Course 67564 2022/23

Bayesian Machine Learning
Recitation 4: Linear Regression

Prof. Yair Weiss *TA: Roy Friedman*

1 Problem Definition

Given a vector of features $x \in \mathbb{R}^d$, the simplest regression model is a function given by the weighted combination of the input features (sometimes called regressors, explanatory variables, covariates or some other name):

$$y = \theta_0 + \sum_i \theta_i x_i \tag{1.1}$$

where we want to predict the value of y (sometimes called the response), and the θs are the parameters of our model (sometimes called the regression coefficients). The problem of finding the θs that estimate y the best is known as *linear regression*[1]. The weight θ_0 is called the *bias term,* which allows the model to learn the intercept (so that y doesn't have to be 0 at $x = 0$). We can rewrite everything in vector form by defining $\boldsymbol{x} \triangleq [1, x_1, ..., x_d]^T$ and $\boldsymbol{\theta} \triangleq [\theta_0, \theta_1, ..., \theta_d]$:

$$y = \boldsymbol{x}^T \boldsymbol{\theta} \tag{1.2}$$

Of course, we usually want to predict many ys at the same time, not just one. Suppose we have N feature vectors $\boldsymbol{x}_i \in \mathbb{R}^d$ (here they include the intercept term at the beginning) and a single output vector $\boldsymbol{y} \in \mathbb{R}^N$. We can rewrite the above for all of the outputs at once by defining:

$$H \triangleq \begin{bmatrix} - & \boldsymbol{x}_1 & - \\ - & \boldsymbol{x}_2 & - \\ & \vdots & \\ - & \boldsymbol{x}_N & - \end{bmatrix} \tag{1.3}$$

Which lets us rewrite everything in the elegant form:

$$\boldsymbol{y} = H\boldsymbol{\theta} \tag{1.4}$$

(from now on we will stop writing vectors in bold, so that $\boldsymbol{y} \equiv y$ and $\boldsymbol{\theta} \equiv \theta$, for ease of notation, but remember that all the variables are vectors). The matrix H is sometimes called the *observation matrix* or the *design matrix.*

In the real world we usually encounter noise when sampling the function values y, which we will explicitly model by adding a noise term:

$$y = H\theta + \eta \tag{1.5}$$

The noise doesn't have to be Gaussian, but must always have zero mean. Anyway, we will model it as Gaussian for now (and this is the manner that the problem is usually presented):

$$\eta \sim \mathcal{N}\left(0, I\sigma^2\right) \tag{1.6}$$

with $\sigma^2 > 0$. We now see that the likelihood $y|\theta$ is an affine transformation of a Gaussian, which is also Gaussian:

$$y|\theta \sim \mathcal{N}\left(H\theta, I\sigma^2\right) \tag{1.7}$$

Figure 1 shows an example of the linear regression task.

[1]Bishop 3.1; Murphy 7.3

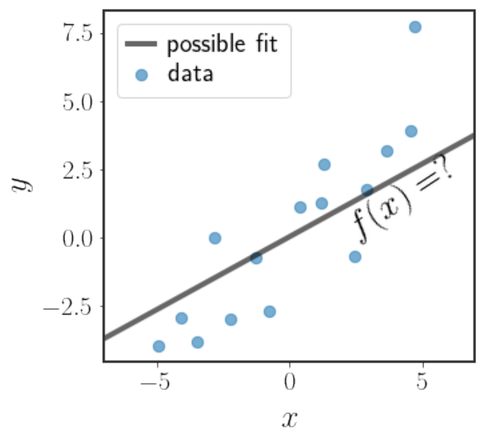

Figure 1: an example of the linear regression task. Training points are given and are assumed to originate from a linear function plus some noise, which we will attempt to find given the points.

2 Basis Functions

Notice that while the model is linear in the features x, we can always define new features that are a non-linear function of x, so that:

$$h \triangleq h(x) \tag{2.1}$$

where $h : \mathbb{R}^d \to \mathbb{R}^p$. The solution to the linear regression problem defined above will of course be linear in h, but **will not be linear in the original features** x. In this case, we would say that we have p *basis functions* such that:

$$h = h(x) = [h_1(x), h_2(x), ..., h_p(x)]^T \tag{2.2}$$

and each $h_i(\cdot)$ is called a *basis function* and has the form of $h_i : \mathbb{R}^d \to \mathbb{R}$.

Common basis functions

Polynomial basis functions (PBF) are the simplest kind of basis functions we will see in the course. In these basis functions, the features are powers of the input variables x up to a certain degree. For instance, if $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, then the features for a PBF of degree 2 are 1, x_1, x_2, x_1^2, x_1x_2, x_2^2. For degree 3, we would add x_1^3, $x_1^2x_2$, $x_1x_2^2$, x_2^3 and so on for any degree we want to use. These are called *polynomial* basis functions since the form $\theta^T h(x)$ for degree q describes a polynomial function of degree q (polynomial in x).
Another commonly used basis function is:

$$h_j(x) = \exp\left[-\frac{\|x-\mu_j\|^2}{2s^2}\right] \tag{2.3}$$

which is called, for obvious reasons, the *Gaussian basis function.* Note that while this basis function is related to the Gaussian distribution, it doesn't need to be a proper distribution. One way to use the Gaussian basis function is to decide, ahead of time, where K different centers μ_j will be placed, and use the distances from each of them as the features. This has the obvious limitations that *we* are the ones that chose where the centers should be placed, and so they may not be optimal for the data. An example use case of such basis functions is given in figure 2; notice how the learned function has hills and valleys corresponding to the locations of the basis functions.
Of course, there are many more possible basis functions that can be used. Since there are many possible basis functions and some may work better than others on our data, we may run into issues of how to choose the best model. Later on we will discuss how to select which basis function to use from the myriad possibilities, but for now we will ignore this problem, since the optimization process for linear regression doesn't depend on the specific basis function that is used.

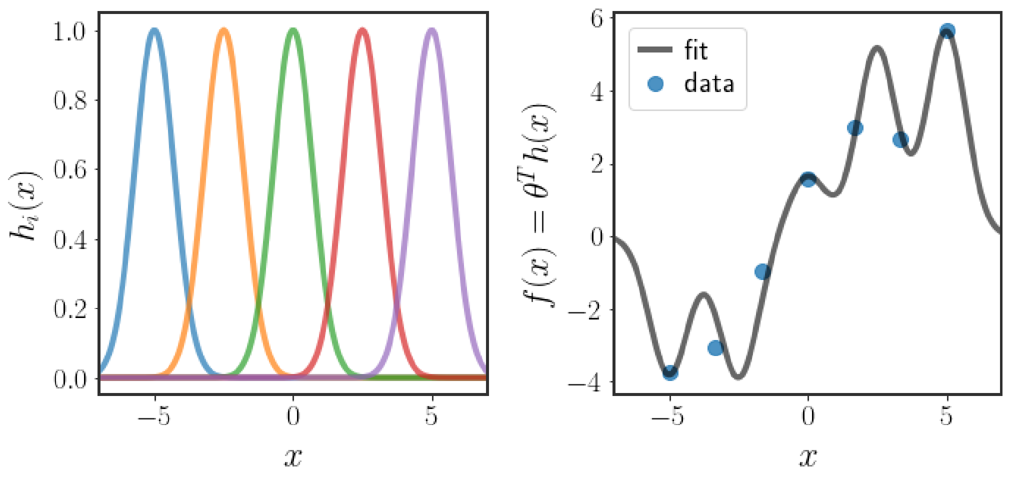

Figure 2: an example for the use of Gaussian basis functions and how they can be used in order to fit some data. On the left are the basis functions themselves, all of which are Gaussian basis functions with different centers, while on the right is the ML fit of some data points with these basis functions.

3 Classical Linear Regression

In the classical formulation of linear regression, the ML estimate for θ is used to define the function $y(x)$. Let's find this estimate. First, let's write out the log-likelihood explicitly:

$$\ell(y\,|\,\theta) = \sum_{i=1}^{N} \log \mathcal{N}\left(y_i\,|\,h(x_i)^T\theta, I\sigma^2\right)$$

$$= -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\|y_i - h(x_i)^T\theta\|^2 + \text{const} \tag{3.1}$$

$$= -\frac{1}{2\sigma^2}\|y - H\theta\|^2 + \text{const} \tag{3.2}$$

where we will ignore terms that are constant with respect to θ for now. Finding the maximum of this log-likelihood is often called *least squares* as we are trying to minimize a sum of square functions over θ; the function we are trying to maximize is the negative of the loss:

$$L = \frac{1}{2}\|y - H\theta\|^2 \tag{3.3}$$

We can differentiate the log-likelihood by θ and equate to 0 to find the minimum (the function is quadratic, so the only stationary point is the minimum):

$$\frac{\partial}{\partial\theta}L = H^T(H\theta - y) \overset{!}{=} 0 \tag{3.4}$$

$$H^T H\theta = H^T y \tag{3.5}$$

$$\Rightarrow \hat{\theta}_{ML} = \left(H^T H\right)^{-1} H^T y \tag{3.6}$$

An example of fitting points using the ML solution and polynomial basis functions can be seen in figure 4.

3.1 Geometry of Least Squares

We can gain further insight into the ML solution by looking at the geometry of the least squares solution[2].

[2]The analysis that follows is a more fleshed out version of the same given in Bishop 3.1.2

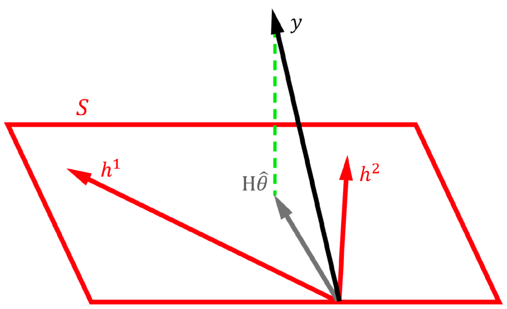

Figure 3: Geometrical interpretation of the least squares solution

Let's look at the N dimensional vector $y = (y_1, y_2, ..., y_N)^T$ and the columns of H, which we will denote as $h^{(j)}$ for now. Some subspace $\mathcal{S}$ is spanned by the d column vectors in H, which leaves us with three possible scenarios that we have to think about when talking about the ML solution $\hat{\theta}_{\text{ML}} = \left(H^T H\right)^{-1} H^T y$:

1. There exists some θ such that $H\theta = y$ exactly
2. y is in the orthogonal complement of $\mathcal{S}$, i.e. $H^T y = 0$
3. y is a combination of the above two scenarios, in which case we can rewrite it as a part that lies in the subspace plus an orthogonal part: $y = y_{\mathcal{S}} + y_{\perp} \triangleq H\theta_{\mathcal{S}} + y_{\perp}$

Plugging this into the least squares solution, the prediction $\hat{y}$ that we will make is:

$$\begin{aligned}\hat{y} &= H\hat{\theta}_{\text{ML}} = H\left(H^T H\right)^{-1} H^T y \\ &= H\left(H^T H\right)^{-1} H^T \left(H\theta_{\mathcal{S}} + y_{\perp}\right) \\ &= H\left(H^T H\right)^{-1} H^T H\theta_{\mathcal{S}} + H\left(H^T H\right)^{-1} H^T y_{\perp} \\ &= H\theta_{\mathcal{S}} = y_{\mathcal{S}}\end{aligned}$$

The geometrical interpretation of the above is that $H\hat{\theta}_{ML}$ is the projection of y onto $\mathcal{S}$, the subspace that the basis functions in H are able to span. You may find it easier to see this visually as in figure 3.

While very simple, the ML solution for linear regression is prone to problems. Specifically, notice that when the basis functions are expressive enough to completely fit the points, then they will always do so, even when we might think that such a solution doesn't make much sense. This is called overfitting, an example of which can be seen in figure 4 (center). In fact, this will happen whenever the basis functions span a subspace with the same rank as the number of data points. When this subspace has a larger rank than the number of points, then there are an infinite number of solutions!

3.2 Ridge Regression

As we add more and more basis functions, we will start overfitting at some point - something we would really like to avoid. *Regularization* seeks to reduce the amount of overfitting by adding some restrictions to the values that the weights can take. Usually this is done by adding a penalty term for θ to what we are trying to minimize, like so:

$$L_R = \frac{1}{2}\|y - H\theta\|^2 + \lambda E_R(\theta)$$

where λ is the *regularization coefficient* that controls the relative weight between the least squares expression and the regularization penalty E_R. The simplest form of regularization is given by the norm of the weights:

$$E_R(\theta) = \frac{1}{2}\theta^T\theta = \frac{1}{2}\|\theta\|^2 \tag{3.7}$$

Adding this term to the least squares objective the loss function becomes:

$$L_R = \frac{1}{2}\|y - H\theta\|^2 + \frac{\lambda}{2}\|\theta\|^2 \tag{3.8}$$

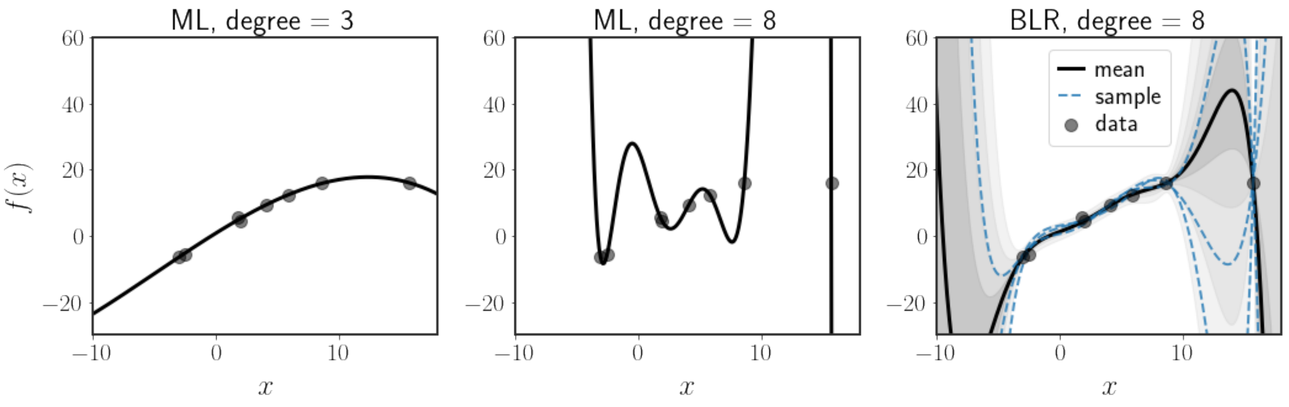

Figure 4: fitting a linear regression model with polynomial basis functions to data. The left most is the ML solution using 3rd order polynomials, the middle using 8th order polynomials while the right most is a Bayesian linear regression model using 8th order polynomials. The model in the center illustrates how the ML solution is prone to overfitting, when using basis functions that are expressive enough to completely describe the data points, unlike the case on the left. Unlike the ML solution, a well calibrated prior can avoid these problems of overfitting, while also defining a distribution over possible solutions. The black line is the MMSE estimate for BLR while the blue lines are samples from the posterior. Notice how the posterior describes high uncertainty outside the bounds of the data points, while being very certain around the data points themselves.

This is only one possible choice for regularization, but is useful since it's easy to find the optimal solution with this regularization. Minimizing L_R with respect to θ, we get:

$$\hat{\theta} = \left(H^T H + I\lambda\right)^{-1} H^T y \tag{3.9}$$

Linear regression with this regularization is often called *ridge regression*. This type of regularization obviously adds more constraints to the optimal values of θ since the weights are penalized based on their magnitude, unlike before. Another way to think about this is that λ suggests a certain budget that the whole model gets for fitting the weights - if $\|y - H\theta\|^2$ is constant and we change the value of λ, then it directly controls the total magnitude of θ.

4 Bayesian Linear Regression

Of course, we can also have a prior over the parameters θ[3]. Since a linear transformation of a Gaussian is also a Gaussian, and since we wrote x as a linear transformation of θ, it will make sense for our prior to have a Gaussian form. We will define our prior as:

$$\theta \sim \mathcal{N}\left(\mu_\theta, \Sigma_\theta\right) \tag{4.1}$$

where $\mu_\theta \in \mathbb{R}^d$ and $\Sigma_\theta \in \mathbb{R}^{d\times d}$; an example for such a prior can be seen in figure 5 (left). The posterior $p\left(\theta | y\right)$ in this case is also a Gaussian distribution given by:

$$p\left(\theta \,|\, y\right) = \mathcal{N}\left(y \,|\, \mu_{\theta|D}, C_{\theta|D}\right) \tag{4.2}$$

with the following parameters:

$$\mu_{\theta|D} = C_{\theta|D}\left(H^T \frac{1}{\sigma^2} y + \Sigma_\theta^{-1}\mu_\theta\right) \tag{4.3}$$

$$C_{\theta|D} = \left(\Sigma_\theta^{-1} + \frac{1}{\sigma^2} H^T H\right)^{-1} \tag{4.4}$$

The posterior describes a reweighing of the prior distribution given the observed points, as can be seen in figure 5 (right).

[3]Bishop 3.3; Murphy 7.6

The optimal estimator (in the BMSE sense), as we have previously seen, is given by the expectation of the posterior:

$$\mathbb{E}\left[\theta|D\right] = C_{\theta|D}\left(H^T\frac{1}{\sigma^2}y + \Sigma_\theta^{-1}\mu_\theta\right) \tag{4.5}$$

Connection to classic linear regression

Since our posterior is a Gaussian, the MMSE and MAP estimates coincide, so let's look at $\mathbb{E}\left[\theta|y\right]$ when the prior is non-informative; this will show us how the MMSE and ML solutions are related. We can do so by defining:

$$\theta_R \sim \mathcal{N}\left(0, \alpha I\right) \tag{4.6}$$

At the limit $\alpha \to \infty$, this prior becomes closer and closer to uniform on the whole space. Using this prior, the expectation over the posterior becomes:

$$\begin{aligned}\mathbb{E}\left[\theta_R|y\right] &= \left(I\frac{1}{\alpha} + \frac{1}{\sigma^2}H^TH\right)^{-1}H^T\frac{1}{\sigma^2}y \\ &= \sigma^2\left(I\frac{\sigma^2}{\alpha} + H^TH\right)^{-1}H^T\frac{1}{\sigma^2}y \\ &= \left(I\frac{\sigma^2}{\alpha} + H^TH\right)^{-1}H^Ty \qquad (4.7) \\ &\overset{\alpha\to\infty}{=} \left(H^TH\right)^{-1}H^Ty \qquad (4.8)\end{aligned}$$

And we have the MLE solution! But wait, we didn't do that just for show. If we rewind back to equation 4.7, notice that this looks suspiciously similar to equation 3.9, where we saw the solution to ridge regression. Now we can give a more informative explanation to ridge regression; instead of saying something a bit hand wavey as "we are trying to avoid overfitting", we can say that ridge regression is the same as Bayesian linear regression with a prior of the form given in equation 4.6. Equivalently, using θ_R as a prior is like trying to solve ridge regression with the regularization:

$$\lambda = \frac{\sigma^2}{\alpha} \tag{4.9}$$

In this sense, if we are very unsure about the prior ($\alpha \gg 1$) then the regularization will be very light, while if we are very sure (α is small), then we will heavily penalize solutions that are far from what we expected.

4.1 Woodbury Matrix Identity

There is an equivalent way of writing the parameters of the posterior. To show this equivalent form, we will need to learn about the *Woodbury matrix identity*. The identity is given by:

$$\left(A + UCV\right)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1}VA^{-1} \tag{4.10}$$

where we assumed that A and C are invertible, while U and V don't even have to be square. Before we continue to use this on the covariance of the posterior, we should talk about when to use this identity. Obviously, if all of the matrices A and C are square and have the same dimensions and we know nothing about their inverses, using this identity will not help us at all. However, many times we will be confronted with an equation similar to the left hand side of equation 4.10, where we actually know what the inverse of A is directly, or know that the inverses of A and C are rather simple to compute.

Example: Low rank matrices

Let's look at an example. Suppose we want to find:

$$\left(I_n\beta + \frac{1}{\alpha}AA^T\right)^{-1} \tag{4.11}$$

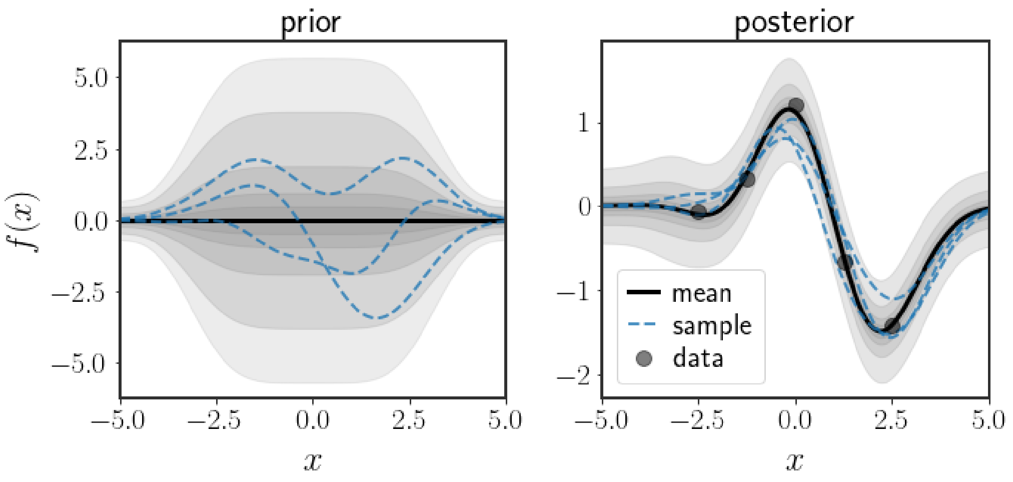

Figure 5: Bayesian linear regression with 10 Gaussian basis functions with means in equally spaced in the range $[-2.5, 2.5]$. On the left is an example of the prior described by the model, as well as a few sampled functions from the prior. The posterior given some data points is shown on the right, as well as sampled functions from the newly defined distribution which describes plausible functions given the observed data points.

where $A \in \mathbb{R}^{n\times m}$ such that $n \gg m$; in this sense AA^T is a *low rank* matrix since its rank (at most m) is much smaller than the full rank (n). In this case, inverting the bigger $n\times n$ matrix will be much less efficient than inverting a small $m \times m$ matrix. We can now put the identity to good use:

$$\begin{aligned}\left(I_n\beta + \frac{1}{\alpha}AA^T\right)^{-1} &= \left(I_n\beta\right)^{-1} - \left(I_n\beta\right)^{-1}A\left(\left(I_m\frac{1}{\alpha}\right)^{-1} + A^T\left(I_n\beta\right)^{-1}A\right)^{-1}A^T\left(I_n\beta\right)^{-1} \\ &= \frac{1}{\beta}I_n - \frac{1}{\beta^2}A\left(I_m\alpha + \frac{1}{\beta}A^TA\right)^{-1}A^T \\ &= \frac{1}{\beta}I_n - \frac{1}{\beta}A\left(I_m\alpha\beta + A^TA\right)^{-1}A^T \qquad (4.12)\end{aligned}$$

Notice that the matrix A^TA is an $m\times m$ matrix, so we end up only needing to invert $m \times m$ matrices, possibly avoiding many unneeded computations.

4.2 Equivalent Form

We now turn back to the covariance we found in equation 4.4. Using the Woodbury identity:

$$\begin{aligned}C_{\theta|D} &= \left(\Sigma_\theta^{-1} + \frac{1}{\sigma^2}H^TH\right)^{-1} \\ &= \Sigma_\theta - \Sigma_\theta H^T\left(\sigma^2 I + H\Sigma_\theta H^T\right)^{-1}H\Sigma_\theta \qquad (4.13) \\ &\triangleq \Sigma_\theta - \Sigma_\theta H^T M^{-1} H\Sigma_\theta \qquad (4.14)\end{aligned}$$

while this doesn't look particularly helpful, sometimes the number of samples (the first dimension of H) will be much smaller than the feature space (the dimension of Σ_θ), in which case we will want to invert in the sample dimension.

It will also be helpful to look at the mean of the posterior in this notation:

$$\begin{aligned}
\mu_{\theta|D} &= C_{\theta|D}\left(H^T\frac{1}{\sigma^2}y+\Sigma_\theta^{-1}\mu_\theta\right)\\
&= \left(\Sigma_\theta-\Sigma_\theta H^T M^{-1}H\Sigma_\theta\right)\left(H^T\frac{1}{\sigma^2}y+\Sigma_\theta^{-1}\mu_\theta\right)\\
&= \left(I-\Sigma_\theta H^T M^{-1}H\right)\mu_\theta+\left(\Sigma_\theta H^T\frac{1}{\sigma^2}-\Sigma_\theta H^T M^{-1}H\Sigma_\theta\frac{1}{\sigma^2}H^T\right)y\\
&= \left(I-\Sigma_\theta H^T M^{-1}H\right)\mu_\theta+\frac{1}{\sigma^2}\Sigma_\theta H^T\left(I-M^{-1}H\Sigma_\theta H^T\right)y\\
&= \left(I-\Sigma_\theta H^T M^{-1}H\right)\mu_\theta+\frac{1}{\sigma^2}\Sigma_\theta H^T M^{-1}\left(M-H\Sigma_\theta H^T\right)x\\
&= \mu_\theta-\Sigma_\theta H^T M^{-1}H\mu_\theta+\frac{1}{\sigma^2}\Sigma_\theta H^T M^{-1}\left(\sigma^2 I+H\Sigma_\theta H^T-H\Sigma_\theta H^T\right)y\\
&= \mu_\theta+\Sigma_\theta H^T M^{-1}\left(y-H\mu_\theta\right) \qquad (4.15)
\end{aligned}$$

And now we have our equivalent form for the mean of the posterior as well as the covariance:

$$\mu_{\theta|D} = \mu_\theta+\Sigma_\theta H^T M^{-1}\left(y-H\mu_\theta\right) \qquad (4.16)$$

$$C_{\theta|D} = \Sigma_\theta-\Sigma_\theta H^T M^{-1}H\Sigma_\theta \qquad (4.17)$$

5 Extras: Numerical Stability of Matrix Inversion

Up to this point, we have mostly dealt with the theoretical aspects of linear regression, where inverting matrices assumed to be easy and exact. In practice, we can very quickly run into problems that arise from the numerical precision of our machines. What does this actually mean? Well, since numbers in our computers are stored using a finite number of bits (not that an infinite number of bits would have helped), there are only so many numbers we can store and there are operations that are not exact. This is especially crucial when working with very small numbers, where the precision of the float saved is of the same order of magnitude of the number itself, or when dividing very large numbers. Numerical stability is an integral part of any practical application of machine learning, so we'll periodically look at solutions for problems that may arise from numerical stability.

The act of inverting matrices is inherently very unstable. This is especially true when we are inverting very large matrices. One of the problems that cause such instabilities is when the matrix is very close to singular, which will usually be the case in this course. As a consequence, it is usually better *not* to invert the matrix directly, if we can get away with it. For instance, if at any point we want to find:

$$x = A^{-1}b \qquad (5.1)$$

for some matrix A, then we can instead solve the linear set of equations for x:

$$Ax = b \qquad (5.2)$$

The result will, mathematically, be the same. However, the algorithm for solving a linear set of equations is inherently much more numerically stable (also faster) than the implementations of matrix inversions. In `numpy`, this can be done by:

$$x = \text{np.linalg.solve}(A, b) \qquad (5.3)$$

(this also works for stacks of A matrices and b vectors, or when b is a rectangular matrix).

Unfortunately, there are times when we can't do this. When we need to calculate explicitly the inverses of certain types of matrices, we should opt to use the most numerically stable, dedicated, method we can. For instance, if we would like to calculate the pseudo-inverse of a matrix:

$$H^\dagger = \left(H^T H\right)^{-1} H^T \qquad (5.4)$$

then, instead of calculating the inverse of $H^T H$ directly, we can use the dedicated function for calculating the pseudo-inverse. In `numpy`, this is:

$$H^\dagger = \text{np.linalg.pinv}\left(H\right) \qquad (5.5)$$

Usually, these dedicated functions were created with numerical stability in mind, which makes them a good option.

The algorithm working behind the scenes when calling `np.linalg.solve` is Gaussian elimination. Where we can run into problems is when the largest number in a row (for instance) is very small. Specifically, when we are working with PD matrices, the diagonal contains the largest values of each row (check this for yourself). So a more hands-on approach for adding stabilization to matrix inversion to PD matrices (but less advised, unless absolutely necessary) is by using the following approximation for any PD matrix A:

$$A^{-1} \approx (A + I\epsilon)^{-1} \tag{5.6}$$

for very small ϵ. This adds stability if we expect the eigenvalues of A to be positive but very close to 0 (so that A is pretty close to singular). Of course, the result will be less accurate, but will be much more stable. Another way to justify this is if we think of A as the covariance matrix for a Gaussian distribution, then the addition of ϵ (for very small ϵ) to the diagonal will change the variance by a very small amount, while ensuring that A truly is PD. Alternatively, you can find the LU factorization or Cholesky decomposition of the matrix:

$$A = LU \Rightarrow A^{-1} = U^{-1}L^{-1} \tag{5.7}$$