Splines (Problem 1)

Mohamad Lakkis

May, 2024

This PDF is to explain some of the coding part in the *Splines (Problem 1)*

Introduction: understanding the problem of splines

The goal of this problem is to understand the concept of splines and how they can be used in regression. What we are doing is that we are working with this form,

$$f(x) = \sum_{j=1}^{N} N_j(x)\theta_j$$

Where $N_j(x)$ are the basis functions and θ_j are the coefficients. So we can look at this as a generalized linear model with the basis functions applied to the original predictors as the new predictors. (i.e. look at $N_j(x)$ as x_j). The RSS that we want to minimize is very similar to the one we used in ridge regression, but now we are working with a generalized ridge regression. //

$$RSS(\theta, \lambda) = (y - N\theta)^T(y - N\theta) + \lambda\theta^T\Omega_N\theta$$

Note that we are working with a generalized ridge Regression and not generalized OLS, since if we remove the penalty term, we can get the RSS to be 0, by choosing any function that interpolates the training data.
To see the linear form, where in the case of the smoothing we have N bases located at the N training points:

$$f(x) = \beta_1 N_1(x) + ... + \beta_N N_N(x)$$

And so we can see intuitively (as in ridge), the solution is linear in y:

$$\hat{\theta} = (N^T N + \lambda\Omega_N)^{-1} N^T y$$

Note that, the choice of the basis functions is independent of the choice of the penalty term (λ). And following from $\hat{\theta}$, we get $\hat{f}(x)$

$$\hat{f}(x) = N(x)\hat{\theta} = N(x)(N^T N + \lambda\Omega_N)^{-1} N^T y = S_\lambda y$$

With S_λ is the smoothing matrix.
Similar to the ridge regression, we can also choose λ by instead choosing the Effective degrees of freedom. The relation is modeled as:

$$EDOF = tr(S_\lambda)$$

Note that for computation power sometimes in fitting a smoothing spline we might use a smaller number of knots than the number of training points(i.e. we are also reducing the number of bases). This is discussed in the ESL book!

- Throughout this problem, we will be only varying the Y(i.e. ϵ and not the X's), so all of the expectation and variances are with respect to $Y|X$.

Overview of the python Libraries used

In order to fit the smoothing splines, we will be using a special case of GAM number of features equal to 1, which are available in the pyGAM library. We will be choosing the splines as the first term. And using the degree of the spline to be 3. But in order to get a smoothing splines we need to put the penalties to auto.

- Note that in pyGAM they have a default of 20 splines, so we will be using that as the number of knots. AND NOT the number of training points (for computation power, as mentioned).
- The build columns method will give us the basis function evaluated at the point you want, so N(x) in the above formula. (on the training points I will just denoted by N), but we are able to compute this N on new points as well.
- When you apply the method to find the coefficients, keep in mind that the last one is the intercept, and the rest are the coefficients of the basis functions. So for example $\hat{f}(x_0) = N(x_0)\hat{\theta}+\hat{\theta_0}$, where $N(x_0)$ is obtained by using build columns on x_0, and $\hat{\theta}$ is the coefficients of the basis functions. And $\hat{\theta_0}$ is the intercept(i.e. last coefficient).
- In the code I provide two methods to get $\hat{\theta}$, by directly using their code or by doing my own implementation(they are the same). $\hat{\theta} = (N^T N + \lambda\Omega_N)^{-1}N^T y$, with everything on the training data x in play.
- Now, in order to get the $\lambda\Omega_N$ matrix, we use the build penalties method. Let's call it P
- Lastly, to compute S_λ, it is simply $N(x)(N^T N + P)^{-1}N^T$, with N(x) is the basis functions evaluated at the training point in this case.

Computing the expectation (pointwise) $E[\hat{f}_\lambda(x_0)]$(Mean at each point)

- Note that x_0, could be a training point and could be any point that is coming from the distribution of $X \sim U[0,1]$.

$$E_Y(\hat{f}_\lambda(x_0)|X = x) = E_Y(N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T y|X = x)$$

Since only y is random,

$$= N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T E_Y(y)$$

$$= N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T E_\epsilon(f(x_0) + \epsilon)$$

But since $E_\epsilon(\epsilon) = 0$, and from the linearity of the expectation, we get:

$$= N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T E_\epsilon(f(x))$$

but notice that $f(x)$ is independent of ϵ, so we can take it out of the expectation:

$$= N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T f(x)$$

So now we know that $E_Y(y|X = x) = f(x)$, so in my code I can choose the true $f(x)$, since I am nature in the simulated environment, or I can get many samples of y and take the average of them to get the expectation, which with n samples will be very close to the true $f(x)$. (What will guarantee that is that in our simulated environment, the $\epsilon \sim N(0,1)$, so $E_Y(y|X = x)$ is $f(x)$.)

Computing the variance (pointwise) $Var[\hat{f}_\lambda(x_0)]$(Variance at each point)

$$Var_Y(\hat{f}_\lambda(x_0)|X = x) = Var_Y[N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T y|X = x]$$

Since only y is random, and the variance is with respect to $Y|X = x$, we get:

$$= N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T Var_Y(y|X = x)N[(N^T N + \lambda\Omega_N)^{-1}]^T N(x_0)^T$$

Since $(N^T N + \lambda\Omega_N)$ is symmetric, we can remove the transpose from the second term, and we get:

$$= N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T Var_Y(y|X = x)N(N^T N + \lambda\Omega_N)^{-1}N(x_0)^T$$

But now notice that $Var_Y(y|X = x) = Var_Y(f(x) + \epsilon|X = x) = Var_Y(\epsilon|X = x) = \sigma I$,and $\sigma = 1$ so we get:

$$= N(x_0)(N^T N + \lambda\Omega_N)^{-1}N^T N(N^T N + \lambda\Omega_N)^{-1}N(x_0)^T$$

Computing the Bias (pointwise) $Bias[\hat{f}_\lambda(x_0)]$(Bias at each point)

$$Bias_Y(\hat{f}_\lambda(x_0)|X = x) = f(x_0) - E_Y[\hat{f}_\lambda(x_0)|X = x]$$

And notice that $f(x_0)$, we know it(since we are nature) or we can approximate it by taking many samples of y and taking the average of them at x_0.
And notice that we can get $E_Y[\hat{f}_\lambda(x_0)|X = x]$ from the previous section.

Computing the MSE (pointwise) $MSE[\hat{f}_\lambda(x_0)]$(MSE at each point)

$$MSE_Y(\hat{f}_\lambda(x_0)|X = x) = Bias^2_Y(\hat{f}_\lambda(x_0)|X = x) + Var_Y(\hat{f}_\lambda(x_0)|X = x)$$

And notice that now we are able to get the Bias and the Variance from the previous sections.

Getting the EPE (Expected Prediction Error) for the whole model

$$EPE = E_X[E_Y(MSE[\hat{f}_\lambda(x_0)]|X = x)] + \sigma^2$$

And we can get all quantities easily by simulation by taking many X train points and then taking for each one many y train points and evaluating these fitted models on points x_0.

Getting the Cross-Validation score

$$CV(\hat{f}_\lambda) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}^{(-i)}(x_i))^2$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{f}_\lambda(x_i)}{1 - S_\lambda(i,i)}\right)^2,$$

Getting the standard error pointwise

$$SE(\hat{f}_\lambda(x_0)|X = x) = \frac{\sqrt{Var_Y(\hat{f}_\lambda(x_0)|X = x)}}{\sqrt{n}}$$

And we can get the variance from the previous section. And notice that n = 1, since we working with the pointwise standard error.

Pointwise Percentile

- At each point x_0, we have many $\hat{f}_\lambda$ depending on each y train that we are choosing (since we are varying only over y). So let's say that we had 100 y train points, we will have 100 $\hat{f}_\lambda(x_0)$, so we can get the 5th percentile, 50th percentile, and 95th percentile of these 100 $\hat{f}_\lambda(x_0)$, very easily !

Plots and Graphs

- In the first figure: we can see how as we increase the df_λ, we are getting a more flexible model, and we can see that the bias is decreasing, but the variance is increasing as expected.
- We can see that based on lowest EPE we have the best $df_\lambda = 9$, which is shown as being very close to the true function f.
- This is also shown in the second figure, where we can see that the bias is decreasing and the variance is increasing as we increase the df_λ more clearly
- From the mean (i.e. $E(\hat{f}(x))$ it is very close to the fitted spline, and even the 90th and 50th percentile, shows how all of the models are ranging around the fitted spline.)
- We can also see How at the boundaries, the variance is increasing, especially when we don't use a linear spline in the last section of the knots. (because in PYGAM we are fitting in all of them a degree 3 spline).
- The last figure is just a dummy example to showcase the point that we talked about in the section of "calculating the bias" by either taking the true $f(x_0)$ or approximating it by taking many samples of y and taking the average of them at x_0. And we can that for the average to be very close to the true $f(x_0)$, we need to keep on increasing n.
- Final Note, is that from the last figure we can now clearly see how the bias is decreasing as we increase df_λ (i.e. we are closer to the true function $f(x)$)

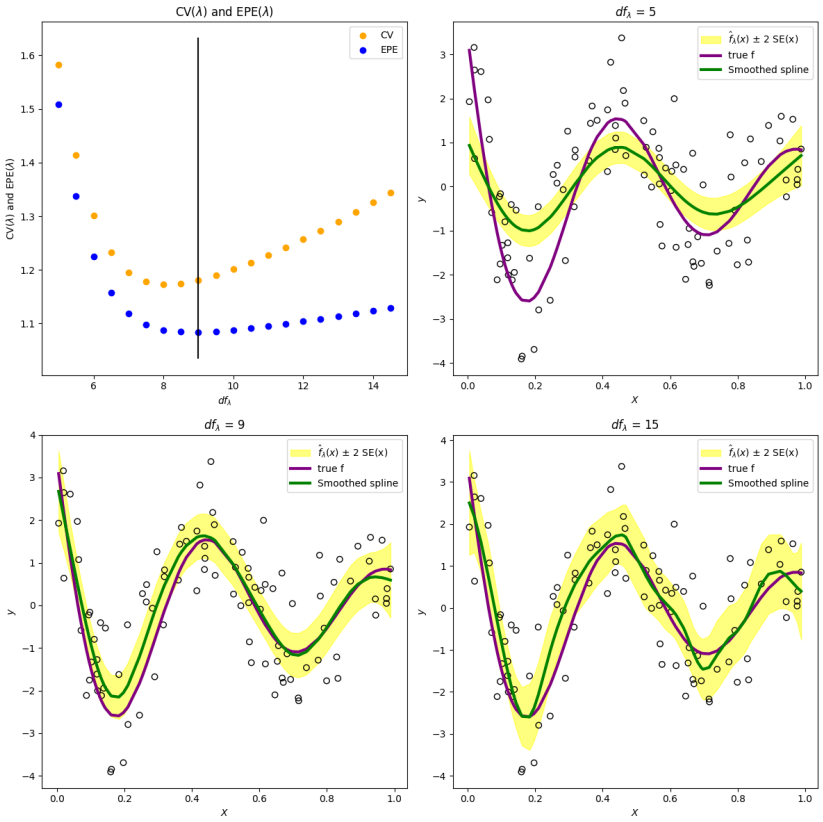

Figure 1: Figure 5.9

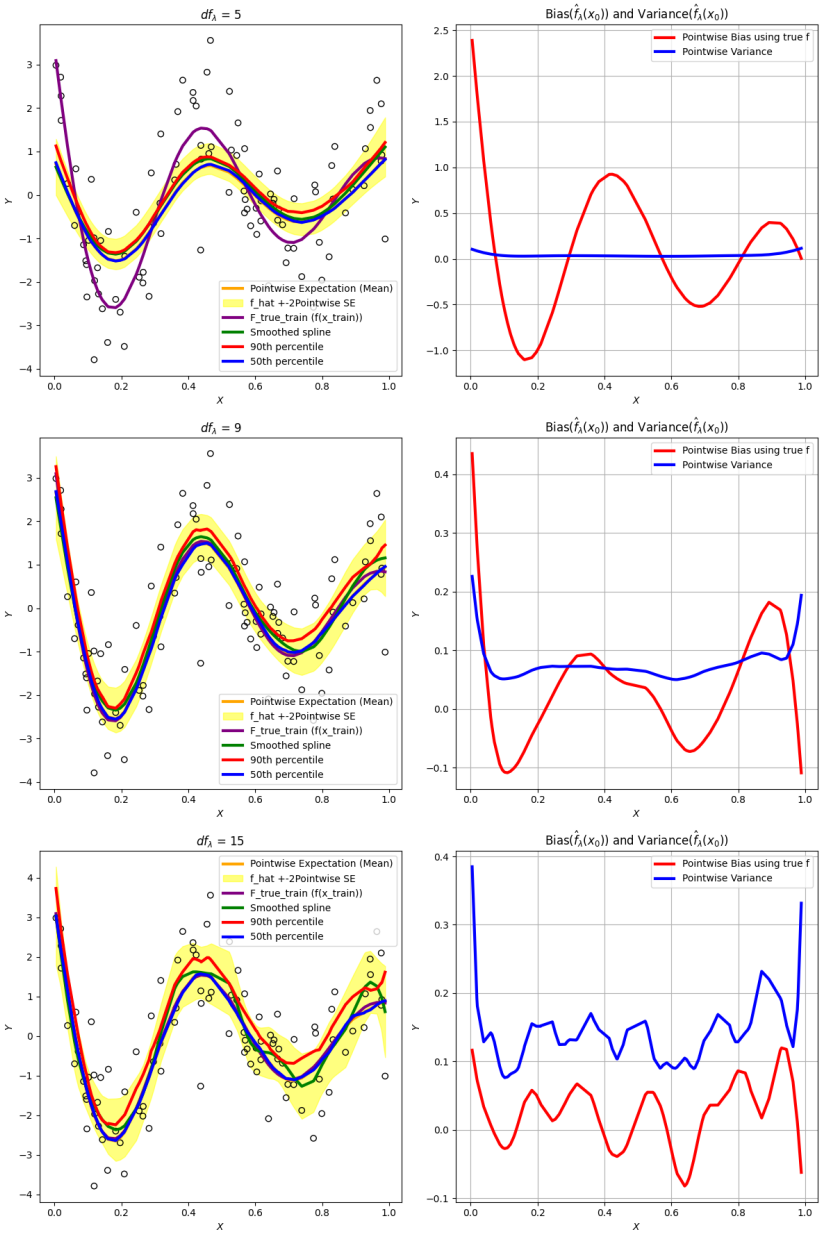

Figure 2: Further Analysis

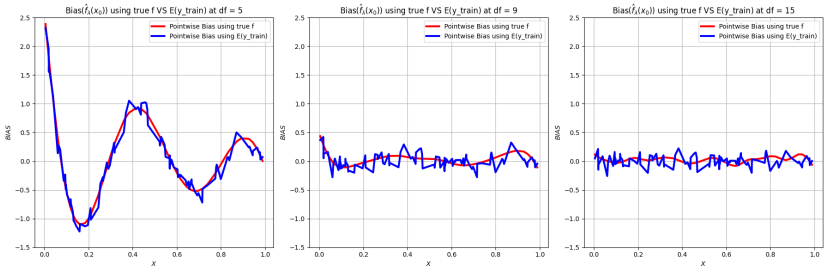

Figure 3: Bias 1 VS Bias 2