446 Section 4 $\leftarrow$ (3 - η(-1))

TA: Varun Ananth

Plans for today!

1. This
2. Reminders
3. Train/Val/Test Problems
4. Gradient Descent
5. Generalized Least Squares
6. Ridge/LASSO (if time)

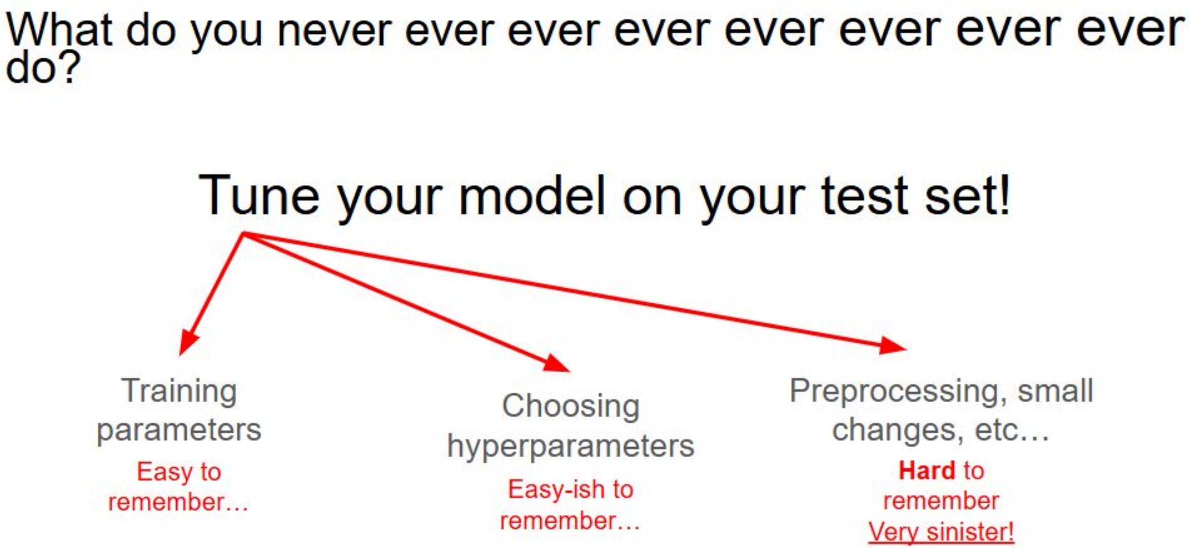
Reminders

- HW1 was due yesterday
  - Remember the late day policy!
- HW2 is released
- Midterm in a week...
  - February 7th, Friday

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!



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!

Calculate a mean just on the train data, and use this to de-mean both the train <u>and</u> test datasets

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

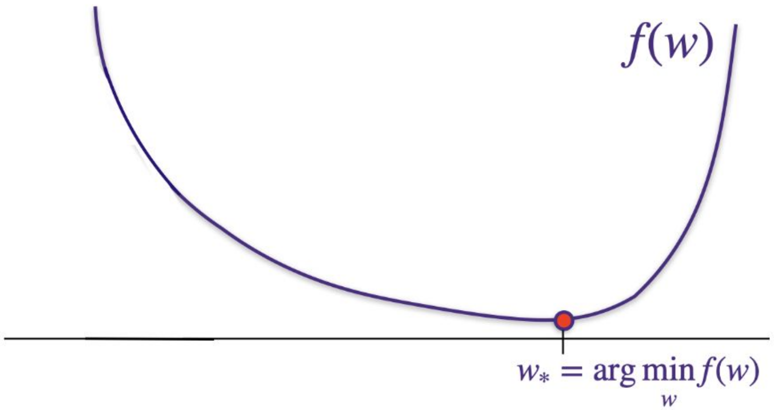
- Tradeoff between more data and better test error estimate

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 $w_* = \arg\min_w f(w)$:



2a

(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

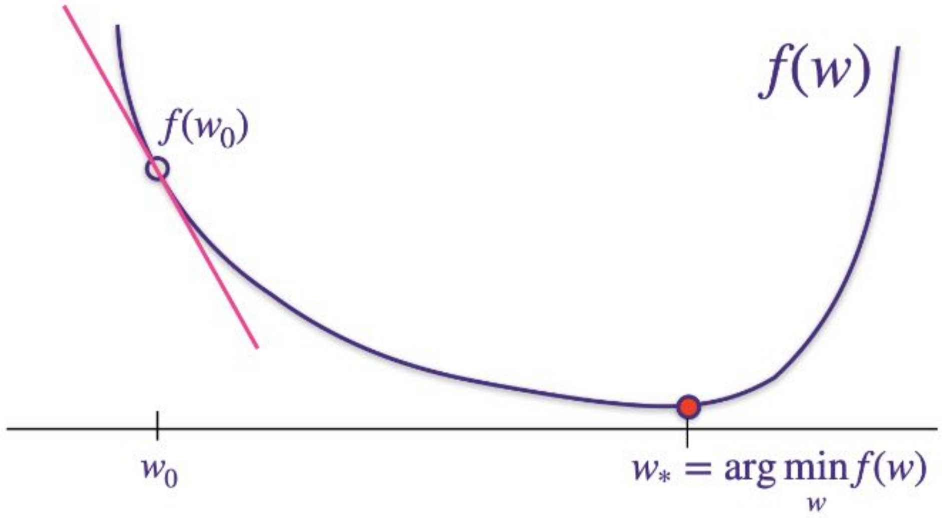
$$\boxed{f(x) \approx f(a) + \frac{f'(a)}{1!}(x-a)} + \underbrace{\frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 \cdots}_{\text{Better and better approximations}}$$

Exact at a, close around a

2a (answer)

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



2b

(b) Now, let us choose some $\eta > 0$ that is *very small*. With this very small η, 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*$_0$)

2b (answer)

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

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

2c

(c) Given your expression for $f(w_1)$ from part (b), explain why, if η 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?

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

2c (answer)

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

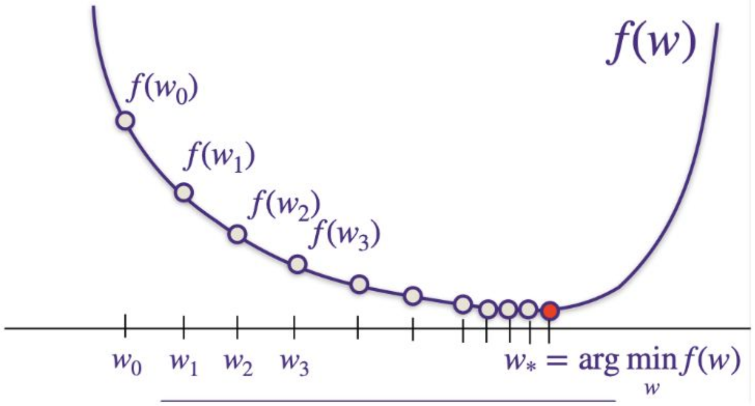
2d (answer)

Gradient descent is written as:

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

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

We visualize as:



Convergence guarantees iff convex!

Generalized Least Squares

Least Squares Proof(s)

Has shown up…

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

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

Should look familiar…

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

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

3.2a

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

(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?

3.2a (answer)

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

(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 ω as well. This makes sense intuitively as well because if the observations are scaled up, the model should also be.

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 $X \in \mathbb{R}^{n \times d}$?

3.2b (answer)

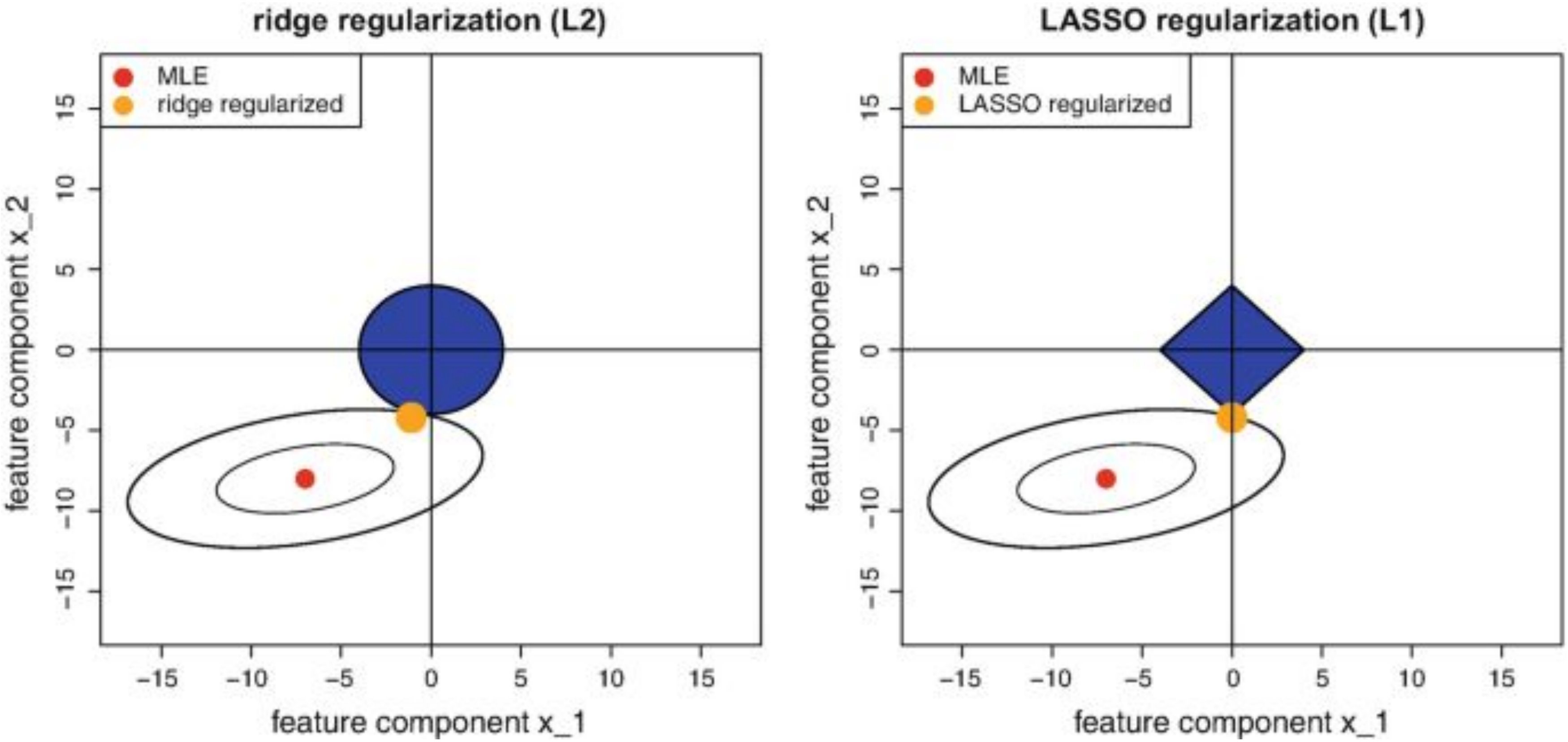
$$\widehat{\omega}_{\text{general}} = (X^\top X + \lambda D)^{-1} X^\top y$$

(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 ω. 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 ω must compensate for it by reducing by a factor of 2.

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

Questions/Chat Time!