

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 02 | Part 1

Introduction

News

- ▶ Lab 01 released. Due Sunday @ 11:59 pm.
- ▶ HW 01 released. Due Wednesday @ 11:59 pm.
 - ▶ \LaTeX template available (optional).
- ▶ Math self-check released.

Last Time

- ▶ We saw **nearest neighbor** predictors.
- ▶ They can work well.
- ▶ But they **memorize** the training data rather than **learning** a simpler underlying pattern.

The Main Problem

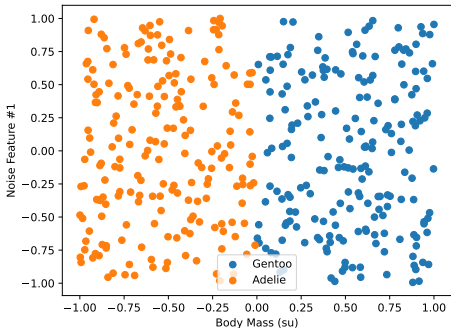
- ▶ Nearest neighbor approaches **do not learn** which features are **useful** and which **are not**.

Example

- ▶ Suppose all Adelie penguins weigh less than all Gentoo penguins.
- ▶ I.e., we can **predict perfectly** based on body mass alone.

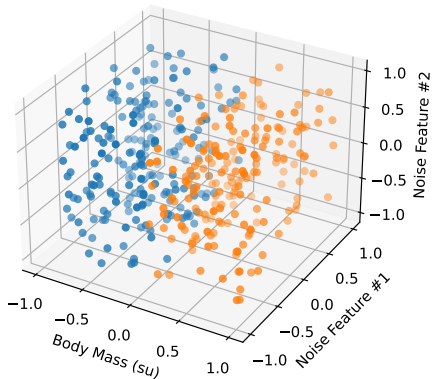
Example: One Noisy Feature

- ▶ Suppose we add a feature that is total noise.
- ▶ Still enough information to perfectly classify.
- ▶ 1-NN: 98% test accuracy.



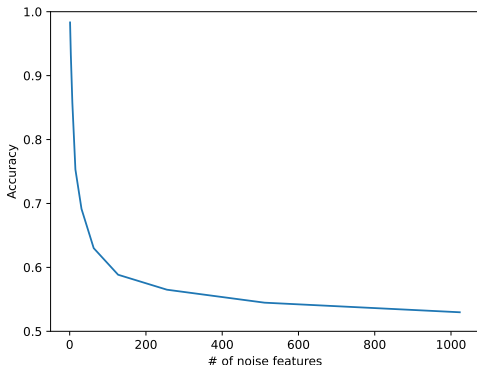
Example: Two Noisy Features

- ▶ Suppose we add another feature that is total noise.
- ▶ *Still* enough information to perfectly classify.
- ▶ 1-NN: 95% test accuracy (**-3%**).



Example: Noisy Features

- ▶ No matter how many noisy features we add, there is enough information to classify perfectly.
- ▶ But 1-NN performance **degrades** with # of (noisy) features:



Explanation

- ▶ Euclidean distance treats all features the same.
 - ▶ Even those that are pure noise.
- ▶ NN does not **learn** which features are useful.¹
- ▶ Distance becomes less meaningful as *noisy* features are added.

¹For extensions of kNN which learn a distance metric from data, see: (Weinberger and Saul, 2009; Goldberger et al., 2005; Shalev-Shwartz et al., 2004)

The Rest of DSC 140A

- ▶ We'll explore three different paradigms for learning from data.
 - ▶ **Part 1:** Empirical Risk Minimization
 - ▶ **Part 2:** Probabilistic Modeling
 - ▶ **Part 3:** Tree-based Methods

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 02 | Part 2

Empirical Risk Minimization

Prediction

- ▶ Prediction is the most common task in ML:
 - ▶ given: a **feature vector**, \vec{x}
 - ▶ predict: an **output target**, y .
- ▶ Example:
 - ▶ given: years of experience and college GPA
 - ▶ predict: salary

Prediction Functions

- ▶ **Informally:** we think experience, GPA, etc., are predictive of salary.
- ▶ **Formally:** we think there is a function H that takes in (experience, GPA) and outputs a good prediction of the salary.

$$H(\text{experience, GPA}) \rightarrow \text{predicted salary}$$

Prediction Functions

- ▶ In general, a **prediction function**² H takes in a feature vector and outputs a predicted label.

$$H(\vec{x}) \rightarrow y$$

²Sometimes called a **hypothesis** function.

Example Prediction Function

$$\begin{aligned} H(\text{experience, GPA}) = & \$50,000 \\ & + \$10,000 \times \text{experience} \\ & + \$5,000 \times \text{GPA} \end{aligned}$$

Goal

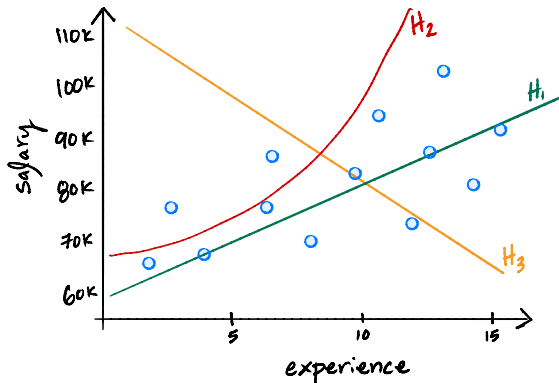
- ▶ There are many possible prediction functions.
- ▶ How do we pick a good one?
- ▶ One that **works well on unseen, future data.**
- ▶ **Problem:** we **don't know** the future.

Data

- ▶ **Assumption:** the future will be like the past.
- ▶ So a prediction function that works well on past data will likely work well on future data.
- ▶ **Idea:** can use past data to “measure” how a good prediction function is, select between them.

Example

- ▶ $H_1(x) = 60,000 + 10,000x$
- ▶ $H_2(x) = 70,000 + 200x^2$
- ▶ $H_3(x) = 110,000 - 15,000x$

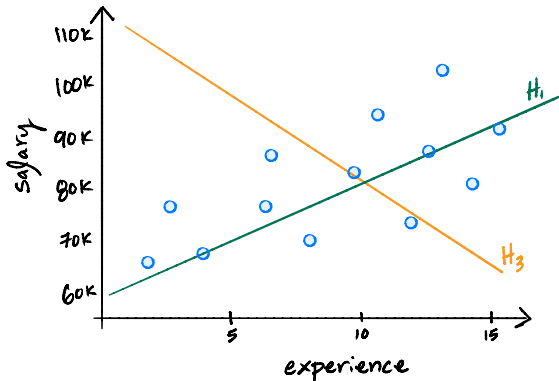


Fit

- ▶ We preferred H_1 over H_2 and H_3 because it “fit” the data better.
- ▶ How do we formally **quantify** how well a prediction function fits the data?

Measuring Errors

- **Idea:** measure the difference between the prediction and the correct label.



Loss Functions

- ▶ A **loss function** measures the difference between a prediction $H(\vec{x}^{(i)})$ and the “right answer” y_i .
- ▶ There are many different loss functions. For now, we'll consider two.
- ▶ **Absolute loss:** $\ell_{\text{abs}}(H(\vec{x}^{(i)}), y_i) = |H(\vec{x}^{(i)}) - y_i|$
- ▶ **Square loss:** $\ell_{\text{sq}}(H(\vec{x}^{(i)}), y_i) = (H(\vec{x}^{(i)}) - y_i)^2$

Quantifying Overall Fit

- ▶ A loss function measures the difference between a prediction and the correct label for a **single training point**.
- ▶ A good prediction function should make good predictions *on average* over the entire training set.
- ▶ That is, for a good H , the **average loss** should be small.

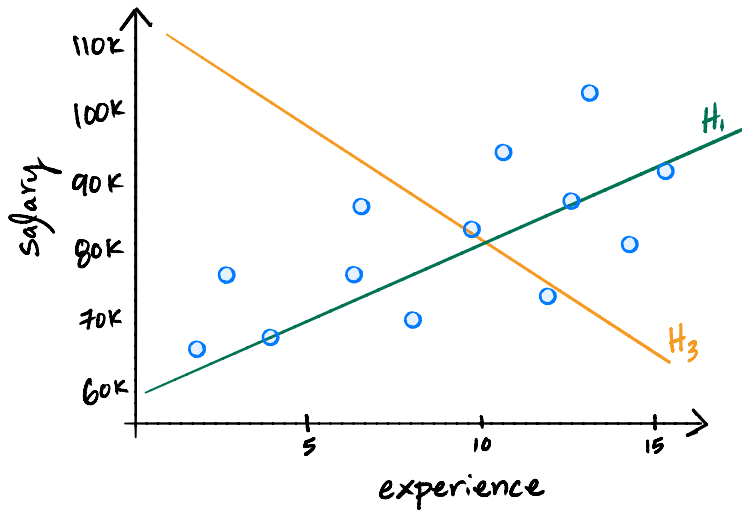
Empirical Risk

- ▶ The average loss on the training set, also called the **empirical risk**, is defined to be:

$$R(H) = \frac{1}{n} \sum_{i=1}^n \ell(H(\vec{x}^{(i)}), y_i)$$

- ▶ It is a function of H , but it also depends on:
 - ▶ The training data, $\mathcal{X} = (\vec{x}^{(1)}, y_1), \dots, (\vec{x}^{(n)}, y_n)$
 - ▶ The particular choice of loss function ℓ

Example



Terminology

- ▶ We might say: “the empirical risk with respect to absolute loss”. This means:

$$R(H) = \frac{1}{n} \sum_{i=1}^n |H(\vec{x}^{(i)}) - y_i|$$

- ▶ Or, “the empirical risk with respect to square loss”. This means:

$$R(H) = \frac{1}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}) - y_i)^2$$

Terminology

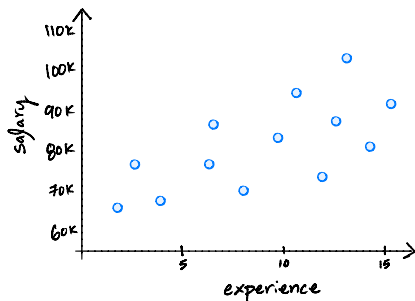
- ▶ We might be quick and say “risk” instead of “empirical risk”.

Minimizing Empirical Risk

- ▶ Empirical risk measures the “fit” of a prediction function to the training data.
- ▶ **Idea:** find a prediction function H that has the smallest empirical risk.

Exercise

Consider the data shown below, and assume absolute loss.



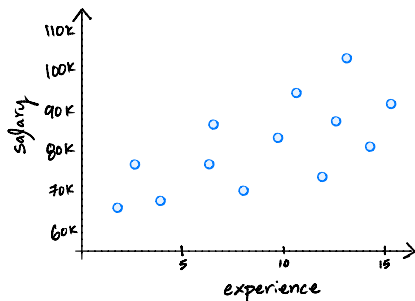
Sketch a prediction function H that minimizes the empirical risk.

Problem

- ▶ It is **too easy** to find a prediction function that has **zero** empirical risk.
 - ▶ Simply memorize the training data.
 - ▶ We want to learn a simpler pattern.
- ▶ Instead, we will restrict our search for prediction functions to a smaller set of (simple) functions.
- ▶ This set is called the **hypothesis class**.

Exercise

Consider the data shown below, and assume absolute loss.



Sketch a **linear** prediction function H that minimizes the empirical risk.

Empirical Risk Minimization

- ▶ The learning strategy we have just derived is called **empirical risk minimization (ERM)**.
- ▶ Step 1: choose a **hypothesis class**
 - ▶ for example, linear functions
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: find H minimizing **empirical risk**

ERM is a Recipe

- ▶ By choosing different hypothesis classes and losses, we derive different learning algorithms.
- ▶ Some choices for Step 1 & 2 make Step 3 easier or harder.
- ▶ We'll see different choices in the coming weeks.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 02 | Part 3

Linear Prediction Functions

A Simple Choice

- ▶ ERM asks us to choose a hypothesis class.
- ▶ Let's start with a simple one: **linear functions**.
- ▶ This choice will take us quite far.

Linear Functions

- ▶ A **linear prediction function** of one feature has the form:

$$H(x) = w_0 + w_1x$$

- ▶ In general, a linear prediction function of d features has the form:

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d$$

- ▶ w_0, w_1, \dots, w_d are the **parameters** or **weights**.

Interpreting Weights

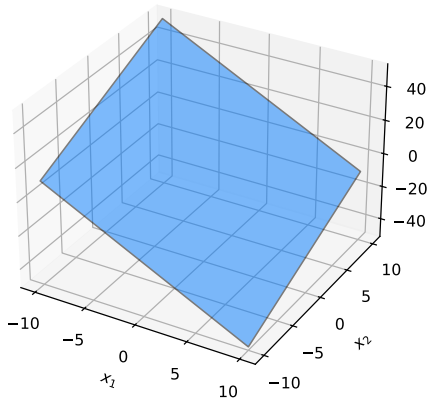
$$H(\vec{X}) = w_0 + w_1X_1 + w_2X_2 + \dots + w_dX_d$$

- ▶ w_0 (the **bias**) determines the prediction when all features are zero.
- ▶ w_1 determines how much the prediction changes when x_1 increases by one unit
- ▶ Similarly for w_2, \dots, w_d

Interpreting Weights

- ▶ When plotted, linear prediction functions are:
 - ▶ straight lines when $\vec{x} \in \mathbb{R}^1$
 - ▶ planes when $\vec{x} \in \mathbb{R}^2$
 - ▶ hyperplanes when $\vec{x} \in \mathbb{R}^d$
- ▶ w_i is the **slope** of the hyperplane in the x_i direction.

Example



$$w_0 = 1, \quad w_1 = -3, \quad w_2 = 2$$

$$H(\vec{x}) = 1 - 3x_1 + 2x_2$$

Parameter Vectors

- ▶ The parameters of a linear function can be packaged into a **parameter vector**, \vec{w} .
- ▶ **Example:** if $H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3$ then $\vec{w} = (w_0, \dots, w_3)^T$.
- ▶ If $\vec{x} \in \mathbb{R}^d$, then $\vec{w} \in \mathbb{R}^{d+1}$.

Parameterization

- ▶ A linear function $H(\vec{x})$ is **completely specified** by its parameter vector.
 - ▶ Can work either with the function, H , or vector, \vec{w} .
- ▶ Sometimes write $H(\vec{x}; \vec{w})$.
- ▶ Example: $\vec{w} = (8, 3, 1, 5, -2, -7)^T$ specifies

$$H(\vec{x}; \vec{w}) = 8 + 3x_1 + 1x_2 + 5x_3 - 2x_4 - 7x_5$$

Compact Form

- Recall the **dot product** of vectors \vec{a} and \vec{b} :

$$\vec{a} = (a_1, a_2, \dots, a_d)^T \quad \vec{b} = (b_1, b_2, \dots, b_d)^T$$

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + \dots + a_d b_d$$

- Observe:

$$\begin{aligned} H(\vec{X}; \vec{W}) &= w_0 + w_1 x_1 + \dots + w_d x_d \\ &= \underbrace{(w_0, w_1, \dots, w_d)^T}_{\vec{W}} \cdot \underbrace{(1, x_1, \dots, x_d)^T}_{?} \end{aligned}$$

Compact Form

- The **augmented feature vector** $\text{Aug}(\vec{x})$ is the vector obtained by adding a 1 to the front of \vec{x} :

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \quad \text{Aug}(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

- With augmentation, we can write:

$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d \\ &= \vec{w} \cdot \text{Aug}(\vec{x}) \end{aligned}$$

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 02 | Part 4

ERM for Linear Predictors

Empirical Risk Minimization

To create a new ML algorithm:

- ▶ Step 1: choose a **hypothesis class**
 - ▶ We've chosen linear functions
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: find H minimizing **empirical risk**

Loss Functions

- ▶ Next, we need to choose a loss function.
- ▶ Choice depends on the problem at hand.
- ▶ Let's focus on **regression** for now.
- ▶ The **absolute loss** is a natural first choice.

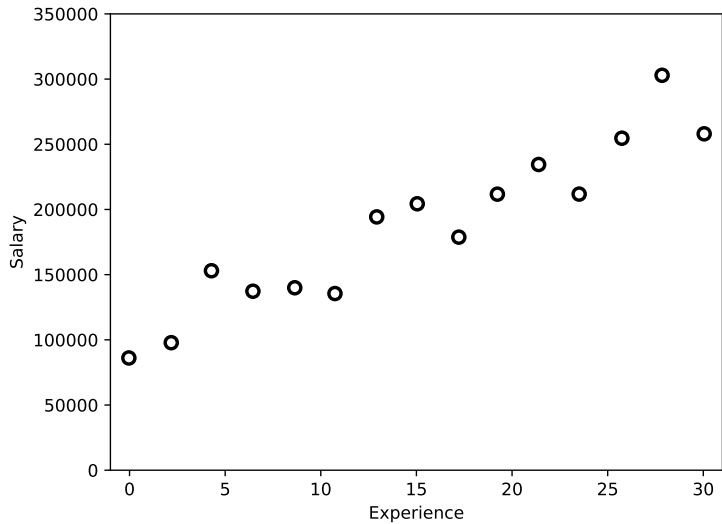
Empirical Risk w.r.t. Absolute Loss

- Now that we have assumed $H(\vec{x})$ is linear, we can write the empirical risk w.r.t. the absolute loss as:

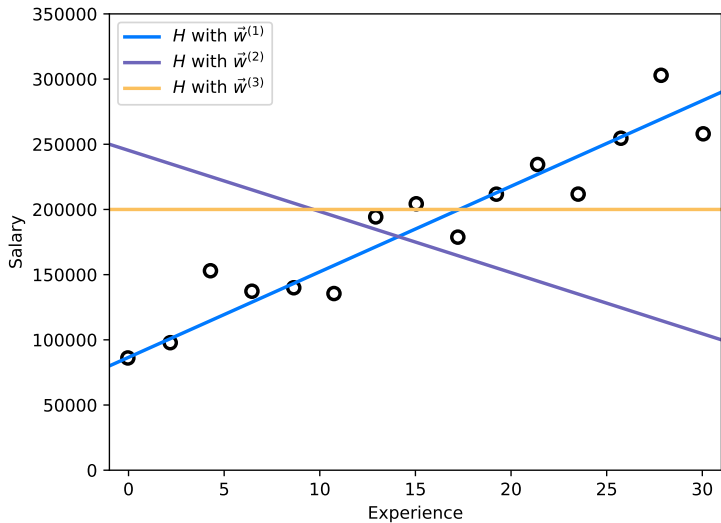
$$\begin{aligned} R_{\text{abs}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n |H(\vec{x}^{(i)}) - y_i| \\ &= \frac{1}{n} \sum_{i=1}^n |\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i| \end{aligned}$$

- A function of \vec{w} , since H is totally specified by \vec{w} .

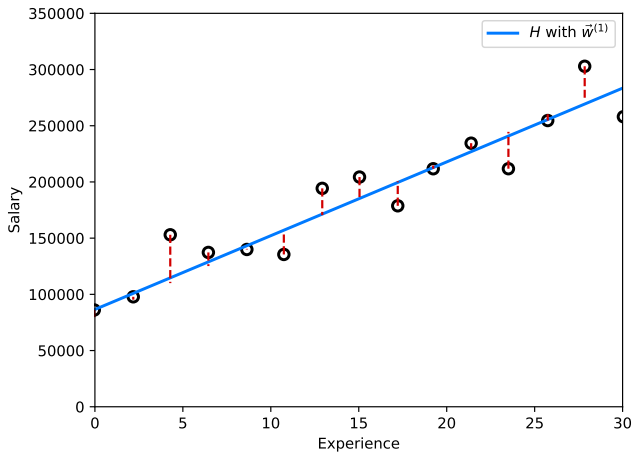
Example



Example

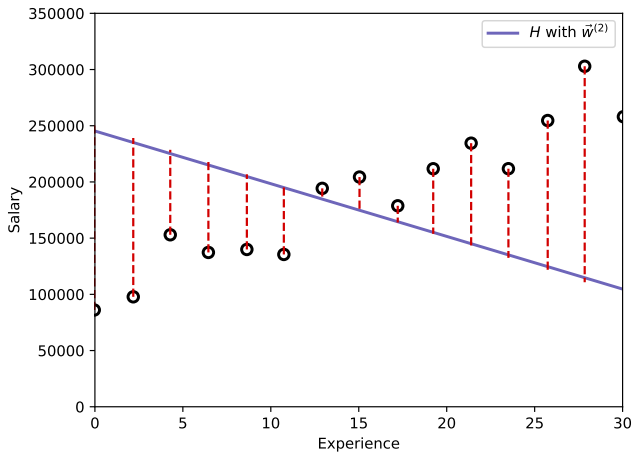


Example



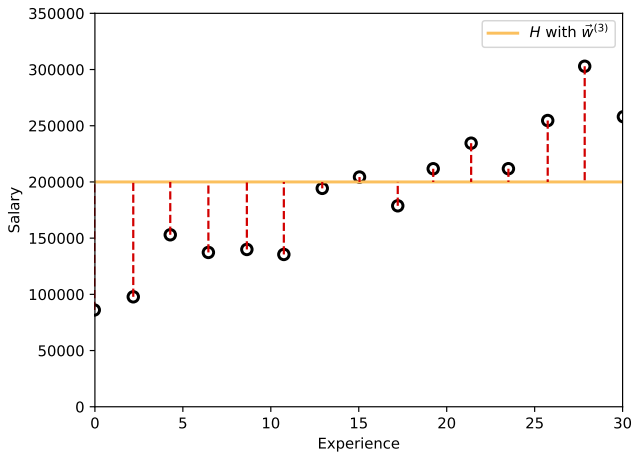
$$R_{\text{abs}}(\vec{w}^{(1)}) \approx 17,000$$

Example



$$R_{\text{abs}}(\vec{w}^{(2)}) \approx 90,000$$

Example



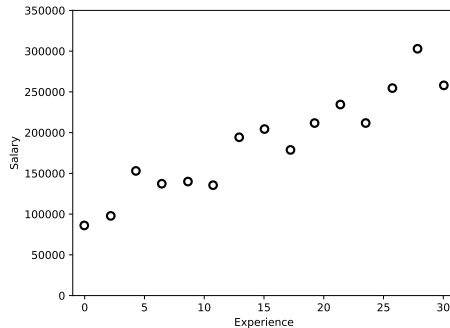
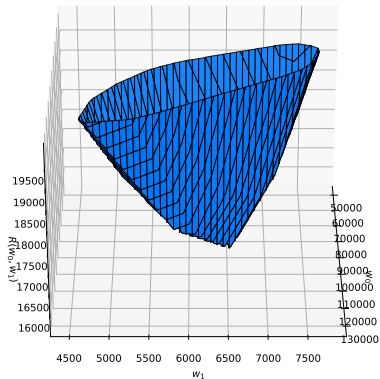
$$R_{\text{abs}}(\vec{w}^{(3)}) \approx 50,000$$

Risk Surface

- ▶ Can imagine plotting $R_{\text{abs}}(\vec{w})$ for all values of \vec{w} .
- ▶ This is called the **risk surface**.
- ▶ A \vec{w} that makes the surface lowest minimizes the empirical risk.

Risk Surface

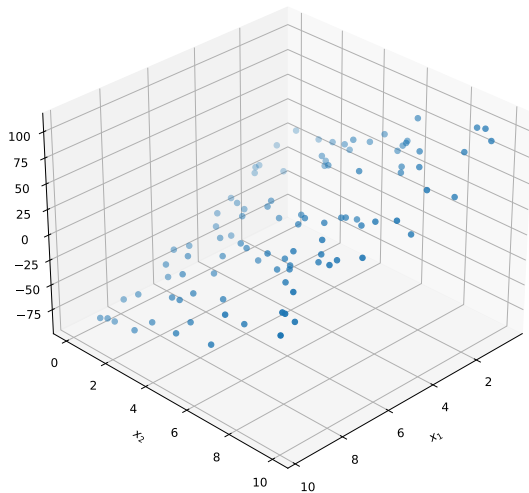
Plot of $R_{\text{abs}}(\vec{w})$



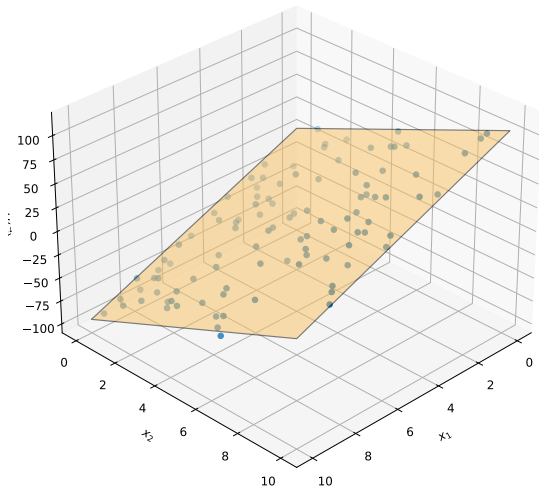
More Features

- ▶ With 2 features, we fit a plane instead of a line.
 - ▶ With ≥ 3 features, we fit a hyperplane.
- ▶ We can no longer easily visualize the risk surface.
- ▶ But the idea is the same: find the \vec{w} that minimizes the empirical risk.

Example

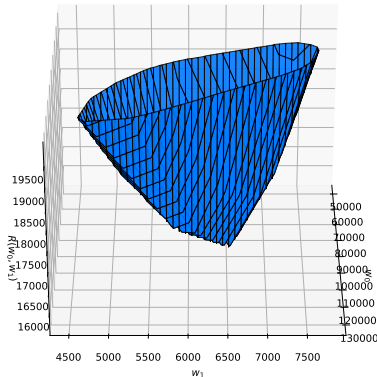


Example



Minimizing Empirical Risk

- How do we find the \vec{w} that minimizes $R_{\text{abs}}(\vec{w})$ (the empirical risk with respect to the absolute loss)?

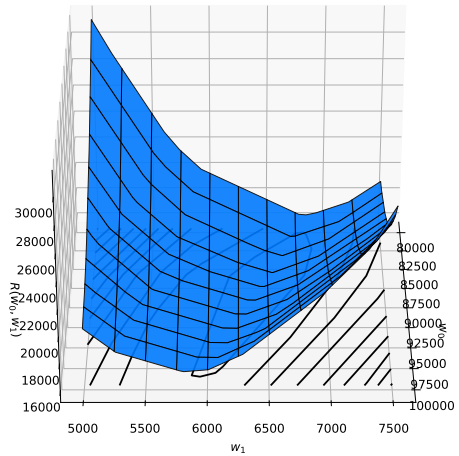


Calculus

- ▶ We know how to use calculus to find the minimum of a function:
 1. Find the **gradient** $\frac{d}{d\vec{w}} R_{\text{abs}}(\vec{w})$.
 2. Set it equal to zero, solve for \vec{w} .
 3. This finds places where $R_{\text{abs}}(\vec{w})$ is flat; check that it is a minimum (and not a maximum or saddle point).

Problem

- ▶ $R_{\text{abs}}(\vec{w})$ is **not differentiable**.
- ▶ There are places where the gradient (slope) is not defined.
- ▶ These appear as “cusps” or “sharp creases” in the risk surface.



Another Loss?

- ▶ We cannot use the usual calculus approach to minimize $R_{\text{abs}}(\vec{w})$.
 - ▶ We'll come back to this in a later lecture.
- ▶ Instead, let's see if the square loss is any better.

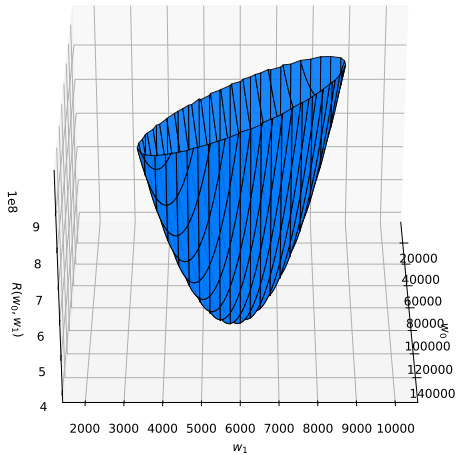
Empirical Risk w.r.t. Square Loss

- ▶ Assuming $H(\vec{x})$ is linear, we can write the empirical risk w.r.t. the square loss as:

$$\begin{aligned} R_{\text{sq}}(\vec{W}) &= \frac{1}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}) - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\vec{W} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i)^2 \end{aligned}$$

- ▶ $R_{\text{sq}}(\vec{W})$ is called the **mean squared error (MSE)**.

Risk Surface



Good News!

- ▶ The mean squared error is **differentiable**.
- ▶ Now, we'll try to find the \vec{w} that minimizes $R_{sq}(\vec{w})$ with calculus.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 02 | Part 5

Least Squares

Minimizing the MSE

- ▶ **Goal:** minimize $R_{sq}(\vec{w})$ with respect to \vec{w} .
- ▶ **Calculus Approach:** Find **gradient** of $R_{sq}(\vec{w})$; set to zero; solve for \vec{w} .
- ▶ We'll rely on results from vector calculus.

First Step: Rewrite Risk

- ▶ Step one: rewrite R_{sq} in vector form.
- ▶ We will find:

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \\ &= \frac{1}{n} \|X\vec{w} - \vec{y}\|^2 \end{aligned}$$

Recall

- If $\vec{u} = (u_1, u_2, \dots, u_k)^T$, then:

$$\|\vec{u}\|^2 = \vec{u} \cdot \vec{u} = \sum_{i=1}^k u_i^2$$

- So, if $\vec{a} = (a_1, \dots, a_k)^T$ and $\vec{b} = (b_1, \dots, b_k)^T$:

$$\begin{aligned}\|\vec{a} - \vec{b}\|^2 &= (\vec{a} - \vec{b}) \cdot (\vec{a} - \vec{b}) \\ &= \sum_{i=1}^k (a_i - b_i)^2\end{aligned}$$

First Step: Rewrite Risk

- ▶ Define $p_i = \text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}$, and let $\vec{p} = (p_1, \dots, p_n)^T$.
 - ▶ \vec{p} is a vector of the predictions on training set.
 - ▶ Note: $\vec{p} \in \mathbb{R}^n$, not \mathbb{R}^d !
- ▶ Then:

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2 \\ &= \frac{1}{n} \|\vec{p} - \vec{y}\|^2 \end{aligned}$$

First Step: Rewrite Risk

- Define the (augmented) **design matrix**, X :

$$X = \begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \longrightarrow & & \\ \text{Aug}(\vec{x}^{(2)}) \longrightarrow & & \\ \vdots & & \\ \text{Aug}(\vec{x}^{(n)}) \longrightarrow & & \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{pmatrix}$$

First Step: Rewrite Risk

- **Observe:** $\vec{p} = X\vec{w}$.

$$\underbrace{\begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) & \longrightarrow & \\ \text{Aug}(\vec{x}^{(2)}) & \longrightarrow & \\ \vdots & & \vdots \\ \text{Aug}(\vec{x}^{(n)}) & \longrightarrow & \end{pmatrix}}_X \underbrace{\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix}}_{\vec{w}} = \underbrace{\begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \cdot \vec{w} \\ \text{Aug}(\vec{x}^{(2)}) \cdot \vec{w} \\ \vdots \\ \text{Aug}(\vec{x}^{(n)}) \cdot \vec{w} \end{pmatrix}}_{\vec{p}}$$

First Step: Rewrite Risk

- Therefore, the MSE can be written:

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n (\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2 \\ &= \frac{1}{n} \|\vec{p} - \vec{y}\|^2 \\ &= \frac{1}{n} \|X\vec{w} - \vec{y}\|^2 \end{aligned}$$

Goal

- Find $\vec{w} \in \mathbb{R}^{d+1}$ minimizing:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \|X\vec{w} - \vec{y}\|^2$$

- **Step Two:** find gradient, set to zero, solve.

Step Two: Find Gradient

- We want to compute:

$$\frac{d}{d\vec{w}} [R_{sq}(\vec{w})] = \frac{d}{d\vec{w}} \left[\frac{1}{n} \|X\vec{w} - \vec{y}\|^2 \right]$$

- $\frac{dR_{sq}}{d\vec{w}}$ is the **gradient** of R_{sq} .

- It is the vector of partial derivatives:

$$\frac{dR_{sq}}{d\vec{w}} = \left(\frac{\partial R_{sq}}{\partial w_0}, \frac{\partial R_{sq}}{\partial w_1}, \dots, \frac{\partial R_{sq}}{\partial w_d} \right)^T$$

Good to know...

$$(A + B)^T = A^T + B^T$$

$$(AB)^T = B^T A^T$$

$$\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u} = \vec{u}^T \vec{v} = \vec{v}^T \vec{u}$$

$$(\vec{u} + \vec{v}) \cdot (\vec{w} + \vec{z}) = \vec{u} \cdot \vec{w} + \vec{u} \cdot \vec{z} + \vec{v} \cdot \vec{w} + \vec{v} \cdot \vec{z}$$

$$\|\vec{u}\|^2 = \vec{u} \cdot \vec{u}$$

Step Two: Find Gradient

- Expand norm to make gradient easier.

$$\|X\vec{w} - \vec{y}\|^2 =$$

=

=

=

Exercise

Consider:

$$\vec{w}^T X^T X \vec{w} - 2 \vec{y}^T X \vec{w} + \vec{y}^T \vec{y}$$

1. What type of object *should* it be?
 - Scalar, vector, or matrix?
2. What type of object *is* it?

Step Two: Find Gradient

$$\begin{aligned}\frac{d}{d\vec{w}} [R_{sq}(\vec{w})] &= \frac{1}{n} \frac{d}{d\vec{w}} [\vec{w}^T X^T X \vec{w} - 2\vec{y}^T X \vec{w} + \vec{y}^T \vec{y}] \\ &= ?\end{aligned}$$

Idea

- ▶ While we could compute each of: $\frac{\partial R_{sq}}{\partial w_0}$, $\frac{\partial R_{sq}}{\partial w_1}$,
- ▶ there's an easier way using matrix-vector calculus.

Exercise

If you had to guess, which of the following is equal to $\frac{d}{d\vec{w}} [\vec{w}^T X^T X \vec{w}]$?

1. X
2. \vec{w}
3. $2X^T X \vec{w}$
4. $2X \vec{w}$

Claims

► $\frac{d}{d\vec{w}} [\vec{w}^T X^T X \vec{w}] = 2X^T X \vec{w}$

► $\frac{d}{d\vec{w}} [\vec{y}^T X \vec{w}] = X^T \vec{y}$

► $\frac{d}{d\vec{w}} [\vec{y}^T \vec{y}] = 0$

How?

- ▶ General procedure: expand, differentiate, gather
 1. Expand $\vec{v}^T \vec{u}$ until coordinates u_1, \dots, u_k are visible.
 2. Compute $\partial d / \partial u_1, \partial d / \partial u_2, \dots, \partial d / \partial u_k$.
 3. Gather result in vector form.

Step Two: Find Gradient

- We claimed

$$\frac{d}{d\vec{w}} [\vec{w}^T X^T X \vec{w}] = 2X^T X \vec{w} \quad \frac{d}{d\vec{w}} [\vec{y}^T X \vec{w}] = X^T \vec{y} \quad \frac{d}{d\vec{w}} [\vec{y}^T \vec{y}] = 0$$

- So:

$$\begin{aligned} \frac{d}{d\vec{w}} [R_{\text{sq}}(\vec{w})] &= \frac{1}{n} \frac{d}{d\vec{w}} [\vec{w}^T X^T X \vec{w} - 2\vec{y}^T X \vec{w} + \vec{y}^T \vec{y}] \\ &= \end{aligned}$$

Solution

- We have found:

$$\frac{d}{d\vec{w}} [R_{sq}(\vec{w})] = \frac{1}{n} (2X^T X \vec{w} - 2X^T \vec{y})$$

- To minimize $R_{sq}(\vec{w})$, set gradient to zero, solve:

$$2X^T X \vec{w} - 2X^T \vec{y} = 0 \implies X^T X \vec{w} = X^T \vec{y}$$

- This is a system of equations in matrix form, called the **normal equations**.

The Normal Equations

- ▶ The least squares solutions for \vec{w} are found by solving the **normal equations**:

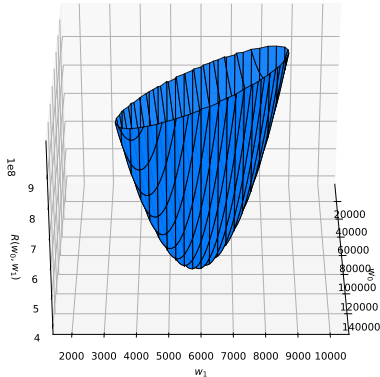
$$X^T X \vec{w} = X^T \vec{y}$$

- ▶ Mathematically, solved by:

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

A Direct Solution

- $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$ is exactly at the bottom of the risk surface.



Linear Least Squares Regression

► To train:

- Given a training set $\{(\vec{x}^{(1)}, y_1), \dots, (\vec{x}^{(n)}, y_n)\}$...
- 1. Construct $n \times (d + 1)$ augmented **design matrix**, X .
- 2. Solve the **normal equations**: $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$.

► To predict:

- Given a new \vec{x} , compute $H(\vec{x}) = \text{Aug}(\vec{x}) \cdot \vec{w}^*$.

Linear Least Squares Regression

The first algorithm we've derived from the ERM framework:

- ▶ Step 1: choose a **hypothesis class**
 - ▶ We've chosen linear functions
- ▶ Step 2: choose a **loss function**
 - ▶ We've chosen the square loss
- ▶ Step 3: find H minimizing **empirical risk**
 - ▶ We've found a direct solution: $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$

Compare to k -Nearest Neighbors

- ▶ **Then:** k -NN did not learn the relative importance of features.
- ▶ **Now:** Linear least squares learns a weight for *each* feature.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 02 | Part 6

From Theory to Practice

Implementation

- ▶ `sklearn.linear_model.LinearRegression`
- ▶ But linear least squares is very simple to implement in numpy:

```
> # training
> w = np.linalg.solve(X.T @ X, X.T @ y)
> # prediction on a new example, x
> # (you'll need to define augment)
> augment(x) @ w
```

Augmentation

- ▶ One easy way to implement augment:

```
def augment(x):  
    return np.array([1, *x])
```

- ▶ This code only works for a single example.
- ▶ To augment an array of examples, use `np.ones` and `np.column_stack`.

Don't Invert!

- ▶ Don't actually compute $(X^T X)^{-1}$.
- ▶ That is, avoid `np.linalg.inv`
- ▶ Inverting a matrix can be **slow** and **numerically unstable**.

Practical Issues

- ▶ You'll sometimes run into technical issues when using least squares.
- ▶ But we have the theoretical tools to understand and address them.

Issue: “Singular Matrix” Error

- ▶ You're training a regression model to predict house prices.
- ▶ Two of your features are 1) size in square feet and 2) size in square yards.

```
np.linalg.solve(X.T @ X, X.T @ y)
```

LinAlgError Traceback (most recent call last)

Cell In[69], line 1

----> 1 np.linalg.solve(X.T @ X, X.T @ y)

File /nix/store/jmldnh8fda65lps694qff2yn9c4fnawp-python3-3.11.8-env/lib/python3.11/site-packages/numpy/linalg/_linalg.py, line 407

```
407 signature = 'DD->D' if isComplexType(t) else 'dd->d'
408 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 409 r = gufunc(a, b, signature=signature, extobj=extobj)
411 return wrap(r.astype(result_t, copy=False))
```

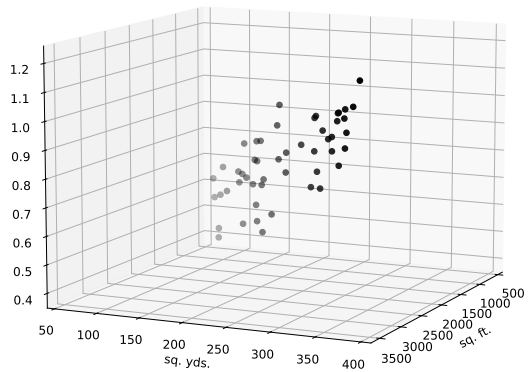
File /nix/store/jmldnh8fda65lps694qff2yn9c4fnawp-python3-3.11.8-env/lib/python3.11/site-packages/numpy/linalg/_linalg.py, line 111

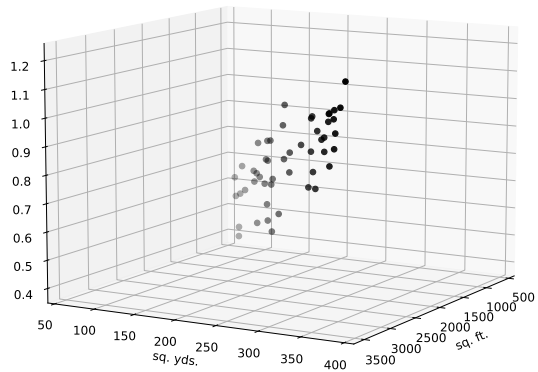
```
111 def _raise_linalgerror_singular(err, flag):
--> 112     raise LinAlgError("Singular matrix")
```

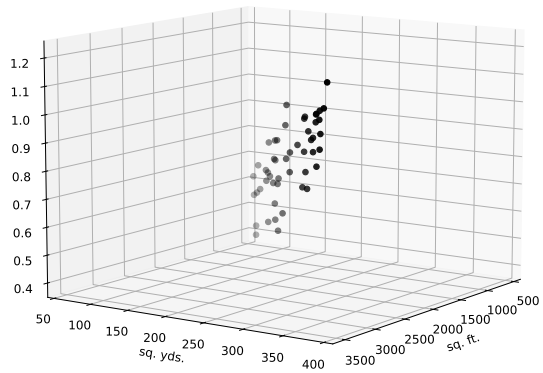
LinAlgError: Singular matrix

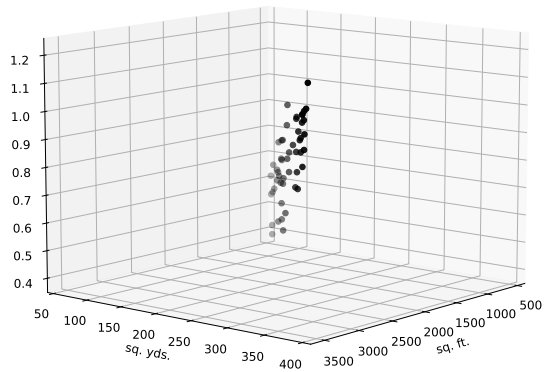
Issue: “Singular Matrix” Error

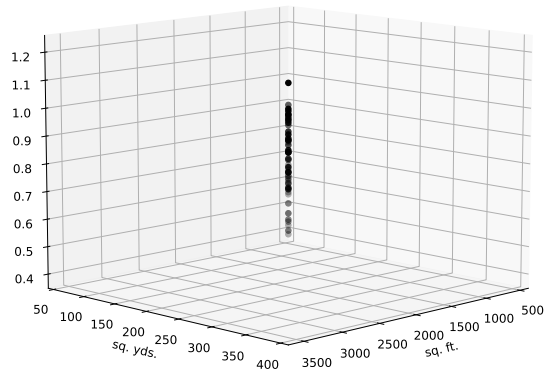
- ▶ Let's look at the data.





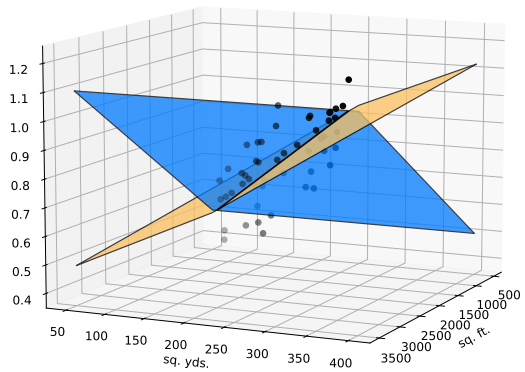


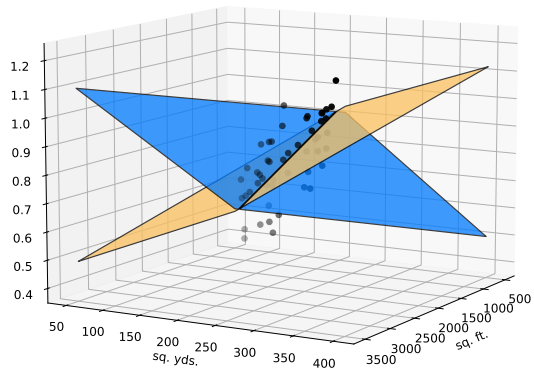


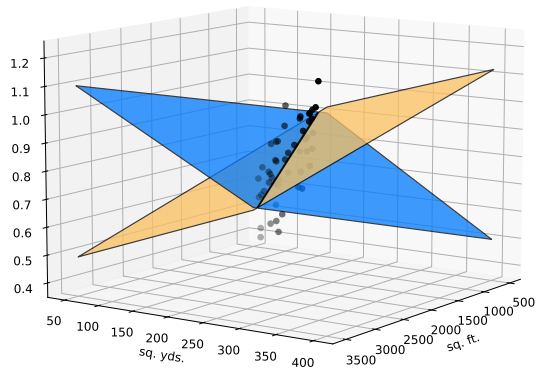


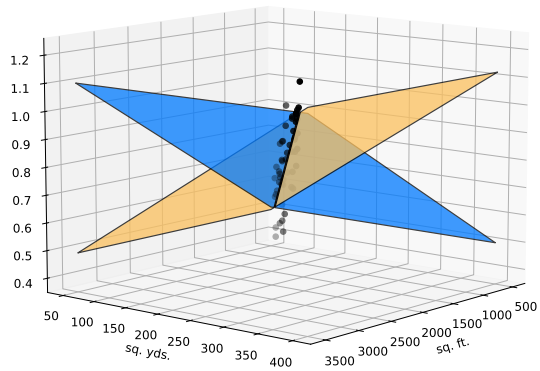
Issue: “Singular Matrix” Error

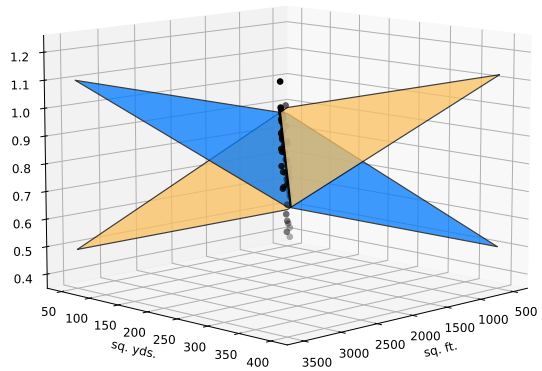
- ▶ The data aren't truly 3-dimensional.
- ▶ There are **infinitely many** planes with the same empirical risk.
- ▶ That is, there are infinitely many solutions to the normal equations.
 - ▶ This is why the matrix is singular.

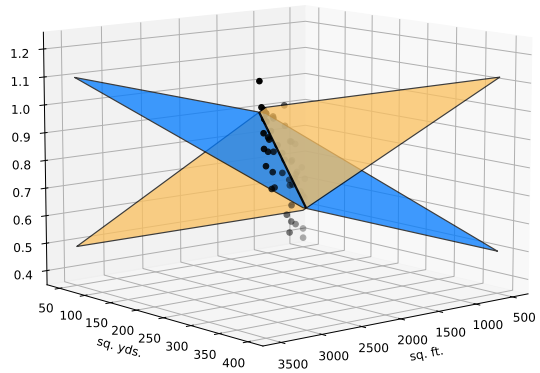


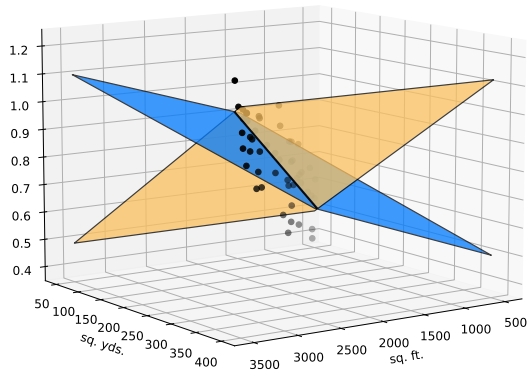


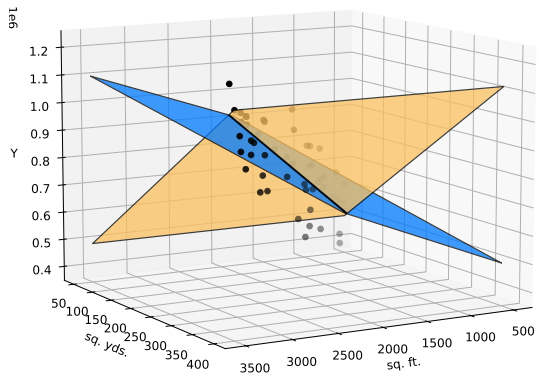


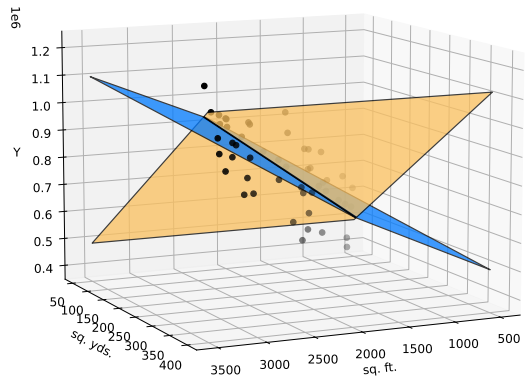












Multicollinearity

- ▶ The situation where one feature is a linear combination of others is called **multicollinearity**.
- ▶ Can happen because the features are redundant, or because of chance.
- ▶ One fix: remove one of the redundant features.
 - ▶ We'll see another fix in lecture on **regularization**.

Issue: Time

```
[*]: np.linalg.solve(X.T @ X, X.T @ y)
```

- ▶ Solving a linear system in d unknowns takes $\Theta(d^3)$ time.
- ▶ Fine for small number of features, but can be slow when using many features.
- ▶ Next time: an approach for efficiently minimizing risk when data is very large.