Cross-Validation, cnt'd

Quick recap and birds'-eye view of where we are

- **Linear (least squares) regression for predicting y from x**
  - **Can also do nonlinear, engineer features.**
- **How do we know if our model works well/which features we should include/how complex a set of features we should engineer/how high dimensionally should we regress?**
  - **A: We're trying to estimate performance of the model not on our training data, but on the *true* distribution. This is only measurable on never before seen data (e.g., the test set).**
  - **But obviously we want something to guide our model training/selection… cross validation helps here!**

Use *k*-fold cross validation

- Randomly divide training data into *k* equal parts
  - $D_1,\ldots,D_k$
- For each *i*
  - Learn model $f_{D\backslash Di}$ using data point not in D_i
  - Estimate error of $f_{D\backslash Di}$ on validation set D_i:

1	2	3	4	5
Train	Train	Validation	Train	Train

$$\text{error}_{\mathcal{D}_i} = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j,y_j)\in\mathcal{D}_i} (y_j - f_{\mathcal{D}\backslash\mathcal{D}_i}(x_j))^2$$

- k-fold cross validation error is average over data splits:

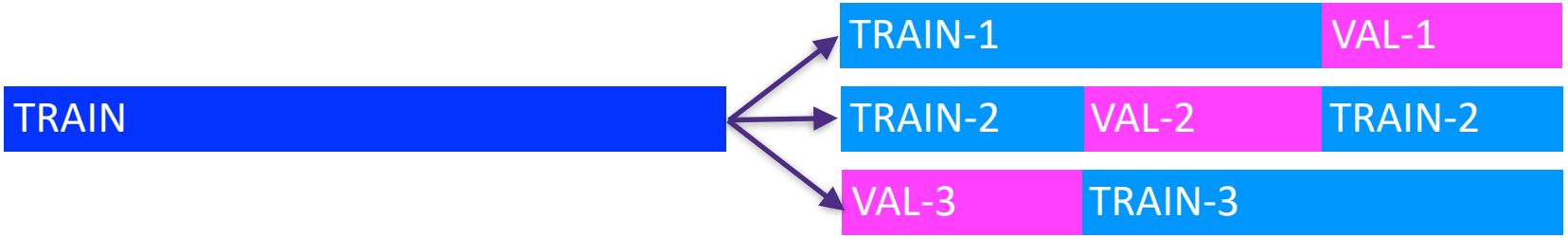
$$error_{k-fold} = \frac{1}{k}\sum_{i=1}^{k} error_{\mathcal{D}_i}$$

- k-fold cross validation properties:
  - Much faster to compute than LOO
  - More (pessimistically) biased – using much less data, only *n(k-1)/k*
  - Usually, k = 10

Recap

- Given a dataset, begin by splitting into

- Model selection: Use k-fold cross-validation on **TRAIN** to train predictor and choose magic parameters such as degree



- Model assessment: Use **TEST** to assess the accuracy of the model you output
  - **Never ever ever ever ever train or choose parameters based on the test data**

Example 1

- **You wish to predict the stock price of zoom.us given historical stock price data**
- **You use all daily stock price up to Jan 1, 2020 as TRAIN and Jan 2, 2020 - April 13, 2020 as TEST**
- **What's "wrong" with this procedure?**

  - ML assumes data is i.i.d.; is this?
    - Nope! Future data is o.o.d. (out of distribution) from the training data

  - Why is "wrong" in square quotes?
    - You might still want to do this. It's economically valuable, if you could.

Example 2

- **Given 10,000-dimensional data and n examples, we pick a subset of 50 dimensions that have the highest correlation with labels in the entire dataset:**

50 indices j that have largest $\frac{|\sum_{i=1}^{n} x_{i,j} y_i|}{\sqrt{\sum_{i=1}^{n} x_{i,j}^2}}$

- **After picking our 50 features, we then break data into train and test dataset.**
- **We train linear regression on these selected features on the training set. We compute the test error and report it**
- **What's wrong with this procedure?**
  - Need to hold out the test data first, otherwise you're contaminating your test data.
    - This happens all the time in ML class projects. Thought the accuracy was 90%. When fixed, it went down to 60%.
  - Top 50 features might be redundant with each other.

Recap

- Learning is…
  - Collect some data
    - E.g., housing info and sale price
  - Randomly split dataset into TRAIN, VAL, and TEST
    - E.g., 80%, 10%, and 10%, respectively
  - Choose a hypothesis class or model
    - E.g., linear with non-linear transformations
  - Choose a loss function
    - E.g., least squares on TRAIN
  - Choose an optimization procedure
    - E.g., set derivative to zero to obtain estimator, cross-validation on VAL to pick num. features
  - Justifying the accuracy of the estimate
    - E.g., report TEST error

Bias-Variance Tradeoff

We keep talking about training error, test error, the bias of our estimator… what are the rules of thumb that relate these (and other quantities)?

Short answer: the more complicated your model/training procedures, the lower your training error can be… but the higher your chance of overfitting to your training data.

Optimal Prediction

True distribution, don't actually have access to it

Goal: Predict $Y \in \mathbb{R}$ **given** $X \in \mathbb{R}^d$ **if** $(X, Y) \sim P_{XY}$

Find function η that minimizes

Tower rule: E[Y] = E$_X$[E[Y|X]]

$$\mathbb{E}_{XY}[(Y - \eta(X))^2] = \mathbb{E}_X\left[\mathbb{E}_{Y|X}[(Y - \eta(x))^2 | X = x]\right]$$

(Hint: for any x, $\eta(x) = c_x$ where c_x minimizes $\mathbb{E}_{Y|X}[(Y - c_x)^2 | X = x]$)

$$\frac{d}{dc}\mathbb{E}_{Y|X}[(Y - c)^2 \,|\, X = x] = 0$$

$$= \mathbb{E}_{Y|X}[2(Y - c)(-1) \,|\, X = x] = 0$$

$$= -2[\mathbb{E}_{Y|X}[Y \,|\, X = x] - c] = 0$$

$$c_x = \boxed{\mathbb{E}_{Y|X}[Y \,|\, X = x] = \eta(x)}$$

Optimal Prediction

Goal: Predict $Y \in \mathbb{R}$ given $X \in \mathbb{R}^d$ if $(X, Y) \sim P_{XY}$

Find function η that minimizes

$$\mathbb{E}_{XY}[(Y - \eta(X))^2] = \mathbb{E}_X\left[\mathbb{E}_{Y|X}[(Y - \eta(x))^2 | X = x]\right]$$

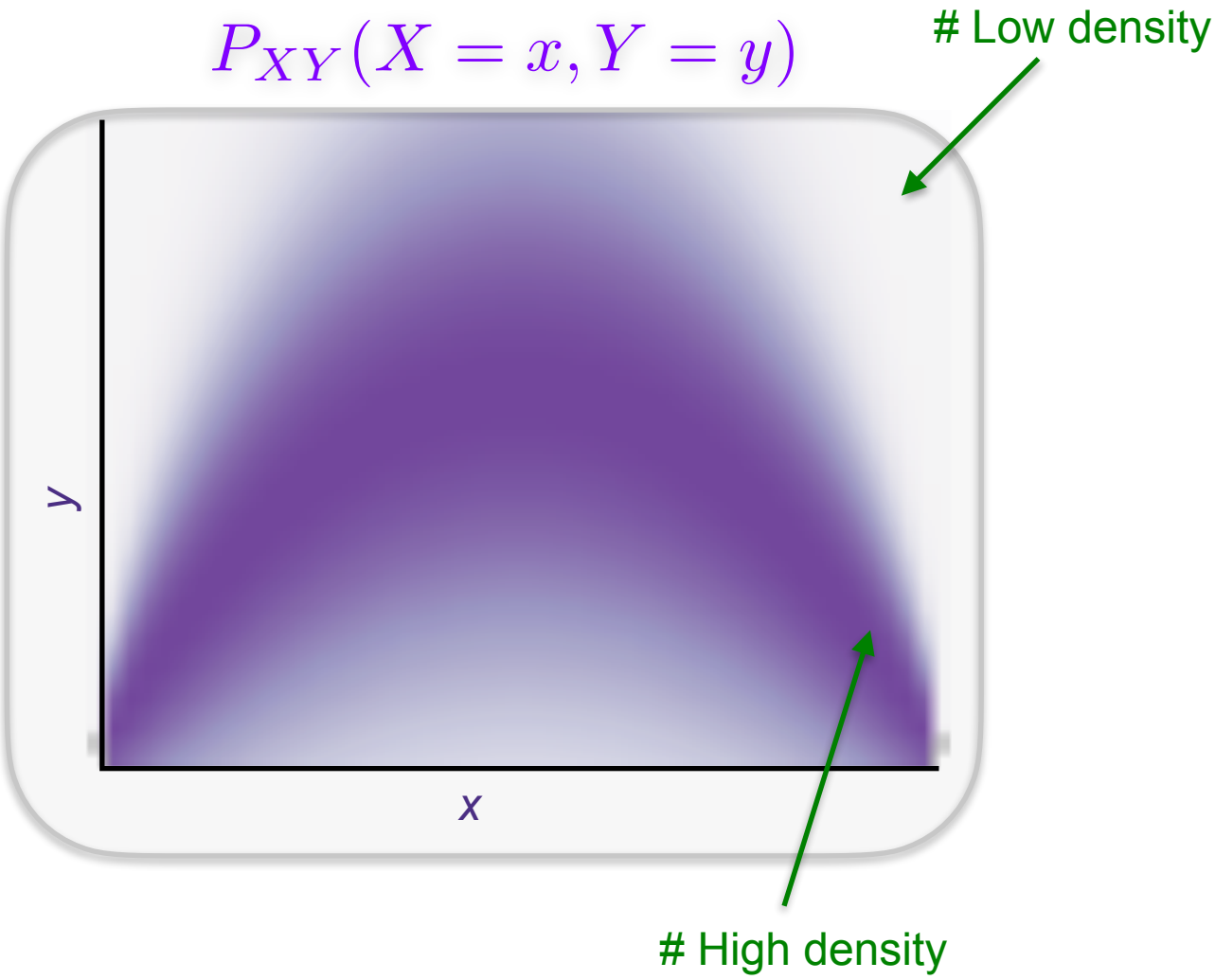
(Hint: for any x, $\eta(x) = c_x$ where c_x minimizes $\mathbb{E}_{Y|X}[(Y - c_x)^2 | X = x]$)

$$\begin{aligned} 0 &= \frac{d}{dc_x}\mathbb{E}_{Y|X}[(Y - c_x)^2 | X = x] \\ &= \mathbb{E}_{Y|X}[\frac{d}{dc_x}(Y - c_x)^2 | X = x] \\ &= \mathbb{E}_{Y|X}[-2(Y - c_x) | X = x] = -2\mathbb{E}_{Y|X}[Y | X = x] + 2c_x \end{aligned}$$

Squared Error Optimal Predictor: $\eta(x) = \mathbb{E}_{Y|X}[Y | X = x]$

Statistical Learning

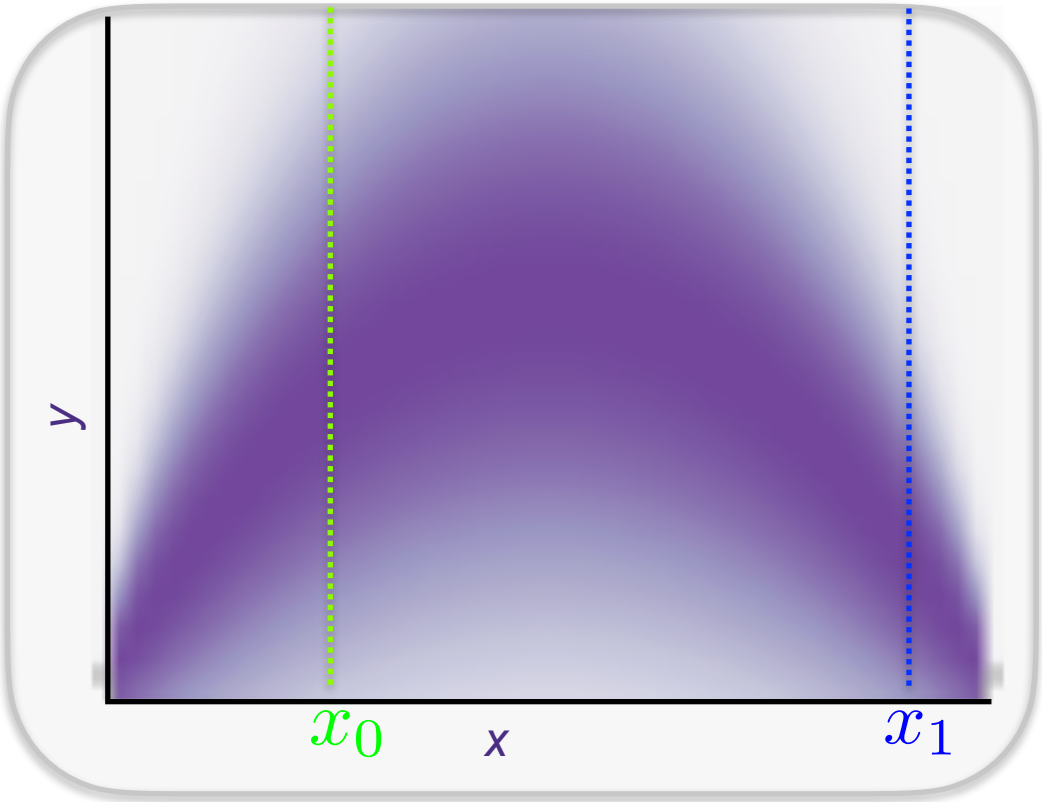
$$\mathbb{E}_{XY}[(Y - \eta(X))^2]$$



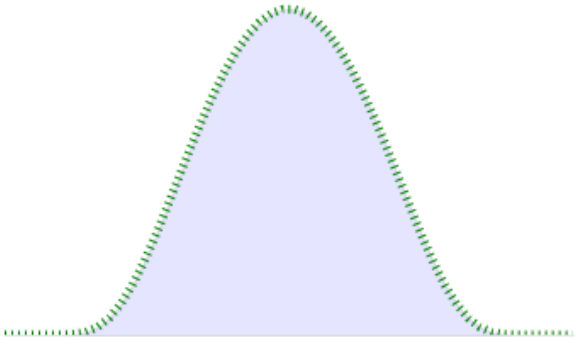
Statistical Learning

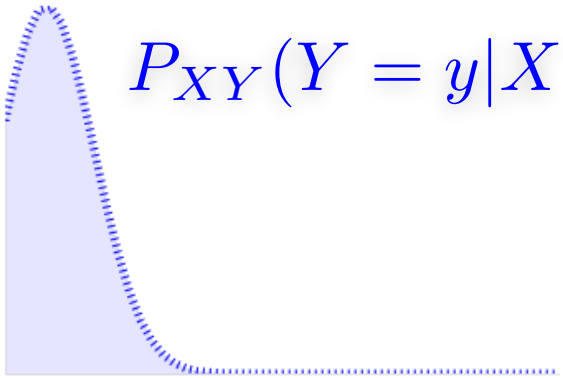
$$\mathbb{E}_{XY}[(Y - \eta(X))^2]$$

$P_{XY}(X = x, Y = y)$



$P_{XY}(Y = y|X = x_0)$

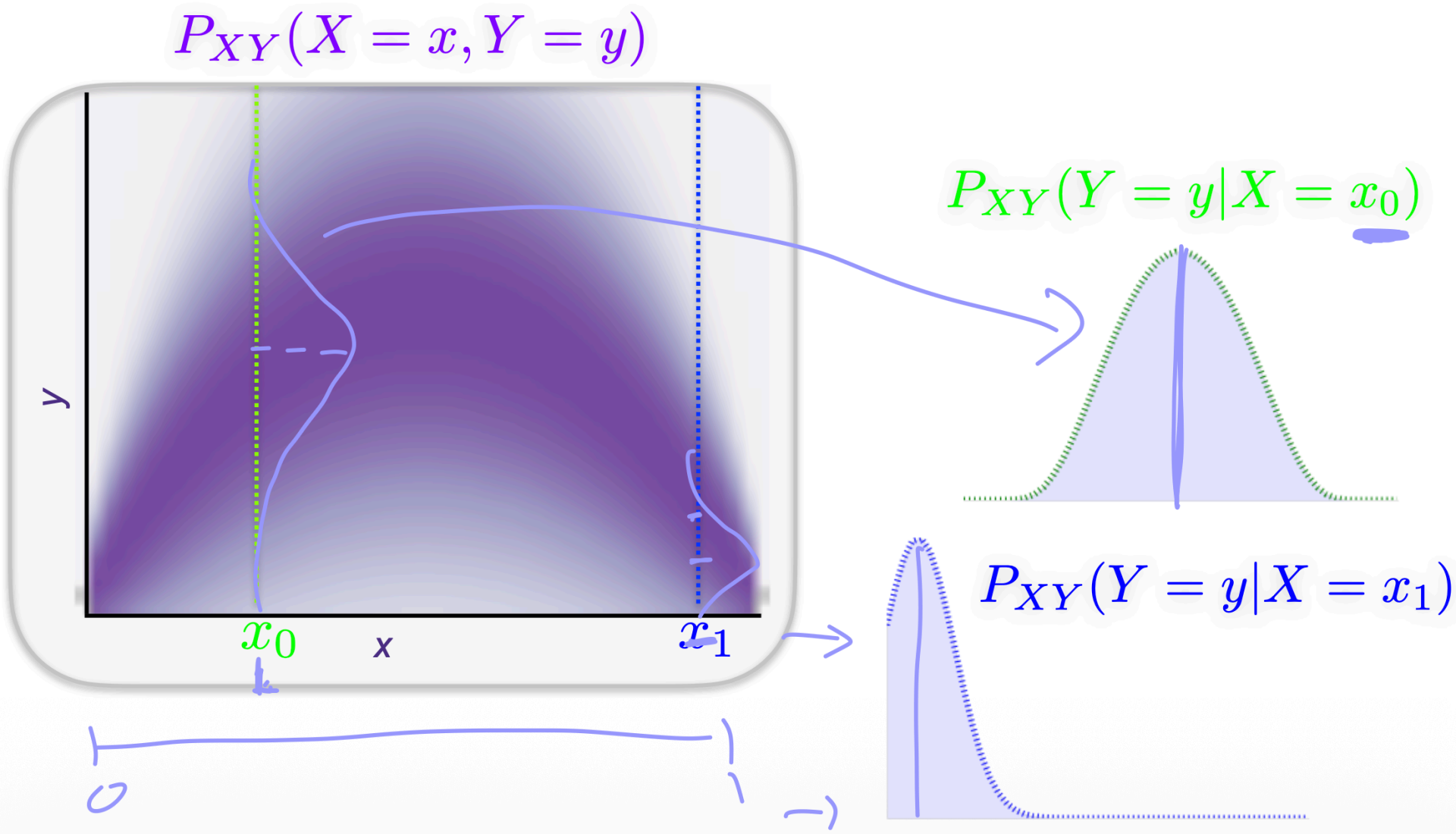




$P_{XY}(Y = y|X = x_1)$

Statistical Learning

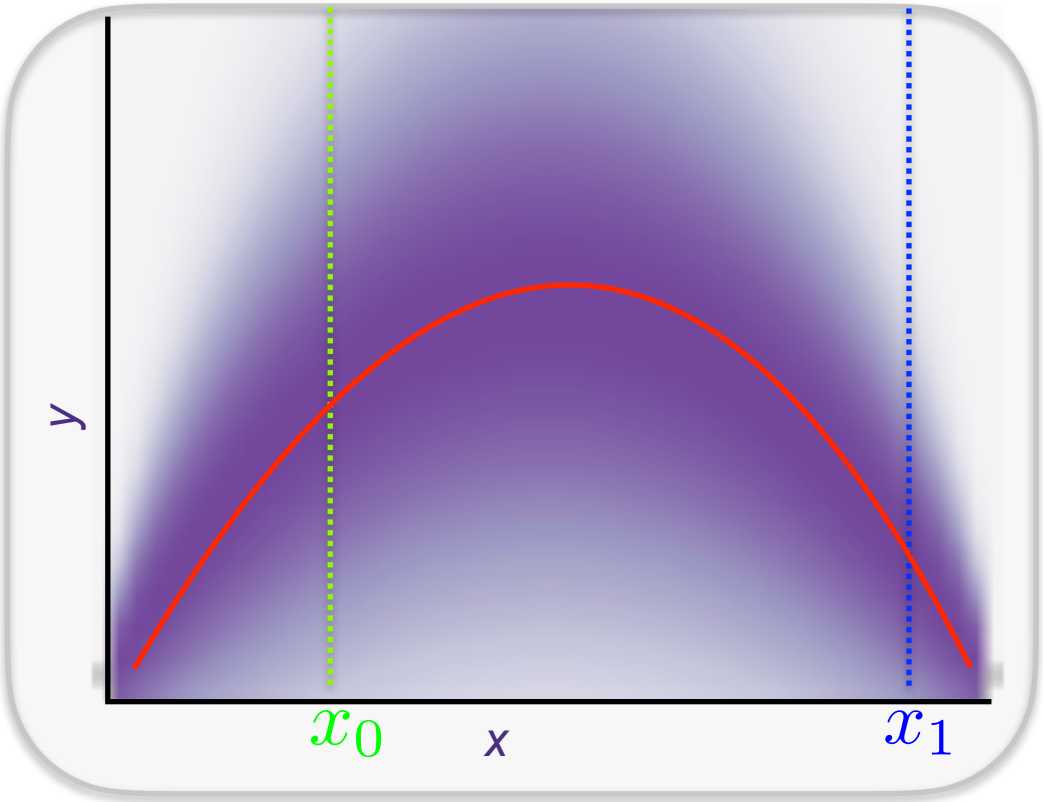
$$\mathbb{E}_{XY}[(Y - \eta(X))^2]$$



Statistical Learning

$$\mathbb{E}_{XY}[(Y - \eta(X))^2]$$

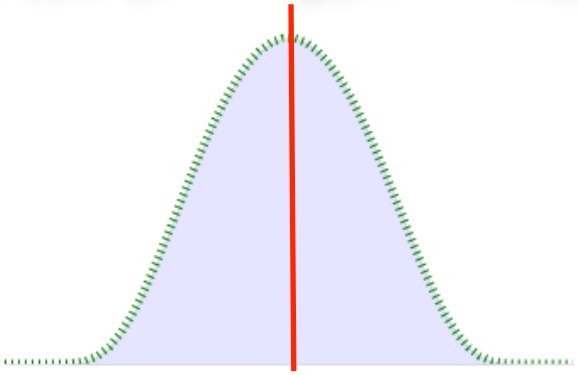
$P_{XY}(X = x, Y = y)$

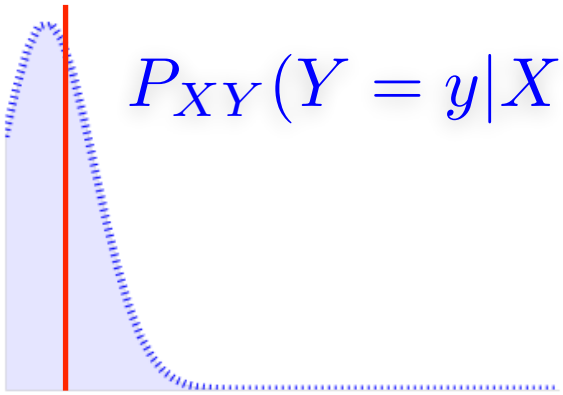


Ideally, we want to find:

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x]$$

$P_{XY}(Y = y|X = x_0)$

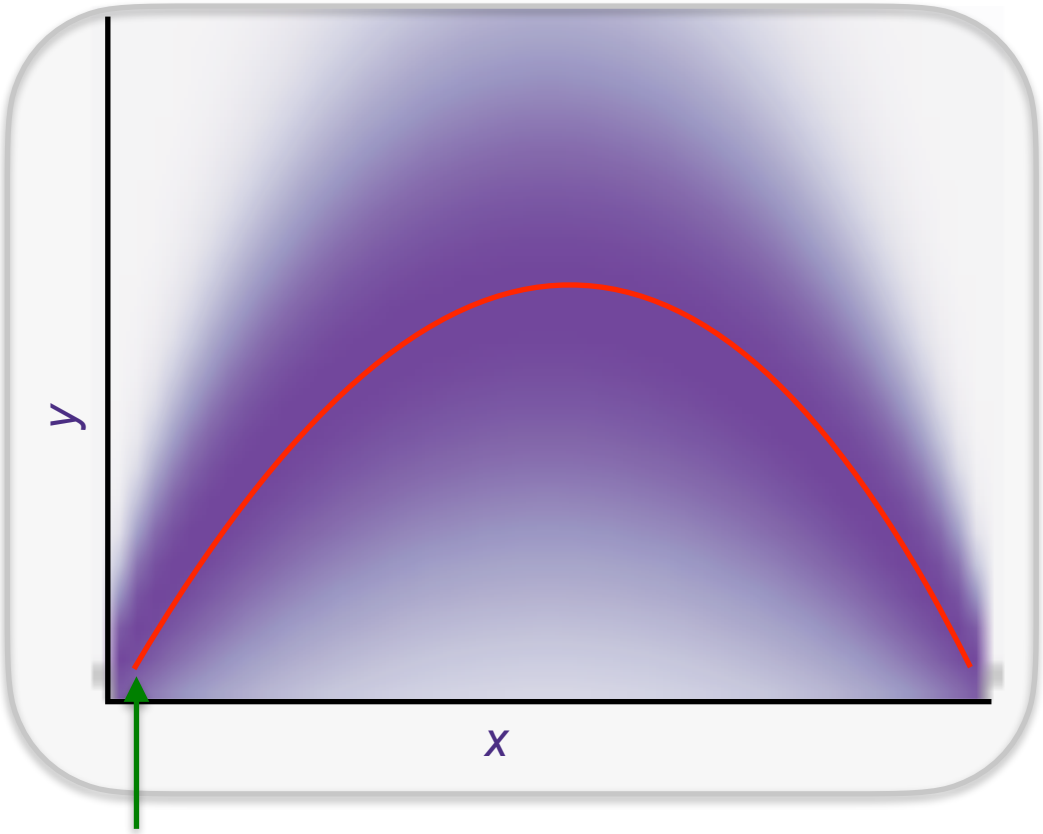




$P_{XY}(Y = y|X = x_1)$

Statistical Learning

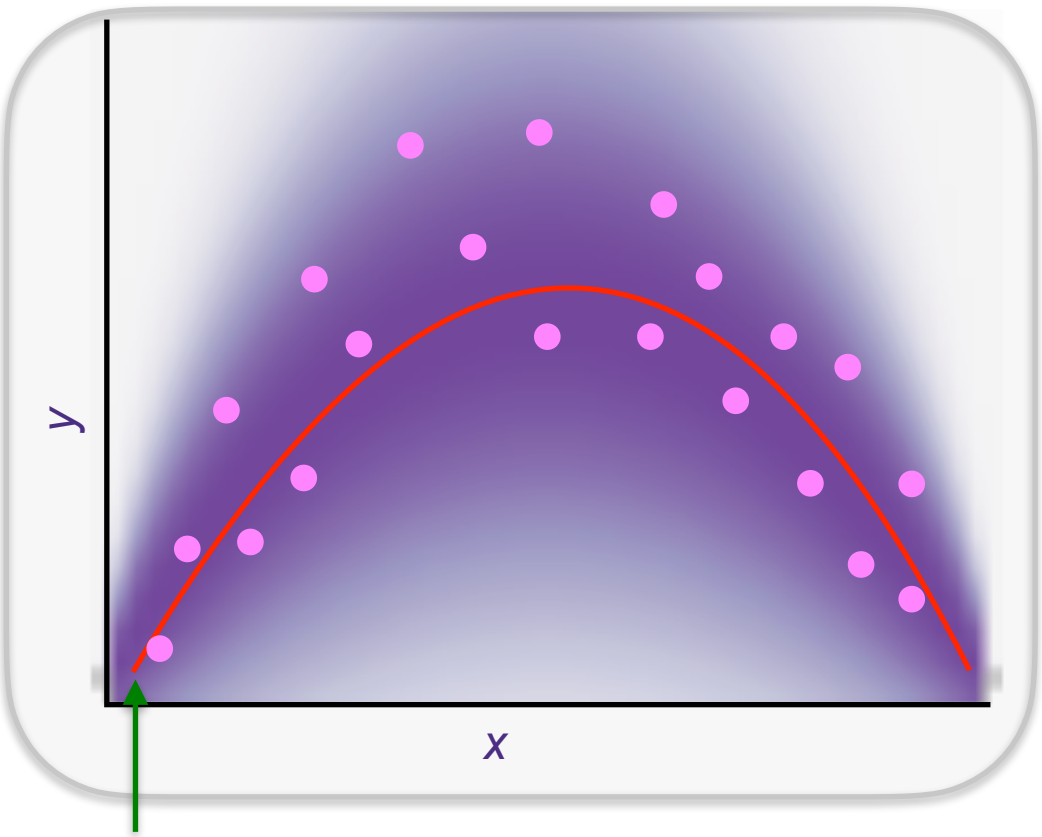
$P_{XY}(X = x, Y = y)$



Ideally, we want to find:

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x]$$

Statistical Learning



Ideally, we want to find:

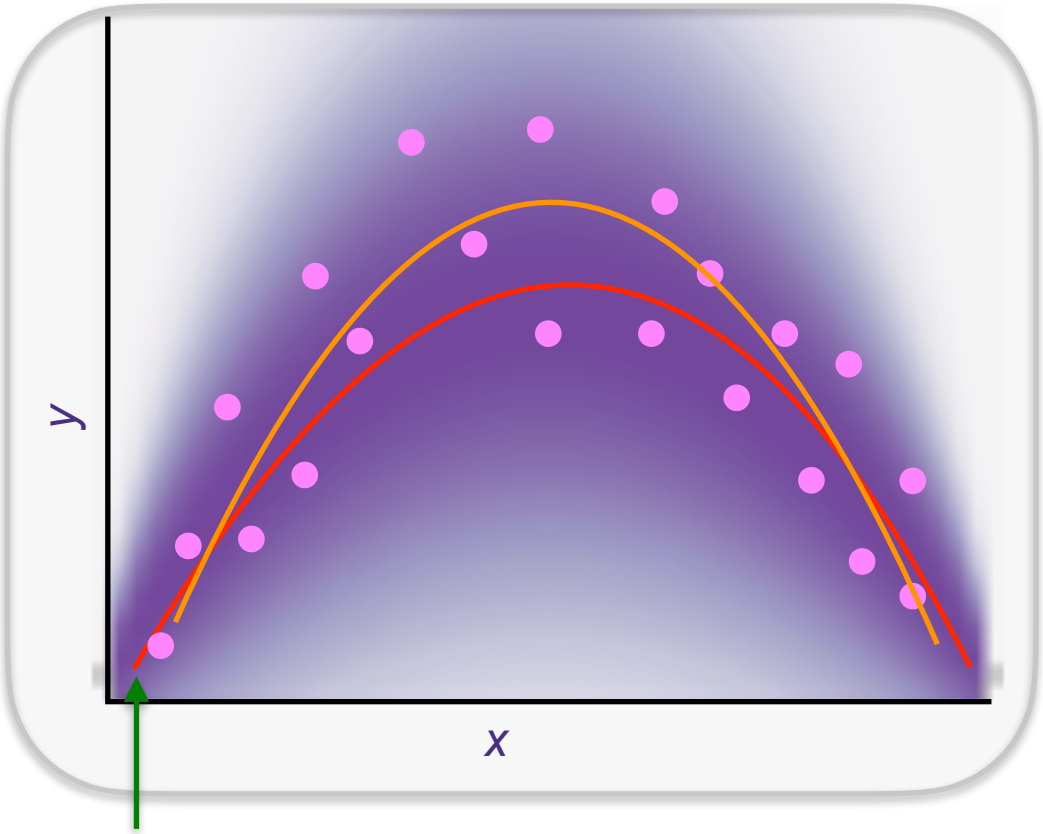
$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x]$$

But we only have samples:

$(x_i, y_i) \overset{i.i.d.}{\sim} P_{XY}$ for $i = 1, \ldots, n$

Statistical Learning

$P_{XY}(X = x, Y = y)$



η(x)

Ideally, we want to find:

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x]$$

But we only have samples:

$(x_i, y_i) \overset{i.i.d.}{\sim} P_{XY}$ for $i = 1, \ldots, n$

and are restricted to a function class (e.g., linear) so we compute:

$$\widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$\hat{f} \neq \eta(x)$ Why?

n is limited

could have chosen wrong model class *F*

Empirical Reconstruction Error (ERE)

What is the $\mathbb{E}[\hat{f}]$?

$$\mathbb{E}_{XY}[\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[(y_i - f(x_i))^2]$$ # By linearity of expectation

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[(y - f(x))^2]$$ by IID: $\mathbb{E}[x_i]$ for any $i = \mathbb{E}[x] \forall x$

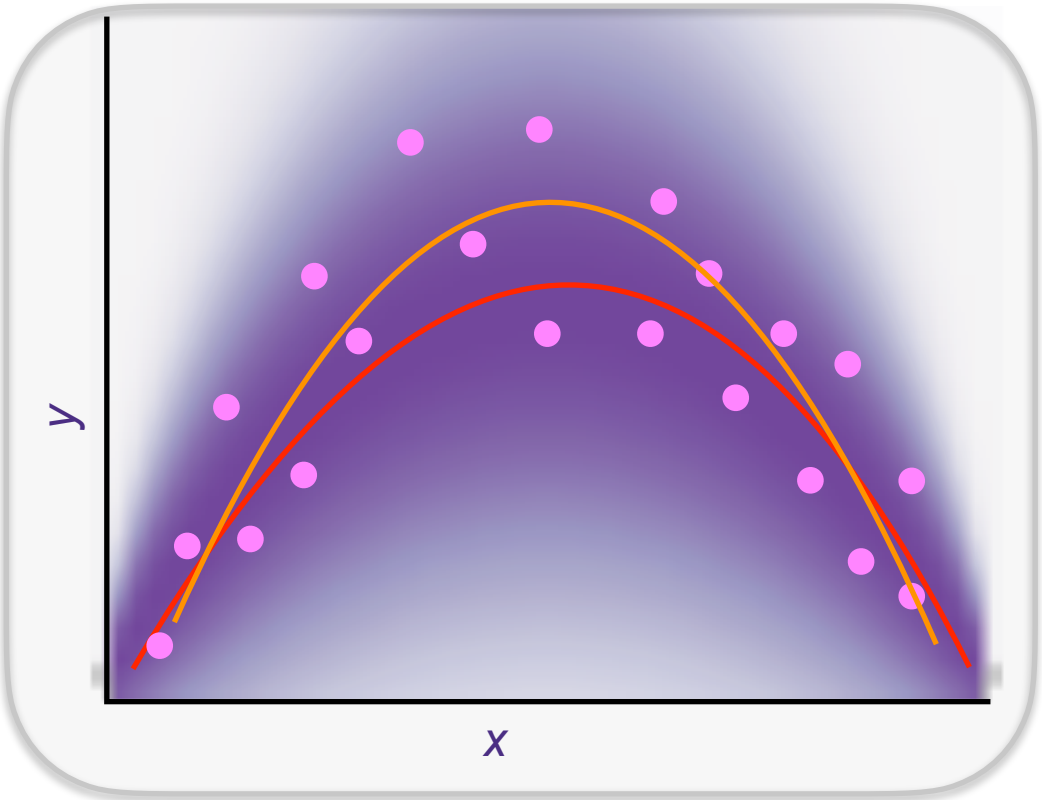
$$= \mathbb{E}[(y - f(x))^2]$$ # Yet again, it works out to the mean squared error!

Minimizing ERE → minimizing squared error loss in general

(in expectation….)

Statistical Learning

$P_{XY}(X = x, Y = y)$



Ideally, we want to find:

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x]$$

But we only have samples:

$(x_i, y_i) \overset{i.i.d.}{\sim} P_{XY}$ for $i = 1, \ldots, n$

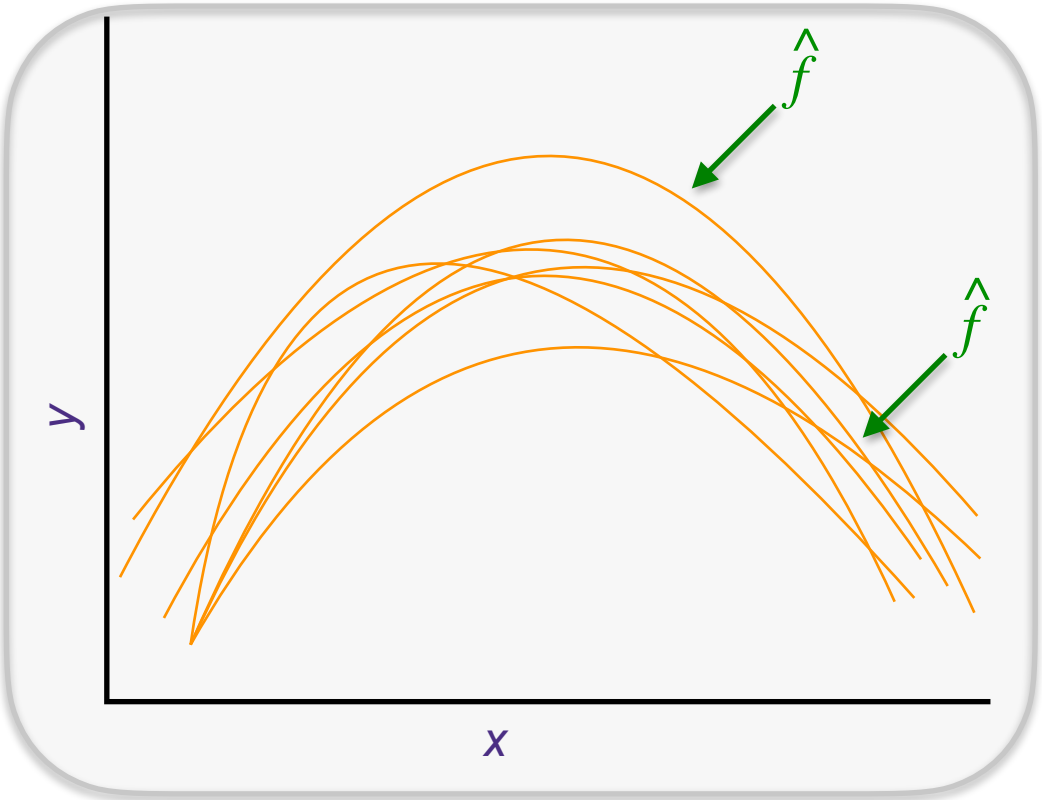
and are restricted to a function class (e.g., linear) so we compute:

$$\widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

We care about future predictions: $\mathbb{E}_{XY}[(Y - \widehat{f}(X))^2]$

Statistical Learning

$P_{XY}(X = x, Y = y)$



Ideally, we want to find:

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x]$$

But we only have samples:

$(x_i, y_i) \overset{i.i.d.}{\sim} P_{XY}$ for $i = 1, \ldots, n$

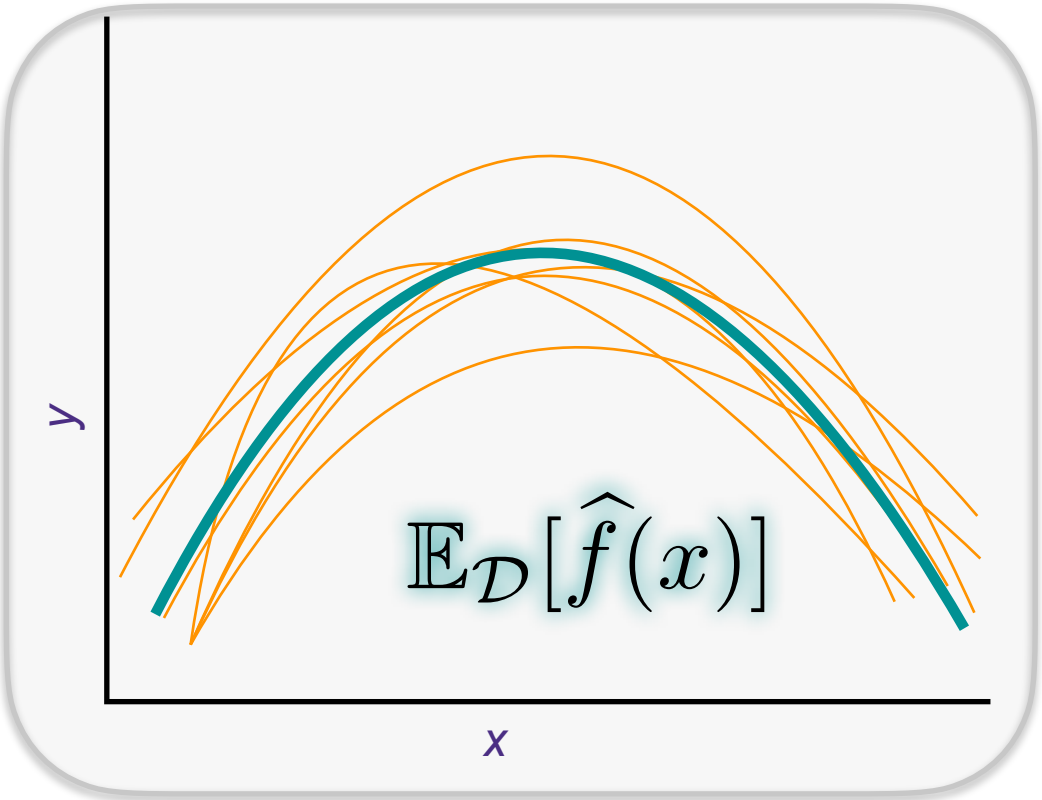
and are restricted to a function class (e.g., linear) so we compute:

$$\widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Each draw $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ results in different $\widehat{f}$

Statistical Learning

$P_{XY}(X = x, Y = y)$



Ideally, we want to find:

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x]$$

But we only have samples:

$(x_i, y_i) \overset{i.i.d.}{\sim} P_{XY}$ for $i = 1, \ldots, n$

and are restricted to a function class (e.g., linear) so we compute:

$$\widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Each draw $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ results in different $\widehat{f}$

Bias-Variance Tradeoff

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x] \qquad \widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$\mathbb{E}_{Y|X}[\mathbb{E}_{\mathcal{D}}[(Y - \widehat{f}_{\mathcal{D}}(x))^2] \big| X = x] = \mathbb{E}_{Y|X}[\mathbb{E}_{\mathcal{D}}[(Y - \eta(x) + \eta(x) - \widehat{f}_{\mathcal{D}}(x))^2] \big| X = x]$$

Bias-Variance Tradeoff

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x] \qquad \widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$\mathbb{E}_{Y|X}[\mathbb{E}_{\mathcal{D}}[(Y - \widehat{f}_{\mathcal{D}}(x))^2]\big|X = x] = \mathbb{E}_{Y|X}[\mathbb{E}_{\mathcal{D}}[(Y - \eta(x) + \eta(x) - \widehat{f}_{\mathcal{D}}(x))^2]\big|X = x]$$

$$= \mathbb{E}_{Y|X}\Big[\mathbb{E}_{\mathcal{D}}[(Y - \eta(x))^2 + 2(Y - \eta(x))(\eta(x) - \widehat{f}_{\mathcal{D}}(x)) + (\eta(x) - \widehat{f}_{\mathcal{D}}(x))^2]\big|X = x\Big]$$

$$= \mathbb{E}_{Y|X}[(Y - \eta(x))^2\big|X = x] + \mathbb{E}_{\mathcal{D}}[(\eta(x) - \widehat{f}_{\mathcal{D}}(x))^2]$$

irreducible error
Caused by stochastic label noise

learning error
Caused by either using too "simple" of a model or not enough data to learn the model accurately

Bias-Variance Tradeoff

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x] \qquad \widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$\mathbb{E}_{\mathcal{D}}[(\eta(x) - \widehat{f}_{\mathcal{D}}(x))^2] = \mathbb{E}_{\mathcal{D}}[(\eta(x) - \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] + \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] - \widehat{f}_{\mathcal{D}}(x))^2]$$

Bias-Variance Tradeoff

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x] \qquad \widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$\mathbb{E}_{\mathcal{D}}[(\eta(x) - \widehat{f}_{\mathcal{D}}(x))^2] = \mathbb{E}_{\mathcal{D}}[(\eta(x) - \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] + \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] - \widehat{f}_{\mathcal{D}}(x))^2]$$

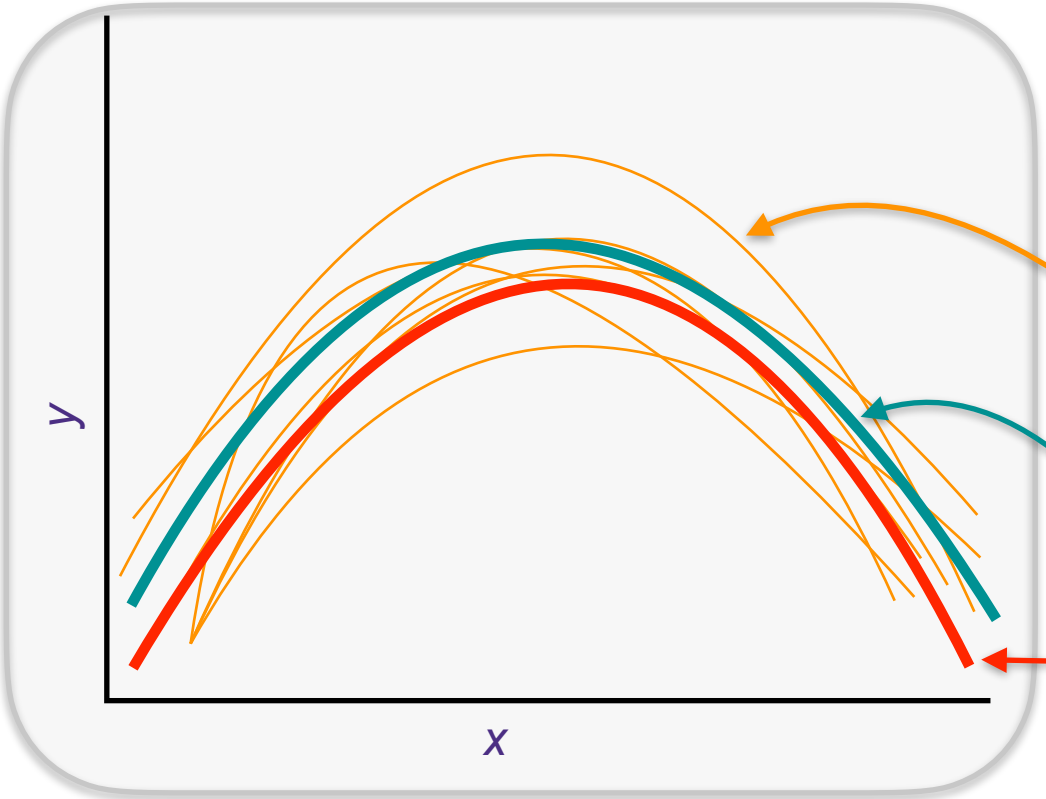
$$= \mathbb{E}_{\mathcal{D}}[(\eta(x) - \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)])^2 + 2(\eta(x) - \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)])(\mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] - \widehat{f}_{\mathcal{D}}(x))$$
$$+ (\mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] - \widehat{f}_{\mathcal{D}}(x))^2]$$

$$= (\eta(x) - \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)])^2 + \mathbb{E}_{\mathcal{D}}[(\mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] - \widehat{f}_{\mathcal{D}}(x))^2]$$

biased squared **variance**

Statistical Learning

$$\underbrace{\mathbb{E}_{\mathcal{D}}[(\eta(x) - \widehat{f}_{\mathcal{D}}(x))^2]}_{\text{learning error}} = \underbrace{(\eta(x) - \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)])^2}_{\text{biased squared}} + \underbrace{\mathbb{E}_{\mathcal{D}}[(\mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] - \widehat{f}_{\mathcal{D}}(x))^2]}_{\text{variance}}$$



$$\widehat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$\mathbb{E}_{\mathcal{D}}[\widehat{f}(x)]$$

$$\eta(x) = \mathbb{E}_{Y|X}[Y|X = x]$$

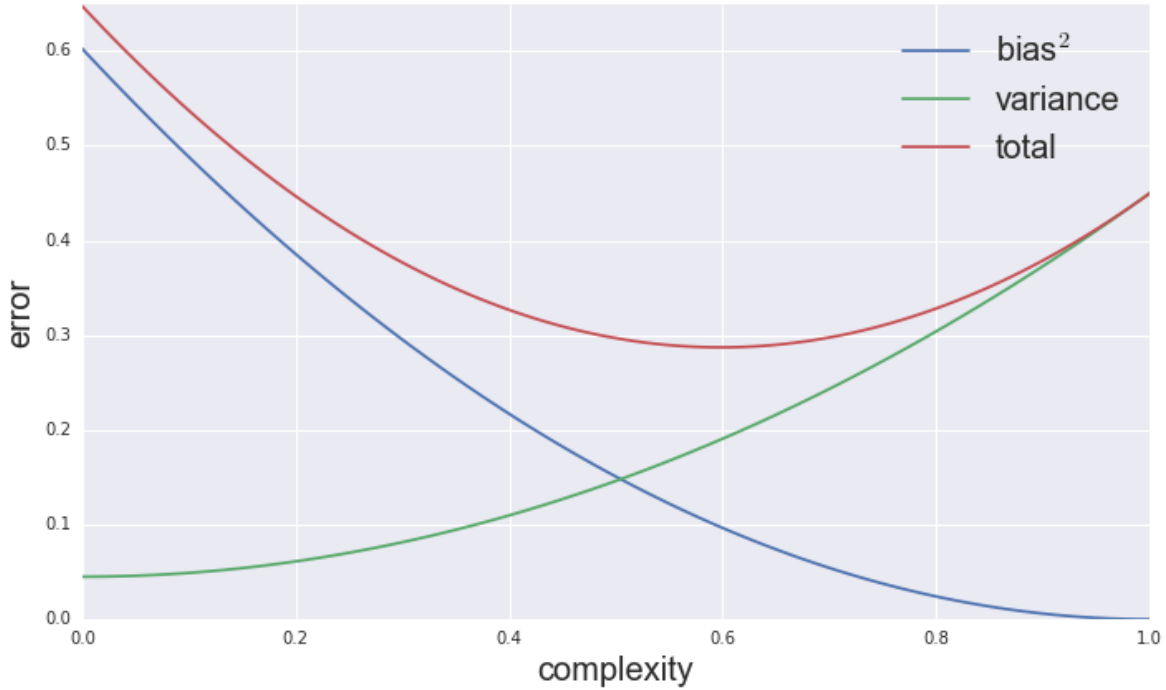
Bias-Variance Tradeoff

$$\mathbb{E}_{Y|X}[\mathbb{E}_{\mathcal{D}}[(Y - \widehat{f}_{\mathcal{D}}(x))^2]\big|X = x] = \mathbb{E}_{Y|X}[(Y - \eta(x))^2\big|X = x]$$

irreducible error

$$+(\eta(x) - \mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)])^2 + \mathbb{E}_{\mathcal{D}}[(\mathbb{E}_{\mathcal{D}}[\widehat{f}_{\mathcal{D}}(x)] - \widehat{f}_{\mathcal{D}}(x))^2]$$

biased squared **variance**



Bias-Variance Demo

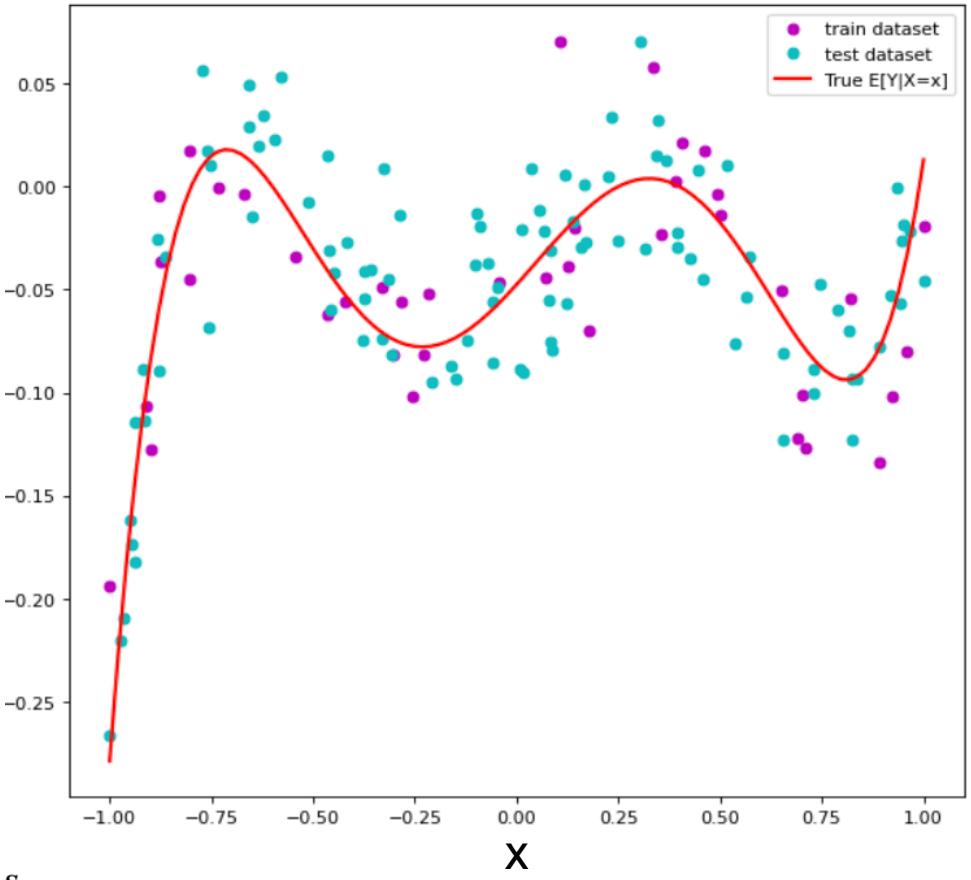
See colab notebook

Bias-variance tradeoff

Let's define a ground truth $\eta(x) = E[Y|X = x]$ to be a degree-5 polynomial.
Then, define $P(Y|X = x) = \mathcal{N}(x, \sigma^2)$

In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt

# Define \eta = E[Y|X] as a 5-th order polynomial
# (this is the same function we used in demo_polynomial.ipynb)
eta = lambda x: (x-.99)*(x-.4)*(x-.25)*(x+.6)*(x+.8)

# Generate samples from P(X,Y)
def sample_Pxy(n, noise_sigma=0.03):

    # Sample inputs x from P(x).
    x = np.random.uniform(-1,1,n)
    x = np.sort(x)
    x[0]=-1
    x[n-1]=1

    # Draw samples from P(Y|X)
    y = eta(x) + noise_sigma*np.random.randn(n)

    return x,y

# Generate training data.
n_train = 40 # sample size
x, y = sample_Pxy(n=n_train)

# Generate test data.
n_test = 100
x_, y_ = sample_Pxy(n=n_test)

# Plot the samples and ground truth.
t = np.linspace(-1,1,100)
y0 = eta(t)
fig=plt.figure(figsize=(9, 8), dpi= 80, facecolor='w', edgecolor='k')
plt.plot(x,y,'mo',label='train dataset')
plt.plot(x_,y_,'co',label='test dataset')
plt.plot(t,y0,'r-', linewidth=2, label='True E[Y|X=x]')
plt.legend()
plt.show()
```



In typical scenarios, we only have one set of samples, which we separate into S_{test} and S_{train}

- it is critical that those two sets do not overlap and the sets are chosen randomly to ensure that the test error is independent of the training error, and also the test samples are coming frmo the same distribution as the new samples that will come in the future

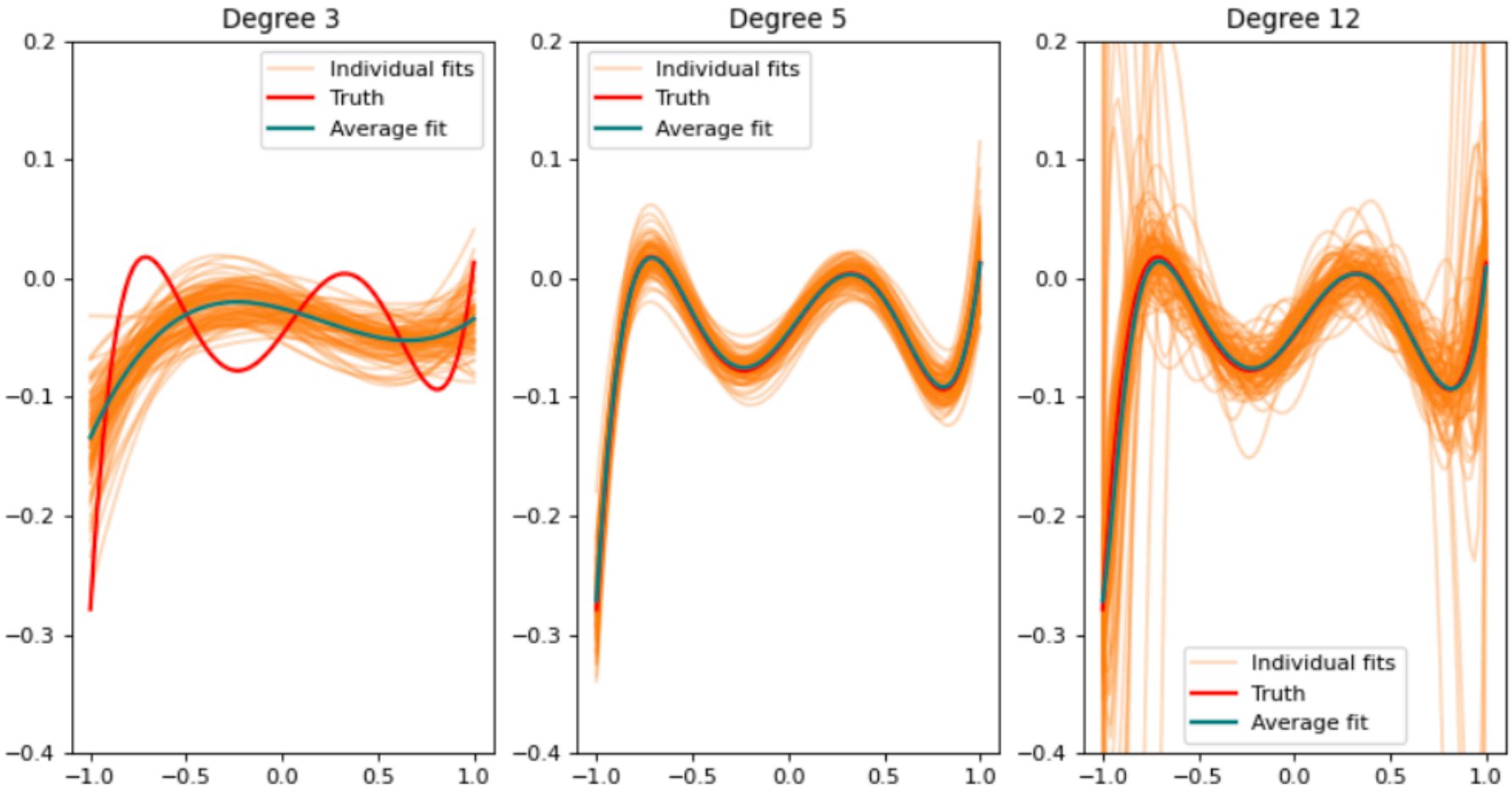
However, in order to understand how the test error behaves (theoretically), we consider the expected test error, and call it true error: i.e. $\mathcal{L}_{\text{true}} = \mathbb{E}[\mathcal{L}_{\text{test}}]$

- test error is an **unbiased** estimate of the true error
- true error is unobservable (we canot copmute it, given finite samples)
- but we care the most about the true error
- so we use test error as a surrogate or an approximation

In order to compute the true error, we simulate a process where we get many fresh samples, and train new predictor each time with the fresh set of samples. It is important to understand that the resulting predictor $f_{S_{\text{train}}}(\cdot)$ is a random function, where the randomness comes from the fresh random training data set S_{train}. We will draw many such random functions, plot them, and see how test, train, true errors behave.

In [2]:

```
num_runs = 100
yhat_simple = np.zeros((num_runs,len(t)))
yhat_complex = np.zeros((num_runs,len(t)))
yhat_justright = np.zeros((num_runs,len(t)))

def poly_features(x, degree):
    """
    Generates a matrix where each column corresponds to x raised to a power from 0 to 'degree'.

    Parameters:
    x (numpy array): The input data.
    degree (int): The maximum degree of the polynomial features.

    Returns:
    numpy array: A matrix where the columns are [1, x, x^2, ..., x^degree].
    """
    return np.vstack([x**i for i in range(degree + 1)]).T

run=0
while run < num_runs:
    n = 40 # sample size
    x,y = sample_Pxy(n)

    # degree-3 polynomial linear regression
    p=3
    X = poly_features(x, degree=p)
    w = np.matmul(np.matmul(np.linalg.inv(np.matmul(X.T,X)),X.T),y)
    T_ = poly_features(t, degree=p)
    yhat_simple[int(run)] = np.matmul(T_,w)
    yh = np.matmul(X,w)
    X_ = poly_features(x_, degree=p)
    y_h= np.matmul(X_,w)
    w_ = w

    # degree-5 polynomial linear regression
    p=5
    X = poly_features(x, degree=p)
    w = np.matmul(np.matmul(np.linalg.inv(np.matmul(X.T,X)),X.T),y)
    T_ = poly_features(t, degree=p)
    yhat_justright[int(run)] = np.matmul(T_,w)
    yh = np.matmul(X,w)
    X_ = poly_features(x_, degree=p)
    y_h= np.matmul(X_,w)
    w_ = w

    # degree-12 polynomial linear regression
    p=12
    X = poly_features(x, degree=p)
    w = np.array(p+1)
    w = np.matmul(np.matmul(np.linalg.inv(np.matmul(X.T,X)),X.T),y)
    T_ = poly_features(t, degree=p)
    yhat_complex[int(run)] = np.matmul(T_,w)
    yh1 = np.matmul(X,w)
    X_ = poly_features(x_, degree=p)
    y_h1= np.matmul(X_,w)

    run=run+1

def build_plot(yhat, title):
    ave_fit = np.zeros(np.size(t))
    for irun in range(1, num_runs):
        plt.plot(t,yhat[int(irun)], color='tab:orange',alpha=0.3)
        ave_fit = ave_fit + yhat[int(irun)]
    plt.plot(t,yhat[0],'tab:orange',alpha=0.3,label='Individual fits')
    plt.plot(t,y0,'r-', linewidth=2, label='Truth')
    plt.plot(t,(1/float(num_runs))*ave_fit, color='teal', linewidth=2,label='Average fit')
    plt.legend()
    plt.title(title)
    axes = plt.gca()
    axes.set_ylim([-0.4,0.2])

fig=plt.figure(figsize=(12, 6), dpi= 80, facecolor='w', edgecolor='k')
plt.subplot(1,3,1)
build_plot(yhat_simple, 'Degree 3')
plt.subplot(1,3,2)
build_plot(yhat_justright, 'Degree 5')
plt.subplot(1,3,3)
build_plot(yhat_complex, 'Degree 12')
plt.show()

```

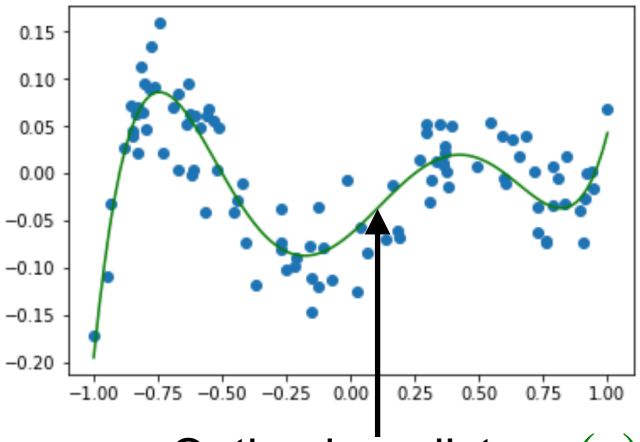


Takeaways

- True function has degree 5, with additive noise
- Degree 3 fit has very high bias because the teal line, which is the average of the fits, is very different from true red line (because 3 < 5). Each individual fit (orange) varies about the average fit (teal) moderately indicating moderate variance. The overall error (bias^2 + variance) of each individual fit is high.
- Degree 12 has very low bias because the teal line is almost equal to the red (because 12 > 5). But each individual fit varies a lot about its average teal line indicating high variance. The overall error (bias^2 + variance) of each individual fit is high.
- Degree 5 has very low bias because the teal line is almost equal to the red (because 5 = 5). And each individual fit varies only a little about its average teal line indicating small variance. The overall error (bias^2 + variance) of each individual fit is low.

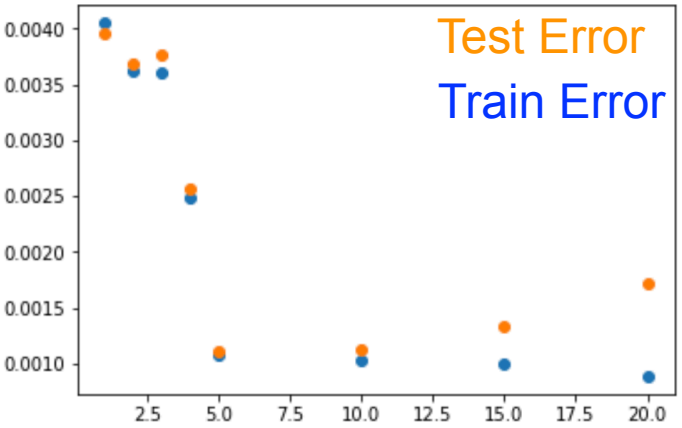
Overfitting

Test error vs. model complexity



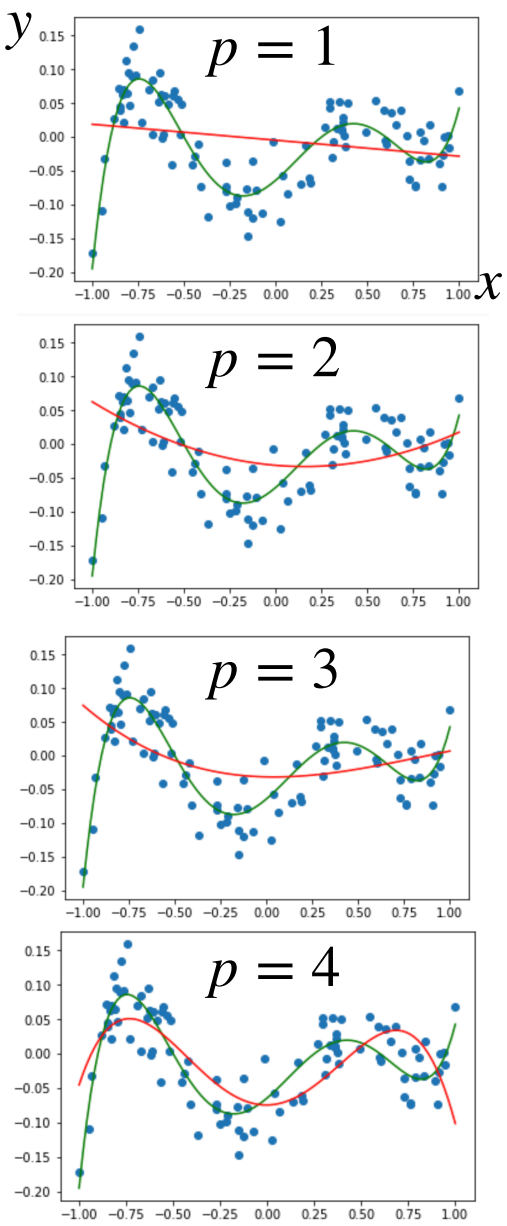
Optimal predictor $\eta(x)$ is degree-5 polynomial

Error

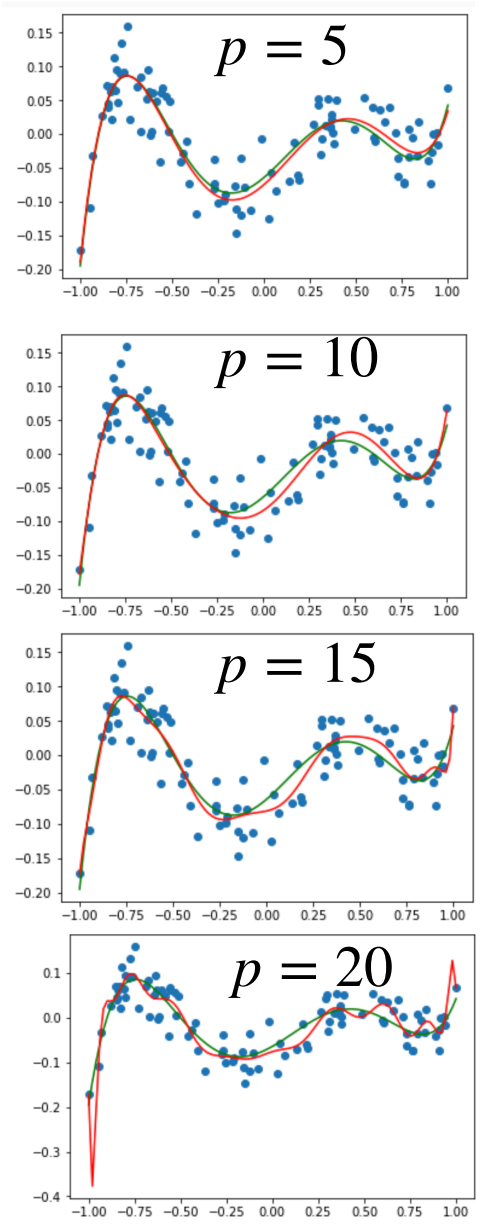


degree p of the polynomial regression

Simple model: # bias
Model complexity is below the complexity of $\eta(x)$

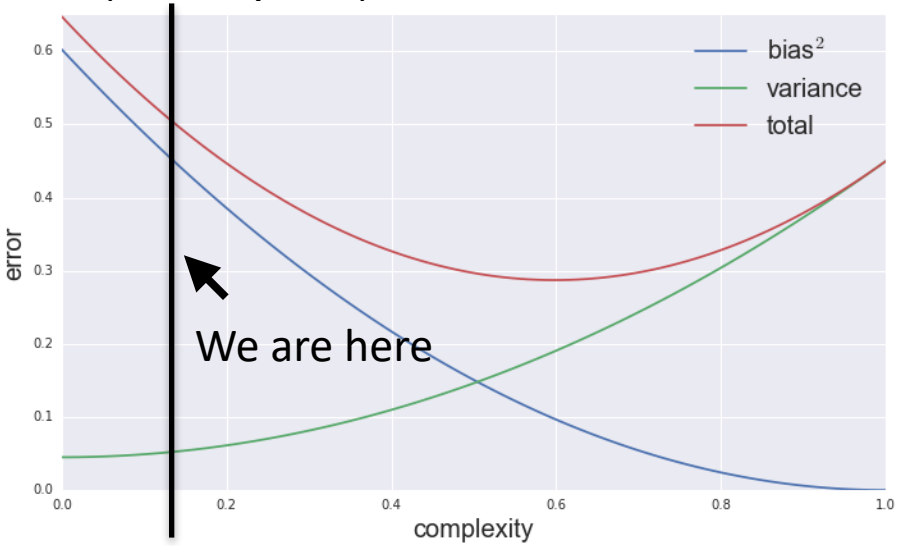


Complex model:
Fits noise in train data, diverging from $\eta(x)$

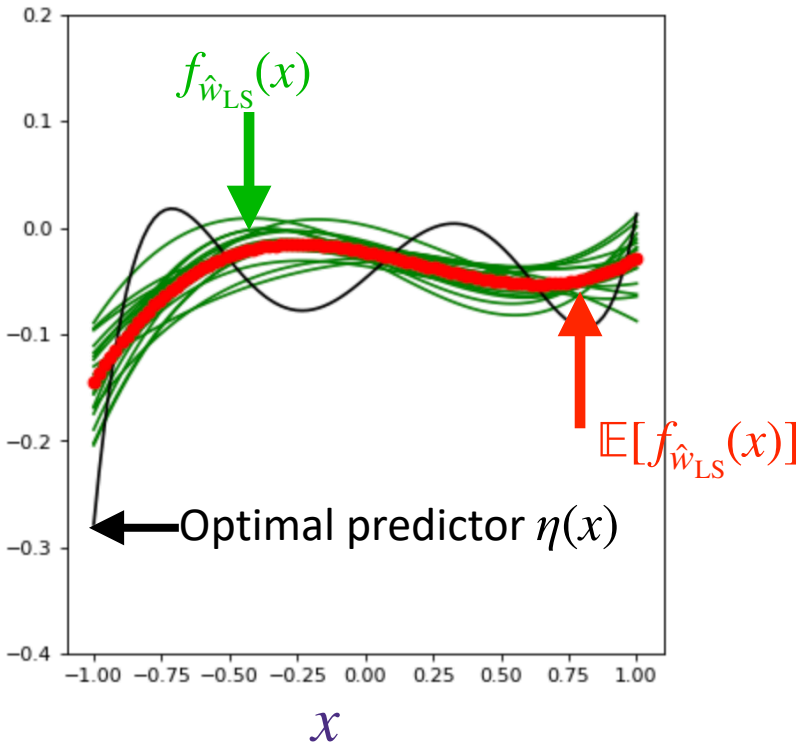


Recap: Bias-variance tradeoff with simple model

(Conceptual) bias variance tradeoff



With degree-3 polynomials, we underfit



- When model **complexity is low** (lower than the optimal predictor $\eta(x)$)
  - Bias^2 of our predictor, $\left(\eta(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)]\right)^2$

    # Large
  - Variance of our predictor, $\mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)] - \hat{f}_{\mathcal{D}}(x)\right)^2\right]$

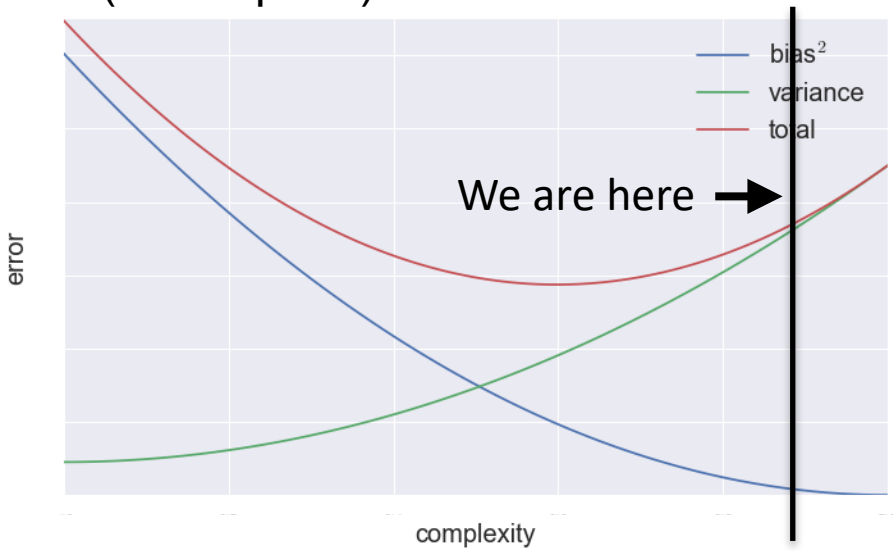
    # Small

- **If we have more samples (larger n), then**
  - What happens to bias? # stays the same
  - What happens to variance? # decreases
  - What happens to overall test error?

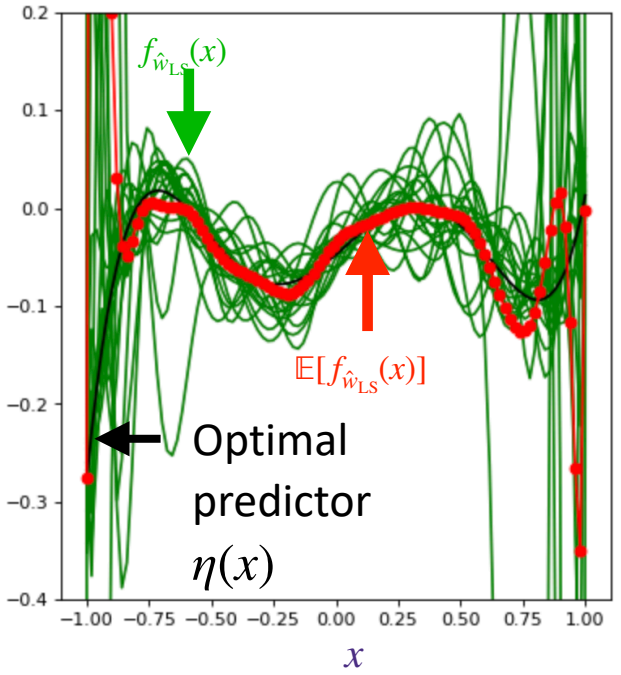
    # Stays high because bias was dominating our error term

Recap: Bias-variance tradeoff with complex model

(Conceptual) bias variance tradeoff



With degree-20 polynomials, we overfit



- When model **complexity is high** (higher than the optimal predictor $\eta(x)$)
  - Bias^2 of our predictor,
    $\left(\eta(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)]\right)^2$

    # Low
  - Variance of our predictor,
    $\mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)] - \hat{f}_{\mathcal{D}}(x)\right)^2\right]$

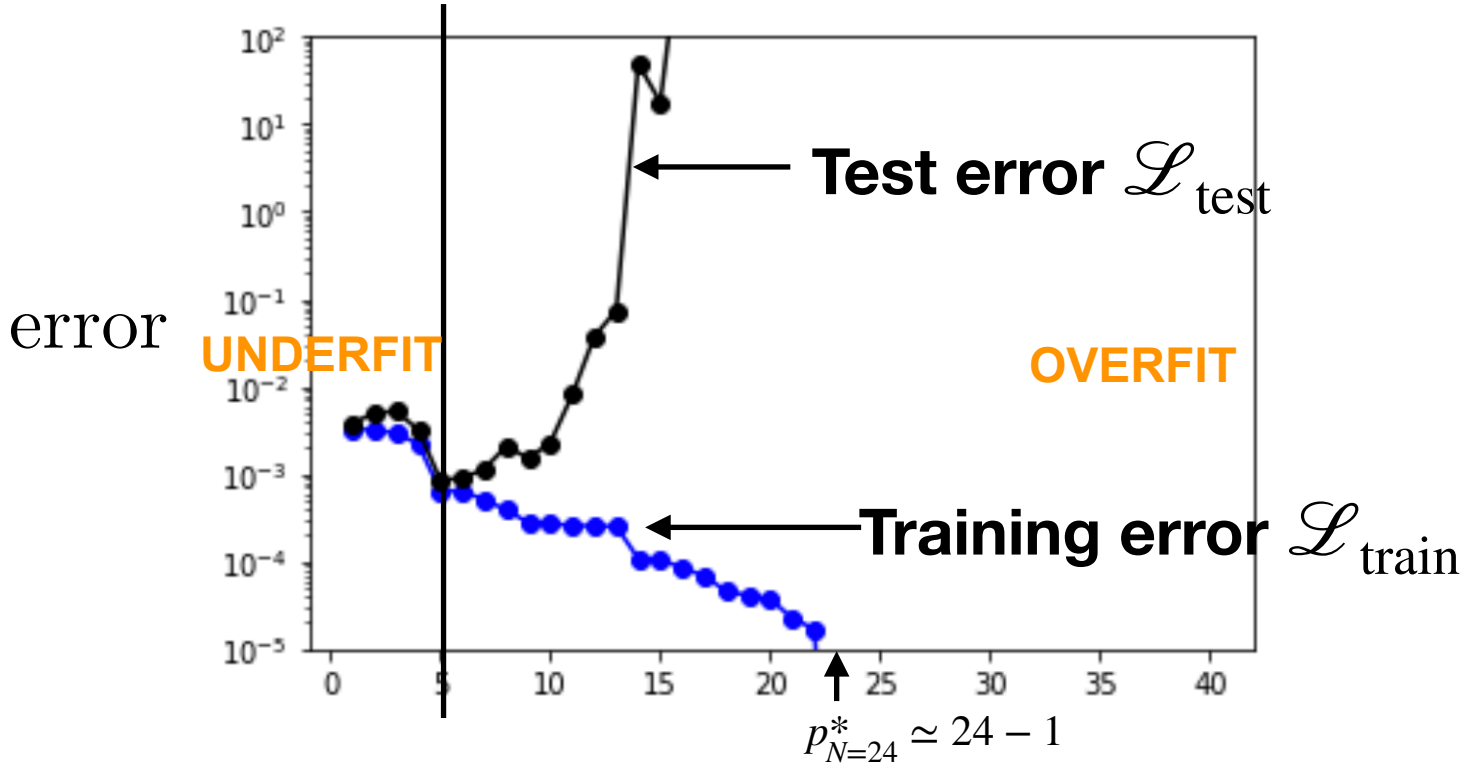
    # High

- **If we have more samples (larger n), then**
  - What happens to bias? # stays the same
  - What happens to variance? # decreases
  - What happens to overall test error?

    # decreases

Optimal model complexity depends on dataset size

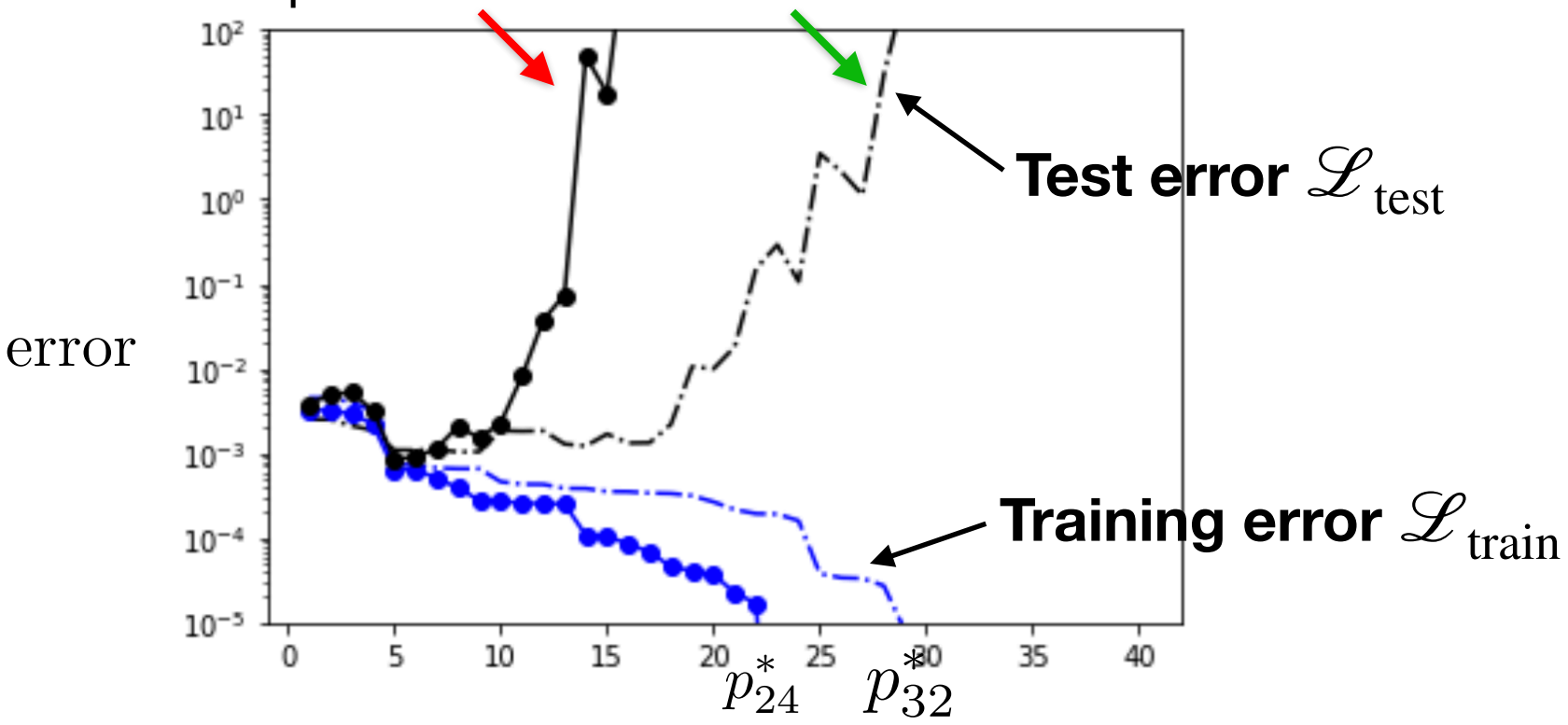
- Assume ***N=30***



- Given sample size N there is a threshold, p^*_N, where training error is zero
- Training error is **always** monotonically non-increasing
- Test error has a trend of going down and then up, but fluctuates

Variance decreases with more data, letting you fit more complex models

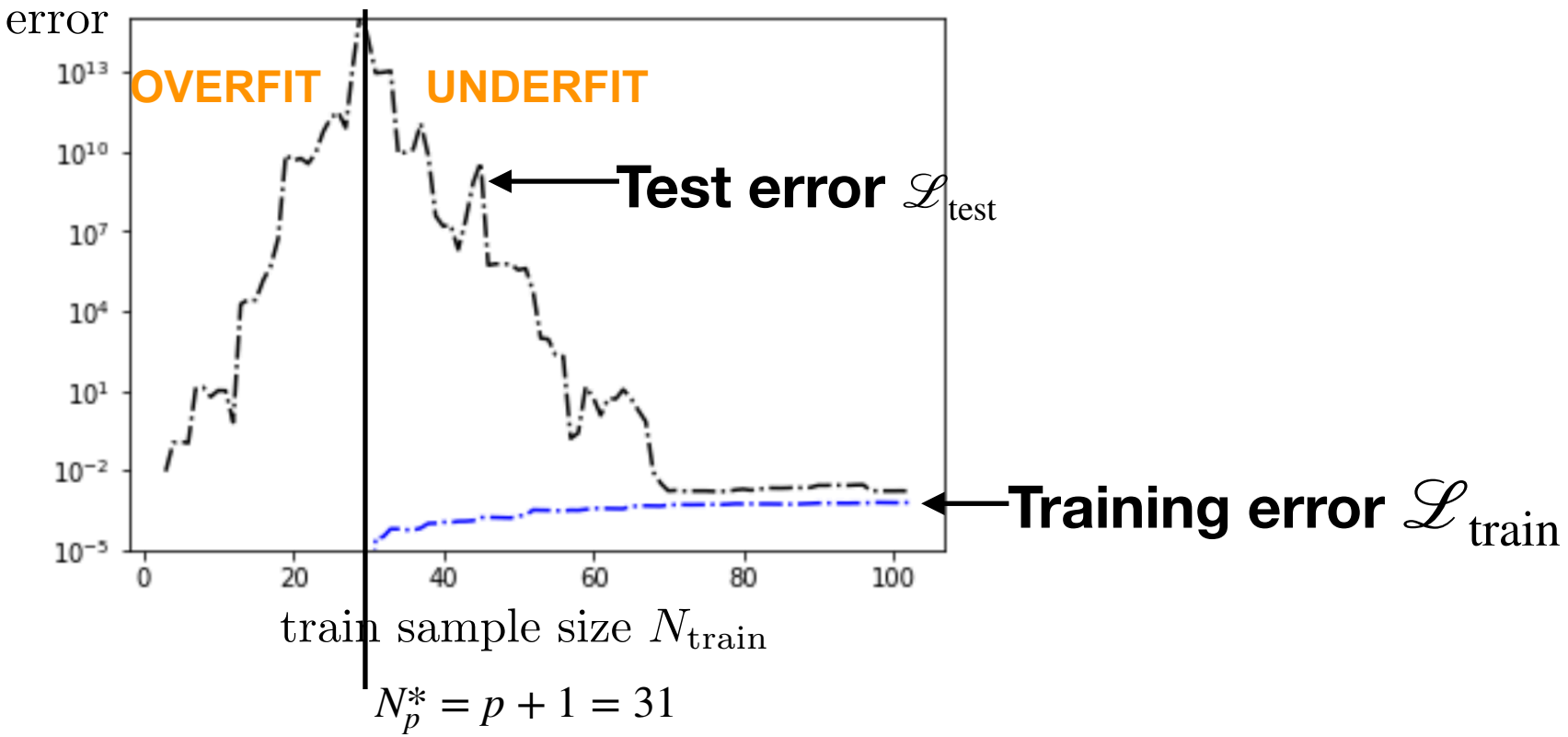
- Now compare ***N=30*** *case to* ***N=40***



- **The threshold, p_N^*, moves right as dataset size increases**
- Training error tends to increase, because more points need to fit
- Test error tends to decrease, because Variance decreases

Variance decreases with more data, letting you fit more complex models

- Choose model complexity ***p=30***, **vary dataset size n**



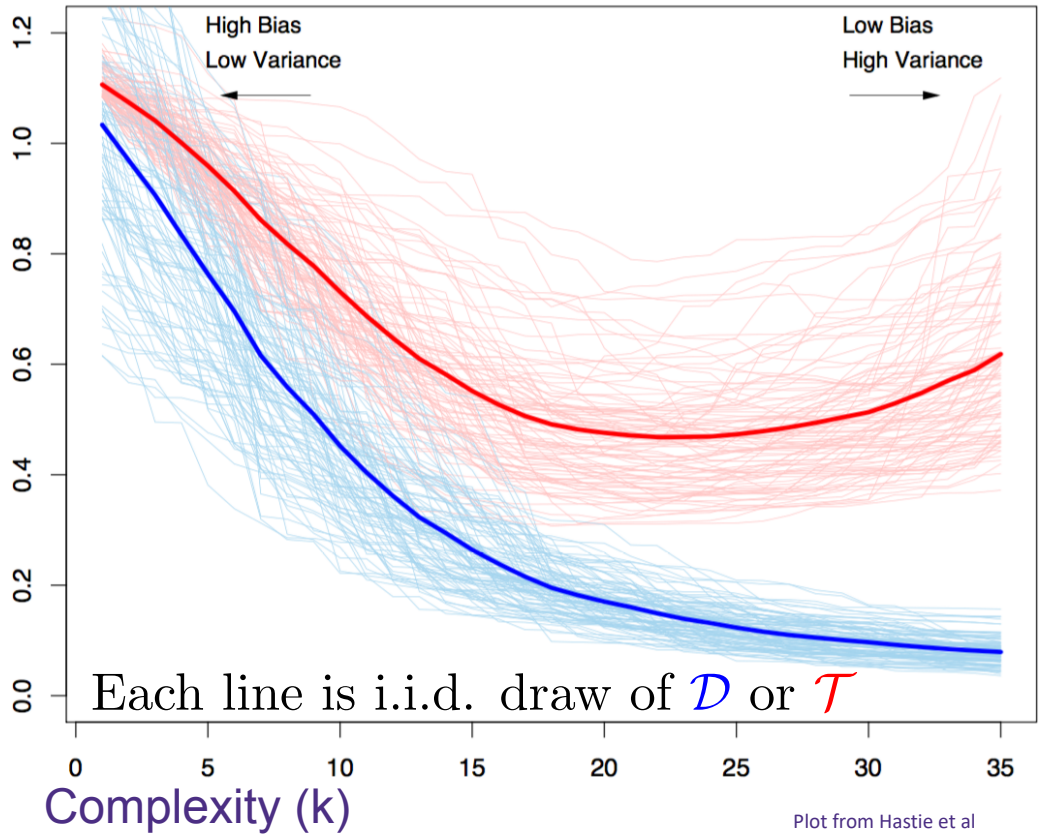
- There is a threshold, N_p^*, below which training error is zero (extreme overfit)
- Above the threshold, test error tends to decrease
- Training error tends to increase (harder to fit so much data with simple model)

Regularization helps avoid overfitting

Training set error as a function of model complexity

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \dots$$

$$\widehat{f}_{\mathcal{D}}^{(k)} = \arg\min_{f \in \mathcal{F}_k} \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} (y_i - f(x_i))^2$$



Plot from Hastie et al

TRAIN error:

$$\mathcal{D} \overset{i.i.d.}{\sim} P_{XY}$$

$$\frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} (y_i - \widehat{f}_{\mathcal{D}}^{(k)}(x_i))^2$$

TRUE error:

$$\mathbb{E}_{XY}[(Y - \widehat{f}_{\mathcal{D}}^{(k)}(X))^2]$$

TEST error:

$$\mathcal{T} \overset{i.i.d.}{\sim} P_{XY}$$

$$\frac{1}{|\mathcal{T}|} \sum_{(x_i, y_i) \in \mathcal{T}} (y_i - \widehat{f}_{\mathcal{D}}^{(k)}(x_i))^2$$

Important: $\mathcal{D} \cap \mathcal{T} = \emptyset$

Training set error as a function of model complexity

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \dots$$

$$\widehat{f}_{\mathcal{D}}^{(k)} = \arg\min_{f \in \mathcal{F}_k} \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} (y_i - f(x_i))^2$$

TRAIN error is **optimistically biased** because it is evaluated on the data it trained on. **TEST error** is **unbiased** only if *T* is never used to train the model or even pick the complexity k.

TRAIN error:

$$\mathcal{D} \overset{i.i.d.}{\sim} P_{XY}$$

$$\frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} (y_i - \widehat{f}_{\mathcal{D}}^{(k)}(x_i))^2$$

TRUE error:

$$\mathbb{E}_{XY}[(Y - \widehat{f}_{\mathcal{D}}^{(k)}(X))^2]$$

TEST error:

$$\mathcal{T} \overset{i.i.d.}{\sim} P_{XY}$$

$$\frac{1}{|\mathcal{T}|} \sum_{(x_i, y_i) \in \mathcal{T}} (y_i - \widehat{f}_{\mathcal{D}}^{(k)}(x_i))^2$$

Important: $\mathcal{D} \cap \mathcal{T} = \emptyset$

Test set error

- Given a dataset, randomly split it into two parts:
  - Training data: $\mathcal{D}$
  - Test data: $\mathcal{T}$

Important: $\mathcal{D} \cap \mathcal{T} = \emptyset$

- Use training data to learn predictor
  - e.g., $\frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} (y_i - \widehat{f}_{\mathcal{D}}^{(k)}(x_i))^2$
  - use training data to pick complexity k

- Use test data to report predicted performance

$$\frac{1}{|\mathcal{T}|} \sum_{(x_i, y_i) \in \mathcal{T}} (y_i - \widehat{f}_{\mathcal{D}}^{(k)}(x_i))^2$$

Ridge Regression

Regularization in Linear Regression

Recall Least Squares:

$$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

$$= \arg\min_{w} (\mathbf{y} - \mathbf{X}w)^T (\mathbf{y} - \mathbf{X}w)$$

when $(\mathbf{X}^T\mathbf{X})^{-1}$ exists….

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

Regularization in Linear Regression

Recall Least Squares:

$$\begin{aligned}\widehat{w}_{LS} &= \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 \\ &= \arg\min_w (\mathbf{y} - \mathbf{X}w)^T(\mathbf{y} - \mathbf{X}w) \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\end{aligned}$$

when $(\mathbf{X}^T\mathbf{X})^{-1}$ exists….

What if $x_i \in \mathbb{R}^d$ and $d > n$?

Regularization in Linear Regression

Recall Least Squares: $$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

When $x_i \in \mathbb{R}^d$ and $d > n$ the objective function is flat in some directions:

Regularization in Linear Regression

Recall Least Squares: $\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$

When $x_i \in \mathbb{R}^d$ and $d > n$ the objective function is flat in some directions:

Implies optimal solution is *not unique* and unstable due to lack of curvature*:*

- small changes in training data result in large changes in solution
- often the *magnitudes* of *w* are "very large"

Regularization imposes "simpler" solutions by a "complexity" penalty

Sensitivity increases overfitting

- For a linear model,
  $$y \simeq b + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$
  if $|w_j|$ is large then the prediction is **sensitive** to small changes in x_j
- Large ***sensitivity*** leads to overfitting and poor generalization, and equivalently models that overfit tend to have large weights
- Note that b is a constant and hence there is no sensitivity for the offset b

Never regularize b

- In ***Ridge Regression***, we use a regularizer $\|w\|_2^2$ to measure and control the sensitivity of the predictor
- And optimize for small loss and small sensitivity, by adding a ***regularizer*** in the objective (assume no offset for now)

$$\hat{w}_{\text{ridge}} = \arg\min_w \left\{ \sum_{i=1}^{n} (y_i - x_i^T w)^2 + \lambda \|w\|_2^2 \right\}$$

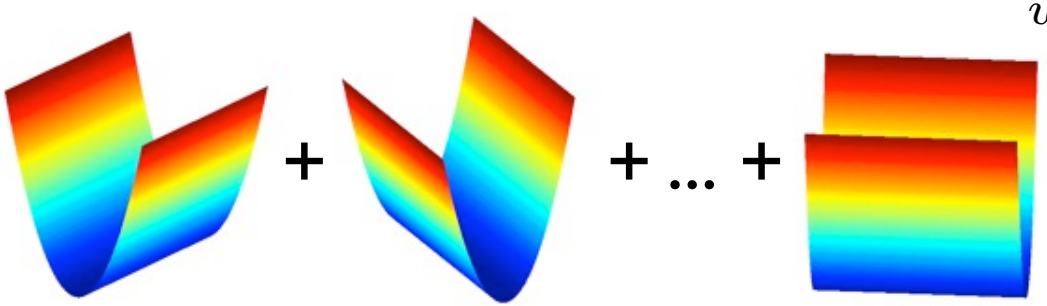
Regularize to make weights smaller / less sensitive. Multiple ways to do that. We'll start with L2 norm

Ridge Regression

if d > n

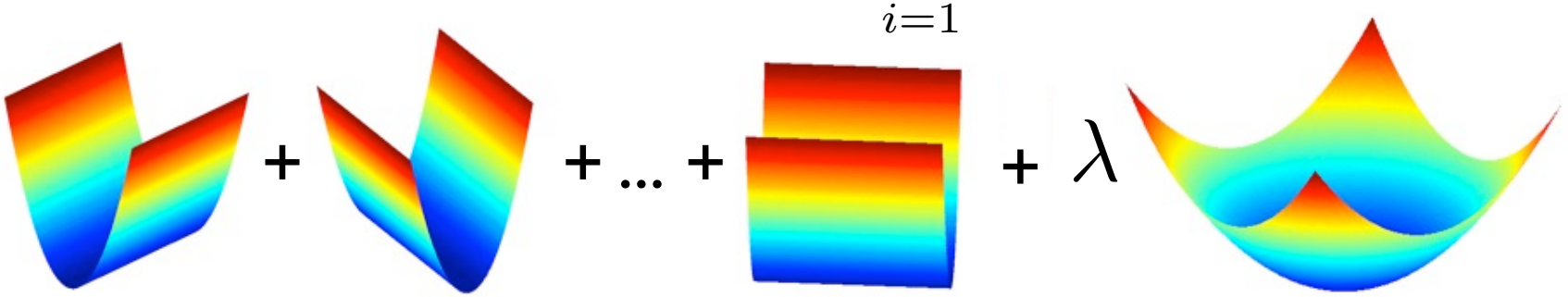
- Old Least squares objective:

$$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$



- Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$



Minimizing the Ridge Regression Objective

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

$w \in \mathbb{R}^{d \times 1}$

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

More identities

$$\|w\|_p = (|w_1|^p + \cdots + |w_d|^p)^{\frac{1}{p}}$$

$$I^d = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \ddots & \\ & & 1 \end{bmatrix}$$

Scalar derivative			Vector derivative		
$f(x)$	$\rightarrow$	$\frac{df}{dx}$	$f(\mathbf{x})$	$\rightarrow$	$\frac{df}{d\mathbf{x}}$
bx	$\rightarrow$	b	$\mathbf{x}^T\mathbf{B}$	$\rightarrow$	$\mathbf{B}$
bx	$\rightarrow$	b	$\mathbf{x}^T\mathbf{b}$	$\rightarrow$	$\mathbf{b}$
x^2	$\rightarrow$	$2x$	$\mathbf{x}^T\mathbf{x}$	$\rightarrow$	$2\mathbf{x}$
bx^2	$\rightarrow$	$2bx$	$\mathbf{x}^T\mathbf{B}\mathbf{x}$	$\rightarrow$	$2\mathbf{B}\mathbf{x}$

Basic identities from calculus/Matrix Cookbook

$$\arg\min_{w} \left\|y - Xw\right\|_2^2 + \lambda \|w\|_2^2$$

$$\arg\min_{w} (Xw - y)^T (Xw - y) + \lambda w^T w$$

From previous derivation

$$\nabla_w [-2y^T Xw + w^T X^T Xw + \lambda w^T w] = 0$$

$$-2X^T y + 2X^T Xw + 2\lambda w = 0$$

$$X^T Xw + \lambda w = X^T y$$

$$(X^T X + \lambda I) w = X^T y$$

$$\hat{w}_{\text{ridge}} = (X^T X + \lambda I)^{-1} \cdot X^T y$$

Shrinkage Properties

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

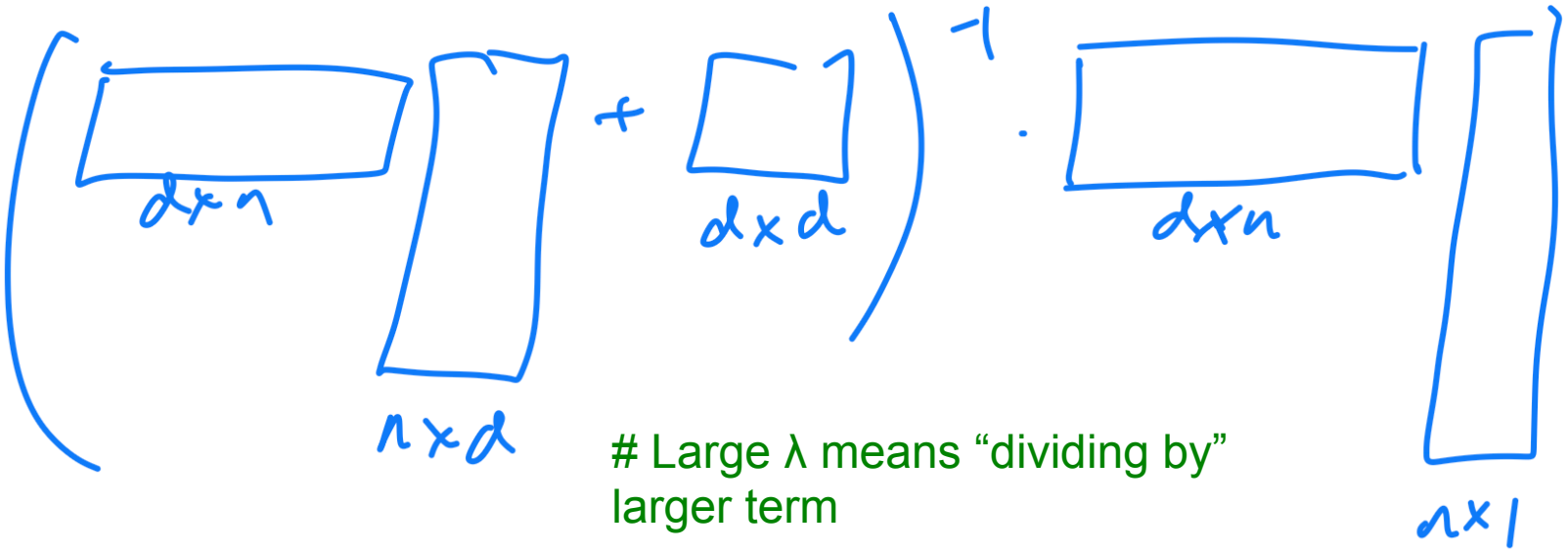
$$= (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{y}$$

$w \in \mathbb{R}^{d\times 1}$

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

What problem does this solve?

Matrix is always invertible



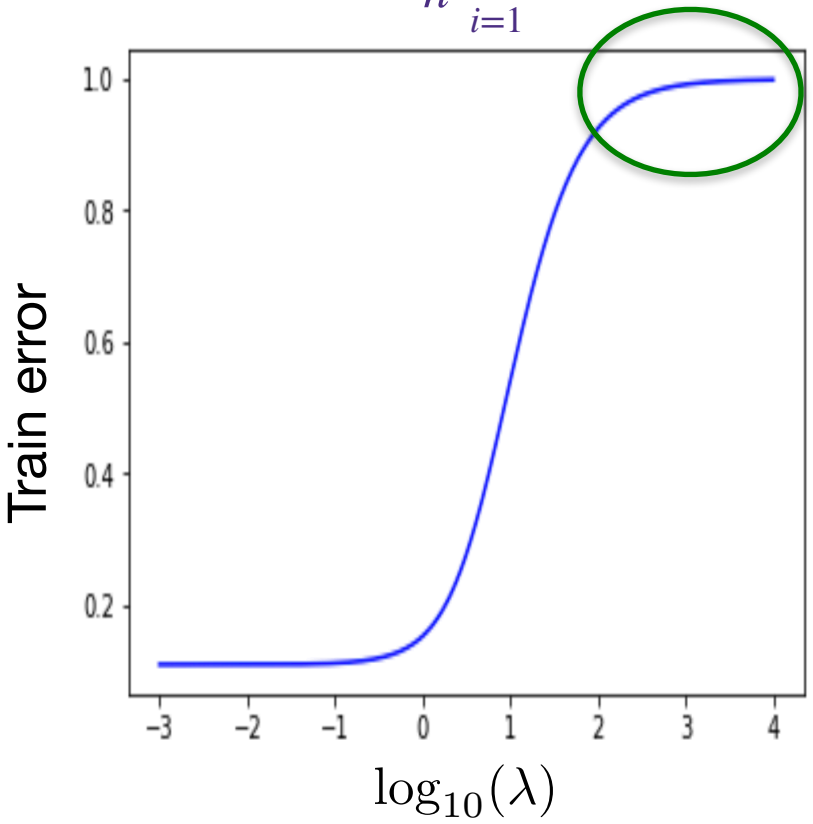
Large λ means "dividing by" larger term

- When $\lambda = 0$, this gives the least squares model
- This defines a family of models hyper-parametrized by λ
- Large λ means more regularization and simpler model
- Small λ means less regularization and more complex model

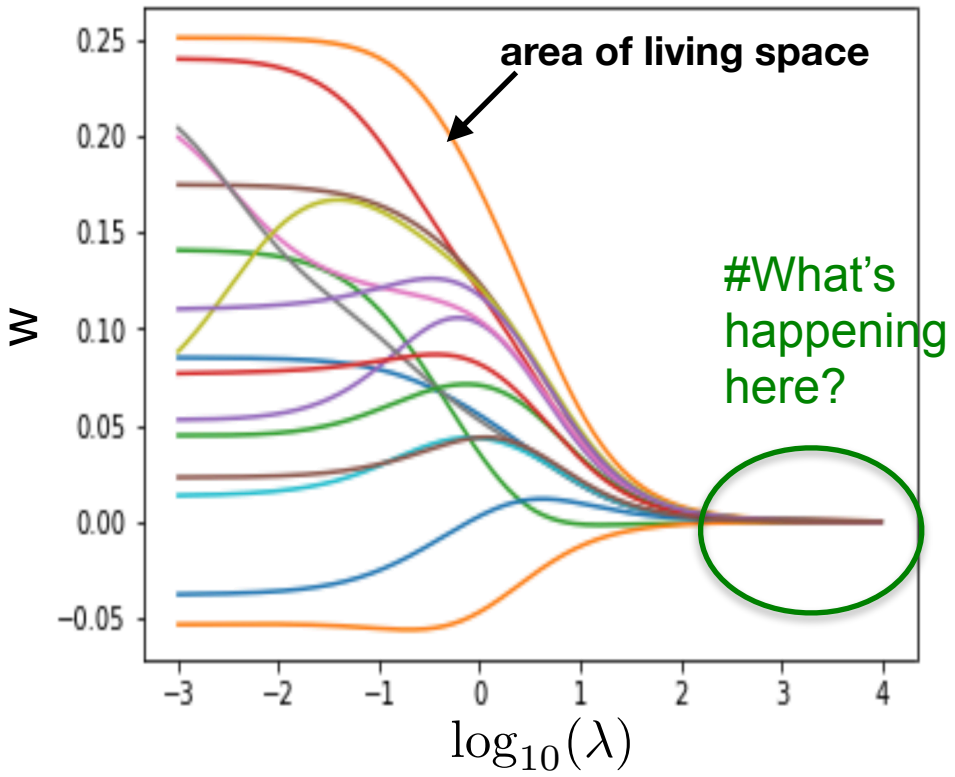
Optimal λ depends on magnitude of features

Ridge regression: minimize $\sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_2^2$

training MSE $\frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \hat{w}_{\text{ridge}}^{(\lambda)})^2$



Housing price predictor $w_i's$

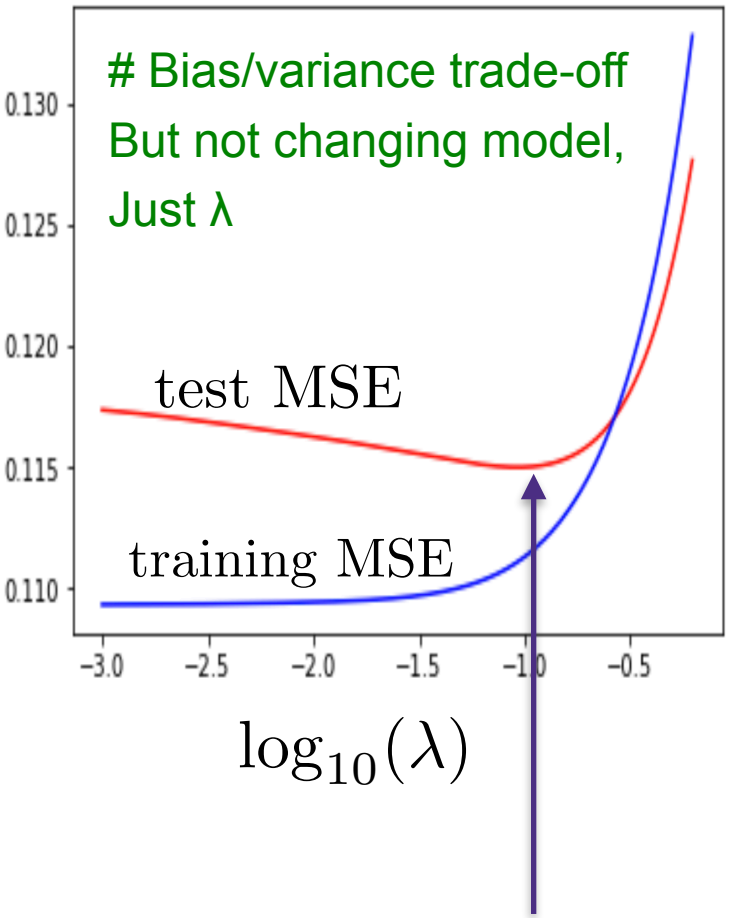


Right plot: called **regularization path**

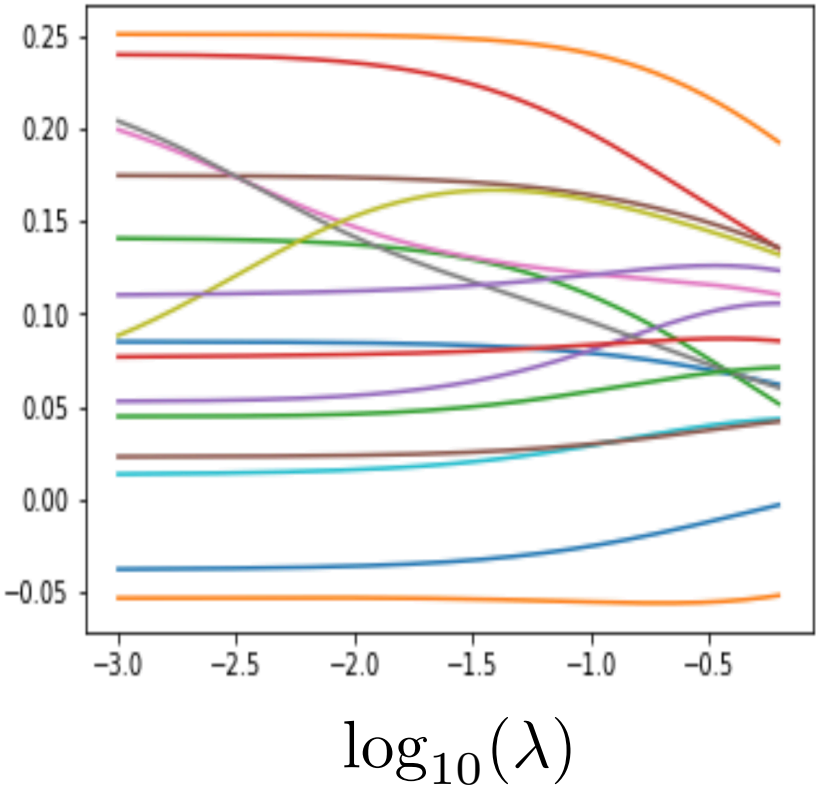
Too much regularization introduces bias / underfitting

Best predictor (area of living space) shrinks slowest

Ridge regression: $\text{minimize} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_2^2$



Housing price predictor $w_i's$



- this gain in test MSE comes from shrinking w's to get a less sensitive predictor
(which in turn reduces the variance)

- this is the role of regularizer

Bias-Variance Properties

Plan: work through an example where we can directly put bias/ variance in terms of λ

- Recall: $\hat{w}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$
- To analyze bias-variance tradeoff, we need to assume probabilistic generative model: $x_i \sim P_X, \ \mathbf{y} = \mathbf{X}w + \epsilon, \ \epsilon \sim \mathcal{N}(0,\sigma^2\mathbf{I})$ for some ground truth model parameter w
- The true error at a sample with feature x is $\mathbb{E}_{y,\mathcal{D}_{\text{train}}|x}[(y - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$

x is a test sample, $\hat{w}_{\text{ridge}}$ is fit to a dataset

Bias-Variance Properties

Plan: work through an example where we can directly put bias/ variance in terms of λ

- Recall: $\hat{w}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$
- To analyze bias-variance tradeoff, we need to assume probabilistic generative model: $x_i \sim P_X, \ \mathbf{y} = \mathbf{X}w + \epsilon, \ \epsilon \sim \mathcal{N}(0,\sigma^2\mathbf{I})$
- The true error at a sample with feature x is

$$\mathbb{E}_{y,\mathcal{D}_{\text{train}}|x}[(y - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$

x is a test sample, $\hat{w}_{\text{ridge}}$ is fit to a dataset

$$= \underbrace{\mathbb{E}_{y|x}[(y - \mathbb{E}[y\,|\,x])^2 \,|\, x]}_{\text{Irreducible Error}} + \underbrace{\mathbb{E}_{\mathcal{D}_{\text{train}}}[(\mathbb{E}[y\,|\,x] - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]}_{\text{Learning Error}}$$

Bias-Variance Properties

Plan: work through an example where we can directly put bias/ variance in terms of λ

- Recall: $\hat{w}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$
- To analyze bias-variance tradeoff, we need to assume probabilistic generative model: $x_i \sim P_X,\ \ \mathbf{y} = \mathbf{X}w + \epsilon,\ \ \epsilon \sim \mathcal{N}(0,\sigma^2\mathbf{I})$
- The true error at a sample with feature x is

$$\mathbb{E}_{y,\mathcal{D}_{\text{train}}|x}[(y - x^T\hat{w}_{\text{ridge}})^2\,|\,x]$$ # x is a test sample, $\hat{w}_{\text{ridge}}$ is fit to a dataset

$$= \mathbb{E}_{y|x}[(y - \mathbb{E}[y\,|\,x])^2\,|\,x] + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(\mathbb{E}[y\,|\,x] - x^T\hat{w}_{\text{ridge}})^2\,|\,x]$$

$$= \mathbb{E}_{y|x}[(y - x^Tw)^2\,|\,x] + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(x^Tw - x^T\hat{w}_{\text{ridge}})^2\,|\,x]$$ # true w

Bias-Variance Properties

Plan: work through an example where we can directly put bias/ variance in terms of λ

- Recall: $\hat{w}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$
- To analyze bias-variance tradeoff, we need to assume probabilistic generative model: $x_i \sim P_X, \;\; \mathbf{y} = \mathbf{X}w + \epsilon, \;\; \epsilon \sim \mathcal{N}(0,\sigma^2\mathbf{I})$
- The true error at a sample with feature x is

$$\mathbb{E}_{y,\mathcal{D}_{\text{train}}|x}[(y - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$ # x is a test sample, $\hat{w}_{\text{ridge}}$ is fit to a dataset

$$= \mathbb{E}_{y|x}[(y - \mathbb{E}[y\,|\,x])^2 \,|\, x] + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(\mathbb{E}[y\,|\,x] - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$

$$= \mathbb{E}_{y|x}[(y - x^Tw)^2 \,|\, x] + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(x^Tw - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$ # true w

$$= \sigma^2 + (x^Tw - \mathbb{E}_{\tilde{\mathcal{D}}_{\text{train}}}[x^T\hat{w}_{\text{ridge}}\,|\,x])^2 + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(\mathbb{E}_{\tilde{\mathcal{D}}_{\text{train}}}[x^T\hat{w}_{\text{ridge}}\,|\,x] - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$

Irreduc. Error — Bias-squared — Variance

Bias-Variance Properties

Plan: work through an example where we can directly put bias/ variance in terms of λ

- Recall: $\hat{w}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$
- To analyze bias-variance tradeoff, we need to assume probabilistic generative model: $x_i \sim P_X, \;\; \mathbf{y} = \mathbf{X}w + \epsilon, \;\; \epsilon \sim \mathcal{N}(0,\sigma^2\mathbf{I})$
- The true error at a sample with feature x is

$$\mathbb{E}_{y,\mathcal{D}_{\text{train}}|x}[(y - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$

x is a test sample, $\hat{w}_{\text{ridge}}$ is fit to a dataset

$$= \mathbb{E}_{y|x}[(y - \mathbb{E}[y\,|\,x])^2 \,|\, x] + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(\mathbb{E}[y\,|\,x] - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$

$$= \mathbb{E}_{y|x}[(y - x^Tw)^2 \,|\, x] + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(x^Tw - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$

true w

$$= \sigma^2 + (x^Tw - \mathbb{E}_{\mathcal{D}_{\text{train}}}[x^T\hat{w}_{\text{ridge}}\,|\,x])^2 + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(\mathbb{E}_{\tilde{\mathcal{D}}_{\text{train}}}[x^T\hat{w}_{\text{ridge}}\,|\,x] - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$

Irreduc. Error — Bias-squared — Variance

nI

Suppose $\mathbf{X}^T\mathbf{X} = n\mathbf{I}$, then $\hat{w}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T(\mathbf{X}w + \epsilon)$

$$= \frac{n}{n+\lambda}w + \frac{1}{n+\lambda}\mathbf{X}^T\epsilon$$

Not realistic but pretend we're sampling independent Gaussians

λ trades off weight on data vs. noise

Bias-Variance Properties

Suppose $\mathbf{X}^T\mathbf{X} = n\mathbf{I}$, then

$$\hat{w}_{\text{ridge}} = \frac{n}{n+\lambda} w + \frac{1}{n+\lambda}\mathbf{X}^T\epsilon$$

- Recall: $\hat{w}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$
- To analyze bias-variance tradeoff, we need to assume probabilistic generative model: $x_i \sim P_X, \quad \mathbf{y} = \mathbf{X}w + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2\mathbf{I})$
- The true error at a sample with feature x is

$$\begin{aligned}
&\mathbb{E}_{y,\mathcal{D}_{\text{train}}|x}[(y - x^T\hat{w}_{\text{ridge}})^2 \,|\, x] \\
&\quad = \mathbb{E}_{y|x}[(y - \mathbb{E}[y\,|\,x])^2 \,|\, x] + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(\mathbb{E}[y\,|\,x] - x^T\hat{w}_{\text{ridge}})^2 \,|\, x] \\
&\quad = \mathbb{E}_{y|x}[(y - x^Tw)^2 \,|\, x] + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(x^Tw - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]
\end{aligned}$$

$$= \sigma^2 + (x^Tw - \mathbb{E}_{\mathcal{D}_{\text{train}}}[x^T\hat{w}_{\text{ridge}}\,|\,x])^2 + \mathbb{E}_{\mathcal{D}_{\text{train}}}[(\mathbb{E}_{\tilde{\mathcal{D}}_{\text{train}}}[x^T\hat{w}_{\text{ridge}}\,|\,x] - x^T\hat{w}_{\text{ridge}})^2 \,|\, x]$$

(verify at home)

$$= \sigma^2 + \frac{\lambda^2}{(n+\lambda)^2}(w^Tx)^2 + \frac{\sigma^2 n}{(n+\lambda)^2}\|x\|_2^2$$

Irreduc. Error | Bias-squared | Variance

- λ trades off bias + variance
- Larger λ →
  - Bigger bias
  - Smaller variance

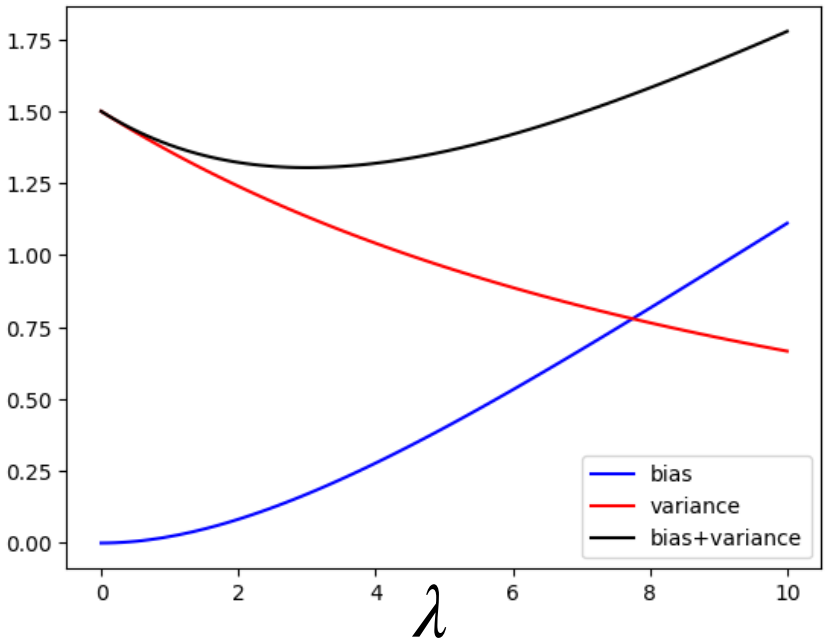
Bias-Variance Properties

Suppose $\mathbf{X}^T\mathbf{X} = n\mathbf{I}$,

- Ridge regressor: $\hat{w}_{ridge} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$
- True error

$$\mathbb{E}_{y,\mathcal{D}_{\text{train}}|x}[(y - x^T\hat{w}_{\text{ridge}})^2 \,|\, x] = \sigma^2 + \underbrace{\frac{\lambda^2}{(n+\lambda)^2}(w^T x)^2}_{\text{Bias-squared}} + \underbrace{\frac{\sigma^2 n}{(n+\lambda)^2}\|x\|_2^2}_{\text{Variance}}$$

d=10, n=20, $\sigma^2 = 3.0, \|w\|_2^2 = 10$



as $\lambda \to 0$,
$\hat{w}_{\text{ridge}} \to \hat{w}_{\text{LS}}$

as $\lambda \to \infty$
$\hat{w}_{\text{ridge}} \to 0$

What you need to know…

- Regularization
  - Penalizes complex models towards preferred, simpler models
- Ridge regression
  - L_2 penalized least-squares regression $\lambda \|w\|_2^2$
  - Regularization parameter trades off model complexity with training error
  - Never regularize the offset!

Simple Variable Selection
LASSO: Sparse Regression

Sparsity

$$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

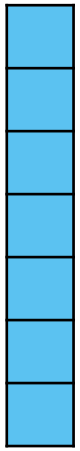
- Vector w is sparse, if many entries are zero

Sparsity

$$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

- **Vector w is sparse, if many entries are zero**
  - **Efficiency**: If size(**w**) = 100 Billion, each prediction is expensive:
    - **If w is sparse, prediction computation only depends on number of non-zeros**

$$\widehat{y}_i = \widehat{w}_{LS}^{\top} x_i = \sum_{j=1}^{d} x_i[j]\widehat{w}_{LS}[j]$$



Sparsity

$$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

- **Vector w is sparse, if many entries are zero**
  - **Interpretability**: What are the relevant dimension to make a prediction?

- How do we find "best" subset among all possible?

Lot size	Dishwasher
Single Family	Garbage disposal
Year built	Microwave
Last sold price	Range / Oven
Last sale price/sqft	Refrigerator
Finished sqft	Washer
Unfinished sqft	Dryer
Finished basement sqft	Laundry location
# floors	Heating type
Flooring types	Jetted Tub
Parking type	Deck
Parking amount	Fenced Yard
Cooling	Lawn
Heating	Garden
Exterior materials	Sprinkler System
Roof type	
Structure style	

Finding best subset: Exhaustive

- Try all subsets of size 1, 2, 3, … and one that minimizes validation error
- Problem?

Finding best subset: Greedy

Forward stepwise:
Starting from simple model and iteratively add features most useful to fit

Backward stepwise:
Start with full model and iteratively remove features least useful to fit

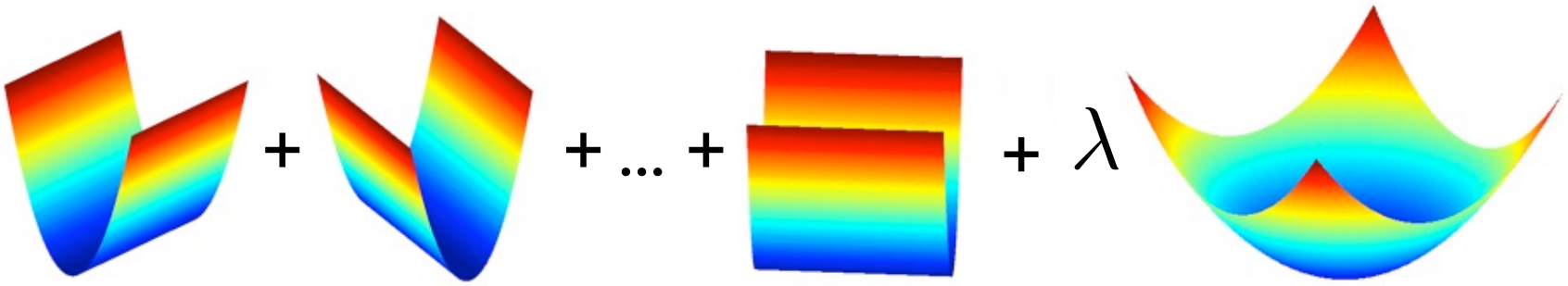
Combining forward and backward steps:
In forward algorithm, insert steps to remove features no longer as important

Lots of other variants, too.

Finding best subset: Regularize

Ridge regression makes coefficients small

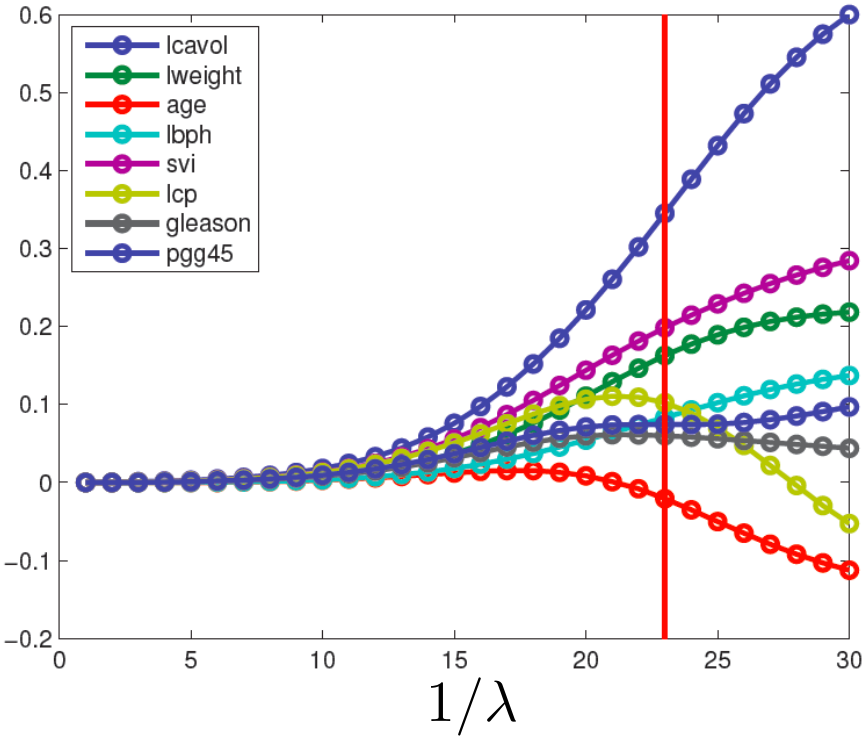
$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$



Finding best subset: Regularize

Ridge regression makes coefficients small

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

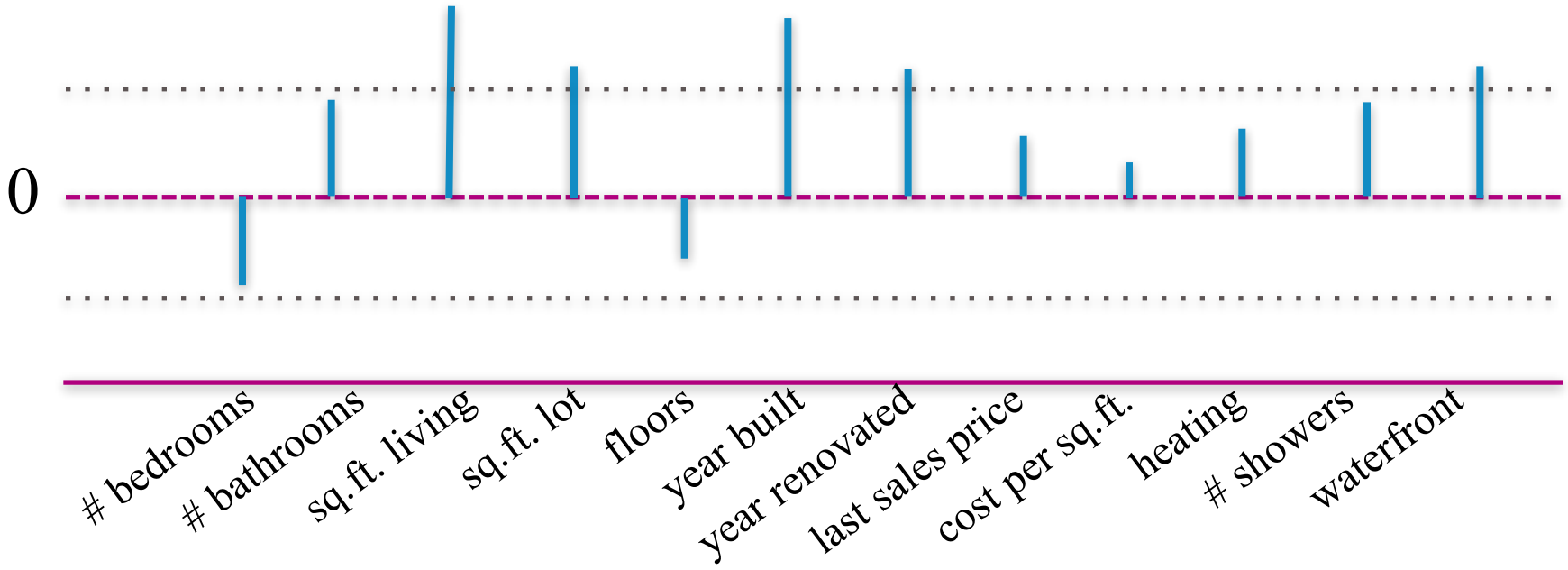


From
Kevin Murphy
textbook

Thresholded Ridge Regression

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

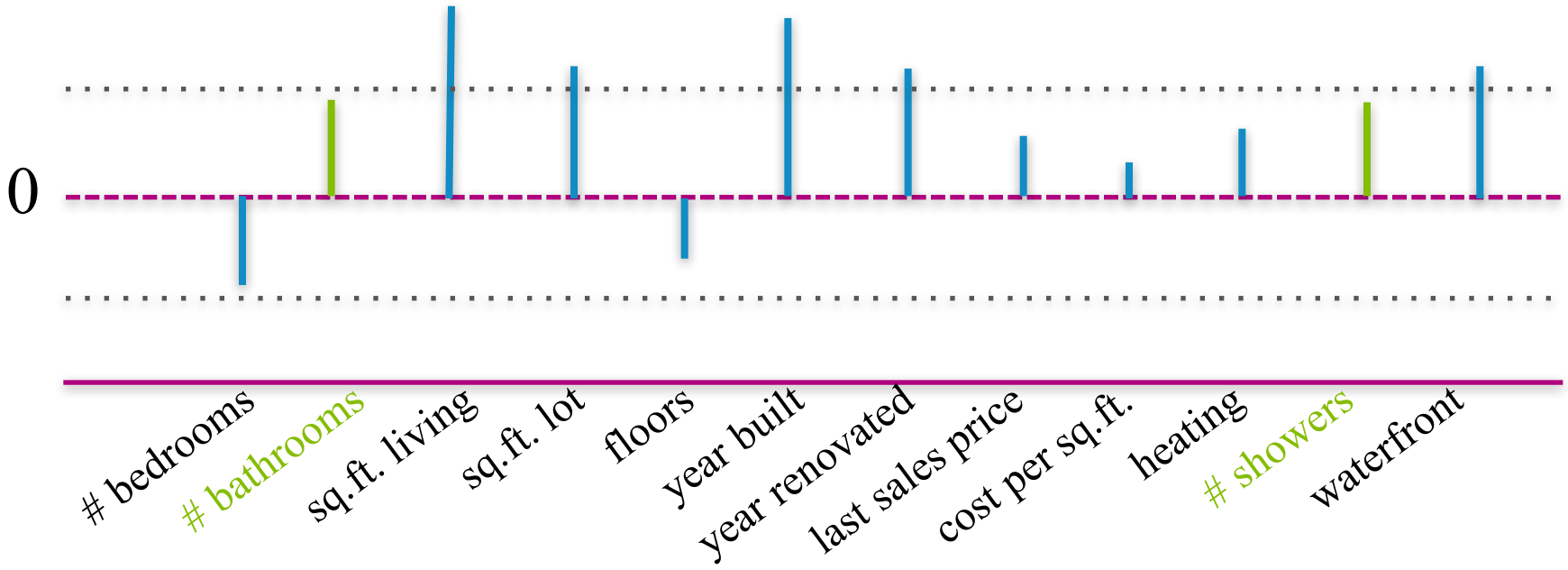
Why don't we just set **small** ridge coefficients to 0?



Thresholded Ridge Regression

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

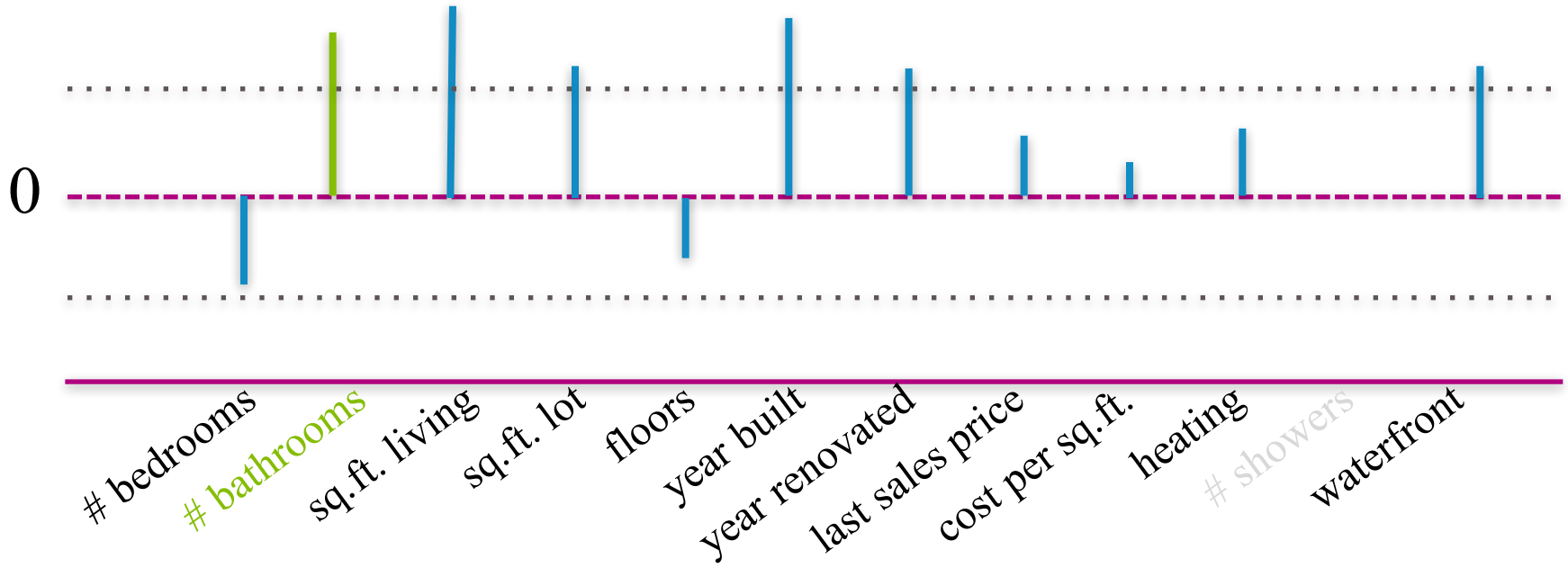
Consider two related features (bathrooms, showers)



Thresholded Ridge Regression

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

What if we **didn't** include showers? Weight on bathrooms increases!

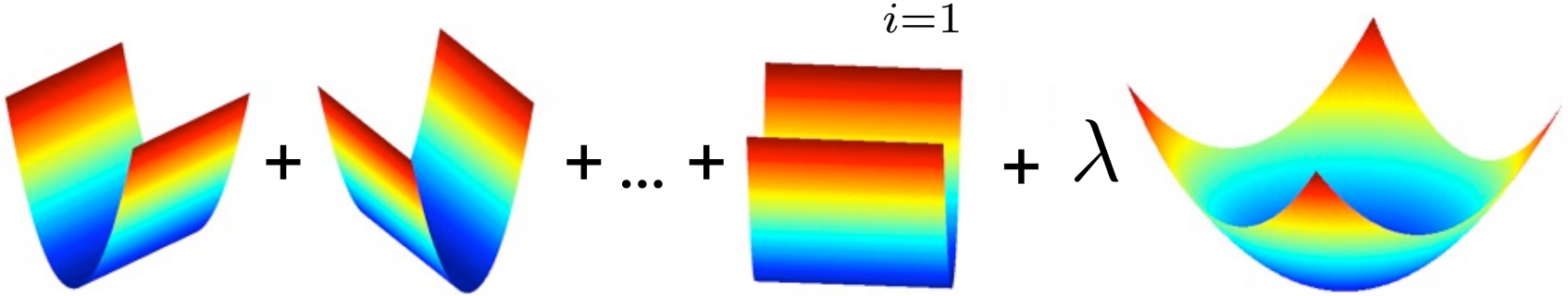


Can another regularizer perform selection automatically?

Recall Ridge Regression

- Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

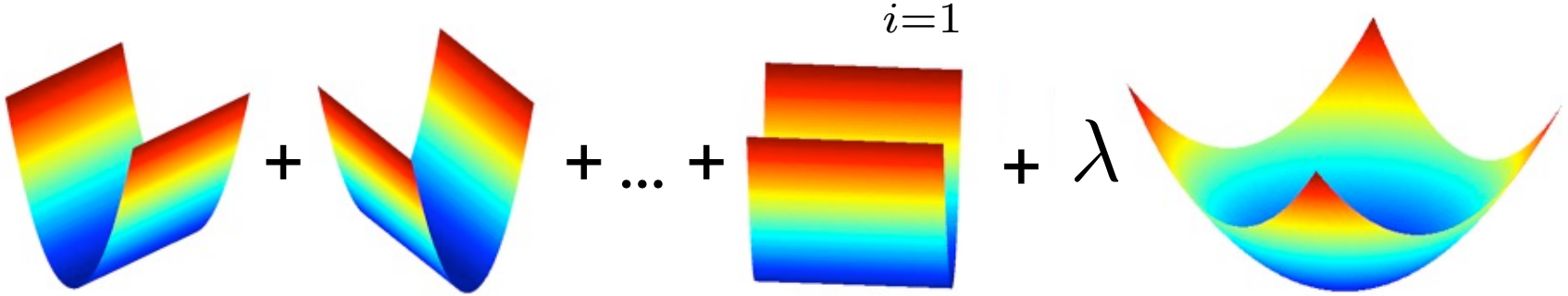


$$\|w\|_p = \left(\sum_{i=1}^{d} |w|^p\right)^{1/p}$$

Ridge vs. Lasso Regression

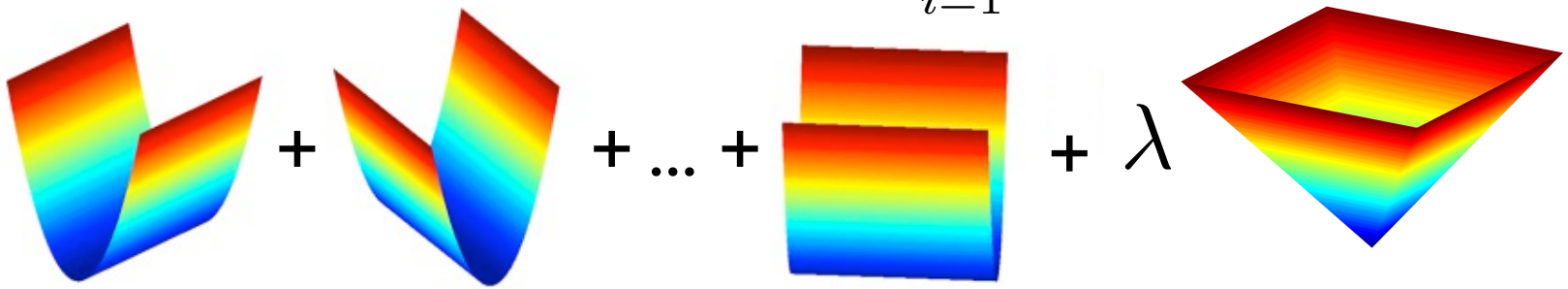
- Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$



- Lasso objective:

$$\widehat{w}_{lasso} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_1$$



Penalized Least Squares

$$\text{Ridge} : r(w) = ||w||_2^2 \qquad \text{Lasso} : r(w) = ||w||_1$$

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda r(w)$$

Penalized Least Squares

$$\text{Ridge} : r(w) = ||w||_2^2 \qquad \text{Lasso} : r(w) = ||w||_1$$

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda r(w)$$

For any $\lambda \geq 0$ for which $\widehat{w}_r$ achieves the minimum, there exists a $\nu \geq 0$ such that

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 \qquad \text{subject to } r(w) \leq \nu$$

Penalized Least Squares

$$\text{Ridge} : r(w) = ||w||_2^2 \qquad \text{Lasso} : r(w) = ||w||_1$$

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda r(w)$$

For any $\lambda \geq 0$ for which $\widehat{w}_r$ achieves the minimum, there exists a $\nu \geq 0$ such that

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 \qquad \text{subject to } r(w) \leq \nu$$