# CSE/STAT 416

## Assessing Performance

Pemi Nguyen
University of Washington
March 30, 2022

Slides by Hunter Schafer

# Logistics

Check EdStem for any announcements or clarifications on logistics

Jupyter Notebooks on EdStem.

Section tomorrow will give you practice on Python and Pandas

HW1 released on Friday

There will be activities to allow you to find pairs

# Linear Regression Model

Assume the data is produced by a line.

$$y^{(i)} = w_0 + w_1 X^{(i)}$$

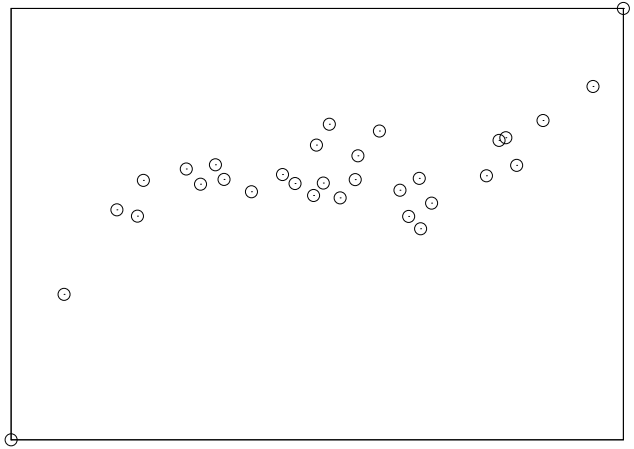
$w_0, w_1$ are the **parameters** of our model that need to be learned

$w_0$ is the intercept ($ of the land with no house)

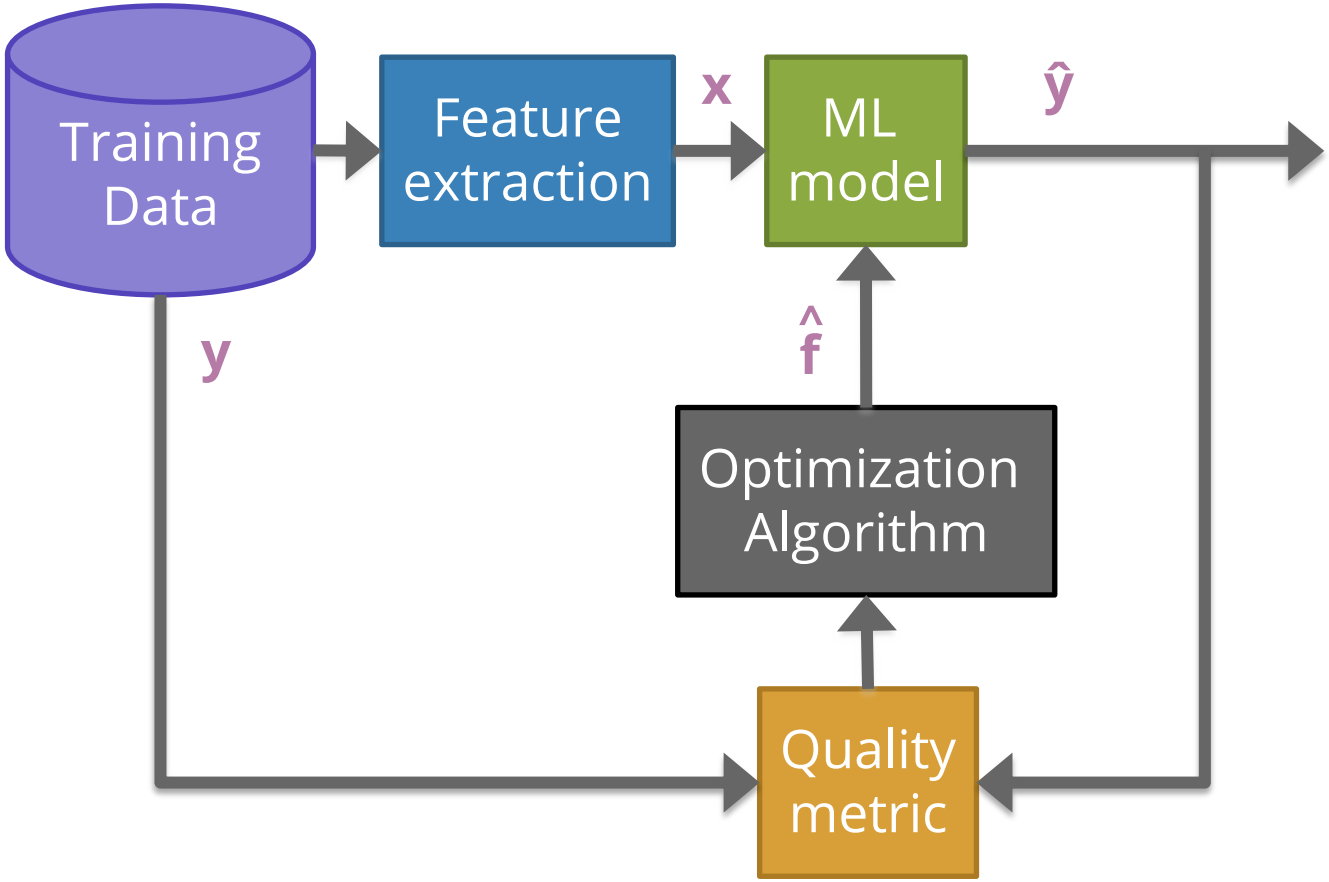
$w_1$ is the slope ($ increase per increase in sq. ft)

Learn estimates of these parameters $\hat{w}_0, \hat{w}_1$ and use them to predict new value for any input scalar $x$!

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x$$



# ML Pipeline


Training Data
Feature extraction
x
ML model
ŷ
y
f̂
Optimization Algorithm
Quality metric

# Linear Regression Recap

**Dataset**

$\{(X^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $X^{(i)} \in \mathbb{R}^d, y \in \mathbb{R}$

**Feature Extraction**

$h(X^{(i)})$: $\mathbb{R}^d \rightarrow \mathbb{R}^D$

$h(X^{(i)}) = (h_0(X^{(i)}), h_1(X^{(i)}), \ldots, h_D(X^{(i)}))$

**Regression Model**

$$y^{(i)} = f(X^{(i)})$$
$$= \sum_{j=0}^{D} w_j h_j(X^{(i)})$$
$$= w^T h(X^{(i)})$$

**Quality Metric / Loss function**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

**Predictor**

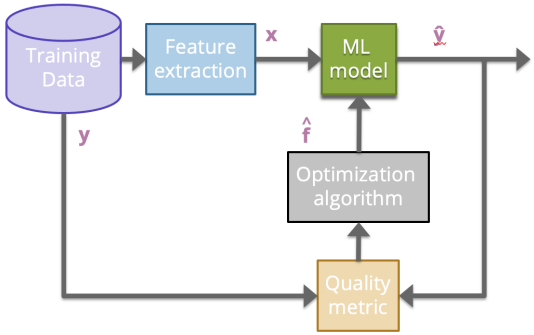
$$\hat{w} = \underset{w}{\operatorname{argmin}}\, MSE(w)$$

**Optimization Algorithm**

Optimized using Gradient Descent

**Prediction**

$$\hat{y}^{(i)} = \hat{w}^T h(X^{(i)})$$



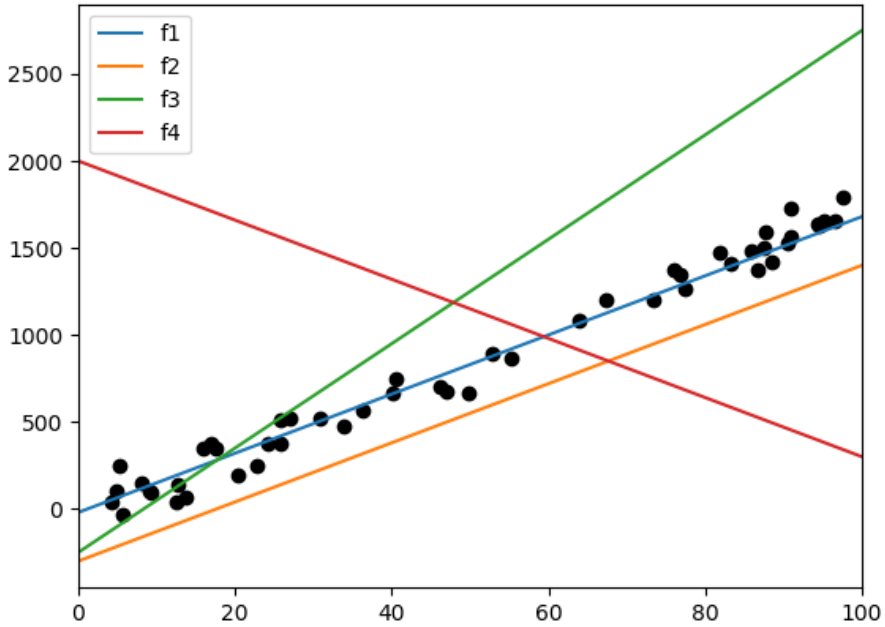
Poll Everywhere

# Think

1 min

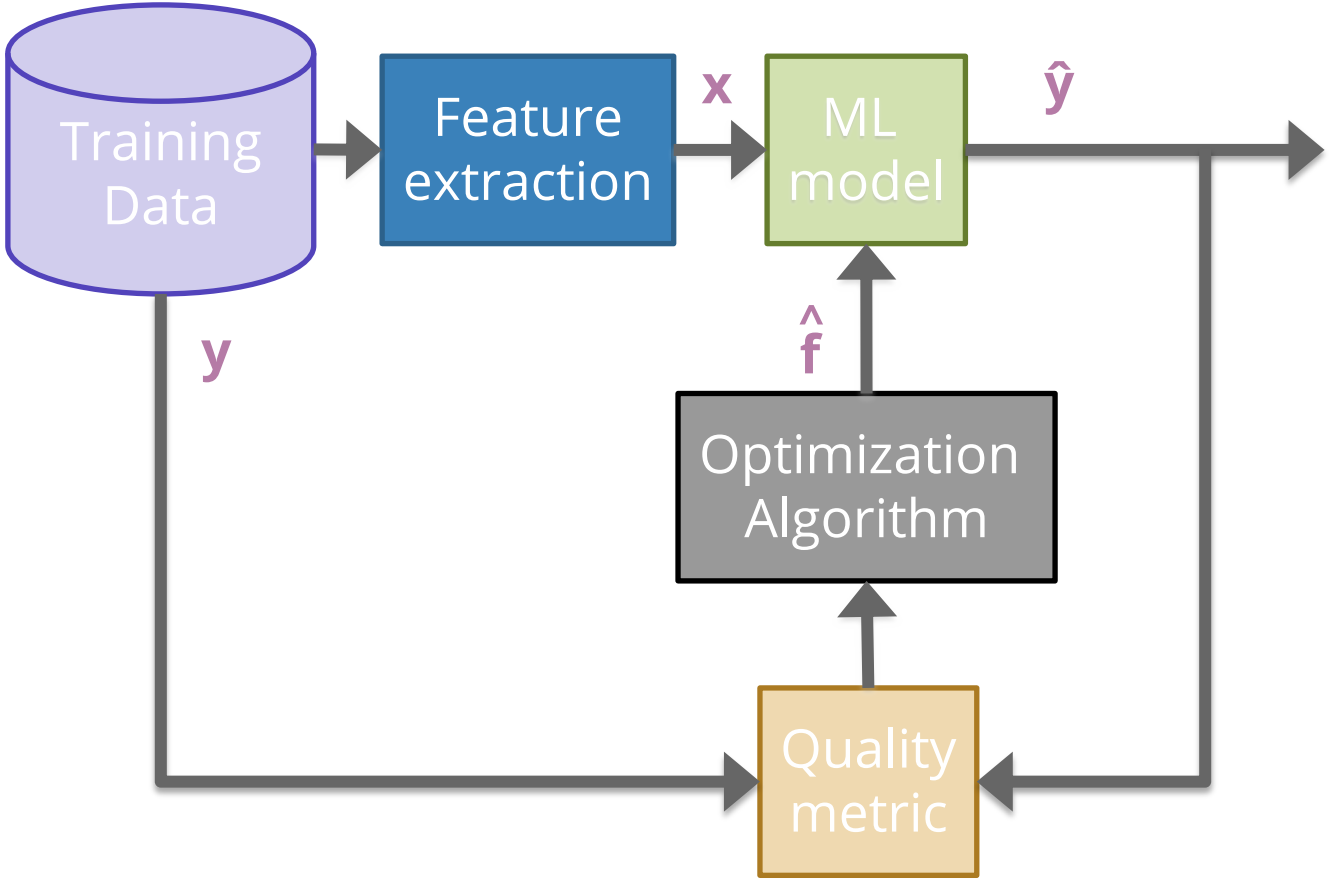
pollev.com/cs416

**Sort the following lines by their MSE on the data, from smallest to largest.** (estimate, don't actually compute)



# Pre-Lecture Video 1

*Feature Extraction*


Training Data
Feature extraction
x
ML model
ŷ
y
f̂
Optimization Algorithm
Quality metric

# Linear Regression Recap

**Dataset**

$\{(X^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $X^{(i)} \in \mathbb{R}^d, y \in \mathbb{R}$

**Feature Extraction**

$h(X^{(i)})$: $\mathbb{R}^d \rightarrow \mathbb{R}^D$

$h(X^{(i)}) = (h_0(X^{(i)}), h_1(X^{(i)}), \ldots, h_D(X^{(i)}))$

**Regression Model**

$$y^{(i)} = f(X^{(i)})$$
$$= \sum_{j=0}^{D} w_j h_j(X^{(i)})$$
$$= w^T h(X^{(i)})$$

**Quality Metric / Loss function**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

**Predictor**

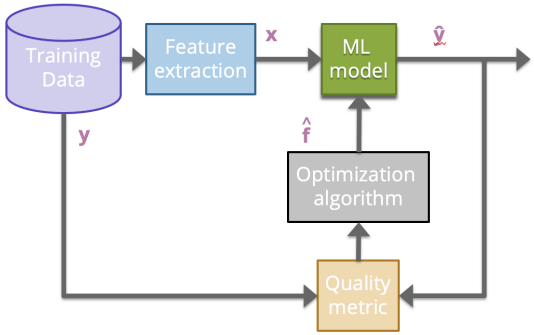
$$\hat{w} = \underset{w}{\operatorname{argmin}}\, MSE(w)$$

**Optimization Algorithm**

Optimized using Gradient Descent

**Prediction**

$$\hat{y}^{(i)} = \hat{w}^T h(X^{(i)})$$

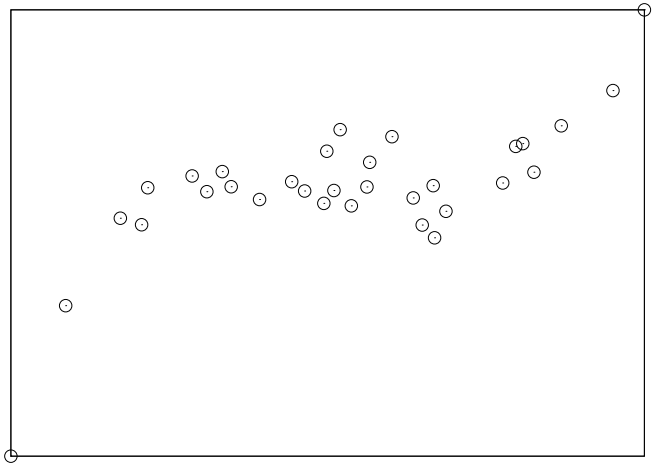


# Higher Order Features

This data doesn't look exactly linear, why are we fitting a line instead of some higher-degree polynomial?

We can! We just have to use a slightly different model!

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$



# Polynomial Regression

**Model**

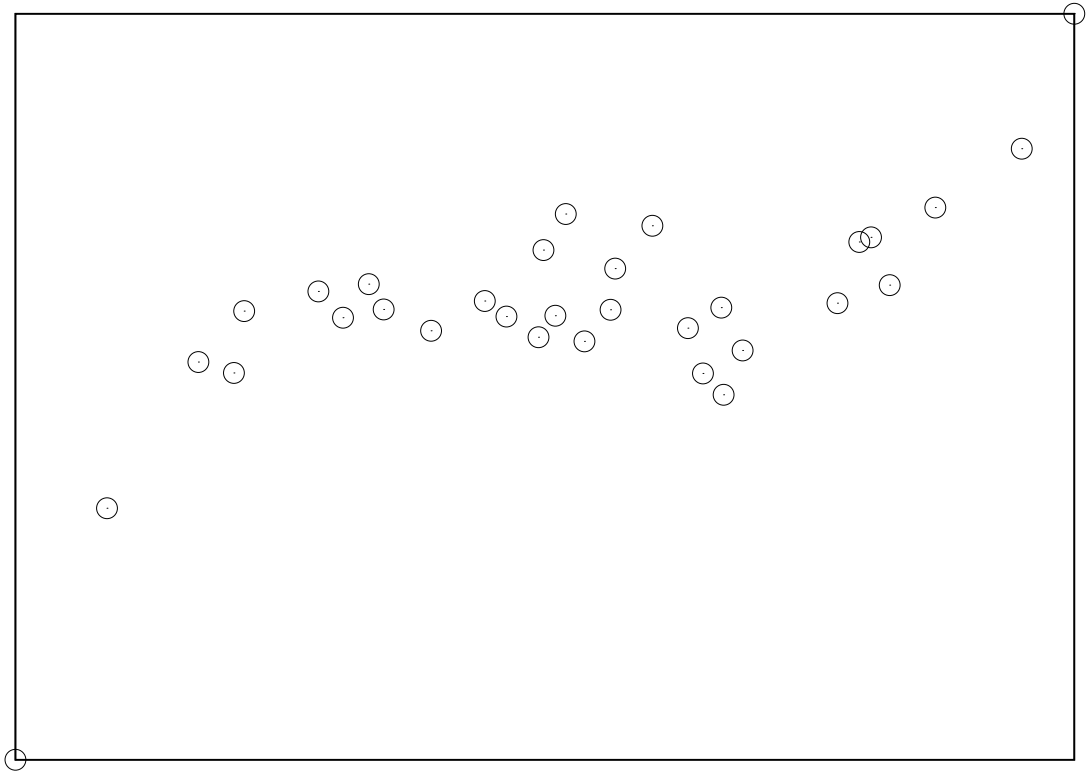
$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d$$

To capture a non-linear relationship in the model, we can transform the original features into more features!

| Feature | Value | Parameter |
|---|---|---|
| 0 | 1 (constant) | $w_0$ |
| 1 | $x$ | $w_1$ |
| 2 | $x^2$ | $w_2$ |
| … | … | … |
| d | $x^d$ | $w_d$ |

How do you train it? Gradient descent (with more parameters)

# Polynomial Regression



How to decide what the right degree? Come back Wednesday!

# Features

**Features** are the values we select or compute from the data inputs to put into our model. **Feature extraction** is the process of reduce the number of features in a dataset by creating new features from the existing ones (and then discarding the original features).

**Model**

$$y = w_0 h_0(x) + w_1 h_1(x) + \ldots + w_D h_D(x)$$

$$= \sum_{j=0}^{D} w_j h_j(x)$$

| Feature | Value | Parameter |
|---|---|---|
| 0 | $h_0(x)$ often 1 (constant) | $w_0$ |
| 1 | $h_1(x)$ | $w_1$ |
| 2 | $h_2(x)$ | $w_2$ |
| … | … | … |
| d | $h_d(x)$ | $w_d$ |

# Adding Other Features

Generally we are given a data table of values we might look at that include more than one feature per house.

Each row is a data point.

Each column (except Value) represents a feature

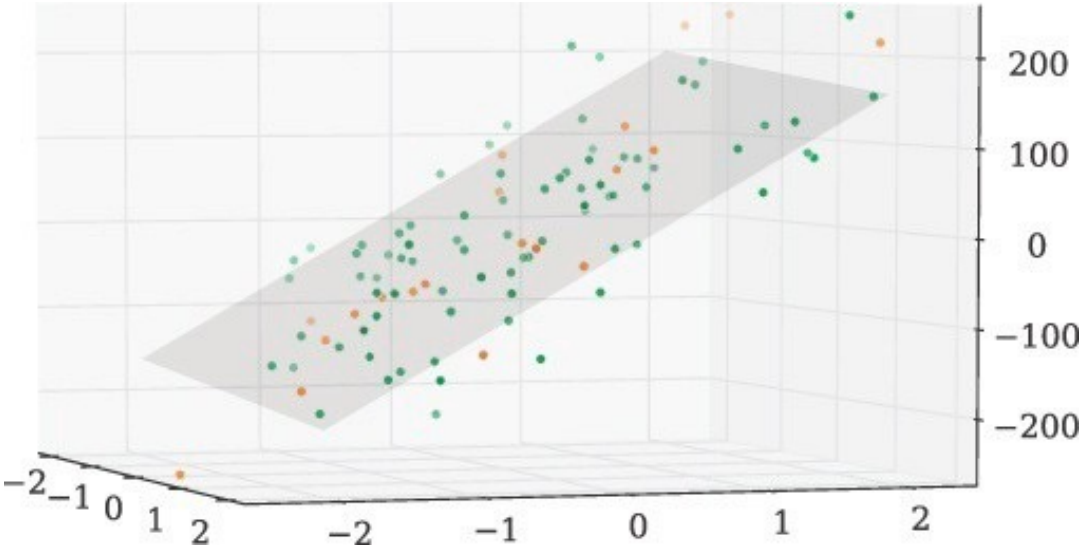
The last column (Price) contains the actual output values

| sq. ft. | # bathrooms | owner's age | … | price |
|---|---|---|---|---|
| 1400 | 3 | 47 | … | 70,800 |
| 700 | 3 | 19 | … | 65,000 |
| … | … | … | … | … |
| 1250 | 2 | 36 | … | 100,000 |

# More Inputs - Visually

Adding more features to the model allows for more complex relationships to be learned

$$y = w_0 + w_1(sq.ft.) + w_2(\#\ bathrooms)$$

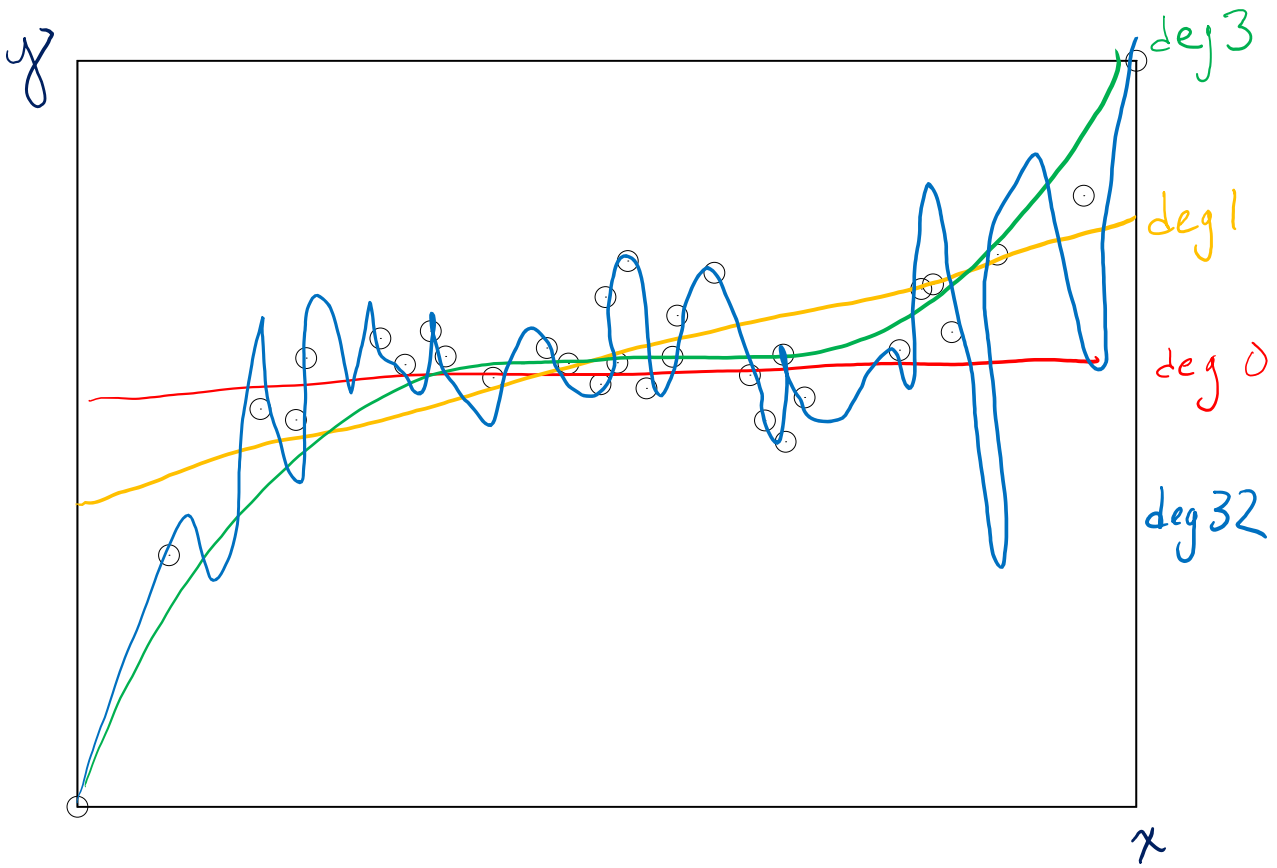



Coefficients tell us the rate of change **if all other features are constant**

# Pre-Lecture Video 2

*Assessing Performance*

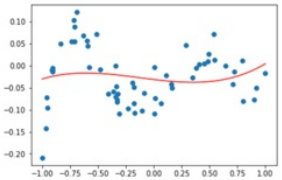
# Polynomial Regression



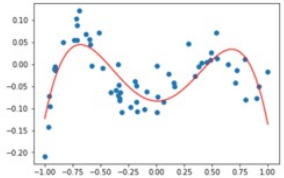

How do we decide what the right choice of $p$ is?

# Polynomial Regression

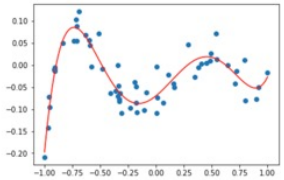
Consider using different degree polynomials on the same training set.



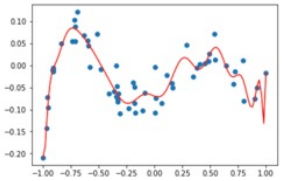
$p = 3$



$p = 4$



$p = 5$



$p = 20$

From estimating with your eyes, which one seems to have the lowest MSE on this dataset?

It seems like minimizing the MSE on the training set is not the whole story here ...

# Why a (near) zero training error is not a good thing?

Why do we train ML models?

We generally want them to do well on **unseen** data.

If we choose the model that minimizes MSE on the data it learned from, we are just choosing the model that can **memorize**, not the one that **generalizes** well.

> **Analogy:** Just because you can get 100% on a practice exam you've studied for hours, it doesn't mean you will also get 100% on the real test that you haven't seen before. When you **overprepares** for an exam through rote memorization, you are not able to generalize your knowledge and transfer what you have learned to solve new, unfamiliar problems.

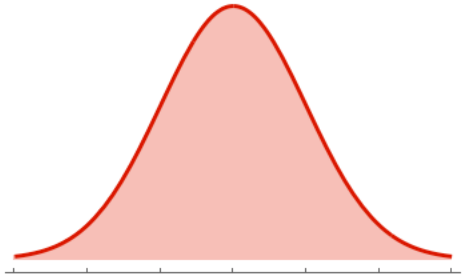
Key Idea: Assessing yourself based on something you learned from generally overestimates how well you will do in the future!

# Performance on unseen data

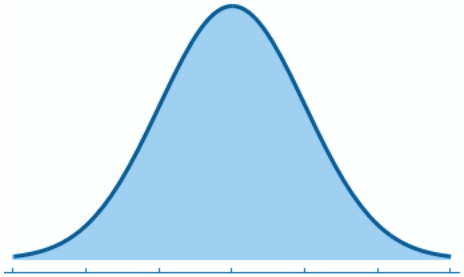
What we care about is how well the model will do on unseen data.

How do we measure this? **True error**

To do this, we need to understand uncertainty in the world



Sq. Ft.



Price | Sq. Ft.

**True Error**

# Model Assessment

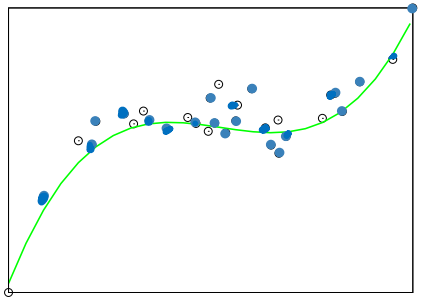
How can we figure out how well a model will do on future data if we don't have any future data?

Estimate it! We can hide data from the model to test it later as an estimate how it will do on future data

We will randomly split our dataset into a **train set** and a **test set**

The train set is to train the model

The test set is to estimate the performance in the future



# Test Error

What we really care about is the **true error,** or how well a model perform on unseen data in the wild, but we can't know that without having an infinite amount of data!

We will use the **test set** to estimate the true error.

**Note:** The train and test set need to be **randomly split** in order for the test set to be truly reflective of data in the real world.

Call the error on the test set the **test error** for a model $\hat{f}$:

$$MSE_{test} = \frac{1}{n} \sum_{i \in Test} \left( y^{(i)} - \hat{f}\left(x^{(i)}\right) \right)^2$$

If the test set is large enough, this can approximate the true error.

# Train/Test Split

If we use the test set to estimate future, how big should it be?

This comes at a cost of reducing the size of the training set though (in the absence of being able to just get more data)

In practice people generally do train:test as either

- 80:20
- 90:10

**Important**: Never train your model on data in the test set!

# Model Complexity

# Model Complexity

There is not a well-defined way to measure the complexity of a model. It depends on the nature of the models.

We usually associate it with the number of parameters. A model with more parameters is usually more complex.

Example with polynomial regression:

- Model 1: $y = w_0 + w_1 x$ (2 parameters)
- Model 2: $y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$ (4 parameters)

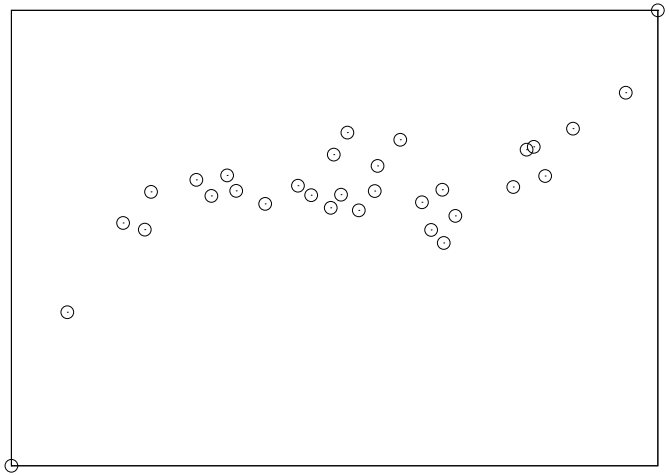
We say that model 2 is more complex than model 1.

# Training Error

What happens to **training error** as we increase model complexity?

Start with the simplest model (a constant function)

End with a very high degree polynomial

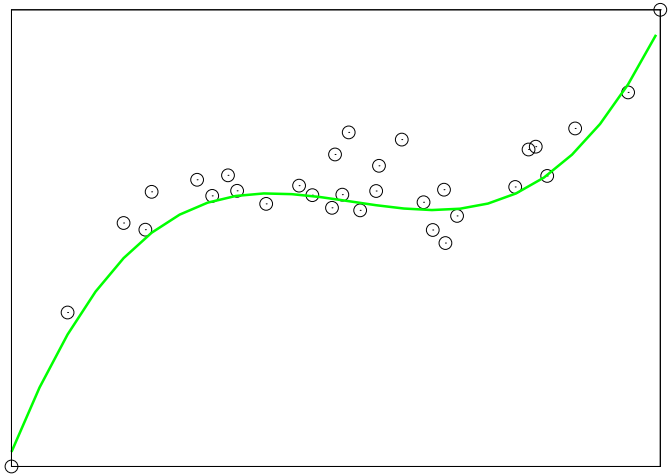


# True Error

What happens to **true error** as we increase model complexity?

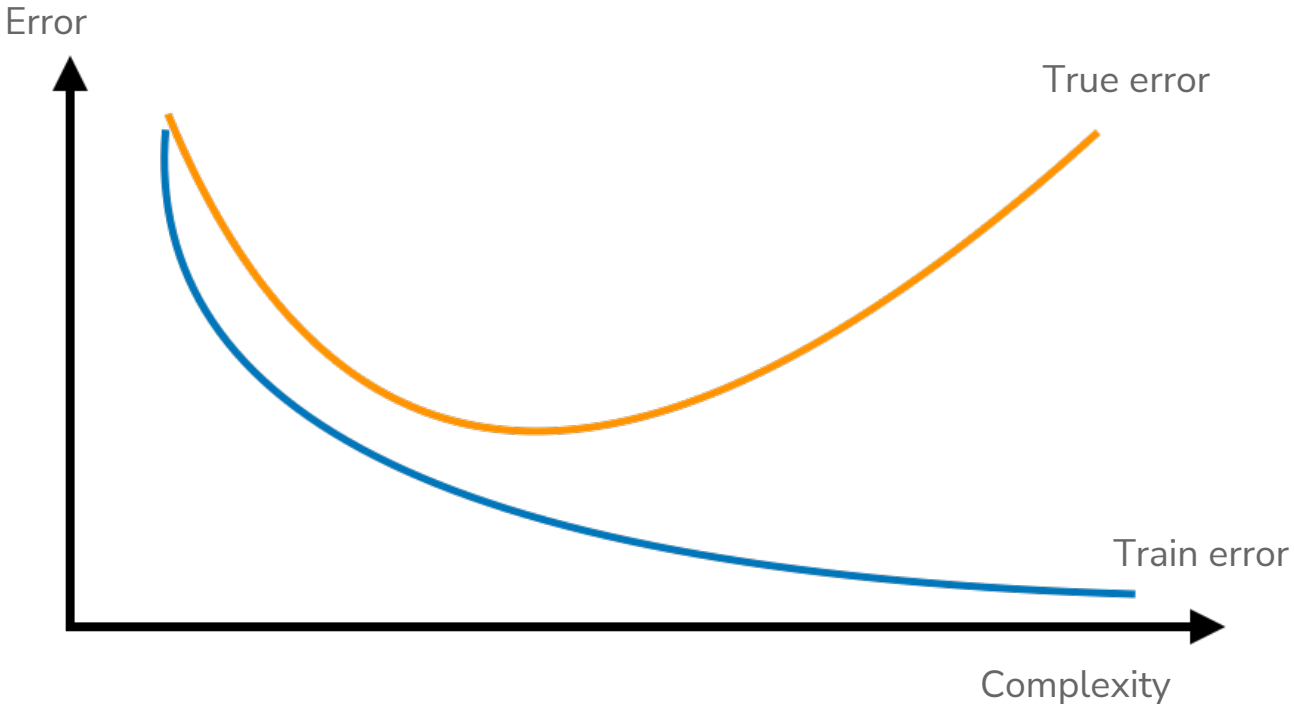
Start with the simplest model (a constant function)

End with a very high degree polynomial



# Train/True Error

Compare what happens to train and true error as a function of model complexity

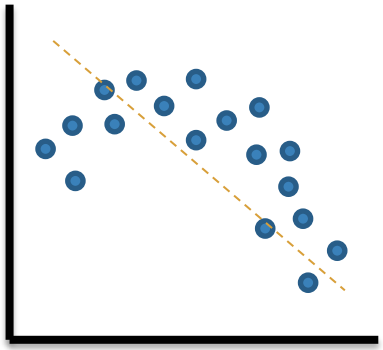


# Underfitting

**Underfitting** happens when a model cannot capture the complex patterns between a training set's features and its output values.

Underfitting is usually easy to fix because we can get a low training error by:

- Removing uninformative features
- Using more complex models (by adding more features or introducing more non-linearities to capture non-linear patterns)



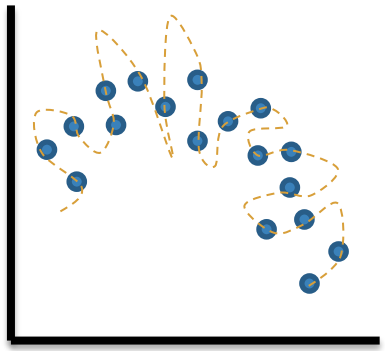
# Overfitting

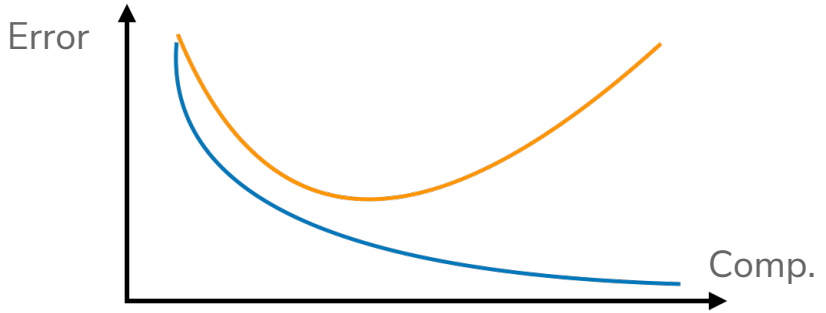
**Overfitting** happens when we too closely match the training data and fail to generalize.

Overfitting occurs when you train a predictor $\hat{w}$ but there exists another predictor $w'$ from the same model class such that:

$$error_{true}(w') < error_{true}(\hat{w})$$

$$error_{train}(w') > error_{train}(\hat{w})$$





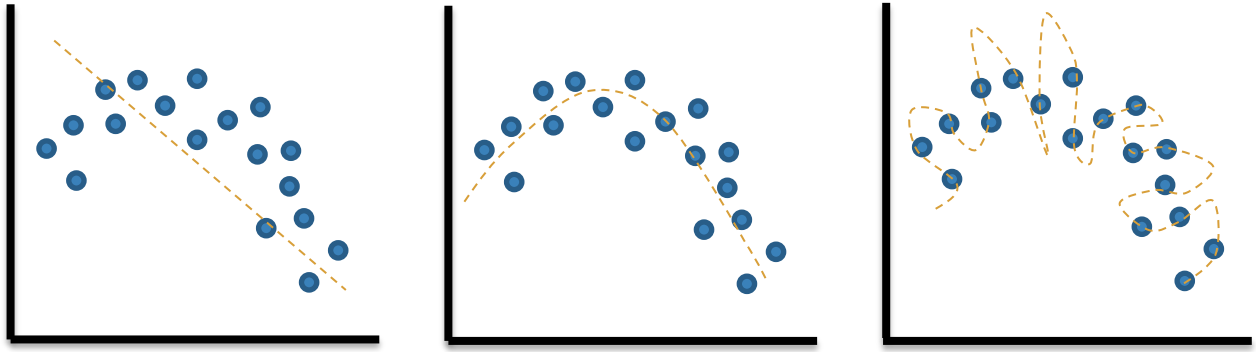
# Underfitting / Overfitting

The ability to overfit/underfit is a knob we can turn based on the model complexity.

More complex => easier to overfit

Less complex => easier to underfit

In a bit, we will talk about how to chose the “just right”, but now we want to look at this phenomena of overfitting/underfitting from another perspective.



Underfitting Optimal Overfitting

# Bias-Variance Tradeoff

# Source of errors in a model

Total errors for a machine learning model comes from 3 types:

**Bias**

**Variance**

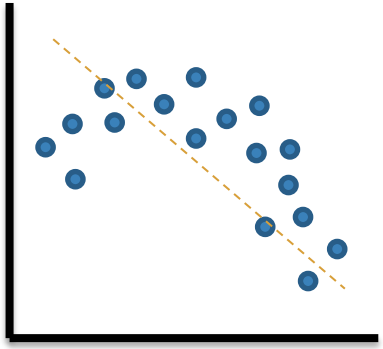
**Irreducible Errors**

Irreducible error is the one that we can't avoid or possibly eliminate. They are caused by elements outside of our control, such as noise from observations.

# Bias

A model that is too simple fails to capture the complex patterns in the dataset, which signifies a fundamental limitation of the model. We call this a **bias** error.

**Bias** is the difference between the average prediction of our model **and** the expected value which we are trying to predict.

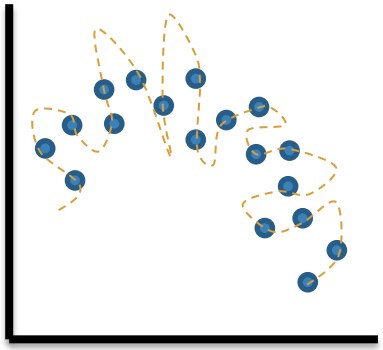


Low complexity (simple) models tend to have high bias.

# Variance

A model that is too complicated for the task overly fits to small fluctuations. The flexibility of the complicated model makes it capable of memorizing answers rather than learning general patterns. This contributes to the error as **variance**.

**Variance** is the variability in the model prediction, meaning how much the predictions will change if a different training dataset is used.



High complexity models tend to have high variance.*

# Bias-Variance Tradeoff

Irreducible error remains unchanged.

Tradeoff between bias and variance:

- Simple models: High bias + Low variance
- Complex models: Low bias + High variance

Source of errors for a particular model $\hat{f}$ using MSE loss function:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{bias}[\hat{f}(x)]^2 + \text{var}(\hat{f}(x)) + \sigma_\epsilon^2$$

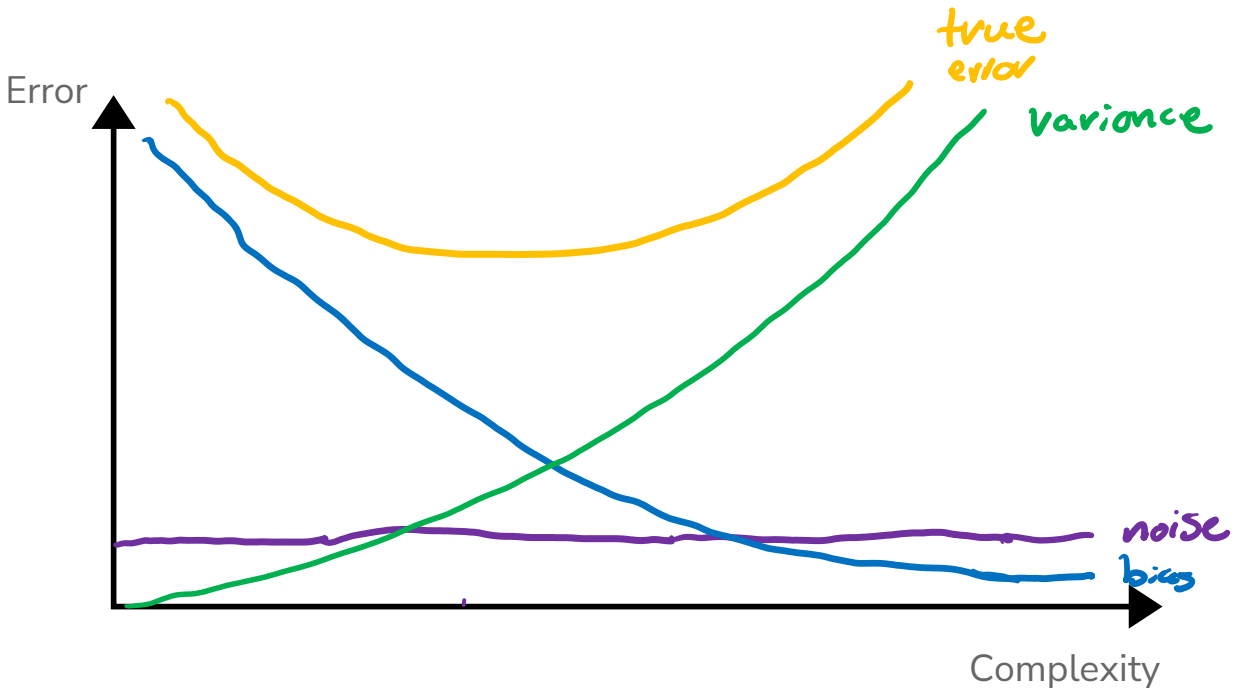
**Error = Biased squared + Variance + Irreducible Error**

# Brain Break

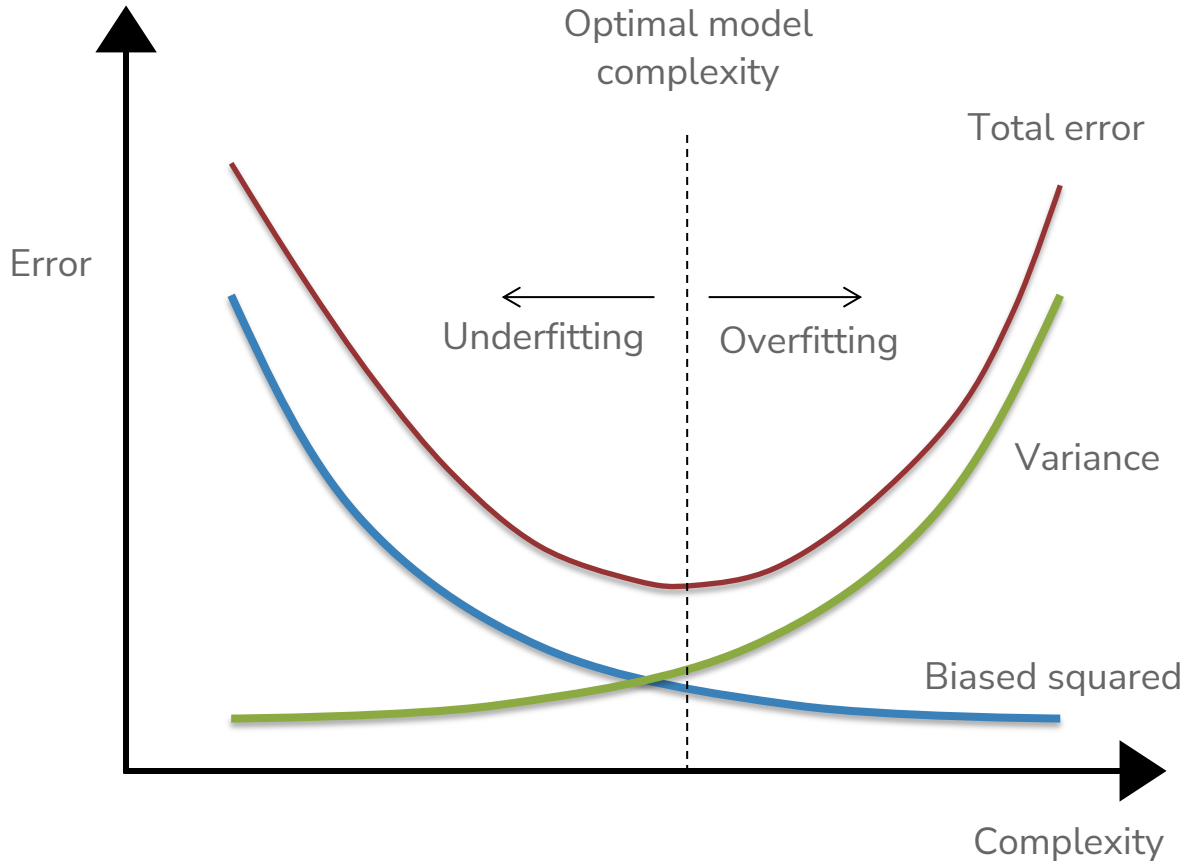
# Bias-Variance Tradeoff

Visually, this looks like the following!

$$Error = Bias^2 + Variance + Irredicible\ Error$$



# Bias – Variance Tradeoff



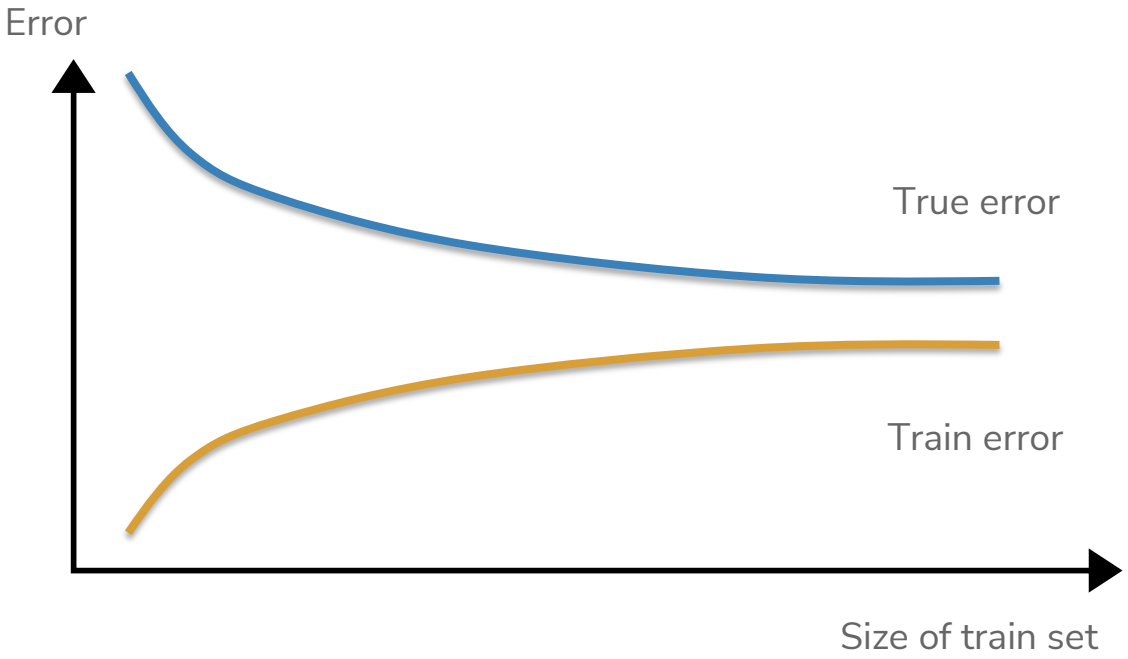
# Dataset Size

So far our entire discussion of error assumes a fixed amount of data. What happens to our error as we get more data?

# Dataset Size

Model complexity doesn't depend on the size of the training set

The larger the training set, the lower the variance of the model, thus less overfitting



# Choosing Complexity

# Choosing Complexity

So far we have talked about the affect of using different complexities on our error. Now, how do we choose the right one?

Poll Everywhere

pollev.com/cs416

**Goal**: Get you actively participating in your learning

Typical Activity

- Question is posed
- **Think** (1 min): Think about the question on your own
- **Pair** (2 min): Talk with your neighbor to discuss question
  - If you arrive at different conclusions, discuss your logic and figure out why you differ!
  - If you arrived at the same conclusion, discuss why the other answers might be wrong!
- **Share** (1 min): We discuss the conclusions as a class

During each of the **Think** and **Pair** stages, you will respond to the question via a Poll Everywhere poll

- The poll will only be open for the last **15** seconds of each of the stage
- Not worth any points, just here to help you learn!

Poll Everywhere

Think

1 min

pollev.com/cs416

**Suppose I wanted to figure out the right degree polynomial for my dataset (we'll try p from 1 to 20). What procedure should I use to do this? Pick the best option**

For each possible degree polynomial p:

- Train a model with degree p on the training set, pick p that has the lowest test error
- Train a model with degree p on the training set, pick p that has the highest test error
- Train a model with degree p on the test set, pick p that has the lowest test error
- Train a model with degree p on the test set, pick p that has the highest test error
- None of the above

Poll Everywhere

Think

2 min

pollev.com/cs416

**Suppose I wanted to figure out the right degree polynomial for my dataset (we'll try p from 1 to 20). What procedure should I use to do this? Pick the best option**

For each possible degree polynomial p:

- Train a model with degree p on the training set, pick p that has the lowest test error
- Train a model with degree p on the training set, pick p that has the highest test error
- Train a model with degree p on the test set, pick p that has the lowest test error
- Train a model with degree p on the test set, pick p that has the highest test error
- None of the above

# Choosing Complexity

We can't just choose the model that has the lowest **train** error because that will favor models that overfit!

It then seems like our only other choice is to choose the model that has the lowest **test** error (since that is our approximation of the true error)

> This is almost right. However, the test set has been **tampered**, thus is no longer is an unbiased estimate of the true error.
>
> We didn't technically train the model on the test set (that's good), but we chose **which model** to use based on the performance of the test set.
>
> - It's no longer a stand in for "the unknown" since we probed it many times to figure out which model would be best.

NEVER EVER EVER touch the test set until the end. You only use it ONCE to evaluate the performance of the best model you have selected during training.

# Choosing Complexity

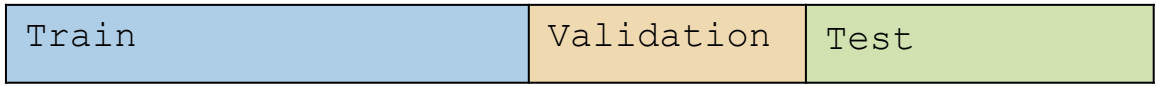
We will talk about two ways to pick the model complexity without ruining our test set.

- Using a validation set
- Doing cross validation

# Validation Set

So far we have divided our dataset into train and test


We can't use Test to choose our model complexity, so instead, break up Train into ANOTHER dataset




We will pick the model that does best on validation. Note that this now makes the validation error of the "best" model a biased estimate of true error. The test error will be an unbiased estimate though since we never looked at it!

# Validation Set

The process generally goes

```
train, validation, test = random_split(dataset)
for each model complexity p:
    model = train_model(model_p, train)
    val_err = error(model, validation)
    keep track of p with smallest val_err
return best p + error(model, test)
```

# Validation Set

**Pros**

Easy to describe and implement

Pretty fast

- Only requires training a model and predicting on the validation set for each complexity of interest

**Cons**

- Have to sacrifice even more training data
- Prone to overfitting

# Cross Validation

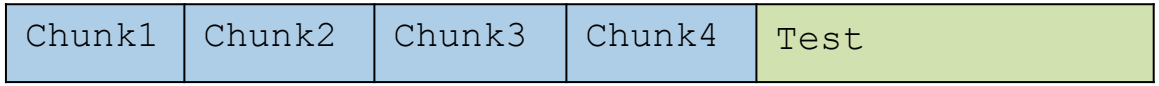
In the pre-lecture videos for next week, we will introduce another way to perform validation called **cross-validation**.

We leave the slides here for reference, but we will cover this in Monday's class.

# Cross-Validation

Clever idea: Use many small validation sets without losing too much training data.

Still need to break off our test set like before. After doing so, break the training set into $k$ chunks.


For a given model complexity, train it $k$ times. Each time use all but one chunk and use that left out chunk to determine the validation error.

# Cross Validation

For a set of hyperparameters, perform Cross Validation on k folds

# Cross-Validation

The process generally goes

```
chunk_1, …, chunk_k, test = split_data(dataset)
for each model complexity p:
    for i in [1, k]:
        model = train_model(model_p, chunks - i)
        val_err = error(model, chunk_i)
    avg_val_err = average val_err over chunks
    keep track of p with smallest avg_val_err
return model trained on train with best p +
error(model, test)
```

# Cross-Validation

**Pros**

- Prevent overfitting: By training the model on multiple folds instead of only 1 training set, this allow the model with the best generalization capabilities.
- Don't have to actually get rid of any training data!
- Help find the optimal model (such as the best set of hyperparameters, the best complexity)

**Cons**

- Very slow. For each model selection, we have to train $k$ times
- Very computationally expensive

# Cross-Validation

For best results, need to make $k$ really big

Theoretical best estimator is to use $k = n$

- Called "Leave One Out Cross Validation”

In practice, people use $k = 5$ to $10$

Nowadays, with the rise of deep learning, people don’t use cross-validation much. Datasets of large models contain millions of training examples, which make them robust and overcome overfitting.

# Recap

**Theme**: Assess the performance of our models

**Ideas:**

Model complexity

Train vs. Test vs. True error

Overfitting and Underfitting

Bias-Variance Tradeoff

Error as a function of train set size

Choosing best model complexity

- Validation set
- Cross Validation