Course 67564	2022/23

Bayesian Machine Learning

Exercise 2: Bayesian Linear Regression

Prof. Yair Weiss — *TA: Roy Friedman*

Deadline: December 8, 2022

1 Theoretical

1.1 Reparameterization of Estimators

Consider three different ways of parameterizing k-th order polynomials:

- $y_\theta(x) = \sum_{n=1}^{k} \theta_n x^n + \eta$
- $y_\alpha(x) = \sum_{n=1}^{k} (10\alpha_n) x^n + \eta$
- $y_\gamma(x) = \sum_{n=1}^{k} \gamma_n^3 x^n + \eta$

where θ, α and γ are the parameters and in each case $\eta \sim \mathcal{N}\left(0, I\sigma^2\right)$ for some known σ^2.

1. Suppose we know how to calculate the posterior probability $p_\theta(\theta|\mathcal{D})$. What are the posterior probabilities $p_\alpha(\alpha|\mathcal{D})$ and $p_\gamma(\gamma|\mathcal{D})$ in terms of $p_\theta(\theta|\mathcal{D})$?

2. Let $k = 1$. Let $\hat{\theta}$ be the MMSE estimator for θ. Will the MMSE solutions $y_{\hat{\alpha}}(x)$ and $y_{\hat{\gamma}}(x)$ necessarily have the same value for any data set and x as $y_{\hat{\theta}}(x)$?

1.2 Sequential Bayesian Linear Regression

Suppose we have two data sets $\mathcal{D}_1$ and $\mathcal{D}_2$ for a regression problem. Both data sets contain pairs (x_i, y_i) and we assume that for both data sets:

$$y_i = \sum_{n=1}^{d} \theta_n h_n(x_i) + \eta \tag{1.1}$$

under some set of basis functions $\{h_i(\cdot)\}_{i=1}^{d}$, where $\eta \sim \mathcal{N}\left(0, I\sigma^2\right)$. We will assume a Gaussian prior for θ of the form:

$$\theta \sim \mathcal{N}(\mu, \Sigma) \tag{1.2}$$

In this question we will try to find how to sequentially update a Bayesian linear regression model, as new data arrives. This is equivalent to training on two different data sets and then merging their estimates in some manner.

3. Show that $p(\theta, \mathcal{D}_1, \mathcal{D}_2) = p(\theta)\, p(\mathcal{D}_1|\theta)\, p(\mathcal{D}_2|\theta)$

4. Find the posterior $p(\theta|\mathcal{D}_1, \mathcal{D}_2)$

5. Suppose we have performed Bayesian regression on both data sets and have calculated $\mu_{\theta|\mathcal{D}_1}$, $\mu_{\theta|\mathcal{D}_2}$, $\Sigma_{\theta|\mathcal{D}_1}$ and $\Sigma_{\theta|\mathcal{D}_2}$. After calculating these values we threw away the data. Show that it is possible to calculate the MMSE estimator for θ given the two data sets using only $\mu_{\theta|\mathcal{D}_i}$ and $\Sigma_{\theta|\mathcal{D}_i}$

2 Practical

We will try to predict the temperatures during the second half of a day using linear regression and a couple of different basis functions. The temperatures we will try to predict are for November 16 2020, supplied in the file `nov162020.npy`. The supplied temperatures are given in 30 minute intervals, so there are 48 points in total.

The supplied file `jerus_daytemps.npy`[1] contains data for the mean daily temperature of November in Givat Ram, Jerusalem, at 8 different hours $T = [02{:}00,\ 05{:}00,\ 08{:}00,\ 11{:}00,\ 14{:}00,\ 17{:}00,\ 20{:}00,\ 23{:}00]$. The hours in T will be the xs for the model, while the ys will be the temperatures. In this exercise we will fit the temperature at a time t using linear regression:

$$y_i(t) = \theta^T h(t) + \eta \tag{2.1}$$

under various choices of basis functions $h(\cdot)$. We will assume $\eta \sim \mathcal{N}\left(0,\ I\sigma^2\right)$ with $\sigma^2 = 0.25$. In addition, we will treat the hours as floats in the range $t \in [0, 24)$.

Before we choose the particular basis functions:

1. Implement a class (or function) for classical linear regression which should receive, during initialization, a set of basis functions $h(\cdot)$. This class should be able to fit a linear regression to a given set of x and ys and to predict new values (see the code skeleton supplied)
2. Implement a class (or function) for Bayesian linear regression. This class should receive, during initialization, a Gaussian prior $\theta \sim \mathcal{N}(\mu, \Sigma)$ and a set of basis functions $h(\cdot)$. This class should be able to find the posterior given a set of x and ys and to predict new values (see the code skeleton supplied)

When implementing the two classes above, pay close attention to numerical stability - you can find more details about numerical stability in the summary of recitation 4, and a few more notes on implementation at the bottom of this page.

2.1 Polynomial Basis Functions

Let's start with the simplest kind of basis functions - polynomial basis functions. These basis functions form a polynomial (as we discussed in class) and are given by:

$$p_d(t) = \begin{pmatrix} 1 \\ t \\ \vdots \\ t^d \end{pmatrix} \in \mathbb{R}^{d+1} \tag{2.2}$$

Recall that when using these basis functions, the linear regression problem is equivalent to fitting a polynomial:

$$y(t) = \theta^T p_d(t) + \eta = \sum_{k=0}^{d} \theta_k t^k + \eta \tag{2.3}$$

Because the numbers x^d can be quite large, using the polynomial basis functions as is may cause numerical problems. To mitigate this, we can use the following (equivalent) basis functions:

$$\tilde{p}_d(t) = \begin{pmatrix} 1 \\ t/d \\ \vdots \\ (t/d)^d \end{pmatrix} \in \mathbb{R}^{d+1} \tag{2.4}$$

How is this equivalent? And why should this help? Well, when we use linear regression in order to fit the functions, we will multiply them by a weight which can reverse this effect. This will help since for large ts, the number t^d is very large, close to what our computers can contain in a float. Dividing the number by the degree is a simple fix that helps with numerical stability.

For $d = [3,\ 7]$, do the following:

[1] Data taken from the Israeli meteorological service

3. Fit a linear regression model to the first half of November 16, whose temperatures can be found in the file `nov162020.npy`, and predict the temperatures for the second half of the day. Plot the true test points (using `pyplot.scatter` for instance) together with the predicted curve. What is the average squared error[2] of the predicted temperatures?

4. Use the historical data provided in order to learn a Gaussian prior over θ. To do this, fit the average daily temperature of each year (available in `jerus_daytemps.npy`) to a polynomial using regular linear regression and extract the weights, $\hat{\theta}_{\mathrm{ML}}$. You should now have N different sets of weights, where N is the number of years. Fit a Gaussian distribution (using MLE) to these sets of weights to get the prior $\theta \sim \mathcal{N}(\mu, \Sigma)$[3]

5. Plot the mean functions described by the prior along with confidence intervals (see Section 3.1 for more information), at 0.1 intervals starting at 0 and ending at 23.9. In the same graph, sample and plot 5 different functions using the learned prior θ. How would you expect these sampled functions to look?

6. Fit a Bayesian linear regression model to the first half of the temperatures of November 16 to get the posterior $p(\theta|\mathcal{D})$ and predict the temperatures in the second half of the day. Plot the true test points (again, with `pyplot.scatter`), the MMSE prediction, the standard deviation of the posterior around the MMSE prediction and 5 sampled functions from the posterior. What is the average squared error of the MMSE?

2.2 Gaussian Basis Functions

Another useful type of basis functions are the Gaussian basis functions. We will define them according to:

$$\varphi(t) = \begin{pmatrix} 1 \\ \varphi_1(t) \\ \vdots \\ \varphi_k(t) \end{pmatrix} \in \mathbb{R}^{k+1} \tag{2.5}$$

where:

$$\varphi_i(t) = e^{-\frac{(t-\mu_i)^2}{2\beta^2}} \tag{2.6}$$

for some β and k values $\{\mu_i\}_{i=1}^{k}$ chosen ahead of time. We will look at the following sets of centers:

- $S_1 = \{\mu_i\}_{i=1}^{3} = \{6, 12, 18\}$
- $S_2 = \{\mu_i\}_{i=1}^{5} = \{4, 8, 12, 16, 20\}$
- $S_3 = \{\mu_i\}_{i=1}^{7} = \{2, 4, 8, 12, 16, 20, 22\}$

This time, comparing between linear regression trained only on a subset of the data and Bayesian linear regression which has a prior doesn't make much sense, so we will only look at the performance of Bayesian linear regression.

7. Implement a function that, given a set of centers S, creates the Gaussian basis functions as defined in equation 2.5. For each of the sets of centers S_1, S_2 and S_3, repeat questions 4-6 with $\beta = 3$

2.3 Cubic Regression Splines

The last, slightly more sophisticated, set of basis functions we will see are called *regression splines*[4]. Regression splines are, simply put, piece wise polynomial functions. A piece wise polynomial function is a function that is made up of segments, *each of which* are defined *by a different polynomial*. In this sense, cubic regression splines are functions made up of segments of independent 3rd order polynomials (hence *cubic*), with the added feature that

[2] The average squared error of the points $\{x_i\}_{i=1}^{N}$ and an estimator $\hat{f}$ compared to a true function f is given by: $\mathrm{E}\left(\hat{f}\right) = \frac{1}{N}\sum_{i=1}^{N}\left(f(x_i) - \hat{f}(x_i)\right)^2$

[3] Review recitation 3 if you aren't sure how to fit a Gaussian to data points using MLE

[4] See this blog post or this book for more information

we constrain the 2nd derivative to be continuous. Cubic regression splines are defined by the following set of basis functions:

$$b_\xi(t) = \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \\ (t-\xi_1)_+^3 \\ (t-\xi_2)_+^3 \\ \vdots \\ (t-\xi_k)_+^3 \end{pmatrix} \in \mathbb{R}^{k+4} \tag{2.7}$$

where:

$$(t-a)_+ = \begin{cases} t-a & t-a \geq 0 \\ 0 & t-a < 0 \end{cases} \tag{2.8}$$

and $\xi = \{\xi_i\}_{i=1}^k$ are k values chosen ahead of time. The segments of the cubic spline are defined by the values ξ_i, so that in every interval $[\xi_{i-1}, \xi_i]$ the function is a cubic polynomial (and different for each interval). The ξ_is are usually called *knots*. We will look at the following sets of knots:

- $K_1 = \{12\}$
- $K_2 = \{8, 16\}$
- $K_3 = \{6, 12, 18\}$

Again, if we try to train the classic linear regression on only one half of a day, using intervals beyond it won't make much sense, so we will only look at the first using Bayesian linear regression.

8. Implement a function that, given a set of knots ξ, creates the cubic spline regression basis functions as defined in equation 2.7. For each of the sets of knots K_1, K_2 and K_3, repeat questions 4-6
9. Compare the average squared error of all of the methods from this exercise

3 Implementation Notes

3.1 Confidence Intervals

One of the most important elements of Bayesian statistics and machine learning is that it is (or at least should be) very easy to check how unsure our model is regarding the fit, since we directly model the parameters probabilistically. One way to visualize this is to use something like confidence intervals. Recall that linear regression is defined by:

$$y(x) = h(x)^T \theta + \eta \tag{3.1}$$

where $\eta \sim \mathcal{N}\left(0, I\sigma^2\right)$. When θ is given by $\theta \sim \mathcal{N}(\mu, \Sigma)$, then we actually know the distribution over $y(x)$ as well:

$$y(x) \sim \mathcal{N}\left(h(x)^T \mu,\, h(x)^T \Sigma h(x) + \sigma^2\right) \tag{3.2}$$

Notice that this is a 1D Gaussian. Of course, the posterior will also be a Gaussian:

$$\theta|\mathcal{D} \sim \mathcal{N}\left(\mu_{\theta|\mathcal{D}},\, C_{\theta|\mathcal{D}}\right) \tag{3.3}$$

so we will also know the distribution of y given the data:

$$y(x)\,|\mathcal{D} \sim \mathcal{N}\left(h(x)^T \mu_{\theta|\mathcal{D}},\, h(x)^T C_{\theta|\mathcal{D}} h(x) + \sigma^2\right) \tag{3.4}$$

The uncertainty over the value of the prediction $y(x)|\mathcal{D}$ is directly given by the variance, or the standard deviation, in the above distribution:

$$\sigma_{y|\mathcal{D}} = \sqrt{\text{var}\left[y(x)|\mathcal{D}\right]} = \sqrt{h(x)^T C_{\theta|\mathcal{D}} h(x) + \sigma^2} \tag{3.5}$$

Now, we want to plot the mean prediction $h(x)^T \mu_{\theta|\mathcal{D}}$ (as it is most likely), but it will also be informative to shade in the interval cI $= \left[h(x)^T \mu_{\theta|\mathcal{D}} - \sigma_{y|\mathcal{D}},\ h(x)^T \mu_{\theta|\mathcal{D}} + \sigma_{y|\mathcal{D}}\right]$. Since the posterior (and prior) is a Gaussian, this means that the interval cI is *where the model expects most of the data points to be*. If most points don't fall in this area, we can be fairly sure that the prior we chose isn't all that good.

See the function `confidence_interval_example()` in `ex2_utils.py` for an example of how to plot the confidence intervals and how the plots should look.

3.2 Debugging

Since this is the first practical exercise in the course, let's talk a bit about debugging. When implementing models for the first time, it may not be obvious what the required behavior should be. However, there are usually simple cases that are easily verifiable. For instance, let's say we want to implement classical linear regression. *By definition*, linear regression tries to fit a straight line to data points. So if you sample points from the function:

$$y = \theta_0 + \theta_1 x \tag{3.6}$$

then linear regression should be able to find it completely. In addition, as we saw in class, if you sample many points (and honestly you don't need that many), then Bayesian regression should also fit the line exactly. Small sanity checks of this type can help you quite a bit. Of course, if you want to check that it works for more complicated functions, say polynomials, then sample points according to $y = \sum_{k=0}^{d} \theta_k x^k$ and linear regression in the same space should be able to get an exact fit.

The second thing we need to pay close attention to is numerical stability, but it's a bit harder to directly check if you run into problems in this regard... To build an intuition for how solutions with numerical instabilities look, try to use the definition of polynomial basis functions in equation 2.2 (the unnormalized version) and compare it to what you get when using the definition in equation 2.4, especially for the 7th order polynomials. One of these should fit the data pretty well, while the other one will seem to diverge rapidly (it shouldn't be hard to see the difference). Another way to notice numerical instabilities is to use what is theoretically supposed to happen as a starting point. The (theoretical) behavior we would expect, with 8 data points, is that 8 degrees of freedom should fit the data pretty well. Since we know this, if something unexpected pops up, we can start suspecting the role of numerical stability in our results.

4 Submission Guidelines

Submit a single zip file named "`ex2_<YOUR ID>.zip`". This file should contain your code, along with an "`ex2.pdf`" file in which you should write your answers to the theoretical part and add the figures/text for the practical part. Please write readable code, as the code will also be checked manually (and you may find it useful in the following exercises). In the submitted code, please make sure that you write a basic main function in a file named "`ex2.py`" that will run (without errors) and produce all of the results that you showed in the pdf of answers that you submitted. The only packages you should use are `numpy` and `matplotlib`.

In general, it is better if you type your homework, but if you prefer handwriting your answers, please make sure that the text is readable when you scan it.

Part of your assignment will be graded by submitting your answers through Moodle, at this link. In each of the questions, write the answer to the corresponding question for grading. These answers will be graded automatically, so write only numeric values where needed.

5 Supplementary Code

In the file `ex2utils.py` you can find an example of how to load the supplied data as well as a few helper functions. You can use this code as you see fit, and change any part of it that you want, just be sure to submit it as well if you change it. Finally, we have also supplied an outline code which you can use to get started in `ex2.py`. You don't have to use the format we outlined, but your code must run without errors and you must submit the plots required in the exercise description.

Good luck!