# 446 Section 4 ← (3 - η(-1))

TA: Sankar Harilal

## Plans for today!

1. Reminders
2. Train/Val/Test Review
3. Gradient Descent
4. Generalized Least Squares
5. Importance of Regularization in Least Squares
6. Ridge/LASSO (if time)

## Reminders

- HW1 was due yesterday
  - Remember that you have 5 late days!
- HW2 was released yesterday; due Wednesday, Feb 11

Some tips:

- Use office hours to your advantage
  - Student TA OH for homework questions
  - Professor OH for conceptual questions
- Skim the homeworks the day they are assigned and try one problem
  - Motivates you to get things done on time, starting an untouched assignment can be daunting
- Keep a tab open with the lecture slides while you do the homework for reference

## Train/val/test

## What do you never ever ever ever ever ever ever ever ever do?

**Train/tune your model on your test set!**

Training parameters

Easy to remember…

Choosing hyperparameters

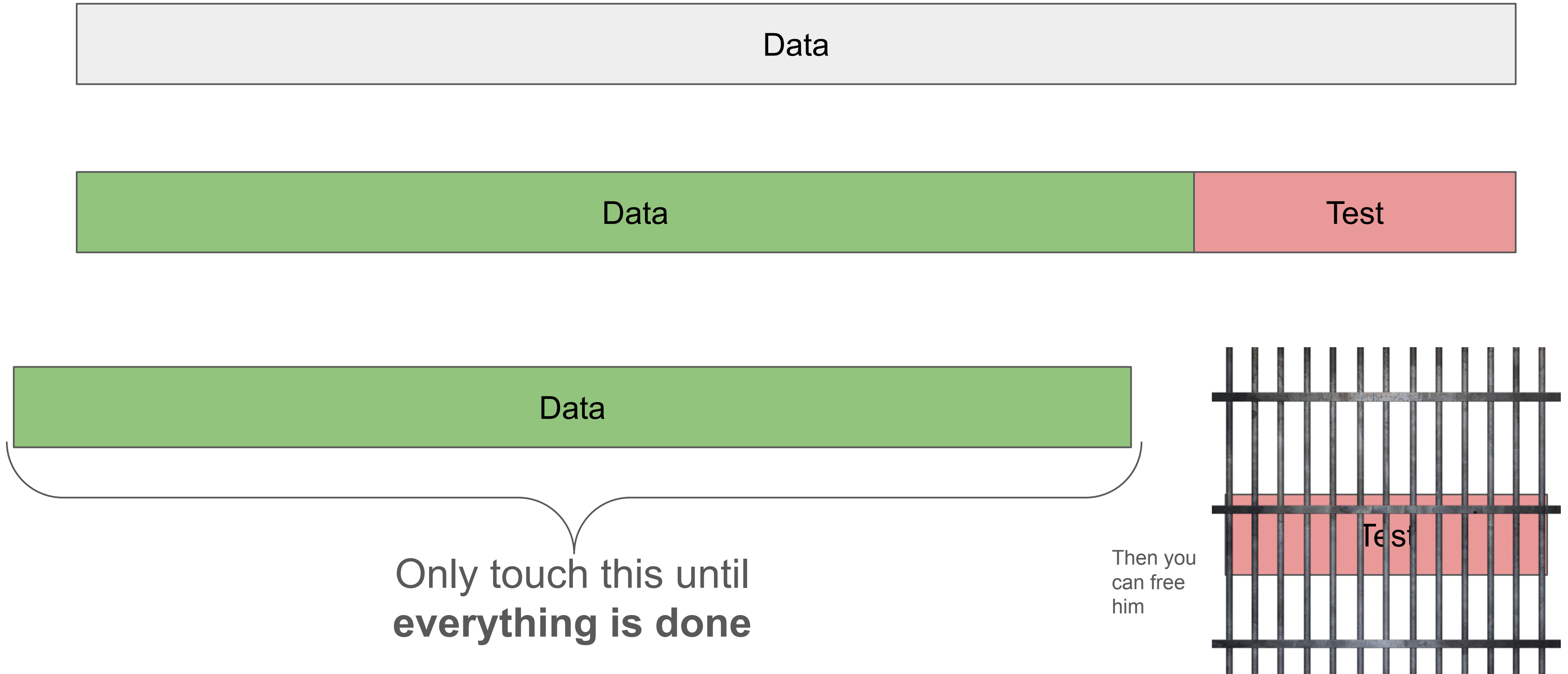
Easy-ish to remember…

Preprocessing, small changes, etc…

**Hard** to remember

Very sinister!

## Easiest way to combat this?



## Bonus Questions:

**Q:** If you have *N* data points, what choice of *k* for k-fold validation would give you the same train/val splits as LOOCV?

**A:** *N*

**Q:** How would you get the **most** “pessimistic” estimate on your validation set?

**A:** Only train on 1 example and validate on the rest

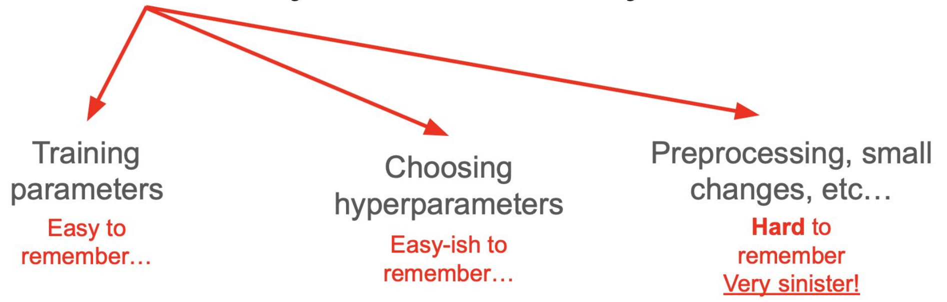
## Problems 1.1, 1.2

You are given blocks of code, and something is wrong/not totally right with how they deal with the data.

*Identify them and propose solutions!*

What do you never ever ever ever ever ever ever ever ever do?

Train/tune your model on your test set!



## 1.1. Program 1

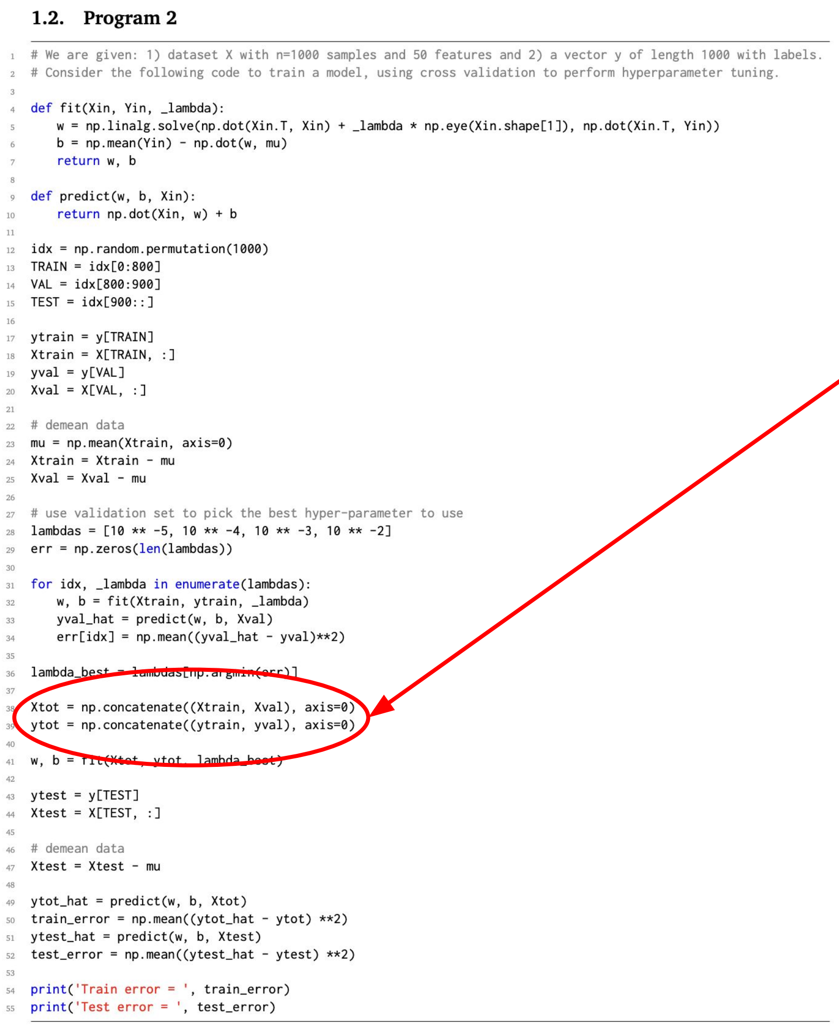
```python
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector

mu = np.mean(X, axis=0)
X = X - mu

idx = np.random.permutation(1000)
TRAIN = idx[0:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN, :]

# solve for argmin_w ||Xtrain*w - ytrain||_2
w = np.linalg.solve(np.dot(Xtrain.T, Xtrain), np.dot(Xtrain.T, ytrain))

b = np.mean(ytrain)

ytest = y[TEST]
Xtest = X[TEST, :]

train_error = np.dot(np.dot(Xtrain, w)+b - ytrain,
                     np.dot(Xtrain, w)+b - ytrain ) / len(TRAIN)
test_error = np.dot(np.dot(Xtest, w)+b - ytest,
                     np.dot(Xtest, w)+b - ytest ) / len(TEST)

print('Train error = ', train_error)
print('Test error = ', test_error)
```

*mu* is calculated from the **entire** data (train + test), intertwining them!

*This is bad!*

**Correct procedure:**

- **Split into train and test**
- **Compute the mean of the train data ($\mu_{train}$)**
- **De-mean both train and test data using $\mu_{train}$**



**1.2. Program 2**

```python
# We are given: 1) dataset X with n=1000 samples and 50 features and 2) a vector y of length 1000 with labels.
# Consider the following code to train a model, using cross validation to perform hyperparameter tuning.

def fit(Xin, Yin, _lambda):
    w = np.linalg.solve(np.dot(Xin.T, Xin) + _lambda * np.eye(Xin.shape[1]), np.dot(Xin.T, Yin))
    b = np.mean(Yin) - np.dot(w, mu)
    return w, b

def predict(w, b, Xin):
    return np.dot(Xin, w) + b

idx = np.random.permutation(1000)
TRAIN = idx[0:800]
VAL = idx[800:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN, :]
yval = y[VAL]
Xval = X[VAL, :]

# demean data
mu = np.mean(Xtrain, axis=0)
Xtrain = Xtrain - mu
Xval = Xval - mu

# use validation set to pick the best hyper-parameter to use
lambdas = [10 ** -5, 10 ** -4, 10 ** -3, 10 ** -2]
err = np.zeros(len(lambdas))

for idx, _lambda in enumerate(lambdas):
    w, b = fit(Xtrain, ytrain, _lambda)
    yval_hat = predict(w, b, Xval)
    err[idx] = np.mean((yval_hat - yval)**2)

lambda_best = lambdas[np.argmin(err)]

Xtot = np.concatenate((Xtrain, Xval), axis=0)
ytot = np.concatenate((ytrain, yval), axis=0)

w, b = fit(Xtot, ytot, lambda_best)

ytest = y[TEST]
Xtest = X[TEST, :]

# demean data
Xtest = Xtest - mu

ytot_hat = predict(w, b, Xtot)
train_error = np.mean((ytot_hat - ytot) **2)
ytest_hat = predict(w, b, Xtest)
test_error = np.mean((ytest_hat - ytest) **2)

print('Train error = ', train_error)
print('Test error = ', test_error)
```

The final model is trained on BOTH the training and validation sets.

*This is… eh…*

**Your hyperparameters selected on just the train data may not hold for train + val:**

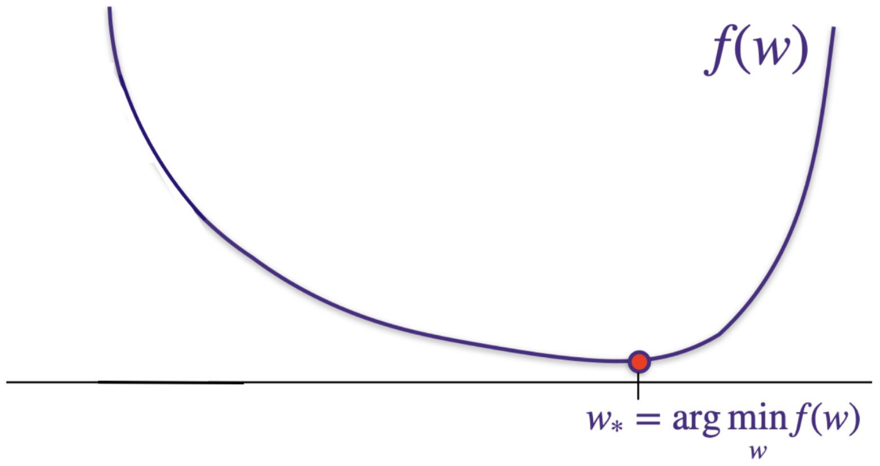
- **More data is good but you should ensure that the hyperparameters you tuned do not depend on the number of elements.**
- **Tradeoff between more data and better test error estimate**

## Gradient Descent

## Gradient Descent

Purpose of this exercise: Understanding how gradient descent relates to approximations, and why it works.

Consider some function $f(w)$, which has some $w_*$ for which $w_* = \arg\min_w f(w)$:



## Question 2a

Let $w_0$ be some initial guess for the minimum of $f(w)$. Gradient descent will allow us to improve this solution.

(a) For some $w$ that is very close to $w_0$, give the Taylor series approximation for $f(w)$ starting at $f(w_0)$.

Remember Taylor expansion?

↳ To approximate a function around a point $a$

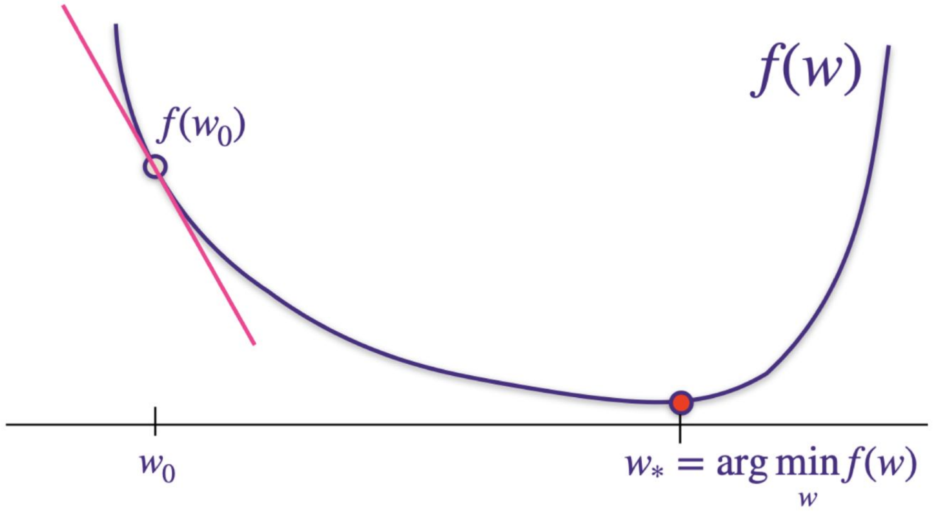
$$\underbrace{\boxed{f(x) \approx f(a) + \frac{f'(a)}{1!}(x-a)}}_{\text{Exact at } a\text{, close around } a} + \underbrace{\frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 \cdots}_{\text{Better and better approximations}}$$

## Question 2a

Let $w_0$ be some initial guess for the minimum of $f(w)$. Gradient descent will allow us to improve this solution.

(a) For some $w$ that is very close to $w_0$, give the Taylor series approximation for $f(w)$ starting at $f(w_0)$.

For $w$ very close to $w_0$, we see that $f(w) \approx f(w_0) + (w - w_0)\left(\frac{df(w)}{dw}\Big|_{w=w_0}\right)$.



## Question 2b

(b) Now, let us choose some $\eta > 0$ that is *very small*. With this very small $\eta$, let's assume that $w_1 = w_0 - \eta\left(\frac{df(w)}{dw}\bigg|_{w=w_0}\right)$. Using your approximation from part (a), give an expression for $f(w_1)$.

**Hint:** Plug in here

$$f(w) \approx f(w_0) + (w - w_0)\underbrace{\left(\frac{df(w)}{dw}\bigg|_{w=w_0}\right)}.$$

Fancy way of saying $f'(w_0)$

(Derivative of f(w) at $w_0$)

## Question 2b

$$w_1 = w_0 - \eta \left( \frac{df(w)}{dw} \bigg|_{w=w_0} \right)$$ ← Given

plug in here

$$f(w_1) \approx f(w_0) + (w_1 - w_0) \left( \frac{df(w)}{dw} \bigg|_{w=w_0} \right)$$

$$= f(w_0) + \left( w_0 - \eta \left( \frac{df(w)}{dw} \bigg|_{w=w_0} \right) - w_0 \right) \left( \frac{df(w)}{dw} \bigg|_{w=w_0} \right)$$

$$= f(w_0) - \eta \left( \frac{df(w)}{dw} \bigg|_{w=w_0} \right)^2$$

## Question 2c

(c) Given your expression for $f(w_1)$ from part (b), explain why, if $\eta$ is small enough and if the function approximation is a good enough approximation, we are guaranteed to move in the "right" direction closer to the minimum $w_*$.

**Remember:** We want to minimize this

**Hint:** Why would this be good?

$$\underbrace{f(w_1)}_{} \approx f(w_0) - \eta \left( \frac{df(w)}{dw} \bigg|_{w=w_0} \right)^{\boxed{2}}$$

## Question 2c

Note that in part (b), the derivative is squared and will always be a nonnegative value. Therefore, $f(w_1) < f(w_0)$.

$$f(w_1) \approx f(w_0) - \eta \left( \frac{df(w)}{dw} \bigg|_{w=w_0} \right)^2$$

**In English:** The loss function after a weight update will always evaluate to be smaller than before the weight update

- If the step size is small enough
- If the approximation is good enough

## Question 2d

(d) Building from your answer in part (c), write a general form for the gradient descent algorithm.

Hint: how could we generalize this equation from part b?

$$w_1 = w_0 - \eta \left( \frac{df(w)}{dw} \bigg|_{w=w_0} \right)$$

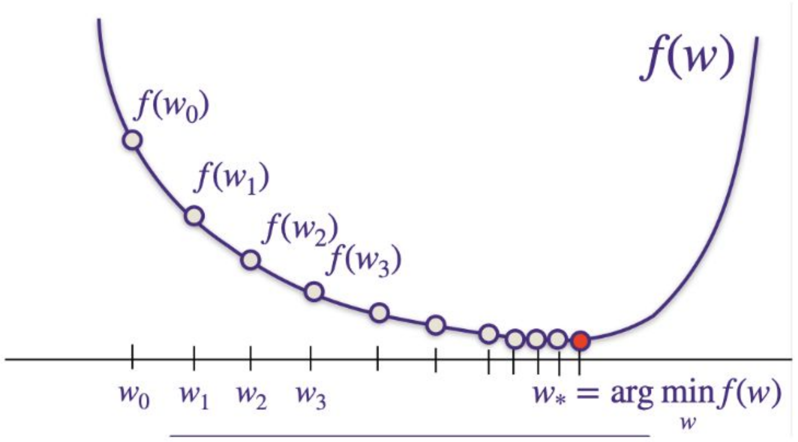
## Question 2d

Gradient descent is written as:

For $k = 0, 1, 2, 3, \ldots, w_{k+1} = w_k - \eta \left( \frac{df(w)}{dw} \bigg|_{w=w_k} \right).$

Note that as $k \to \infty$, $\left( \frac{df(w)}{dw} \bigg|_{w=w_k} \right) \to 0.$

We visualize as:



**Convergence guarantees iff convex!**

## Generalized Least Squares

## Least Squares Proof(s)

Should look familiar…

$$\boxed{\widehat{\omega}_{\text{general}} = (X^\top X + \lambda D)^{-1} X^\top y}$$

Has shown up…

- In lecture (Lecture 2)
- On your homework (A5 Ridge Regression proof)
- And now here!

$$\widehat{\omega}_{\text{general}} = \left( \sum_{i=1}^{n} x_i x_i^\top + \lambda D \right)^{-1} \left( \sum_{i=1}^{n} x_i y_i \right)$$

You can look at the generalized proof in your own time.

## Question 3.2a

$$\widehat{\omega}_{\text{general}} = (X^\top X + \lambda D)^{-1} X^\top y$$

(a) In the simple least squares case ($\lambda = 0$ above), what happens to the resulting $\widehat{\omega}$ if we double all the values of $y_i$?

**Solution:**

As can be seen from the formula $\widehat{\omega} = (X^\top X)^{-1} X^\top y$, doubling $y$ doubles $\omega$ as well. This makes sense intuitively as well because if the observations are scaled up, the model should also be.

## Question 3.2b

$$\widehat{\omega}_{\text{general}} = (X^\top X + \lambda D)^{-1} X^\top y$$

(b) In the simple least squares case ($\lambda = 0$ above), what happens to the resulting $\widehat{\omega}$ if we double the data matrix $X \in \mathbb{R}^{n\times d}$?

**Solution:**

As can be seen from the formula $\widehat{\omega} = (X^\top X)^{-1} X^\top y$, doubling $X$ halves $\omega$. This also makes sense intuitively because the error we are trying to minimize is $\|X\omega - y\|_2^2$, and if the $X$ has doubled, while $y$ has remained unchanged, then $\omega$ must compensate for it by reducing by a factor of 2.

## Importance of Regularization in Least Squares

## Question 3.2c

$$\widehat{\omega}_{\text{general}} = (X^\top X + \lambda D)^{-1} X^\top y$$

(c) Suppose $D = I$ (that is, it is the identity matrix). That is, this is the *ridge* regression setting. Explain why $\lambda > 0$ ensures that the solution exists and the matrix can be inverted.

## 3.2c setup

Let’s do a linear algebra refresher so that we can show off an interesting and actually useful result about the utility of regularization!

$$A : \mathbb{R}^d \to \mathbb{R}^n, \text{ if } d \gg n$$

$$\downarrow$$

$$A : \begin{array}{cc} d & \\ \left[\begin{matrix} \vdots & \vdots & \vdots \end{matrix}\right] & n \end{array}$$

must have a non-empty nullspace
(can show using rank-nullity theorem)

Way to think about nullspaces

$$A \in \mathbb{R}^{n \times d} \quad x \in \mathbb{R}^d$$

Nullspace: Subspace of $\mathbb{R}^d$, contains all solutions to $\underline{Ax = 0}$

In other words, all the vectors are “annihilated” by $A$

This is bad for invertibility

Note: $Null(A) = Null(A^TA)$

$\hookrightarrow$ proof in Section 02 handout

Invertible means $(A^TA)^{-1}(A^TA) = I$, so $(A^TA)^{-1}(A^TA)x = x$

Invertible means $(A^\top A)^{-1}(A^\top A) = I$, so $(A^\top A)^{-1}(A^\top A)x = x$

If $A^\top A$ has a non-empty nullspace, then

$\exists x$ s.t. $(A^\top A)x = 0$

$$\downarrow$$

Makes $(A^\top A)^{-1}\underbrace{(A^\top A)x}_{\text{If this } = 0\text{, no way to recover } x\text{!}} = x$ impossible!

Main idea: If $X \in \mathbb{R}^{n \times d}, d \gg n,$ then

Null$(X)$ and Null$(X^\top X)$ are non-empty

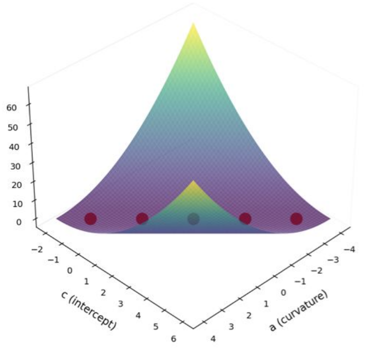
This means $X^\top X$ has no inverse

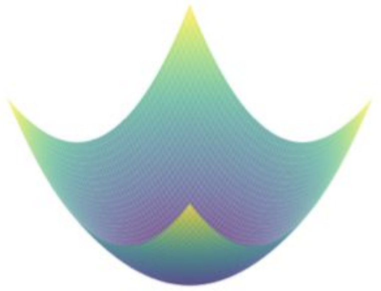
An issue because... $\hat{w} = (X^\top X)^{-1} X^\top y$

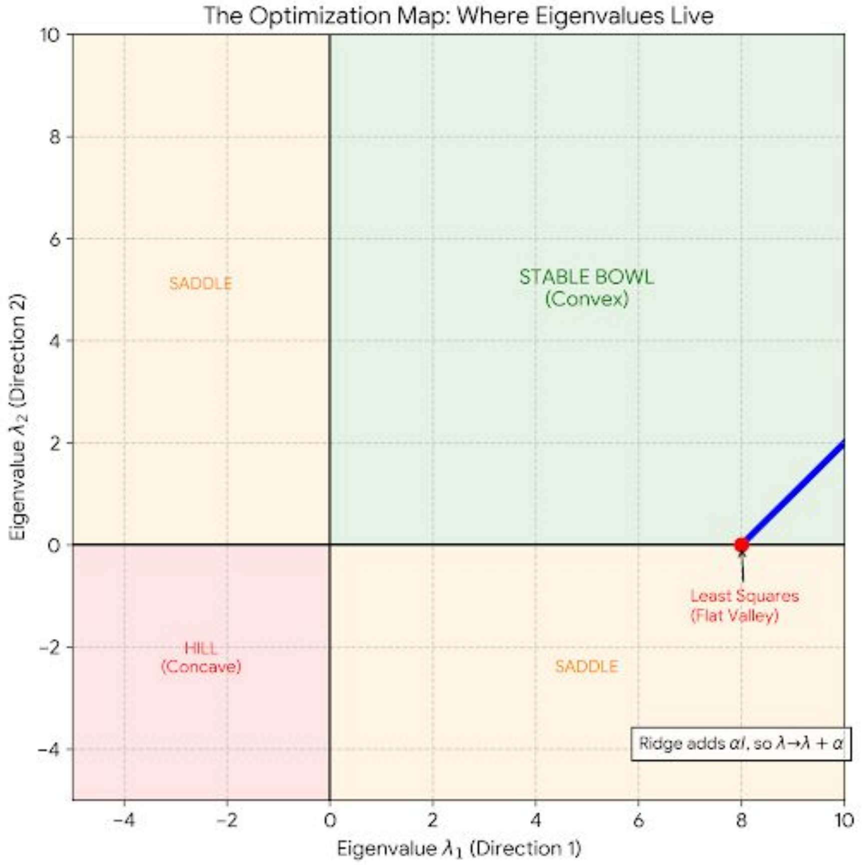
Let's not give up!

## Visualized

- If $X$ has a non-empty null space, matrix $X^T X$ has an eigenvalue of 0. This corresponds to a “flat valley” – no unique solution
- By adding $I$, we shift all the eigenvalues, moving it into a “stable bowl”






The Optimization Map: Where Eigenvalues Live
SADDLE
STABLE BOWL (Convex)
HILL (Concave)
SADDLE
Least Squares (Flat Valley)
Ridge adds $\alpha I$, so $\lambda \to \lambda + \alpha$
Eigenvalue $\lambda_1$ (Direction 1)
Eigenvalue $\lambda_2$ (Direction 2)

Let's add in $\lambda I$ : $\hat{w} = (X^T X + \lambda I)^{-1} X^T Y$

Is $X^T X + \lambda I$ always invertible for $\lambda > 0$?

A matrix $A$ is positive semi-definite if $x^\top A x \geq 0$ and positive definite if $x^\top A x > 0$
**Positive Definite (PD):** All eigenvalues are strictly positive
**Positive Semi-Definite (PSD):** All eigenvalues are 0 or positive

We want to show that $u(X^T X + \lambda I)u > 0 \quad \forall u \in \mathbb{R}^d$

$\hookrightarrow$ Show matrix is positive definite, meaning it must have an inverse

We want to show that $u(X^T X + \lambda I)u > 0 \quad \forall u \in \mathbb{R}^d$

$$u(X^T X + \lambda I)u = \underbrace{u(X^T X)u}_{\text{Is this always >0?}} + u(\lambda I)u$$

L2 norm

$$u(X^T X)u = uX^T Xu = \|Xu\|_2^2 \geq 0 \quad \text{Yes!}$$

$$\begin{aligned} u(X^TX + \lambda I)u &= u(X^TX)u + u(\lambda I)u \\ &\geq u(\lambda I)u \leftarrow \text{this is PD, } \therefore > 0 \\ &> 0 \end{aligned}$$

We have shown $X^TX + \lambda I$ is PD and therefore always invertible if $\lambda > 0$!

$\hookrightarrow$ $\underline{\text{Even if } d \gg n!}$

## Question 3.2c

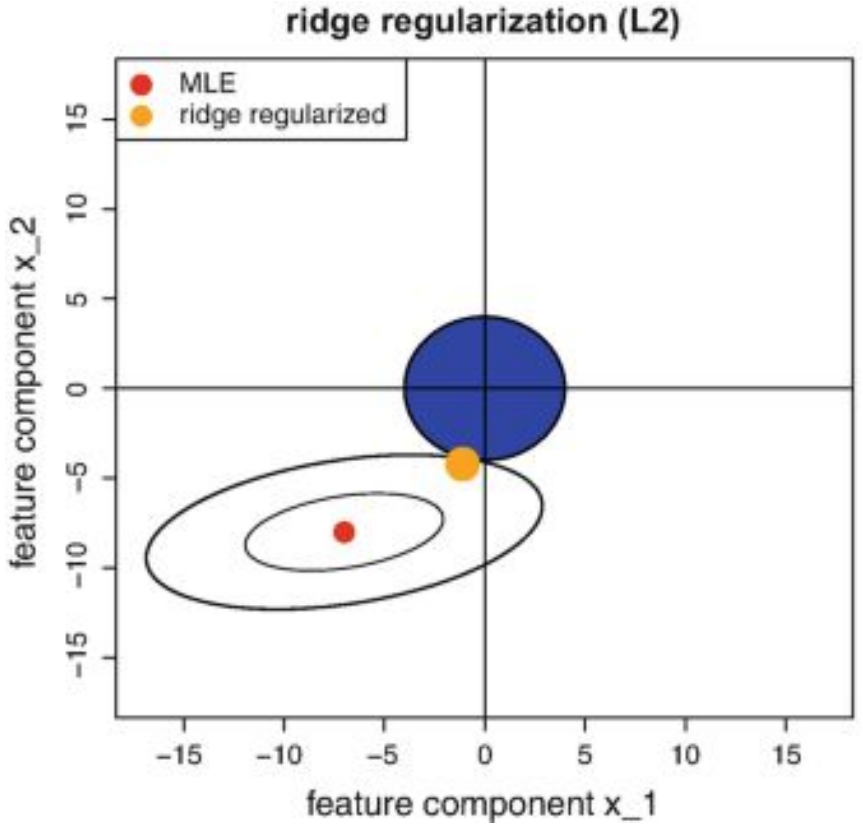
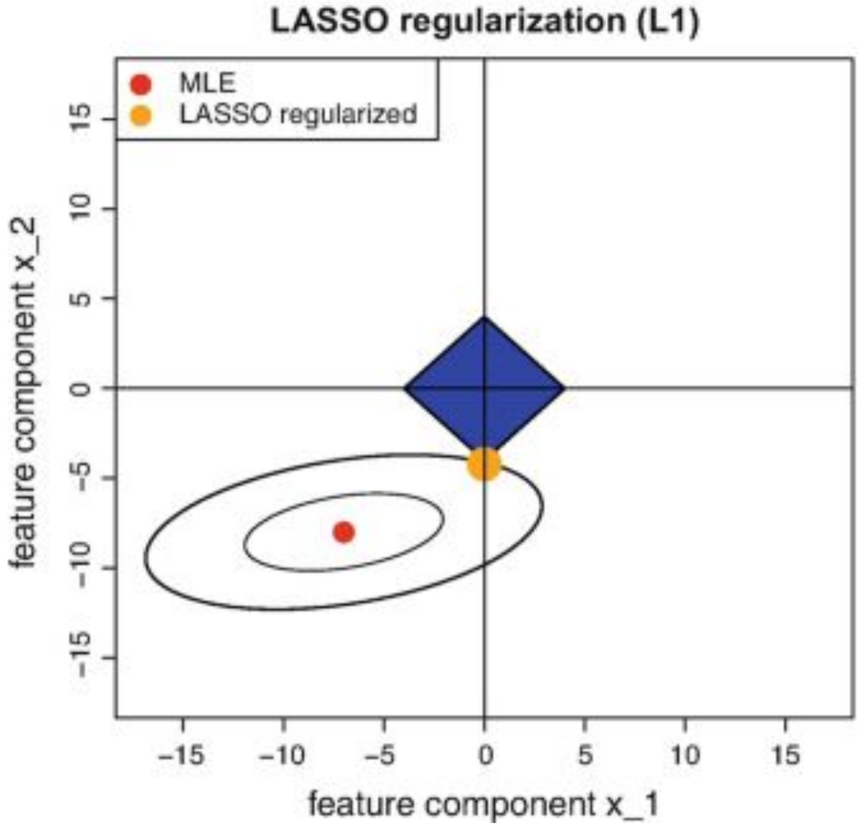
$$\widehat{\omega}_{\text{general}} = (X^\top X + \lambda D)^{-1} X^\top y$$

(c) Suppose $D = I$ (that is, it is the identity matrix). That is, this is the *ridge* regression setting. Explain why $\lambda > 0$ ensures that the solution exists and the matrix can be inverted.

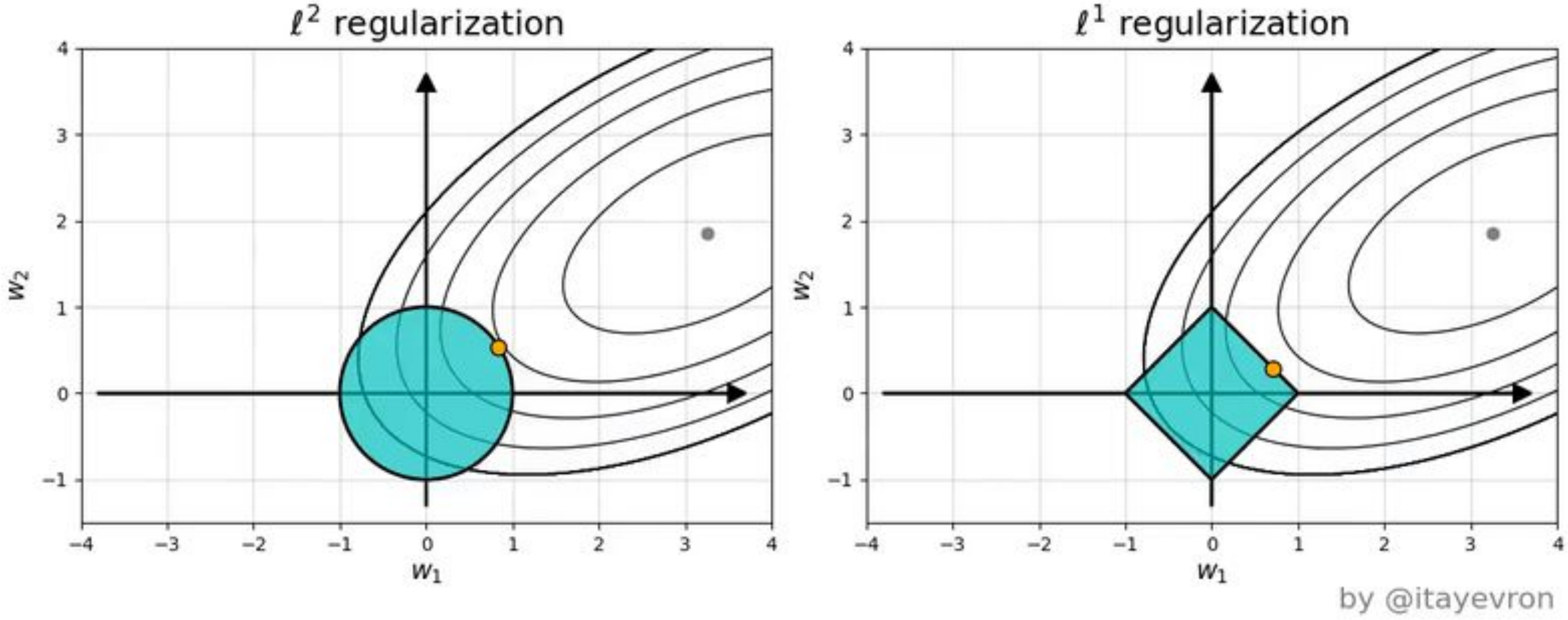
**Solution:**

The solution is $\widehat{\omega} = (X^\top X + \lambda I)^{-1} X^\top y$. We already saw in a previous part that $X^\top X$ is always positive semidefinite, that is, its eigenvalues are at least zero. Adding $\lambda I$, where $\lambda > 0$, ensures that $X^\top X + \lambda I$ is in fact positive *definite*. This helps us have a good condition number.

## Ridge vs. LASSO


ridge regularization (L2)
● MLE
● ridge regularized
feature component x_1
feature component x_2
−15
−10
−5
0
5
10
15

LASSO regularization (L1)
● MLE
● LASSO regularized
feature component x_1
feature component x_2
−15
−10
−5
0
5
10
15

## $\ell^1$ induces sparse solutions for least squares



## Questions/Chat Time!