CS 181 Spring 2025 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 $\{\mathbf{x}_1, \dots \mathbf{x}_N\}$ (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 $\{x_1, \dots, x_n\}$ are all in the interval $[0, 1]$.

1. Assuming $n \geq k$, 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.

Reference: 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.

2.1 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. If we wanted, we could add a $1/N$ or a 25 or whatever we want. Note that when we do optimization, a multiplicative constant or an additive constant does not affect where the point of minimization is. 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.

Now let's take a deeper dive into the expression. We can decompose the expression into intuitive parts.

$$\underbrace{\mathbf{w}^*}_{A} = \underbrace{(\mathbf{X}^\top \mathbf{X})^{-1}}_{B} \underbrace{\mathbf{X}^\top \mathbf{y}}_{C}$$

We will start off with expression C which roughly corresponds to the correlation between the features and the target. One way to think of is, is to consider two random variables X, Y and note that $\mathrm{Cov}(X, Y) = E[XY] - E[X]E[Y]$. So, in this case $E[XY]$ can be approximated by $\sum_{i=1}^n x_i y_i$, which roughly corresponds to $\mathbf{X}^\top \mathbf{y}$. This is indeed this expected value when it is centered, i.e., you you define $\mathbf{X}^* = \mathbf{X} - \mathbf{M}$ where $\mathbf{M}$ is

$$\mathbf{M} = \begin{bmatrix} \mu_1 & \mu_2 & \cdots & \mu_d \\ \mu_1 & \mu_2 & \cdots & \mu_d \\ \vdots & \vdots & \ddots & \vdots \\ \mu_1 & \mu_2 & \cdots & \mu_d \end{bmatrix}$$

and μ_j is the mean of the column j of $\mathbf{X}$. However, you do not need to know this now. It will become necessary when we talk about PCA Regression in the far future.

Expression B corresponds to the covariance of the features. We note that $\mathbf{X}^\top \mathbf{X}$ is the *covariance matrix* of the feature matrix. This can be linked to how $\mathrm{Var}(X) = \mathrm{Cov}(X, X)$. Note that we cannot square, non-square matrices, but you can imagine it as being $E[X^2]$ and thus estimated as $\sum_{i=1}^n x_i^2$.

Expression A corresponds to the features weights, and the equation is telling us how the feature weights are related to the correlation with the target. Specifically we have the following insights:

- If there is a **highly predictive feature**, i.e., if $(\mathbf{X}^\top \mathbf{y})_i$ is large for some i, then we would **expect the corresponding weight to be large**, i.e., w_i^* to be large.
- This interpretation is mostly meaningful when features are approximately orthogonal. If you have highly correlated features, you could have significance transferred from one to the other.
- This shared significance and unstable behavior could lead the contrapositive of the above statement to not be true. Ways to mitigate this is to explore regularization techniques (e.g., Ridge, Lasso) which stabilize weights when features are correlated.

2.1.1 Concept Question

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

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

(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 loss-minimizing $\mathbf{w}^*$.

(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.

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 $\boldsymbol{\Sigma} = I_D$. (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 this class, you will see a lot of problems where you specify a likelihood and take the log as the first step, then optimize the negative log likelihood. You should become comfortable with this process, and expect to see problems on exams that give a data-generating process incorporating random variables and ask you to find up the likelihood and optimize it.

2. 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.

3. We maximized the log likelihood above. 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.

2.4 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 inevitability 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 the column span of $\mathbf{X}$, 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!

It is important emphasize why we have $\hat{\mathbf{y}}$ be in the column space of $\mathbf{X}$. If you think about it, we *should* be predicting $\hat{\mathbf{y}}$ as a function of $\mathbf{X}$. Specifically, in the linear regression case, we are predicting as a linear function of $\mathbf{X}$.

We will define $\mathbf{P} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top$. We can show that $\hat{\mathbf{y}}$ is a projection of $\mathbf{y}$. We note that

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

We see that $\mathbf{P}$ is acting like "hat operator" it takes in $\mathbf{y}$ and outputs $\hat{\mathbf{y}}$. We say that $\mathbf{P}$ is an *orthogonal projection* onto the column space of $\mathbf{X}$ which says that 1) $\mathbf{y}$ is in the column space of $\mathbf{X}$ and 2) $\mathbf{y} - \hat{\mathbf{y}}$ is orthogonal to the column space of $\mathbf{X}$. We will be checking that indeed $\mathbf{P}$ is an orthogonal projection in the homework. We can also take a deeper dive to explain each part of the expression of the fitted values:

$$\hat{\mathbf{y}} = \underbrace{\mathbf{X}}_{A}\underbrace{(\mathbf{X}^\top \mathbf{X})^{-1}}_{B}\underbrace{\mathbf{X}^\top \mathbf{y}}_{C}$$

Starting from the back, expression C again measures how correlated the features are to the response. This is discussed above in the statistical interpretation setting. Expression B dictates normalizing for the covariance between the features. For example, say that the variance of x_3 is very high, then small changes in it would affect the fitted values by a lot. However, since we are inverting the covariance, we canceled that out in a sense. Now expression A rebuilds the vector in the column space using these corrected coefficients.

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 Failures of Linear Regression: Residual Diagnostics

Linear regression is one of the most widely used modeling tools due to tits simplicity and interpretability. Given a design matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ and response vector $\mathbf{y} \in \mathbb{R}^N$. As we discussed above, the residual vector, $r = \mathbf{y} - \hat{\mathbf{y}}$, captures everything model fails to explain. We will see that residuals help us identify inconsistencies with the model assumptions and when indeed linear regression is a good fit for our data.

3.1 Typical Failures

3.1.1 Missing/Bad Features

A frequent failure occurs when the true relationship between inputs and outputs is nonlinear, while the model is linear in the original failures. Suppose the data-generating process satisfies

$$y = w_0 + w_1 x + w_2 x^2 + \epsilon$$

but the model only includes the intercept and linear term. We would see that x^2 would lie *outside* of $C(\mathbf{X})$ and therefore can not be capture by our projection P. In this example, you would realize this when observing the *residual plot*, plotting your residuals with the linear feature (x_i, r_i), and looking at the distribution. We would see a geometric parabola form long the residuals.

3.1.2 Highly Correlated Features

Linear regression can also fail in the presence of strongly correlated features, as we alluded to above. When columns of $\mathbf{X}$ are nearly linearly dependent, the matrix $\mathbf{X}^\top \mathbf{X}$ becomes ill-conditioned, and the projection $\mathbf{P}$ becomes sensitive to small perturbations in the data. Although the fitted values $\hat{\mathbf{y}}$ may remain close to $\mathbf{y}$, the coefficient vector $\mathbf{w}^*$, can vary dramatically in response to noise. In this case we could lose a lot of feature interpretability as well as increase our out-of-sample error.

3.1.3 Outliers

Outlier also introduce another geometric failure model. Observations with unusually large leverage can disproportionately influence the orientation of the projection subspace. Even when such points are few, their distance in feature space can cause the fitted model to rotate in order to reduce their squared residual. Note that this is due to the L_2 norm nature of this approach, and this would be become more dramatic in an L_3 approach and less dramatic in an L_1 approach. This rotation will decrease the fit quality on the rest of the data. You can recognize these points on a residual plot if they have extremely higher magnitude than the rest of the points.

3.1.4 Other Assumption Misalignments

We could finally have other miscellaneous violations through heteroskedastic noise and clustering. Heteroskedastic noise can be caused when the noise, ϵ, is not independent of the point, we could have that the error is a function of the feature, $\epsilon(\mathbf{x})$. This can be fixed if you have some sort of idea of what the relationship between the error and the features are, but otherwise you would need to lay strong assumptions. This can be seen on a residual plot by recognizing areas with high residual variance vs. areas of low residual variance. Clustering can also be identified by finding clusters where residuals seem to be similar.

3.2 Diagnostics and Fixes

Note that what makes linear regression, *linear regression*, is the following: you can regress *linear* coefficients. For example, linear regression cannot solve for β such that

$$y = \beta_1 e^{\beta_0 x}$$

similarly, you cannot find β such that $y = \beta_0\beta_1 + \beta_1 x + \beta_0 x^2$. You must have a *linear combination* of the coefficients. Note that we can model $y = \beta_0 x^{100} e^x \sin(x^2 \cdot \tan(x))$, since β_0 is the linear coefficients of some $x_1 = x^{100} e^x \sin(x^2 \cdot \tan(x))$. This is an important point that you should understand.

Typically, the universal diagnostic for linear regression is observing the residual plot (if you haven't already noticed). By viewing the plot you can see if the residual roughly looks like independent, Gaussian, homoskedastic, noise. If there is reason to believe that it is not. You would maybe employ one or more of these solutions:

1. One option is to **change the loss function**. Ordinary least squares minimizes the L_2 loss, which places disproportionate weight on large residuals. As a result outliers can strongly influence the fitted model. Robust alternatives such L_1 regression (or LASSO) or Huber loss reduce sensitivity to extreme errors, leading to more stable fits when outliers are present.

2. Another approach is to **change the feature representation**. Structured residual patterns, such as curvature, indicate that important components of $\mathbf{y}$ lie outside the column space $C(\mathbf{X})$. Adding non-linear terms— through basis expansion, polynomial terms, or splines— grows the subspace and allows the projection to capture previously missing structure.

3. Now if you tried the above two methods and we still seem to not have a great model fix, it may be necessary to **change the model class entirely**. Sometimes even the top two fixes do not work as it is like putting lipstick on a pig. Linear models do not govern everything so a lot of the time we need to switch the approach to others.

We will discuss a couple of these methods in the weeks to come in lecture.

4 Extending OLS: Change of Basis

4.1 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 $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$. 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.

4.2 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?

4.3 Exercise: Reasoning About Column Spaces

In this part, we will apply the projection interpretation of OLS to reason about feature transformations. For reference, for the value of

1. Assume that $\mathbf{X}$ is a $n \times k$ matrix with rank k so that each column is linearly independent of the others. Suppose that you replace $\mathbf{X}$ with $\mathbf{X}'$, where $\mathbf{X}'$ has an additional column that is a linear combination of the other columns. In terms of train MSE, will linear regression using $\mathbf{X}'$ instead of $\mathbf{X}$ do better, worse, or the same? Does the answer change if you add multiple columns that are linear combinations of the others?

2. Now, suppose that you replace $\mathbf{X}$ with $\tilde{\mathbf{X}}$, where each column of $\tilde{\mathbf{X}}$ equal to the corresponding column of $\mathbf{X}$ with the linear regression of that column on the others subtracted from it. How will a linear regression using $\tilde{\mathbf{X}}$ compare to one using $\mathbf{X}$ in terms of train MSE?

3. Does the answer to the previous part change if you also replace $\mathbf{y}$ with $\tilde{\mathbf{y}}$, where $\tilde{\mathbf{y}}$ is $\mathbf{y}$ residualized against all the columns of $\mathbf{X}$, similarly to $\tilde{\mathbf{X}}$?

4. Why do the previous results not apply to change of basis?

5 Exercise: Linear Regression with Fixed Effects

How can we extend regression if we know more about the data-generating process?

1. Some datasets measure a set of individuals at different points over time, with the individual indexed by i and time by t. Suppose we know that the true relationship between y and x is

   $$y_{it} = \mathbf{x}_{it}^\top \beta + \alpha_i + \varepsilon_{it}.$$

   where $\varepsilon_{it} \sim \mathcal{N}(0, \sigma^2)$, α_i is the constant fixed effect for individual i, and β is the coefficient for the feature vector $\mathbf{x}_{it}$. Additionally, assume that all measurements are taken simultaneously.

   Show that under these assumptions,

   $$y_{it} - E_i[y_{it}] = (\mathbf{x}_{it} - E_i[\mathbf{x}_{it}])^\top \beta + \epsilon_{it},$$

   where $E_i[y_{it}]$ is the average value of y_{it} across all t for individual i and $E_i[\mathbf{x}_{it}]$ is the average value of $\mathbf{x}_{it}$ across all t for individual i.

2. Now, suppose that there is a shared time effect γ_t. The data generating process is now

   $$y_{it} = \mathbf{x}_{it}^\top \beta + \alpha_i + \gamma_t + \varepsilon_{it}.$$

   Show that under these assumptions,

   $$y_{it} - E_i[y_{it}] - E_t[y_{it}] + E[y_{it}] = (\mathbf{x}_{it} - E_i[\mathbf{x}_{it}] - E_t[\mathbf{x}_{it}] + E[\mathbf{x}_{it}])^\top \beta + \varepsilon_{it}.$$

3. Define $\ddot{\mathbf{x}}_{it} = \mathbf{x}_{it} - E_i[\mathbf{x}_{it}] - E_t[\mathbf{x}_{it}] + E[\mathbf{x}_{it}]$, $\ddot{y}_{it} = y_{it} - E_i[y_{it}] - E_t[y_{it}] + E[y_{it}]$, and $\ddot{\mathbf{X}}$ and $\ddot{\mathbf{y}}$ as the matrix and vector formed by stacking the points. Explain how these quantities can be estimated from the data, then solve for β in terms of $\ddot{\mathbf{X}}$ and $\ddot{\mathbf{y}}$.

4. How could you allow for fixed effects by augmenting the data matrix $\mathbf{X}$? What are the disadvantages of this approach compared to the estimator in part (c)?

5. Why is it that on the training set, the MSE of the model with fixed effects is always less than or equal to the MSE of the model without fixed effects i.e. the model fit assuming $y_{it} = \mathbf{x}_{it}^\top \beta + \varepsilon_{it}$?

6. None of the previous derivations require ε_{it} to be normally distributed. What weaker condition would have been sufficient?