(\underline{x} \mid M_1) + \log P(M_1)$$
$$a_2 = \log P(\underline{x} \mid M_2) + \log P(M_2)$$

(a) **Show that:**

$$P(M_1 \mid \underline{x}) = \sigma(a_1 - a_2) = \frac{1}{1 + \exp(-(a_1 - a_2))}$$

By Bayes' Rule:

$$P(M_1 \mid \underline{x}) = \frac{P(\underline{x} \mid M_1) P(M_1)}{P(\underline{x})}$$
$$P(M_2 \mid \underline{x}) = \frac{P(\underline{x} \mid M_2) P(M_2)}{P(\underline{x})}$$

Since M_1, M_2 are the only possible models:

$$P(M_1 \mid \underline{x}) + P(M_2 \mid \underline{x}) = \frac{P(\underline{x} \mid M_1) P(M_1) + P(\underline{x} \mid M_2) P(M_2)}{P(\underline{x})} = 1$$

so:

$$P(\underline{x}) = P(\underline{x} \mid M_1) P(M_1) + P(\underline{x} \mid M_2) P(M_2)$$

Hence:

$$P(M_1 \mid x) = \frac{e^{a_1}}{e^{a_1} + e^{a_2}} = \frac{1}{1 + e^{a_2 - a_1}} = \sigma(a_1 - a_2)$$

as required.

(b) **Now, consider K models, and define:**

$$a_k = \log[P(\underline{x} \mid M_k) P(M_k)]$$

i. **Show that:**

$$\log P(M_k \mid \underline{x}) = a_k - \log \sum_{\ell} \exp(a_\ell)$$

We generalise the work above, which tells us that:

$$P(M_k \mid \underline{x}) = \frac{P(\underline{x} \mid M_k)P(\underline{x})}{\sum_\ell P(\underline{x} \mid M_\ell)P(\underline{x})}$$

Hence:

$$\log P(M_k \mid \underline{x}) = a_k - \log \sum_\ell \exp(a_\ell)$$

as required

ii. **Show that:**

$$\log \sum_k \exp(a_k) = \max_k \, a_k + \log \sum_k \exp\left(a_k - \max_\ell \, a_\ell\right)$$

Explain why the expression is often implemented this way.

$$\begin{aligned}\log \sum_k \exp(a_k) &= \log \sum_k \exp(a_k - \max_\ell \, a_\ell + \max_\ell \, a_\ell) \\ &= \log \exp\left(\max_\ell \, a_\ell\right) \sum_k \exp(a_k - \max_\ell \, a_\ell) \\ &= \max_k \, a_k + \log \sum_k \exp(a_k - \max_\ell \, a_\ell)\end{aligned}$$

If the a_k are extremely small (i.e $a_k < -1000$), the term $\exp(a_k)$ will underflow to 0, so taking the log will result in $-\infty$). By subtracting the maximum, we will have the the largest term in the sum will be $e^0 = 1$. If any term underflows (which happens if a_k is extremely small compared to the maximum), then they would've had no practical effect when added to a value which is ≥ 1.

2. **Consider a dataset $\{\underline{x}^{(n)}, y^{(n)}\}_{n=1}^N$, describing N preparations of cells from some lab experiments. $y^{(n)}$ denotes the fraction of cells that are alive in preparation n. The first input feature of each preparation indicates whether the cells were created in lab A, B, C. The remaining features are real numbers describing experimental conditions, such as temperatures and concentrations of chemicals/nutrients.**

   (a) **Describe how you might represent the first input feature and the output, when learning a regression model to predict the fraction of alive cells in future preparations from the labs. Explain your reasoning.**

   - the input features are categorical variables, so I would use a one-hot-encoding:

     $$A \to (1,0,0) \quad B \to (0,1,0) \quad C \to (0,0,1)$$

     It wouldn't be sensible to do a direct numerical assignment (i.e $B \to 2$) since this isn't interpretable (i.e output from lab B isn't the mean of the outputs from labs A and C)
   - the output should be proportion in the range $[0,1]$
   - however linear regression models will output values in $\mathbb{R}$ (unless we cap the domain); hence, it might be smart to use a **logit transform** to map inputs from $[0,1]$ to $\mathbb{R}$:

     $$logit(p) = \log \frac{p}{1-p}$$

   - however, this might cause a problem if $p = 0$ or $p = 1$; in such a case, we could use an alternative transform:

     $$logit(0.01 + 0.98p)$$

- alternatively, we could fit a classifier to predict whether a preparation will be all dead ($p = 0$) or all alive ($p = 1$); based on the classifier, we could use the logit transform or not

(b) **Compare using the lab identity as an input to your regression with 2 baseline approaches:**

1. **Ignore the lab feature, and treat the data from all labs as if it came from a single lab**
2. **Split the dataset into 3 parts, one for each lab, and train 3 separate regression models**

To do the comparisons, compare standard linear regression, and gaussian process regression. Is it possible for these models (when using the lab identity) to emulate either or both of the 2 baselines?

- **Linear Regression**:
  - for baseline 1, since we ignore the lab, we fit less parameters, so we are less likely to overfit given little data (but we lose information about the idiosyncrasies of each lab)
  - for baseline 2, we have 3 times as many parameters, so we might potentially learn 3 completely different linear models for each lab
  - the one-hot encoded model will be able to fit the same model as the first baseline (just set the lab weights to 0)
  - however, it won't fit to the second baseline: including the one-hot features will only change the bias, but the gradient of the model will be the same; however, the second baseline can potentially learn 3 different gradients
  - in particular, this means that if the conditions of the labs have a significant effect on the fraction of alive cells, we model with one-hot features might underfit (since it learns the same gradient for the 3 labs)
- **Gaussian Process Regression**:
  - the GP regressor could potentially fit to both baselines
  - if we set long lengthscales for the kernel function, the effect of the lab would be negligible (the covariance will be close to 1, so all the features will be seen as correlated), and so, we would get results like baseline 1
  - if we set short lengthscales, the kernel will be near 0, so the feuatres are seen as independent, and thus, we could fit 3 different models, getting results like baseline 2
  - with intermediate lengthscales, we acknowledge that different labs can have different results, but we'd hope that these results are still somewhat related

(c) **There's a debate in the lab about how to represent the other input features: log-temperature or temperature, temperature in Fahrenheit, Celsius or Kelvin? Also, whether to use log concentration, or concentration as inputs to the regression. Discuss ways in which these issues could be resolved. If there is a debate about the output, how could this debate be resolved?**

- we can fit multiple models, one for each set of feature transformations
- we can compare the effect of transformations by using a validation set or marginal likelihood (if we use GPs)
- alternatively, we could create a model in which we include each possible feature transformation (i.e include Fahrenheit, Celsius and Kelvin as 3 distinct features), and hope that regularisation sets sets the weights of redundant features to near 0
- for output transformation, things are a bit more complicated. If we squash outputs into a narrow range of values, then the loss will trivially decrease, even if our model doesn't improve. Hence, we'd have to compare the model by first converting back to the original output, and then computing the loss.

3. **We have discussed that for Gaussian processes, the prior is sampled using 0 mean:**

$$\underline{f} \sim \mathcal{N}(\underline{0}, K)$$

If we knew in advanced the mean $\underline{m}$ of the outputs, we could in principle change this and sample:

$$\underline{f} \sim \mathcal{N}(\underline{m}, K)$$

but in practice, we just subtract the known mean $\underline{m}$ from the data, and once again fit a zero mean model.

(a) **If we don't know the mean $\underline{m}$, we could try sampling it from some prior, which could itself be a Gaussian process:**

$$\underline{m} \sim \mathcal{N}(\underline{0}, K_m)$$
$$\underline{f} \sim \mathcal{N}(\underline{m}, K_f)$$
$$\underline{y} \sim \mathcal{N}(\underline{f}, \sigma_n^2 \mathbb{I})$$

Show that the function values $\underline{f}$ still come from a function drawn from a zero-mean GP. Identify the resulting covariance function of the zero-mean process for $\underline{f}$.

- notice, since:

$$\underline{f} \sim \mathcal{N}(\underline{m}, K_f)$$

then we can write:

$$\underline{f} = \underline{m} + \underline{z}, \qquad \underline{z} \sim \mathcal{N}(\underline{0}, K_f)$$

so:

$$\mathbb{E}[\underline{f}] = \mathbb{E}[\underline{m}] + \mathbb{E}[\underline{z}] = 0$$
$$Cov[\underline{f}] = Cov[\underline{m}] + Cov[\underline{z}] = K_f + K_m$$

so that:

$$\underline{f} \sim \mathcal{N}(\underline{0}, K_m + K_f)$$

(b) **Identify the mean's kernel function k_m for 2 types of mean:**

- **an unknown constant:**

$$m_i = b, \quad b \sim \mathcal{N}(0, \sigma_b^2)$$

- **an unknown linear trend:**

$$m_i = \underline{w}^T \underline{x}^{(i)} + b$$

where:

$$\underline{w} \sim \mathcal{N}(\underline{0}, \sigma_w^2 \mathbb{I}) \qquad b \sim \mathcal{N}(0, \sigma_b^2)$$

- if $\underline{m}$ is a vector with the value $b \sim \mathcal{N}(0, \sigma_b^2)$ repeated then:

$$k_m(\underline{x}^{(i)}, \underline{x}^{(j)}) = Cov[m_i, m_j] = \mathbb{E}[m_i m_j] - \mathbb{E}[m_i]\mathbb{E}[m_j] = \mathbb{E}[b^2] = Var[b] + (\mathbb{E}[b])^2 = \sigma_b^2$$

so:

$$(K_m)_{ij} = \sigma_b^2$$

Thus, if we want to add a random offest to a GP, we can add a constant to every element in the covariance matrix.

- if $\underline{m}$ is linear, we derived that the linear kernel is:

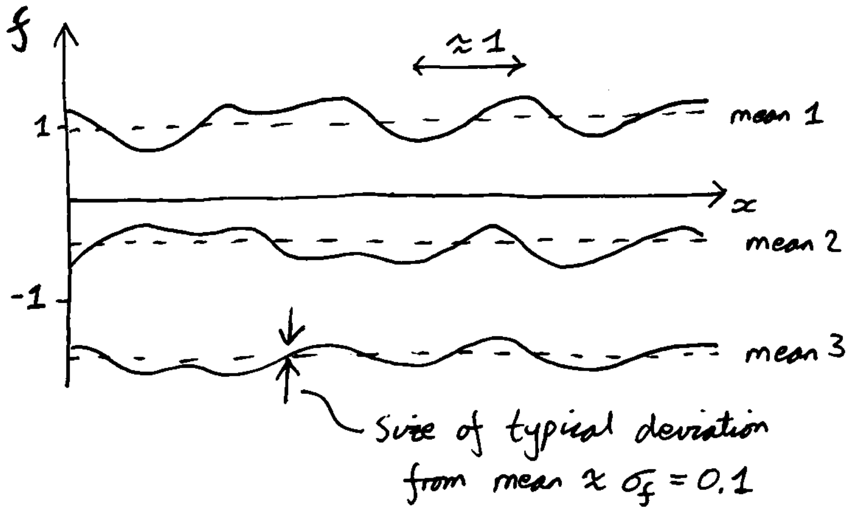
$$k(\underline{x}P^{(i)}, \underline{x}^{(j)}) = \sigma_w^2 \underline{x}^{(i)T} \underline{x}^{(j)} + \sigma_b^2$$

Hence, adding $\sigma_w^2 \underline{x}^{(i)T} \underline{x}^{(j)}$ to K_{ij} int eh covariance matrix allows us to model a random linear trend

(c) **Sketch 3 typical draws from a GP prior with kernel:**

$$k(x^{(i)}, x^{(j)}) = 0.1^2 \exp\left(-\frac{(x^{(i)} - x^{(j)})^2}{2}\right) + 1$$

- this is a standard Gaussian kernel with some noise:
  - we have a variance of $\sigma^2 = 0.1^2$ (we expect an amplitude of around 0.1^2 for the priors)
  - we have a lenghtscale of $\ell = 1$, so we expect turning points at every unit of the axis; the priors should look relatively squiggly, but still smooth
  - we have an added offset of 1



f
≈ 1
1
-1
x
mean 1
mean 2
mean 3
Size of typical deviation from mean ≈ σ_f = 0.1