CS221 Fall 2017 Homework

Alexi Stein – `lexi@stanford.edu`

May 25, 2022

Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1: Kernel Regression

a.)

1

First, begin by expanding Φ. We have

$$\begin{aligned}
\Phi &= \begin{bmatrix} \phi(x^{(1)})^T \\ \phi(x^{(2)})^T \\ \vdots \\ \phi(x^{(n)})^T \end{bmatrix} \\
&= \begin{bmatrix} 1 & (x^{(1)} - x^{(1)})_+ & (x^{(1)} - x^{(2)})_+ & \dots & (x^{(1)} - x^{(n-1)})_+ \\ 1 & (x^{(2)} - x^{(1)})_+ & (x^{(2)} - x^{(2)})_+ & \dots & (x^{(2)} - x^{(n-1)})_+ \\ 1 & (x^{(3)} - x^{(1)})_+ & (x^{(3)} - x^{(2)})_+ & \dots & (x^{(3)} - x^{(n-1)})_+ \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x^{(n)} - x^{(1)})_+ & (x^{(n)} - x^{(2)})_+ & \dots & (x^{(n)} - x^{(n-1)})_+ \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & (x^{(2)} - x^{(1)})_+ & 0 & \dots & 0 \\ 1 & (x^{(3)} - x^{(1)})_+ & (x^{(3)} - x^{(2)})_+ & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x^{(n)} - x^{(1)})_+ & (x^{(n)} - x^{(2)})_+ & \dots & (x^{(n)} - x^{(n-1)})_+ \end{bmatrix} \\
&= L.
\end{aligned}$$

We obtain this lower triangular $L \in \mathbb{R}^{n \times n}$ via the fact that $x^{(1)} < x^{(2)} < \dots x^{(n)}$, which zeros the upper triangular entries under ReLU activation.

Next, as set forth in the problem, we have

$$y^{(i)} = \phi(x^{(i)})^T\theta,$$

which in matrix form amounts to

$$Y = \Phi\theta = L\theta.$$

Now, we know that $\Phi = L$ is a lower triangular matrix in $\mathbb{R}^{n\times n}$ with all positive entries along the diagonal; hence, it is invertible. As $Y \in \mathbb{R}^n$, we are then guaranteed that $\theta^* \in \mathbb{R}^n$ such that Y $= \Phi\theta^* = L\theta^*$. That is, θ^* is the set of coefficients that i.) solves the system exactly and ii.) minimizes the 2-norm. Since the system is solved exactly, each training feature entry $x^{(1)}, \dots, x^{(n)}$ is mapped directly back to the corresponding response entry $y^{(1)}, \dots, y^{(n)}$. The same is true of linear interpolation, where each $x^{(i)}$ is mapped to its original $y^{(i)}$, with a game of connect-the-dots in between. More succinctly, the invertible lower-triangular L allows for θ^* such that for any training pair $(x^{(i)}, y^{(i)})$, we will have

$$y^{(i)} = \phi(x^{(i)})^T\theta^*.$$

Similarly, on examination of s_n, we see

$$\begin{aligned} s_n(x^{(i)}) &= \sum_{i=1}^{n}\left(y^{(i)} + \frac{y^{(i+1)} - y^{(i)}}{x^{(i+1)} - x^{(i)}}(x^{(i)} - x^{(i)})\mathbf{1}(x \in [x^{(i)}, x^{(i+1)})\right) \\ &= \sum_{i=1}^{n}\left(y^{(i)} + 0)\mathbf{1}(x \in [x^{(i)}, x^{(i+1)})\right) \\ &= y^{(i)}. \end{aligned}$$

Hence at the training points (as stated in the problem), the two are equal.

2

First, some notation. Since the rows of Φ are indexed $i = 1, \dots, n$ and the columns $j = 0, \dots, n-1$, we will use

$$\theta^* = \begin{bmatrix} \theta_0^* \\ \vdots \\ \theta_{n-1}^* \end{bmatrix}$$

to index the coefficients. This way, the inner product will align with the column indexing of Φ, with θ_0^* serving as the intercept. Next, consider the expansion of $\phi(x)$ for $x < x^{(1)}$. We will have

$$\phi(x) = \begin{bmatrix} 1 \\ (x - x^{(1)})_+ \\ (x - x^{(2)})_+ \\ \vdots \\ (x - x^{(n-1)})_+ \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Hence, it is necessarily the case that

$$\phi(x)^T\theta^* = \theta_0^*.$$

Similarly, when $x \geq x^{(n)}$, we have

$$\phi(x) = \begin{bmatrix} 1 \\ (x - x^{(1)})_+ \\ (x - x^{(2)})_+ \\ \vdots \\ (x - x^{(n-1)})_+ \end{bmatrix},$$

where the ordering of the $x^{(i)}$ will guarantee that $0 < \phi(x)_j < \phi(x)_\ell$ for all $j < \ell$; in other words, $\phi(x)$ will be a nonzero vector of increasing values (excluding intercept term), we will have

$$\begin{aligned} \phi(x)^T &= \sum_{j=0}^{n-1}(x - x^{(j)})_+\theta_j^* \\ &= \sum_{j=0}^{n-1}(x - x^{(j)})\theta_j^* \\ &= \sum_{j=0}^{n-1}(x - x^{(n)} + x^{(n)} - x^{(j)})\theta_j^* \\ &= \sum_{j=0}^{n-1}(x - x^{(n)})\theta_j^* + \sum_{j=0}^{n-1}(x^{(n)} - x^{(j)})\theta_j^* \\ &= s_n(x^{(n)}) + \sum_{j=0}^{n-1}(x - x^{(n)})\theta_j^* \\ &= s_n(x^{(n)}) + (x - x^{(n)})\sum_{j=0}^{n-1}\theta_j^*; \end{aligned}$$

in other words, it is the prediction of the last support point plus some "residual" or "padding" added on. Again, recall the updated definition of s_n, in avoidance of confusion.

b.)

1

We have, applying the chain rule (trivially) to the ReLU:

$$\begin{aligned}
\frac{d}{dx}s_n(x) &= \frac{d}{dx}\sum_{j=0}^{n-1}(x - x^{(j)})_+\theta_j^* \\
&= \frac{d}{dx}\left[\theta_0^* + \sum_{j=1}^{n-1}(x - x^{(j)})_+\theta_j^*\right] \\
&= \sum_{j=1}^{n-1}\frac{d}{dx}(x - x^{(j)})_+\theta_j^* \\
&= \sum_{j=1}^{n-1}\theta_j^*\frac{d}{dx}\max(0, x - x^{(j)}) \\
&= \sum_{j=1}^{n-1}\theta_j^*\mathbf{1}(x - x^{(j)} > 0)\cdot\frac{d}{dx}x \\
&= \sum_{j=1}^{n-1}\theta_j^*\mathbf{1}(x > x^{(j)})\cdot 1,
\end{aligned}$$

as desired.

2

Take arbitrary $x^{(i)}$. When your x is just underneath $x^{(i)}$, i.e. $x = x^{(i)} - \delta$ for some $\delta < \frac{1}{n-1}$, you are guaranteed that

$$\mathbf{1}(x > x^{(j)}) = 0$$

for all $j \geq i$, since $x^{(j)} \geq x^{(i)} > x$ in such cases. Hence, we have

$$\frac{d}{dx}s_n(x) = \sum_{j=1,\ldots,i-1}\theta_j^*.$$

However, when you take a step just "across" $x^{(i)}$ – more precisely, a step $\Delta \in (x^{(i)} - \delta, x^{(i)} - \delta + \frac{1}{n-1}]$, you will now have $x^{(i)} \leq x' = x + \Delta \leq x^{(i+1)}$ (inequalities by construction), and hence

$$\mathbf{1}(x > x^{(j)}) = 0$$

for all $j \geq i + 1$. This then gives

$$\frac{d}{dx}s_n(x') = \sum_{j=1,\ldots,i}\theta_j^*.$$

The takeway of this is that the gradient is effectively a step function with respect to x: when you go from "just under" $x^{(i)}$ to "just over" $x^{(i)}$[1], you add the next θ_j^* in line to your derivative.

3

When it is the case that $x < x^{(1)}$, you are guaranteed

$$\mathbf{1}(x > x^{(j)}) = 0$$

for all $j \geq 1$. By the derivative calculation in c.), the derivative must be zero.

c.)

1

For this to make sense, I'm treating $M_{j,0} = 0$; I believe the instruction has a mild typo. First, observe by matrix/vector multiplication that

$$\begin{aligned}
(M\theta)_i &= \sum_{j=0}^{n-1} M_{i,j}\theta_j^* \\
&= 0 + \sum_{j=1}^{n-1} M_{i,j}\theta_j^* \\
&= \sum_{j=1}^{n-1} \mathbf{1}(x^{(i)} \geq x^{(j)})\theta_j^* \\
&= \sum_{j=1}^{i} \theta_j^*.
\end{aligned}$$

Then, by the "just under/just over" logic set forth in problem e.), we have that for $x \in (x^{(i)}, \infty]$, we have

$$\frac{d}{dx}s_n(x) \geq \sum_{j=1,\ldots,i} \theta_j^*.$$

Then, it is clear that as $x \to x^{(i)}$ from above, x will at some point permanently enter the interval $(x^{(i)}, x^{(i+1)}]$, and thereafter , we will have

$$\frac{d}{dx}s_n(x) = \sum_{j=1,\ldots,i} \theta_j^*.$$

Intuitively, by approaching from above, we're tiptoeing as close as possible to the step without going down it, and hence the θ_i^* term remains in the summand.

[1]Where "just under/over" are taken to mean within $1/(n-1)$, the interval size

2

Recall we have $x = \{(j-1)/(n-1)\}_{j=0}^{n}$ We have, by the evenly-spaced construction of x

$$\begin{aligned}
Q(s) &= \int_0^1 \left(\frac{d}{dz}s_n(z)\right)^2 dz \\
&= \int_0^{1/(n-1)} \left(\frac{d}{dz}s_n(z)\right)^2 dz + \int_{1/(n-1)}^{2/(n-1)} \left(\frac{d}{dz}s_n(z)\right)^2 dz + \ldots + \int_{(n-2)/(n-1)}^{(n-1)/(n-1)} \left(\frac{d}{dz}s_n(z)\right)^2 dz \\
&= \sum_{i=1}^{n-1} \int_{z=x^{(i)}}^{x^{(i+1)}} \left(\frac{d}{dz}s_n(z)\right)^2 dz \\
&= \sum_{i=1}^{n-1} \int_{z=x^{(i)}}^{x^{(i+1)}} (M\theta)_i^2 dz \\
&= \sum_{i=1}^{n-1} (M\theta)_i^2 \int_{z=x^{(i)}}^{x^{(i+1)}} 1 \cdot dz \\
&= \sum_{i=1}^{n-1} (M\theta)_i^2 \cdot \left(\frac{1}{n-1}\right) \\
&= \frac{1}{n-1} \sum_{i=1}^{n-1} (M\theta)_i^2
\end{aligned}$$

The parallel between the two forms then becomes obvious.

3

As detailed in lecture and confirmed by intuition/inspection, a λI regularization term corresponds to penalizing the second derivative of the spline function, whereas a $\lambda M^T M$ regularization term corresponds to penalizing the first derivative of the spline function. In other words, λI enforces smoothness, whereas $\lambda M^T M$ enforces flatness. So on first pass, we might expect our $\hat{\theta}$ under $M^T M$ to reflect a flatter/more-zero-like solution – however, that is not the case.

In fact, since we have a linear interpolation (i.e. must hit every training point), the solution can neither be made more linear (it already is) nor flatter (would no longer interpolate), so both sets of coefficients must be the same. $\theta^*_{(1)} = \theta^*_{(2)}$. So question largely becomes trivial, as the (linear) interpolation constraints have deterministic consequences in terms of linearity and flatness.

d.) Penalized Kernel Regression

1

Our aim is to minimize objective

$$\begin{aligned} J(\theta; X, Y, \lambda) &= n^{-1}||Y - \Phi\theta||_2^2 + n^{-1}\lambda||M\theta)||_2^2 \\ &= \frac{1}{n}(Y - \Phi\theta)^T(Y - \Phi\theta) + \frac{\lambda}{n}(\theta^T M^T M\theta) \end{aligned}$$

Taking a gradient w.r.t. θ^T and setting to zero gives:

$$\begin{aligned} 0 &= \nabla_{\theta^T} J(\theta; X, Y, \lambda) \\ &= \nabla_{\theta^T}\left[n^{-1}||Y - \Phi\theta||_2^2 + n^{-1}\lambda||M\theta)||_2^2\right] \\ &= \nabla_{\theta^T}\left[\frac{1}{n}(Y - \Phi\theta)^T(Y - \Phi\theta) + \frac{\lambda}{n}(\theta^T M^T M\theta)\right] \\ &= \frac{-2}{n}\Phi^T(Y - \Phi\theta) + \frac{2\lambda}{n}M^T M\theta \\ &= -\Phi^T(Y - \Phi\theta) + \lambda M^T M\theta \\ \implies \Phi^T Y &= \Phi^T\Phi\theta + \lambda M^T M\theta \\ \Phi^T Y &= (\Phi^T\Phi + \lambda M^T M)\theta \\ \implies \hat{\theta} &= (\Phi^T\Phi + \lambda M^T M)^{-1}\Phi^T Y, \end{aligned}$$

as desired. This is effectively the standard ridge regression proof.

2

No problem presented for 3.d.2; not on HW sheet.

3

Observe that by definition/construction, we have

$$M = \begin{bmatrix} \mathbf{0}_{n\times 1} & L_1 \\ 0 & \mathbf{0}_{n\times 1}^T \end{bmatrix},$$

where $L_1 \in \mathbb{R}^{(n-1)\times(n-1)}$ is a lower triangular matrix with 1's entered into the lower triangular entries. As such, it follows that

$$M^T M = \begin{bmatrix} 0 & \mathbf{0}_{n\times 1}^T \\ \mathbf{0}_{n\times 1} & L_1^T L_1. \end{bmatrix}$$

And since L_1 consists only of ones, $L_1^T L_1$ takes the form

$$L_1^T L_1 = \begin{bmatrix} n-1 & n-2 & n-3 & n-4 & \dots & 1 \\ n-2 & n-2 & n-3 & n-4 & \dots & 1 \\ n-3 & n-3 & n-3 & n-4 & \dots & 1 \\ n-4 & n-4 & n-4 & n-4 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & 1 & 1 & 1. \end{bmatrix}$$

Hence, in full, we have (keeping the 1-indexing of the rows and 0-indexing of the columns):

$$M^T M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & n-1 & n-2 & n-3 & n-4 & \dots & 1 \\ 0 & n-2 & n-2 & n-3 & n-4 & \dots & 1 \\ 0 & n-3 & n-3 & n-3 & n-4 & \dots & 1 \\ 0 & n-4 & n-4 & n-4 & n-4 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

4

See attached RMarkdown file/pdf.

5

See attached RMarkdown file/pdf.

6

Here, we plug x – the new "y" – into the quasi-normal equations, to obtain

$$\hat{\theta} = (\Phi^T\Phi + \lambda M^T M)^{-1}\Phi^T x$$

and hence

$$\hat{y} = \Phi\hat{\theta} = \Phi(\Phi^T\Phi + \lambda M^T M)^{-1}\Phi^T x.$$

HW4 Addendum

Isaac Kleisle-Murphy

5/23/2022

d.)

(4)

First, we instantiate our `relu_kernel_regressor()` function, which performs the regression via the `relu` kernel.

```r
vstack <- function(x){
  # Stacks a vector vertically, with the vector^T
  # repeated over rows
  lapply(1:length(x), function(i){
    matrix(x, nrow=1)
  }) %>%
    do.call("rbind", .)
}

hstack <- function(x){
  # Stacks a vector horizontally, with the vector
  # repeated over columns
  lapply(1:length(x), function(i){
    matrix(x, ncol=1)
  }) %>%
    do.call("cbind", .)
}

relu_kernel_regressor <- function(x, y, lambda=1.0){
  ### create order stats
  xsorted = sort(x)
  ysorted = matrix(y[order(x)], ncol=1)
  N = length(x)

  ### Make Gram matrix
  Phi = cbind(
    1.,
    (hstack(xsorted) - vstack(xsorted))[, 1:(N-1)]
  )
  ### relu
  Phi[Phi < 0] = 0.

  ### make mask
  M = Phi
  ### zero intercepts
  M[, 1] = 0.
  ### insert 1's
  M[M > 0] = 1.

  ### solve
  theta = (
    solve(
      t(Phi) %*% Phi
      + lambda * t(M) %*% M
    ) %*% (
      t(Phi) %*% ysorted
    )
  )
  ### insample fit
  yhat = Phi %*% theta

  return(
    list(
      theta=theta,
      yhat=yhat
    )
  )
}
```

To convince you that it works, consider the following regression, where data is generated via

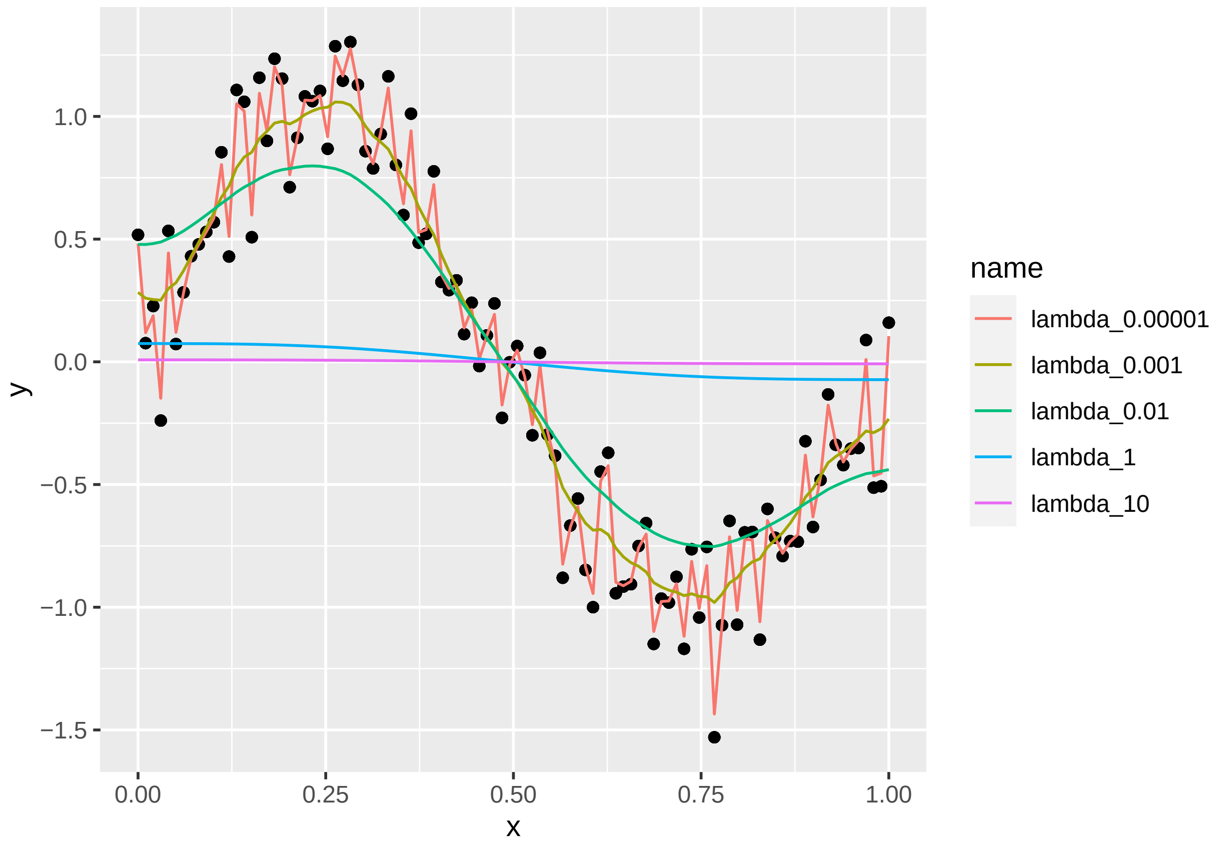
$$y_i = sin(2\pi x_i) + \epsilon_i$$

where

$$\epsilon_i \overset{iid}{\sim} N(0,1)$$

and the x_i are $n = 100$ evenly spaced points in $[0, 1]$. We have, for $\lambda = 1e-5, 1e-3, 1e-2, 1, 10$:

```
set.seed(205)
## make data
x = seq(0, 1, length.out=100)
y = rnorm(length(x), sin(x * (2 * pi)), .25)[order(x)]
x = sort(x)


fit_l0.00001 = relu_kernel_regressor(x, y, 0.00001)
fit_l0.001 = relu_kernel_regressor(x, y, 0.001)
fit_l0.01 = relu_kernel_regressor(x, y, 0.01)
fit_l1 = relu_kernel_regressor(x, y, 1.0)
fit_l10 = relu_kernel_regressor(x, y, 10.0)

plot_df = data.frame(
  x=x,
  y=y,
  lambda_0.00001=fit_l0.00001$yhat,
  lambda_0.001=fit_l0.001$yhat,
  lambda_0.01=fit_l0.01$yhat,
  lambda_1=fit_l1$yhat,
  lambda_10=fit_l10$yhat
) %>%
  tidyr::pivot_longer(
    .,
    cols=paste0(
      "lambda_",
      c("0.00001", "0.001", "0.01", "1", "10")
    )
)

ggplot(plot_df, aes(x=x, y=y))+
  geom_point() +
  geom_path(
    aes(x=x, y=value, color=name)
  )
```



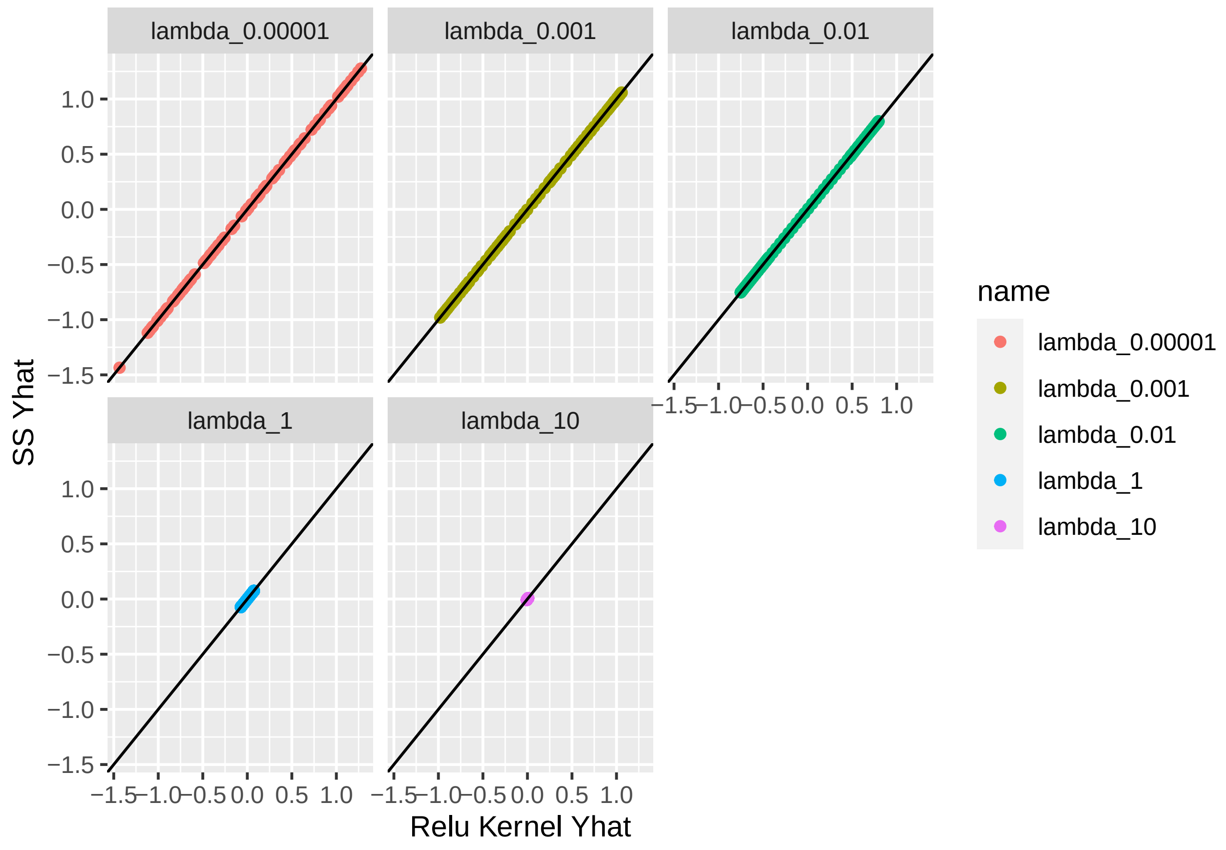
As expected the $\lambda = 1e-5$ induces a near linear interpolation, while the $\lambda = 1, 10$ practically zero out the coefficients and “flatten” the regression. Intuitively, the $||Y - \Phi\theta||_2^2$ wants to linearly interpolate as much as possible and the $||M\theta||_2^2$ term wants to flatten as much as possible. Hence, when λ is tiny, linear interpolation dominates, while when λ is larger, flattening dominates – both of which we see above.

(5)

When `m=1`, the `ss()` function just amounts to a linear interpolation, which as set forth in 1A is the same as our regressor. Hence, when λ is set the same, `ss(m=1)` becomes a linear interpolation (evenly-spaced feature assumption holds, by construction of x) under some level of penalization – which is exactly the penalized relu kernel regressor set forth above. *So the two should be the same.*

As evidence in support of this, consider comparison of in-sample $\hat{y}$ predictions under both of the methods.

```r
ss_l0.00001 = npreg::ss(x, y, m=1, all.knots=T, lambda=0.00001)
ss_l0.001 = npreg::ss(x, y, m=1, all.knots=T, lambda=0.001)
ss_l0.01 = npreg::ss(x, y, m=1, all.knots=T, lambda=0.01)
ss_l1 = npreg::ss(x, y, m=1, all.knots=T, lambda=1.0)
ss_l10 = npreg::ss(x, y, m=1, all.knots=T, lambda=10.0)

spline_df = data.frame(
  x=x,
  y=y,
  lambda_0.00001=ss_l0.00001$y,
  lambda_0.001=ss_l0.001$y,
  lambda_0.01=ss_l0.01$y,
  lambda_1=ss_l1$y,
  lambda_10=ss_l10$y
) %>%
  tidyr::pivot_longer(
    .,
    cols=paste0(
      "lambda_",
      c("0.00001", "0.001", "0.01", "1", "10")
    )
) %>%
  inner_join(
    .,
    plot_df, by=c("x", "y", "name"),
    suffix=c("_relu", "_ss")
)

ggplot(spline_df, aes(x=value_relu, y=value_ss, color=name)) +
  geom_point() +
  facet_wrap(~name) +
  labs(x="Relu Kernel Yhat", y="SS Yhat") +
  geom_abline(slope = 1)
```



Indeed, we see equality between the $\hat{y}$'s under the two methods, as expected

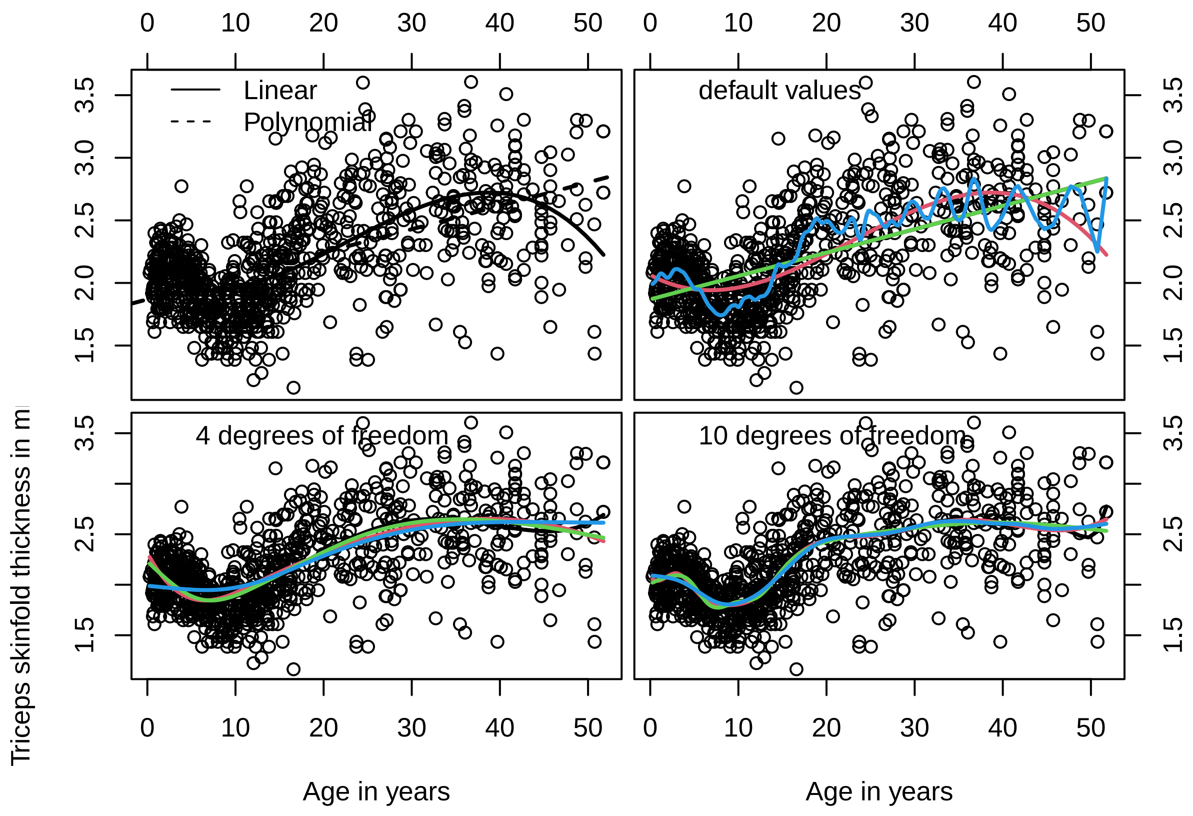
2.)

a.)

First, we reproduce the plots, as provided in the paper Appendix.

```
triceps = read.csv("~/Downloads/triceps/triceps.csv")
x = triceps$age;
y = triceps$lntriceps[order(x)]
x = sort(x)

## function to define spline basis
pbase <- function(x, p) {
  u <- (x - min(x)) / (max(x) - min(x))
  u <- 2 * (u - 0.5)
  P <- outer(u, seq(0, p, by = 1), "^")
  P
}

### setup
vdist <- hdist <- 0.2
layout( matrix(1:4, 2, 2, byrow=TRUE), widths=c(10,10), heights=c(10, 10))
par(mar= c(vdist, 4, 3, hdist))
```

```r
### linear + labels
plot(x, y, ylab="", xlab="Age in years", axes=FALSE)
axis(2); axis(3); box()
abline(lm(y~x), lwd=2, lty=2)
U <- pbase(x,3)
lines(x, U %*% coef(lm(y~U-1)), lwd=2)
legend(0.05, 3.8, c("Linear", "Polynomial"), col=1, lty=1:2, bty="n")
par(mar= c(vdist, hdist, 3, 4))
plot(x, y, ylab="Triceps skinfold thickness in mm (log scale)", xlab="Age in years", axes=FALSE)
axis(3); axis(4); box()

## fit models
fit.poly <- lm(y ~ poly(x))
fit.bs <- lm(y ~ bs(x) )
fit.ns <- lm(y ~ ns(x) )
fit.sp <- smooth.spline(y ~ x)

## add fit lines to the plot
lines(x, predict(fit.poly, data.frame(x=x)), col=1, lwd=2)
lines(x, predict(fit.bs, data.frame(x=x)), col=2, lwd=2)
lines(x, predict(fit.ns, data.frame(x=x)), col=3, lwd=2)
lines(fit.sp, col=4, lwd=2)
legend(0.05, 3.8, "default values", col=1, bty="n")
par(mar= c(5, 4, vdist, hdist))
plot(x, y, ylab="Triceps skinfold thickness in mm (log)", xlab="Age in years", axes=FALSE)
axis(1); axis(2); box()

## fit models
fit.poly.4 <- lm(y~ poly(x,4))
fit.bs.4 <- lm(y~ bs(x, df=4) )
fit.ns.4 <- lm(y~ ns(x, df=4) )
fit.sp <- smooth.spline(y~ x, df=4)

## add fit lines to the plot
lines(x, predict(fit.poly.4, data.frame(x=x)), col=1, lwd=2)
lines(x, predict(fit.bs.4, data.frame(x=x)), col=2, lwd=2)
lines(x, predict(fit.ns.4, data.frame(x=x)), col=3, lwd=2)
lines(fit.sp, col=4, lwd=2)
legend(0.05, 3.8, "4 degrees of freedom", col=1, bty="n")
par(mar= c(5, hdist, vdist, 4))
plot(x, y, ylab="Triceps skinfold thickness in mm (log)", xlab="Age in years", axes=FALSE)
axis(1); axis(4); box()

## fit models
fit.poly.10 <- lm(y~ poly(x,10))
fit.bs.10 <- lm(y~ bs(x, df=10) )
fit.ns.10 <- lm(y~ ns(x, df=10) )
fit.sp <- smooth.spline(y ~ x, df=10)

## add fit lines to the plot
lines(x, predict(fit.poly.10, data.frame(x=x)), col=1, lwd=2)
lines(x, predict(fit.bs.10, data.frame(x=x)), col=2, lwd=2)
lines(x, predict(fit.ns.10, data.frame(x=x)), col=3, lwd=2)
lines(fit.sp, col=4,lwd=2)
legend(0.05, 3.8, "10 degrees of freedom", col=1, bty="n")
```



```
layout(matrix(1, 1, 1))
```

b.)

Then, we make functions to tune the various basis regressions. These are configured below:

```
loo.cv.df <- function(x, y, basis.fn, df, fit.fn=lm){
  #' Note that x and y should be sorted so that x is monotonically increasing
  N = length(x)
  # LOO except for endpoints, so we don't have to predict beyond boundary knots
  sapply(2:(N-1), function(i){
    fit.df = data.frame(
      x = x[(1:N) != i],
      y = y[(1:N) != i]
    )
    i
    ### Case 1: bs, ns, ss, poly
    if(basis.fn %in% c("bs", "ns", "poly")){
      args = ifelse(basis.fn == "poly", "degree=", "df=")
      formula.str = paste0(
        "y ~ ",
        basis.fn,
        "(x, ",
        args,
        df,
        ")"
      )
      fit = fit.fn(
        as.formula(formula.str),
        fit.df
      )

      yhat = (predict(fit, data.frame(x=x[i])))
    }else{
      fit = smooth.spline(fit.df$x, fit.df$y, df=df)
      yhat = predict(fit, data.frame(x=x[i]))$y
      yhat = as.numeric(yhat)
    }

    (y[i] - yhat)^2
  }) %>%
    mean() -> mse.loo
  mse.loo
}

tune.smoother <- function(x, y, basis.fn, df.grid, fit.fn=lm){
  # tune
  sapply(df.grid, function(df){
    loo.cv.df(x, y, basis.fn, df)
  }) -> cv.result
  # print winner
  best.idx = which.min(cv.result)
  df.best = df.grid[best.idx]
  cat("-------------------------------------\n",
      "Tuning results\n  Basis: ", basis.fn,
      "\n  Best DF: ", df.best,
      "\n  LOO MSE: ", cv.result[best.idx], "\n",
      "-------------------------------------\n")

  fit.df = data.frame(x=x, y=y)

  ### Case 1: bs, ns, ss, poly
  if(basis.fn %in% c("bs", "ns", "poly")){
    args = ifelse(basis.fn == "poly", "degree=", "df=")
    formula.str = paste0(
      "y ~ ",
      basis.fn,
      "(x, ",
      args,
      df.best,
      ")"
    )
    fit = fit.fn(
      as.formula(formula.str),
      fit.df
    )
  }else{
    fit = smooth.spline(fit.df$x, fit.df$y, df=df.best)
  }
  return(fit)
}
```

We then tune each of the functions LOO-style, and plot the results (going back to `ggplot2`, for familiarity, at this point). Hyperparam configurations are printed as part of the script.

```
df_seq = c(4, 6, 8, 10, 12, 20)
fit.poly.tune = tune.smoother(x, y, "poly", df_seq)
```

```
## -------------------------------------
##  Tuning results
##   Basis:  poly
##   Best DF:  10
##   LOO MSE:  0.09977551
##  -------------------------------------
```

```
fit.bs.tune = tune.smoother(x, y, "bs", df_seq)
```

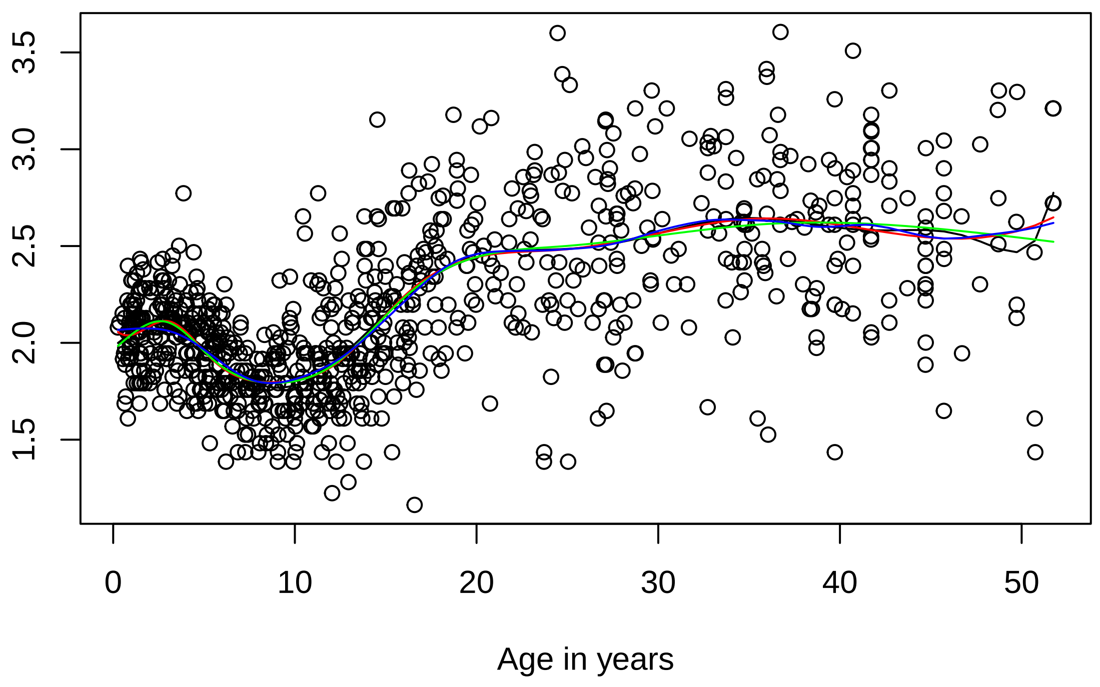
```
## -------------------------------------
##  Tuning results
##   Basis:  bs
##   Best DF:  10
##   LOO MSE:  0.09925858
##  -------------------------------------
```

```
fit.ns.tune = tune.smoother(x, y, "ns", df_seq)
```

```
## -------------------------------------
##  Tuning results
##   Basis:  ns
##   Best DF:  8
##   LOO MSE:  0.0989379
##  -------------------------------------
```

```
fit.sp.tune = tune.smoother(x, y, "smooth.spline", df_seq)
```

```
## -------------------------------------
##  Tuning results
##   Basis:  smooth.spline
##   Best DF:  12
##   LOO MSE:  0.09975969
##  -------------------------------------
## add fit lines to the plot
plot(x, y, ylab="", xlab="Age in years", axes=T)
lines(x, predict(fit.poly.tune, data.frame(x=x)), lty=1)
lines(x, predict(fit.bs.tune, data.frame(x=x)), col="red", lty=1)
lines(x, predict(fit.ns.tune, data.frame(x=x)), col="green", lty=1)
lines(x, predict(fit.sp.tune, x)$y, col="blue", lty=1)
```


Age in years