CS 181 Spring 2024 Section 1: Regression

1 Non-Parametric Approaches: KNN and Kernelized Regression

This section is on regression. In terms of the cube, that means that we are working with labeled data, and that those labels are continuous: for each data point $\mathbf{x} \in \mathbb{R}^D$, there is a corresponding $y \in \mathbb{R}$. Regression models predict that y using $\mathbf{x}$.

Regression models can be either parametric or non-parametric. In this context, a model is *non-parametric* if it makes no assumptions about the structure underlying the data, and *parametric* if it does. Probabilistic models are a subset of parametric models.

We will first go over two forms of non-parametric regression. They do not assume that y does not have some underlying distribution that depends on $\mathbf{x}$, and do not enforce a linear relationship between $\mathbf{x}$ and y.

1.1 K-Nearest Neighbors (KNN)

1. KNN is considered a form of non-parametric regression: for a fixed value of $K = k$ that you choose, there are no other parameters that the model learns.

2. Rundown of the KNN Algorithm:

   (a) Let $\mathbf{x}^*$ be the point that we would like to make a prediction about. Let's find the k nearest points $\{\mathbf{x}_1, \dots \mathbf{x}_k\}$ to $\mathbf{x}^*$, based on some predetermined distance function.

   (b) Denote the true y values of these k points as $\{y_1, \dots y_k\}$.

   (c) Output our prediction $\hat{y}^*$ for our point of interest $\mathbf{x}^*$:

   $$\hat{y}^* = \frac{1}{k} \sum_{i=1}^{k} y_i$$

   The little "hat" in $\hat{y}^*$ indicates that this is our *prediction*, rather than the true y value.

1.2 Kernelized Regression

1. Kernelized Regression is considered to be a smoother, more general extension of KNN. You might come across this concept under the name kernel-weighted average regression. Define $k(\mathbf{x}^*, \mathbf{x}_n)$ as our "kernel function." In Kernelized Regression, we want to take a *weighted average* of all the points in our training data when outputting our prediction for an unknown point. Intuitively, we want to weigh points that are "closer" to our unknown point of interest *more heavily* than points that are "farther" away. As such, our kernel function $k(\mathbf{x}^*, \mathbf{x}_n)$ should be *larger* for a point $\mathbf{x}_n$ closer to our point of interest $\mathbf{x}^*$ than a point $\mathbf{x}_n$ farther away.

2. Importantly, the value of $\mathbf{x}$ that results in the largest value of $k(\mathbf{x}^*, \mathbf{x})$ should be $\mathbf{x}^*$ itself:

$$\arg\max_{\mathbf{x}} k(\mathbf{x}^*, \mathbf{x}) = \mathbf{x}^*$$

3. Rundown of the Kernelized Algorithm:

   (a) Let $\{\mathbf{x}_1, \ldots \mathbf{x}_N\}$ (and their corresponding y values) be *all* of the N points comprising our training data set.

   (b) Let $\mathbf{x}^*$ be our point of interest that we want to make a prediction for. We make our prediction as follows:

$$\hat{y}^* = \frac{\sum_{i=1}^{N} k(\mathbf{x}^*, \mathbf{x}_i) \cdot y_i}{\sum_{j=1}^{N} k(\mathbf{x}^*, \mathbf{x}_j)}$$

   The denominator term normalizes the sum of our weights to equal 1. Compared to the KNN algorithm, the kernelized regression uses all the points in the dataset to predict rather than just the k nearest.

1.3 Concept Questions

Say that you are modeling a problem with a 1-dimensional $x \in \mathbb{R}$, and $\{x_1, \ldots, x_n\}$ are all in the interval $[0, 1]$.

1. Assuming $n \leq k$, What happens to the predictions of the KNN regression as x^* increases from 1 to ∞?

2. Assuming that $\frac{\partial K}{\partial |x^* - x_i|} < 0$ and $\lim_{|x^* - x_i| \to \infty} K(x^*, x_i) = 1$ (the kernel function is strictly decreasing with the distance between x^* and x_i and has a limit at 1), what happens to the predictions of the Kernelized regression as x^* increases from 1 to ∞?

2 Ordinary Least Squares: Three Approaches

Despite its simplicity, linear regression is a widely used and flexible regression formula. It also provides a nice introduction to a variety of methods we will use later in the course.

2.1 Conventions and Matrix Derivatives

In class, we briefly discussed how to find the weights that minimize the least squares loss function. Before we derive the end result in more detail, there are a few important notes notes on the conventions we'll use in this class, and the corresponding matrix derivatives.

1. In this class, if $y \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^D$, then

$$\frac{dy}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_D} \end{bmatrix}$$

   Note that this derivative is a *column* vector!

2. If $\mathbf{y} \in \mathbb{R}^N$ and $\mathbf{x} \in \mathbb{R}^D$, then

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_D} & \frac{\partial y_2}{\partial x_D} & \cdots & \frac{\partial y_N}{\partial x_D} \end{bmatrix}$$

   Note that this derivative is basically like stacking the $\frac{dy_i}{d\mathbf{x}}$ column vectors together like "books on a bookshelf!"

3. By convention, we will treat our vector of y-values, $\mathbf{y}$, as a *column* vector, and we usually write that column vector as $\mathbf{y}$. We will also treat our weight vector, $\mathbf{w}$, as a *column* vector. Finally, we will treat each *individual* data point $\mathbf{x}_i$ as a column vector.

4. *However*, we will treat our data matrix $\mathbf{X}$ (of all the individual data points $\mathbf{x_i}$ combined together), which is also called a "design matrix," as follows. This is not a typo:

$$\mathbf{X} = \begin{bmatrix} \leftarrow \mathbf{x_1}^\top \rightarrow \\ \leftarrow \mathbf{x_2}^\top \rightarrow \\ \vdots \\ \leftarrow \mathbf{x_N}^\top \rightarrow \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1D} \\ x_{21} & \cdots & x_{2D} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{ND} \end{bmatrix}.$$

   D is the number of dimensions in our data (i.e., the dimensions of each data point $\mathbf{x_i}$), and N is the number of datapoints. If we have N observations with labels, $\mathbf{y}$ will be a N by 1 column vector and $\mathbf{X}$ will be a N by D matrix.

2.2 Regression as Loss Minimization

Say we want to model y with a linear combination of the input $\mathbf{x}$. Our regression function is then

$$h(\mathbf{x}; \mathbf{w}) = w_1 x_1 + w_2 x_2 + \ldots + w_D x_D = \sum_{d=1}^{D} w_d x_d = \mathbf{w}^\top \mathbf{x} \tag{1}$$

where $x_j \in \mathbb{R}$ for $j \in \{1, \ldots, D\}$ are the features and $\mathbf{w} = \{w_1, w_2, \ldots, w_D\} \in \mathbb{R}^D$ is the weight parameter. (We will deal with intercepts later).

In order to solve for $\mathbf{w}$, we need some loss function that depends on $\mathbf{w}$. As the name suggests, the least squares loss function depends on the squared distance between the prediction and the true label:

$$\mathcal{L}(\mathbf{w}; \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}, \{\mathbf{y_1}, \ldots, \mathbf{y_N}\}) = \frac{1}{2} \sum_{n=1}^{N} \left(y_n - \mathbf{w}^\top \mathbf{x}_n \right)^2 \tag{2}$$

The presence of the $\frac{1}{2}$ is just to make things neater when we take the derivative. A few remarks on this loss function:

1. Since the values of $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and $\{y_1, \ldots, y_N\}$ are fixed in the optimization problem (we can't change the values of the data), we'll often write the loss function as $\mathcal{L}(\mathbf{w})$ to be concise.

2. $\hat{y_n} = \mathbf{w}^\top \mathbf{x}_n$ is our predicted y value for each $\mathbf{x}_n$, under linear regression. Thus, our loss function could also be written as the following:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y_n - \hat{y_n})^2$$

3. The loss function itself outputs a *scalar* value! The total loss is *always* a scalar real value, and *not* a vector!

We want to find the value of $\mathbf{w}$ that minimizes the loss function, which can be defined as

$$\mathbf{w}^*{}_{\text{OLS}} = \arg\min_{\mathbf{w}} \left(\frac{1}{2} \sum_{n=1}^{N} \left(y_n - \mathbf{w}^\top \mathbf{x}_n \right)^2 \right) \tag{3}$$

where $\mathbf{w}^*{}_{\text{OLS}}$ is the value of $\mathbf{w}$ which minimizes ordinary least squares loss.

To minimize this, we take the derivative of the loss function with respect to $\mathbf{w}$ and set the derivative equal to 0 like we would with any optimization problem. By the power, sum (across the summation), and chain rules, we have the following:

$$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_{n=1}^{N} \left((y_n - \mathbf{w}^\top \mathbf{x}_n) \cdot \frac{d(y_n - \mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} \right) = 0$$

Note that the $\frac{1}{2}$ has cancelled with the 2 that the power rule produces. Let's look at that inner derivative a bit more closely. This is a good example of how to think through a matrix derivative. By rules from single variable calculus, we have:

$$\frac{d(y_n - \mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} = \frac{d(-\mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} = -\frac{d(\mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}}$$

Note that $\mathbf{w}^\top \mathbf{x}_n$ is a scalar function, while $\mathbf{w}$ is a vector. Thus, our resultant derivative should be a column vector. We can expand the product:

$$\mathbf{w}^\top \mathbf{x}_n = w_1 x_{n1} + w_2 x_{n2} + \cdots + w_D x_{nD}$$

For any arbitrary element of $\mathbf{w}$, which we'll call w_i, we have, via single variable calculus:

$$\frac{d(\mathbf{w}^\top \mathbf{x}_n)}{dw_i} = x_{ni}$$

From our notational conventions previously combined with our observation directly above,

$$\frac{d(\mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} = \begin{bmatrix} \frac{\partial(\mathbf{w}^\top \mathbf{x}_n)}{\partial w_1} \\ \frac{\partial(\mathbf{w}^\top \mathbf{x}_n)}{\partial w_2} \\ \vdots \\ \frac{\partial(\mathbf{w}^\top \mathbf{x}_n)}{\partial w_D} \end{bmatrix} = \begin{bmatrix} x_{n1} \\ x_{n2} \\ \vdots \\ x_{nD} \end{bmatrix} = \mathbf{x}_n$$

Now, going back to our derivative of the loss function with respect to $\mathbf{w}$, we have:

$$\begin{aligned}\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_{n=1}^{N}\left((y_n - \mathbf{w}^\top\mathbf{x}_n)\cdot\frac{d(y_n - \mathbf{w}^\top\mathbf{x}_n)}{d\mathbf{w}}\right) &= \sum_{n=1}^{N}\left((y_n - \mathbf{w}^\top\mathbf{x}_n)\cdot -\frac{d(\mathbf{w}^\top\mathbf{x}_n)}{d\mathbf{w}}\right)\\ = \sum_{n=1}^{N}\left((y_n - \mathbf{w}^\top\mathbf{x}_n)\cdot -\mathbf{x}_n\right) &= \sum_{n=1}^{N}\left(-y_n\mathbf{x}_n + (\mathbf{w}^\top\mathbf{x}_n)\mathbf{x}_n\right) = 0\end{aligned}$$

We can move the negative terms to the right hand side:

$$\sum_{n=1}^{N}(\mathbf{w}^\top\mathbf{x}_n)\mathbf{x}_n = \sum_{n=1}^{N} y_n\mathbf{x}_n \tag{4}$$

Note that $\mathbf{w}^\top\mathbf{x}_n$ is a *scalar*, and that $\mathbf{w}^\top\mathbf{x}_n = \mathbf{x}_n^\top\mathbf{w}$. This is important because we can left-multiply or right-multiply by scalars however we want:

$$\sum_{n=1}^{N}(\mathbf{w}^\top\mathbf{x}_n)\mathbf{x}_n = \sum_{n=1}^{N}(\mathbf{x}_n^\top\mathbf{w})\mathbf{x}_n = \sum_{n=1}^{N}\mathbf{x}_n(\mathbf{x}_n^\top\mathbf{w})$$

Since $\mathbf{w}$ does not depend on the value of the index n, we can move it outside the sum:

$$\sum_{n=1}^{N}\mathbf{x}_n(\mathbf{x}_n^\top\mathbf{w}) = \left(\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^\top\right)\mathbf{w}$$

Recall the design matrix X above. If we fully expand the sum, we will actually find that $\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^\top = \mathbf{X}^\top\mathbf{X}$. Similarly, if we fully expand the right side, we will find that $\sum_{n=1}^{N} y_n\mathbf{x}_n = \mathbf{X}^\top\mathbf{y}$. We can then substitute these expressions into equation (4) above to produce

$$\left(\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^\top\right)\mathbf{w} = \sum_{n=1}^{N}\mathbf{x}_n y_n \iff \left(\mathbf{X}^\top\mathbf{X}\right)\mathbf{w} = \mathbf{X}^\top\mathbf{y}$$

Assuming that $\mathbf{X}^\top\mathbf{X}$ is invertible, we can isolate $\mathbf{w}$:

$$\mathbf{w} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

In short, if we minimize our least squares loss function with respect to the weights, we get the following solution:

$$\mathbf{w}^*_{OLS} = \arg\min_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} \tag{5}$$

where $\mathbf{X} \in \mathbb{R}^{N\times D}$. Each row represents one data point and each column represents values of one feature across all the data points. In practice, gradient descent is often used to compute w^*. Today, we just got super lucky that there was a clean-cut closed-form analytical solution. [1]

[1]Note: $(\mathbf{X}^\top\mathbf{X})^{-1}$ is invertible iff X is full column rank (i.e. rank D, which implies $N \geq D$). **What if $(\mathbf{X}^\top\mathbf{X})^{-1}$ is not invertible?** Then, there is not a unique solution for $\mathbf{w}^*$. If $d > N$, computing the pseudoinverse of $\mathbf{X}^\top\mathbf{X}$ will find one solution.

2.2.1 Concept Question

How is a model (such as linear regression) related to a loss function (such as least squares)?

2.2.2 Exercise: Modifying the Loss Function

The OLS loss function can be modified to accomplish different objectives. One common modification is regularization: imposing a penalty on larger weights in order to prevent overfitting. One form of regularization, called Ridge regularization, adds a $\frac{1}{2}||\mathbf{w}||_2^2$ term to the loss function.

Find the weight vector which solves least squares with Ridge regularization described as

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left[\frac{1}{2} \sum_{i=1}^{N} \left(\mathbf{x}_i^\top \mathbf{w} - y_i \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{D} w_j^2 \right]$$

2.3 Regression as Likelihood Maximization

Here's a quick overview of general MLE:

- Given a model and observed data, the **maximum likelihood estimate** (of the parameters) is the estimate that maximizes the probability DENSITY of seeing the observed data under the model.
- It is obtained by maximizing the **likelihood function**, which is the same as the joint PDF of the data, but viewed as a function of the parameters rather than the data.
- Since log is monotonic function, we will often maximize the **log likelihood** rather than the likelihood as it is easier (turns products from independent data into sums) and results in the same solution.

In the context of linear regression, say we assume that y is described by a linear function of $\mathbf{x}$ and a random error term, the error is normal and iid, and we know that the covariance matrix between the error terms $\mathbf{\Sigma} = I_D$. (Note: whenever a covariance matrix is diagonal, it means $\text{Cov}(\epsilon_i, \epsilon_j) = 0$ for all $i \neq j$, which is saying all ϵ_i are independent of the others / the ϵ_i values are iid.). Specifically:

$$y_i = \mathbf{x}_i^\top \mathbf{w} + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, 1)$$

This is equivalent to the expression $\epsilon_i = y_i - \mathbf{x}_i^\top \mathbf{w}$, so the likelihood we are maximizing is then

$$L(\mathbf{w}) = \prod_{n=1}^{N} \frac{1}{(2\pi)^{1/2}} \exp\left(-\frac{1}{2}(y_n - \mathbf{x}_n^\top \mathbf{w})^\top (y_n - \mathbf{x}_n^\top \mathbf{w})\right)$$

We want to maximize this in order to find the MLE, but differentiating under a product is hard. Fortunately, taking the log turns this product into a sum and separates the terms inside of the product:

$$\ell(\mathbf{w}) = \log L(\mathbf{w}) = n \log\left(\frac{1}{(2\pi)^{1/2}}\right) - \frac{1}{2}\sum_{n=1}^{N}\left((y_n - \mathbf{x}_n^\top \mathbf{w})^\top (y_n - \mathbf{x}_n^\top \mathbf{w})\right)$$

Now, notice that the second term is the loss function from above multiplied by -1! We can substitute it in:

$$\ell(\mathbf{w}) = n \log\left(\frac{1}{(2\pi)^{1/2}}\right) - \mathcal{L}(\mathbf{w})$$

And since the first term on the right does not depend on $\mathbf{w}$, differentiating produces

$$\frac{\partial \ell}{\partial \mathbf{w}} = -\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$

So maximizing $\ell(\mathbf{w})$ by setting $\frac{\partial \ell}{\partial \mathbf{w}} = 0$ is solved by the same $\mathbf{w}$ that minimizes the least squares loss function above. A few comments:

1. In general, you won't have the solution for part of the MLE already, so you will have to do all the matrix algebra we did above to solve these problems.

2. We maximized the log likelihood above. Since likelihoods as a product of probabilities are always between 0 and 1, the log likelihood will always be negative. It's common to minimize the negative log likelihood $(-1 \cdot \ell(\mathbf{w}))$ rather than maximize the log likelihood - it's mathematically equivalent and makes it more consistent with loss minimization problems. From a practical standpoint, many optimizers are designed to minimize functions, so when there isn't an analytic solution it's better to be doing gradient descent than ascent.

2.3.1 Exercise: Bayesian Regression

Say that you have some prior knowledge on the weight vector $\mathbf{w}$. Specifically, your prior is that

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_D)$$

Which implies that the prior probability of a specific weight vector is (from the multivariate normal pdf)

$$P(\mathbf{w}) = \frac{\exp\left(\frac{-1}{2\sigma^2}\mathbf{w}^\top \mathbf{w}\right)}{\sqrt{(2\pi)^D |\sigma^2 I_D|}}$$

If as above $L(\mathbf{w}; \mathbf{X}, \mathbf{y})$ is the likelihood of the data given a weight vector, recall that the posterior distribution on $\mathbf{w}$ is defined using Bayes' law as

$$P(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{L(\mathbf{w}; \mathbf{X}, \mathbf{y}) P(\mathbf{w})}{P(\mathbf{X})}$$

Solve for this this MAP (Maximum A Posteriori) estimator. How does the result relate to the previous exercise, both mathematically and conceptually?

2.4 Regression as Projection

The Projection Theorem tells us that the smallest vector between a point and plane is orthogonal to that plane. (If you want to know why, you can prove it using the norm and inner product properties in the math review section from last week). We can leverage this to solve least squares geometrically. Since $\mathbf{y}$ is $N \times 1$, we can treat it as a vector in N-dimensional space. $\mathbf{X}$ is $N \times D$, and so can be considered as D column vectors of size N. If we assume that $N \geq D$ (the same condition for invertibility that we used above), then the span of these vectors will be a subset of the vector space.

$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$ is a $N \times 1$ vector that is in the span of the $\mathbf{X}$ vectors. We want to find the $\mathbf{w}$ that minimizes $\mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\mathbf{w}$, and from the Projection Theorem we know that for the minimizing $\mathbf{w}$, $\mathbf{y} - \mathbf{X}\mathbf{w} \perp \mathbf{X}$. This implies that when we project $\mathbf{y} - \mathbf{X}\mathbf{w}$ into $\mathbf{X}$, we will have

$$\mathbf{0} = \mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) = \mathbf{X}^\top \mathbf{y} - \mathbf{X}^\top \mathbf{X}\mathbf{w}$$

Rearranging terms:

$$\mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top \mathbf{X}\mathbf{w} \iff \mathbf{w}^* = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

so the geometric setup produces the same optimal weights as both the loss and likelihood functions above!

In most of the models we will work with in this course, there will not be a geometric way to set up the problem. However, in this case, setting up the regression as a projection problem finds the optimal weights with much less effort than the other approaches. If possible, working with matrices instead of individual points will often simplify both algebra and code (and significantly speed up that code as well).

3 Extending OLS: Change of Basis

3.1 Takeaways

We allow $h(\mathbf{x}; \mathbf{w})$ to be a non-linear function of the input vector $\mathbf{x} \in \mathbb{R}^D$, while remaining linear in $\mathbf{w} \in \mathbb{R}^M$ by using a basis function $\phi : \mathbb{R}^D \to \mathbb{R}^M$. The resulting basis regression model is below:

$$h(\mathbf{x}; \mathbf{w}) = \sum_{m=1}^{M} w_m \phi_m((\mathbf{x})) = \mathbf{w}^\top \phi(\mathbf{x})$$

To merge the bias term, we can define $\phi_1(\mathbf{x}) = 1$. Some examples of basis functions include polynomial $\phi_m(x) = x^m$, Fourier $\phi_m(x) = \cos(m\pi x)$, and Gaussian $\phi_m(x) = \exp\{-\frac{(x-\mu_m)^2}{2s^2}\}$.

Basis transformations let us model non-linear relationships in the data. Say the true data generating relationship is $f(x) = 1 + x^2$: using the basis transformation $\phi(x) = [1\ x\ x^2]^\top$ would let us perfectly model the function with $\mathbf{w} = [1\ 0\ 1]^\top$ while a linear regression would model the relationship very poorly. This becomes even more important when you have periodic data: if x is something like months since 2010 and y is snowfall, a linear regression without a change of basis won't be able capture the true periodic relationship but a Fourier basis can. If the data has some intercept ($f(x) \neq 0$), you need a basis transformation unless some dimension of $\mathbf{x}$ is constant across all $\mathbf{x}$.

Thinking geometrically, if $r(\mathbf{x})$ is the true regression function we are approximating, so $y_i = r(\mathbf{x}_i)$ for all i, we can think about OLS on untransformed data as the squared-error minimizing projection of r into the space of linear functions spanned by $\mathbf{X}$. When we use a basis transformation, we are instead projecting r into the space of functions spanned by $\phi(\mathbf{X})$. Equivalently, $\phi(\mathbf{X})$ is the new basis of the space of functions we are projecting into.

3.2 Concept Questions

1. What are some advantages and disadvantages to using linear basis function regression to basic linear regression?
2. How do we choose the bases?

4 Exercise: OLS on Augmented Data

Let $\mathbf{X} \in \mathbb{R}^{n \times D}$ be our design matrix and $\mathbf{y}$ be our vector of n target values. Assume $\mathbf{X}$ and $\mathbf{y}$ are both centered, that is assume the mean of each row is 0. Let $\tilde{\mathbf{X}}$ be the $(n+D)$ by D matrix formed by vertically stacking $\mathbf{X}$ on top of aI_D, and let $\tilde{\mathbf{y}}$ be the $(n+D)$-length vector formed by vertically stacking $\mathbf{y}$ on top of a vector of D zeros (0_D).

That is, let $\tilde{\mathbf{X}} = \begin{bmatrix} X_{11} & \cdots & X_{1m} \\ \vdots & \ddots & \vdots \\ X_{n1} & \cdots & X_{nm} \\ a & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & a \end{bmatrix}$ and $\tilde{\mathbf{y}} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \\ 0 \\ \vdots \\ 0 \end{bmatrix}$.

(a) Assuming that $\mathbf{y} \sim \mathcal{N}(\mathbf{X}\mathbf{w}, I_D)$, treat $\mathbf{X}$ as fixed and $\mathbf{w}^*$ as a random variable. On the augmenting data alone **(using aI_D as $\mathbf{X}$ and 0_D as $\mathbf{y}$)** calculate the MLE of $\mathbf{w}^*$, then the expectation and variance of that estimate.

(b) Write the least squares loss function induced by $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{y}}$ in terms of $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{w}$, and a.

(c) Find for the $\mathbf{w}^*$ that minimizes this loss function. *Hint: this should require no algebra.*

(d) Why is this cool? If we assume that $\mathbf{w}$ is normal, how does part (a) relate to the Bayesian regression exercise? State the formal relationship between λ in Ridge regularization, σ^2 in Bayesian regression, and a^2 here.