CSE/STAT 416

Assessing Performance

Karthik Mohan
University of Washington
March 31, 2021

Logistics

- Check EdStem for any announcements or clarifications on assignments - This is our go to source!
- Jupyter Notebooks on EdStem.
- Section tomorrow will give you practice writing some of the code for HW1
- HW1 released
  - Remember there is a HW0 that introduces tools for the course that you are encouraged to finish by Friday, but it's not worth points or anything.

Grade Breakdown

- **Weekly Homework Assignments**
  - **Weight**: 75%
  - **Number**: Approximately 8
  - Each Assignment has two parts that contribute to your grade separately:
    - Programming (60%)
    - Conceptual (20%)
- **Checkpoints**
  - **Weight**: 10%
  - **Number:** Approximately 20 (each lecture, drop 3)
- **Learning Reflections**
  - **Weight**: 10%
  - **Number:** Approximately 10 (each week, drop 1)
- **Participation**
  - **Weight**: 5%
  - **What:** Answer polls in lecture videos within 24 hours of the lecture

Getting Help

The best place to get **asynchronous help** is EdStem. You can post questions (publicly or privately) to get help from peers or members of the course staff.

- You're encouraged to respond with your ideas to other posts!

The best place to get **synchronous help** is office hours, quiz sections or to form a study group.

- Karthik (T,W 4 - 5 pm PST)
- Rahul (M 1 - 3 pm PST)
- Timothy (F 2 - 4 pm PST)
- Svet (?)
- Quiz sections (Thursday)

RECAP LAST LECTURE

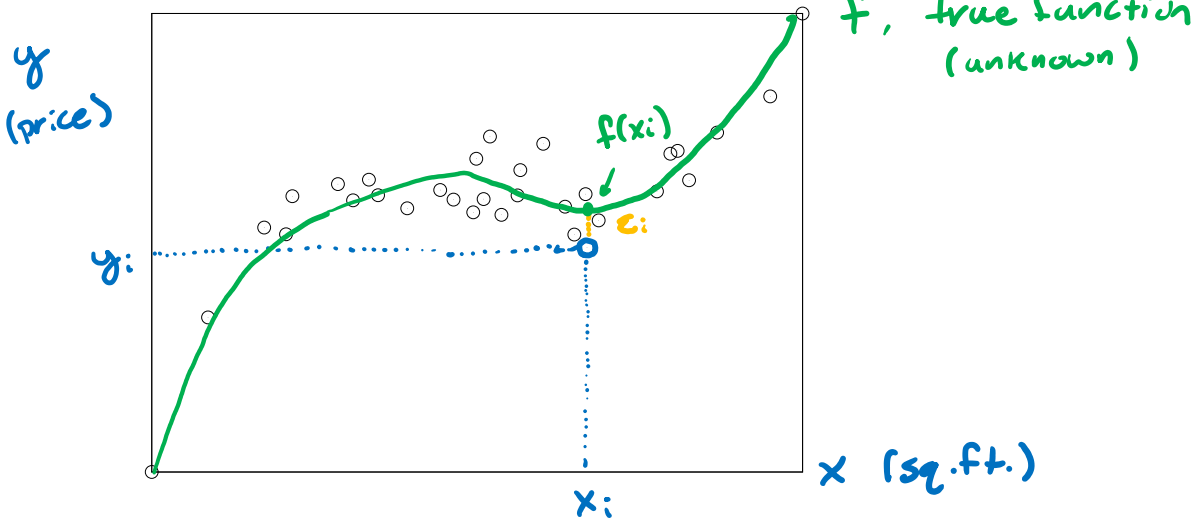
- **TOPICS COVERED LAST WEEK**
  - Linear Regression
  - Case Study: Housing prices
  - ML Models
  - Basic ML pipeline

RECAP LAST LECTURE

- **TOPICS COVERED LAST WEEK**
  - Linear Regression
  - Case Study: Housing prices
  - ML Models
  - Basic ML pipeline

Model

A **model** is how we *assume* the world works



Regression model:

$$y_i = f(x_i) + \varepsilon_i$$

$$E[\varepsilon_i] = 0$$

← "expected value" of noise is 0

"Essentially, all models are wrong, but some are useful."
- George Box, 1987

Linear Regression Model

Model: $y_i = f(x_i) + \epsilon_i$

where $f(x_i) = w_0 + w_1 x_i$

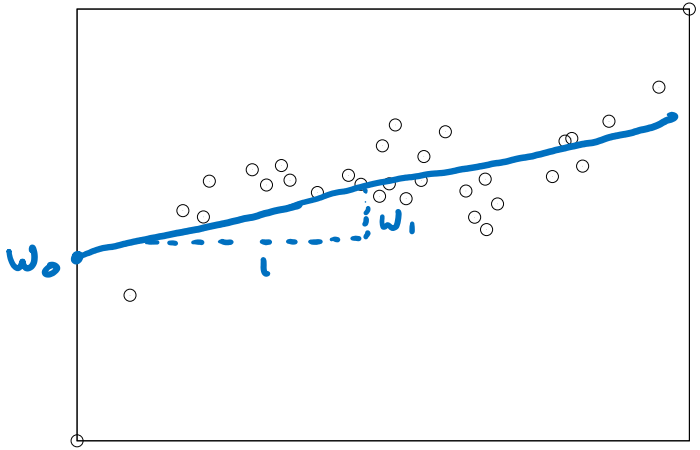
Assume the data is produced by a line.

$$y_i = \underbrace{w_0 + w_1 x_i}_{f(x_i)} + \epsilon_i$$

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 x!



$$\hat{y} = \underbrace{\hat{w}_0 + \hat{w}_1 x}_{\hat{f}(x)}$$

Why don't we add ϵ?

Checkpoint review

Question 1

Suppose we were testing a new drug treatment for a particular disease. We have gathered data from various trials that recorded the average response (a number, where 0 is low response and a higher number is a high response) in an experiment that used a particular dosage (measured in mg).

We suspect there is a linear relationship between the response (y) and the dosage in mg (x), so we decide to model this data with a linear regression model.

$$y_i = w_0 + w_1 x_i + \varepsilon_i$$

Suppose that after training a model, we find $\hat{w}_0 = 1$ and $\hat{w}_1 = 2$. Which of the following interpretations of the learned predictor are true? *Select all that apply.*

- [ ] The predicted response for a 0mg dose is expected to be 0.
- [ ] The predicted response for a 0mg dose is expected to be 1.
- [ ] The predicted response for a 0mg dose is expected to be 2.

Checkpoint review

Question 2

This question has the same setup as the previous, with different interpretations below. Which of the following interpretations of the learned predictor are true? *Select all that apply.*

- [ ] If we were to increase the dosage by 2mg, we expect that the response would increase by 1.
- [ ] If were to increase the dosage by 1mg, we expect that the response would increase by 0.002.
- [ ] If we were to increase the dosage by 1mg, we expect that the response would increase by 2.

Checkpoint review

Question 3

Suppose in the setup for the last question, we had the following dataset.

Drug Dosage	Response
10	19
2	8
3	7
9	16

Consider our learned predictor $\hat{w}_0 = 1$ and $\hat{w}_1 = 2$. What is the RSS of this learned predictor on this training dataset? Enter your answer as a number.

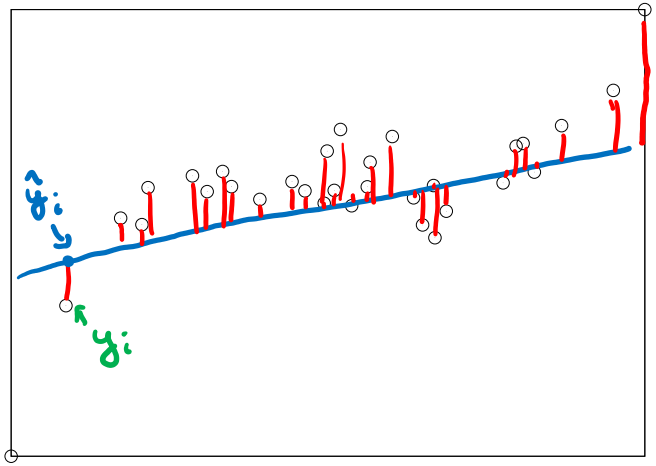
Residual Sum of Squares (RSS)

How to define error? **Residual sum of squares (RSS)**

$$RSS(w_0, w_1) = (y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \dots + (y_n - \hat{y}_n)^2$$

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

$$= \sum_{i=1}^{n} (y_i - \underbrace{(w_0 + w_1 x_i)}_{\hat{y}_i})^2$$



Note:

Also commonly used

Mean Square Error (MSE)

$$MSE(w_0, w_1) = \frac{1}{n} RSS(w_0, w_1)$$

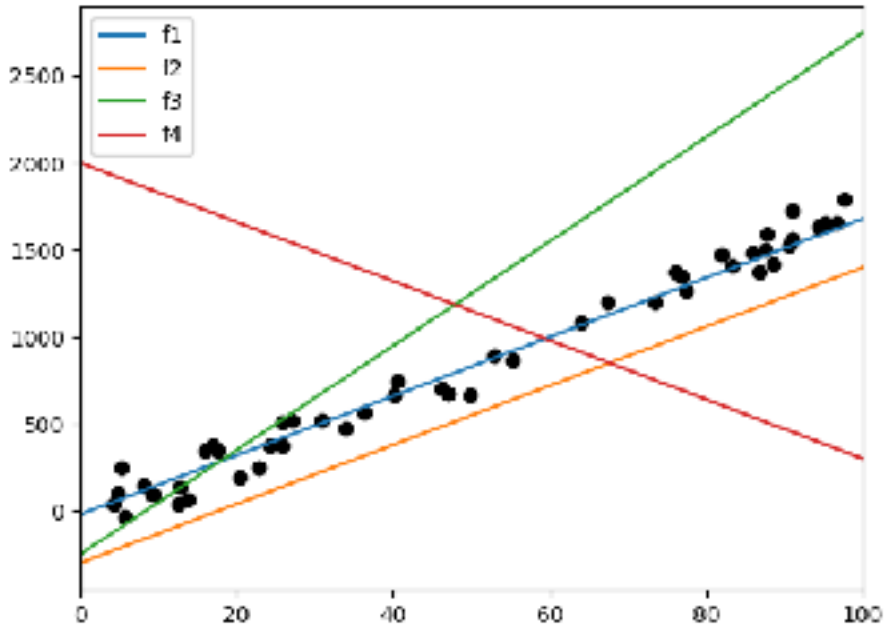
Poll Everywhere

Think

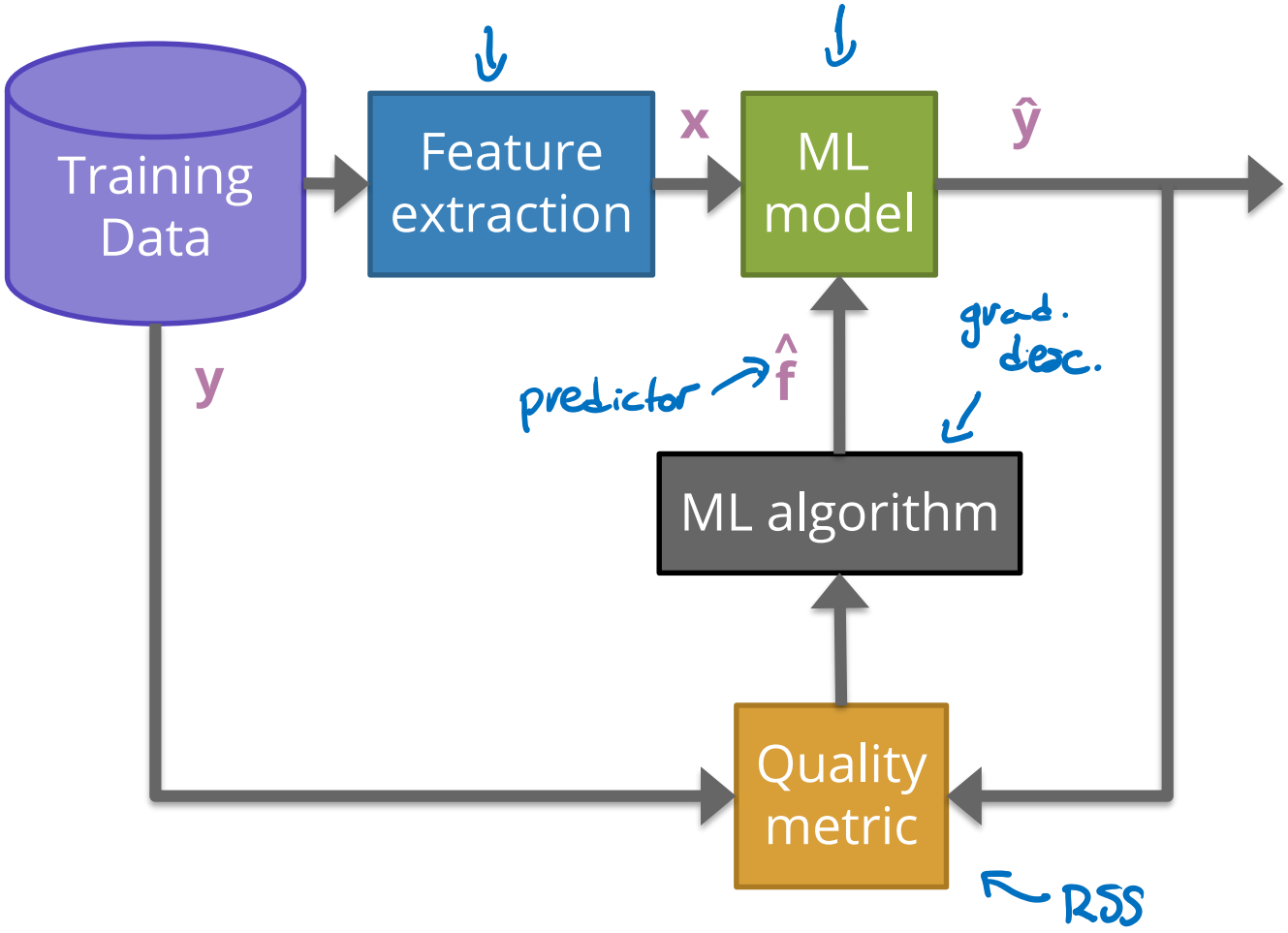
1 min

pollev.com/karthikmohan088

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



ML Pipeline


Training Data
Feature extraction
x
ML model
ŷ
y
predictor
f̂
grad. desc.
ML algorithm
Quality metric
RSS

Notation

Important: Distinction is the difference between a *data input* and a *feature*.

- Data inputs are columns of the raw data
- Features are the values (possibly transformed) for the model (done after our feature extraction $h(x)$)

Data Input: $x_i = (x_i[1], x_i[2], \ldots, x_i[d])$

Output: y_i

- x_i is the i^{th} row
- $x_i[j]$ is the i^{th} row's j^{th} data input
- $h_j(x_i)$ is the j^{th} feature of the i^{th} row

Linear Regression Recap

Notation: $w^T h(x) = \sum_{j=0}^{D} w_j h_j(x)$

Dataset

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

Feature Extraction

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

$h(x) = (h_0(x), h_1(x), \ldots, h_D(x))$

Regression Model

$$y = f(x) + \epsilon$$
$$= \sum_{j=0}^{D} w_j h_j(x) + \epsilon$$
$$= w^T h(x) + \epsilon$$

Quality Metric

$$RSS(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

Predictor

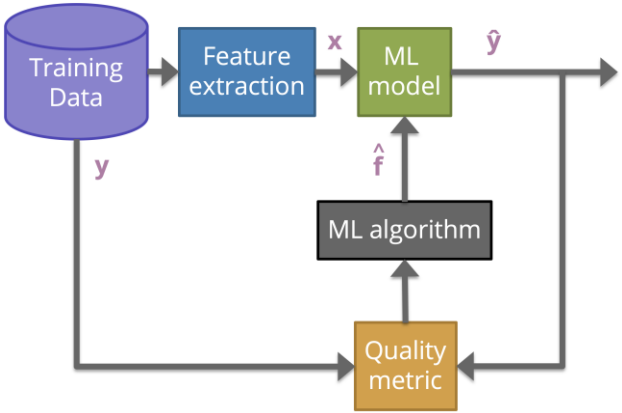
$$\hat{w} = \min_w RSS(w)$$

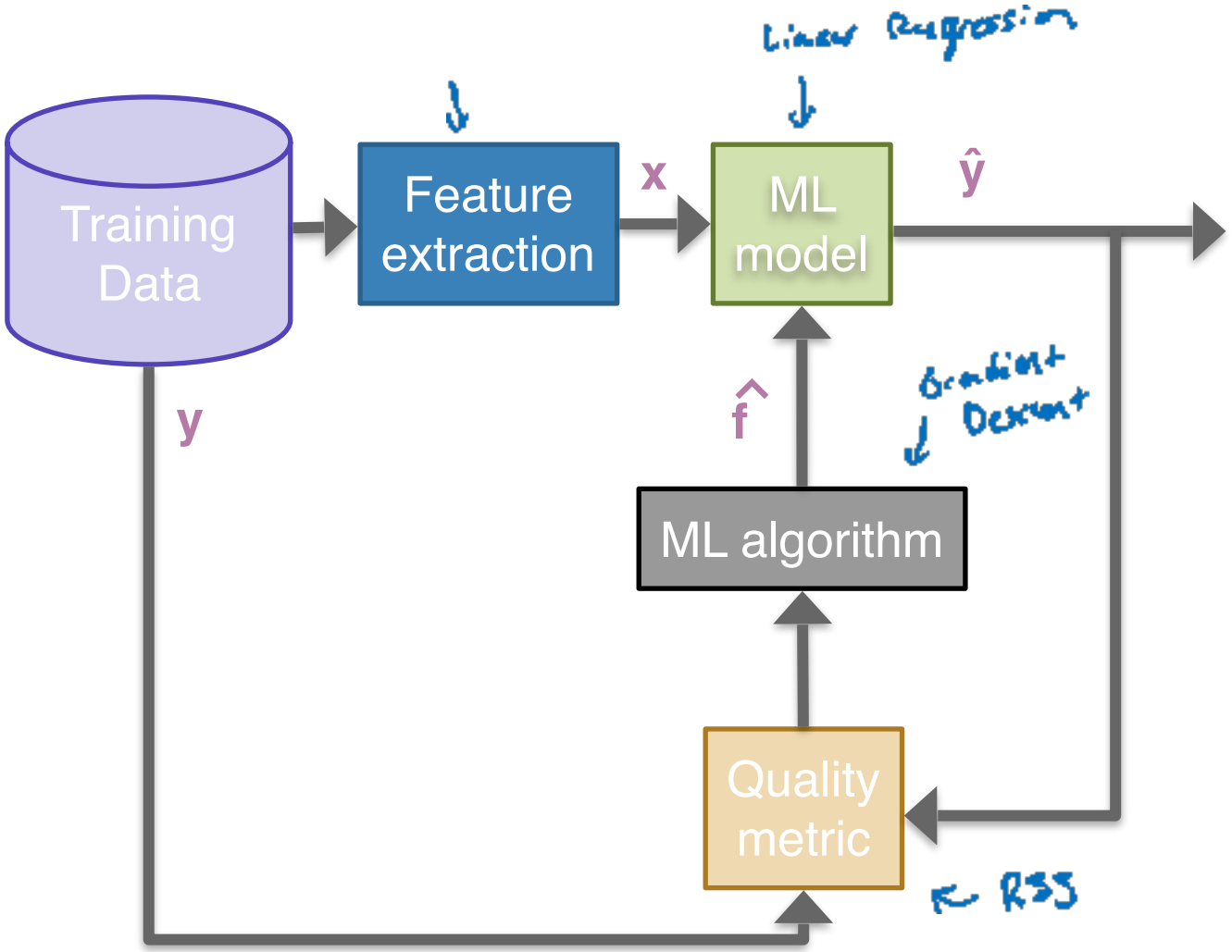
ML Algorithm

Optimized using Gradient Descent

Prediction

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




Linear Regression
Training Data
Feature extraction
x
ML model
ŷ
y
f^
Gradient Descent
ML algorithm
Quality metric
RSS

TODAY'S LECTURE (NEW MATERIAL)

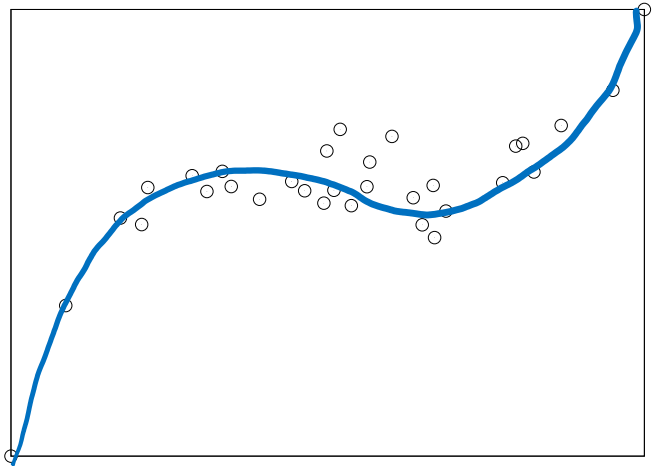
- **TOPICS**
  - Feature Engineering
  - Higher order Features
  - Polynomial Regression

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!

$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + w_3 x_i^3 + \epsilon_i$$



Polynomial Regression

Model

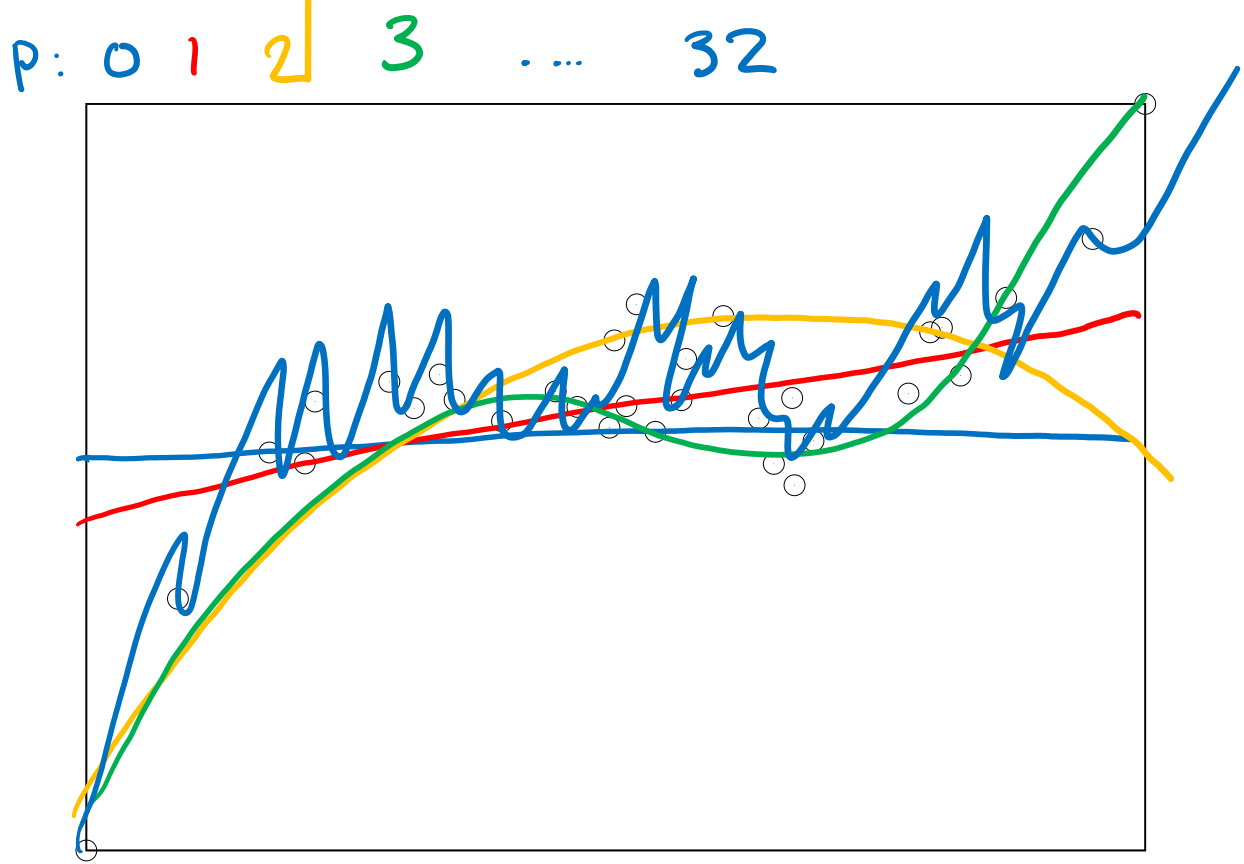
$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \ldots + w_p x_i^p + \epsilon_i$$

Just like linear regression, but uses more features!

Feature	Value	Parameter
0	1 (constant)	w_0
1	x	w_1
2	x^2	w_2
…	…	…
p	x^p	w_p

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

Polynomial Regression



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

Features

$h_j(x_i)$ = feature map

Features are the values we select or compute from the data inputs to put into our model. **Feature extraction** is the process of turning the data into features.

Model

$$y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + \ldots + w_D h_D(x_i) + \epsilon_i$$

$$= \sum_{j=0}^{D} w_j h_j(x_i) + \epsilon_i$$

Feature	Value	Parameter
0	$h_0(x)$ often 1 (constant)	w_0
1	$h_1(x)$ $= x_i$	w_1
2	$h_2(x)$ $= x_i^2$	w_2
…	… $= x_i^3 \cdot \log(x_i)$	…
D	$h_D(x)$ $= e^{x_i}$	w_D

Housing Prices Case Study: Adding Other Inputs

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

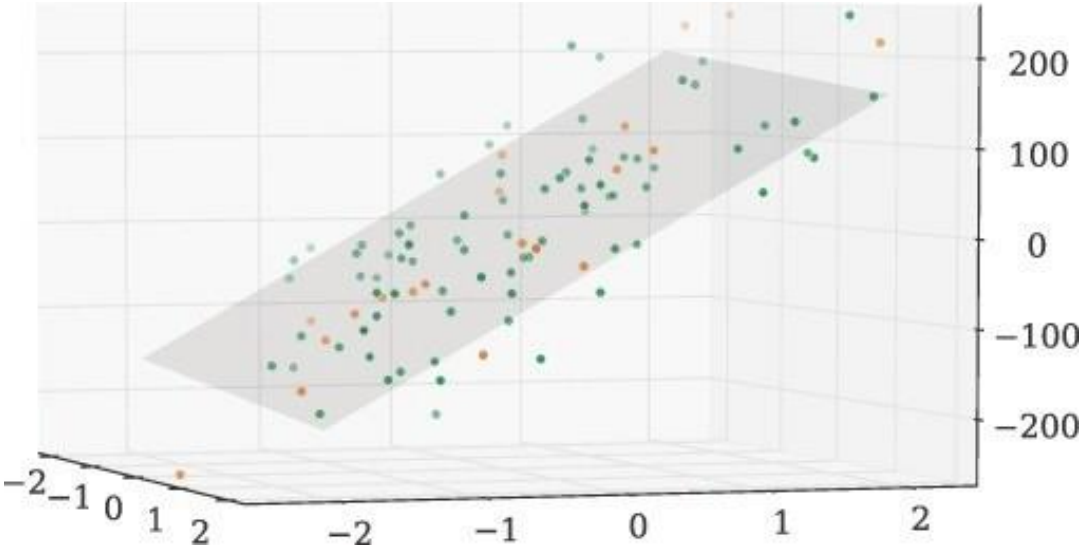
- Each row is a single house.
- Each column (except Value) is a data input.

sq. ft.	# bathrooms	owner's age	…	value
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

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



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

Notation

Important: Distinction is the difference between a *data input* and a *feature*.

- Data inputs are columns of the raw data
- Features are the values (possibly transformed) for the model (done after our feature extraction $h(x)$)

Data Input: $x_i = (x_i[1], x_i[2], \ldots, x_i[d])$

Output: y_i

- x_i is the i^{th} row
- $x_i[j]$ is the i^{th} row's j^{th} data input
- $h_j(x_i)$ is the j^{th} feature of the i^{th} row

Features

You can use anything you want as features and include as many of them as you want!

Generally, more features means a more complex model. This might not always be a good thing!

Choosing good features is a bit of an art.

Feature	Value	Parameter
0	1 (constant)	w_0
1	$h_1(x)$ … $x[1]$ = sq. ft.	w_1
2	$h_2(x)$ … $x[2]$ = # bath	w_2
…	…	…
D	$h_D(x)$ … like $\log(x[7]) * x[2]$	w_D

Linear Regression Recap

Dataset

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

Feature Extraction

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

$h(x) = (h_0(x), h_1(x), \ldots, h_D(x))$

Regression Model

$$y = f(x) + \epsilon$$
$$= \sum_{j=0}^{D} w_j h_j(x) + \epsilon$$
$$= w^T h(x) + \epsilon$$

Quality Metric

$$RSS(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

Predictor

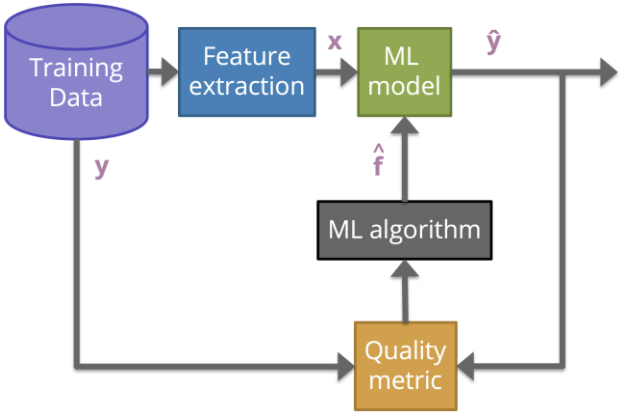
$$\hat{w} = \min_w RSS(w)$$

ML Algorithm

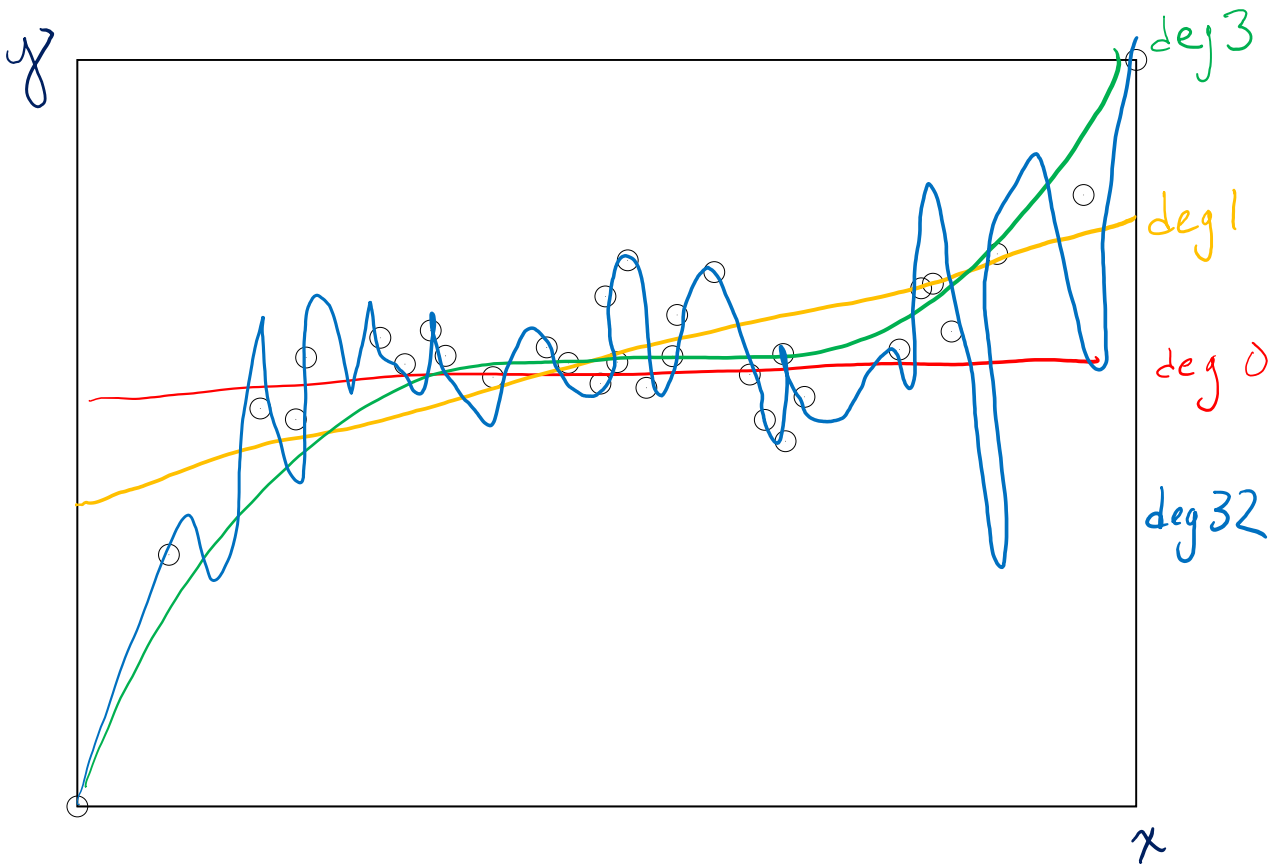
Optimized using Gradient Descent

Prediction

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



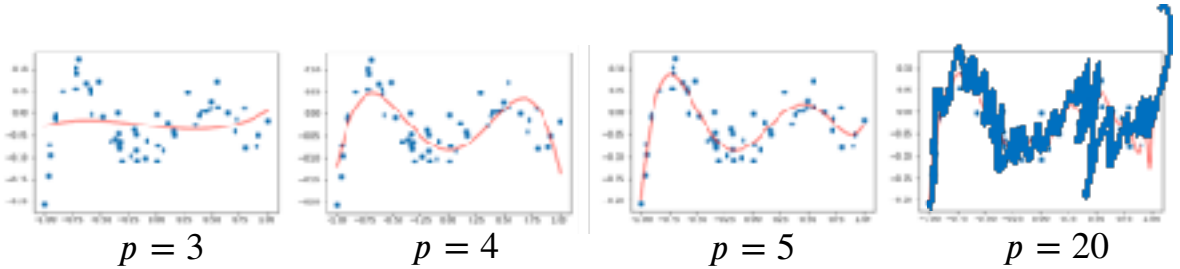
Polynomial Regression



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

Polynomial Regression

Consider using different degree polynomials on the same dataset



$p = 3$ $p = 4$ $p = 5$ $p = 20$

Which one has a lower RSS on this dataset?

It seems like minimizing the RSS is not the whole story here…

Performance

Why do we train ML models?

We generally want them to do well on **future** data

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

- 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.

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

Future Performance

General Loss Function $L(y, \hat{f}(x))$

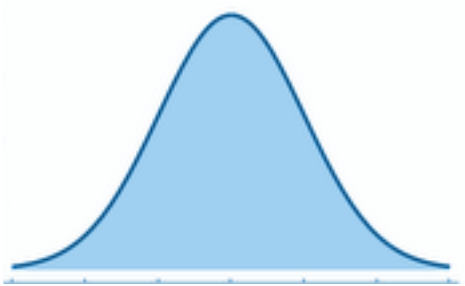
↳ $L(y, \hat{f}(x)) = (y - \hat{f}(x))^2$

What we care about is how well the model will do in the future.

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

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

$y = f(x) + \varepsilon$



Sq. Ft. x

Price | Sq. Ft. $y|x$

True Error

$$E[L(y, \hat{f}(x))] = \sum_{x \in X} \sum_{y \in Y} L(y, \hat{f}(x))\, p(x, y)$$

↑ all possible (x, y) pairs

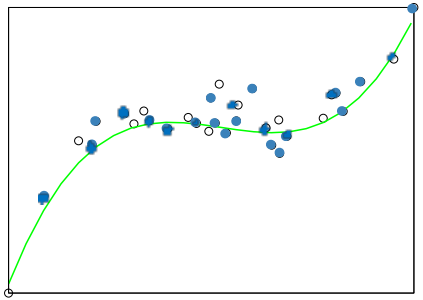
Model Assessment

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,** but we can't know that without having an infinite amount of data!

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

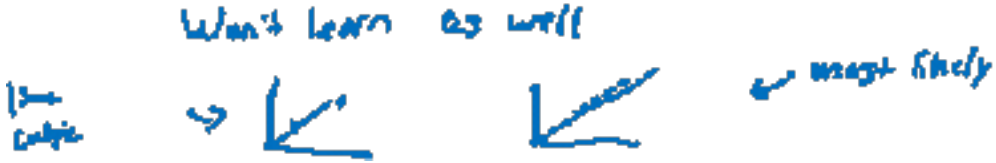
Call the error on the test set the **test error**

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!

Poll Everywhere

Think

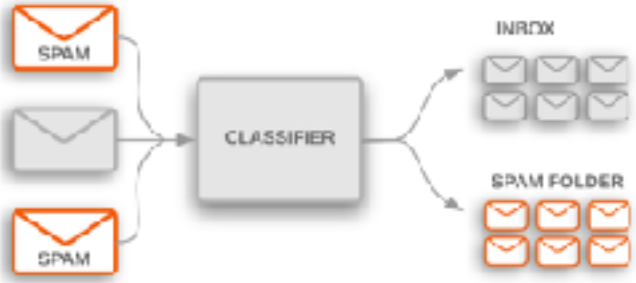
1 min

pollev.com/
karthikmohan088

SPAM DETECTION

You are tasked with coming up with a machine learning model to classify emails as spam or not spam. You start off with a data set of 1000 emails of which you know that 10 of them are spam and the remaining are not spam.

When you test your ML model on a new set of 100 emails, your model correctly predicts all emails as not spam. What can be said about the success of your ML model given what you have seen so far?

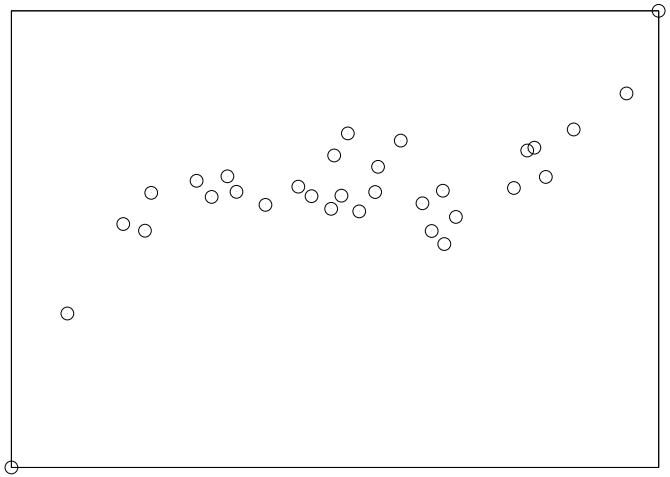


Model Complexity

Train Error

What happens to training error as we increase model complexity?

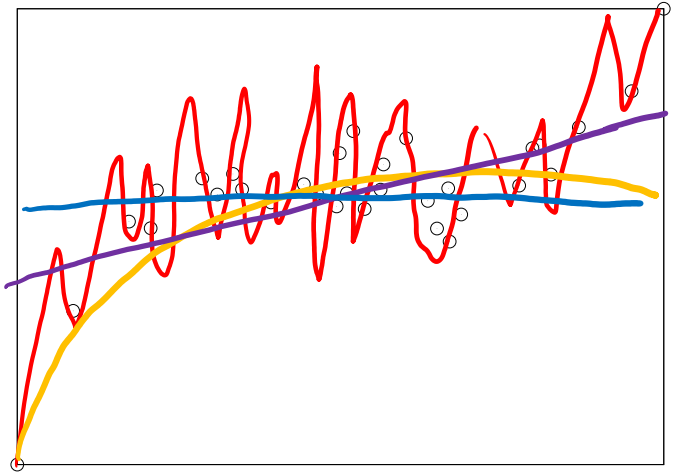
- Start with the simplest model (a constant function)
- End with a very high degree polynomial



Train Error

What happens to training error as we increase model complexity?

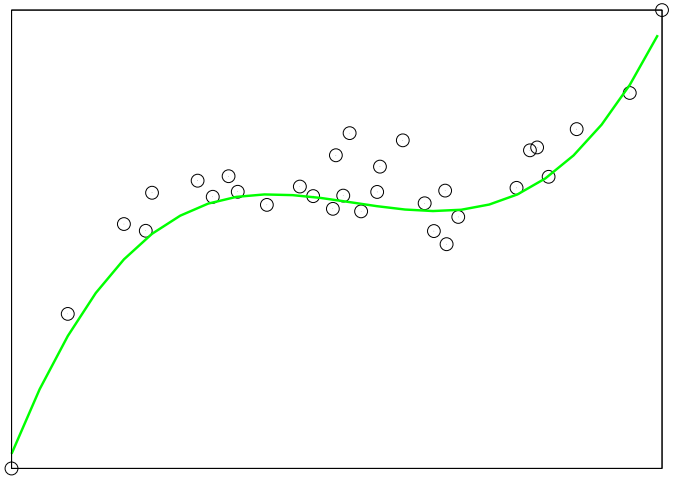
- Start with the simplest model (a constant function)
- End with a very high degree polynomial



True Error

What happens to true error as we increase model complexity?

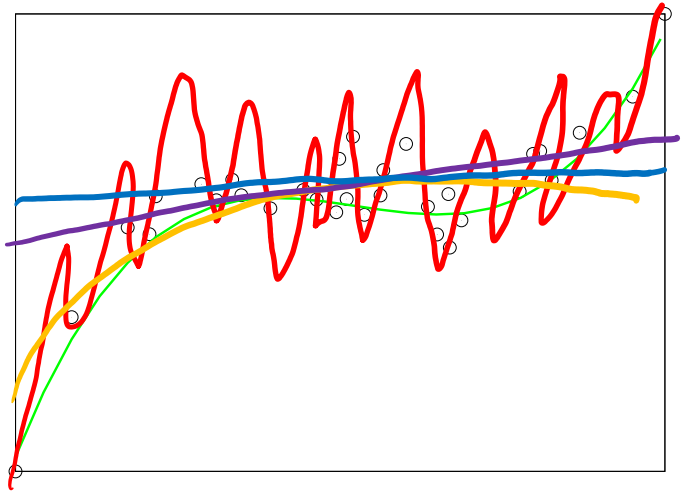
- Start with the simplest model (a constant function)
- End with a very high degree polynomial

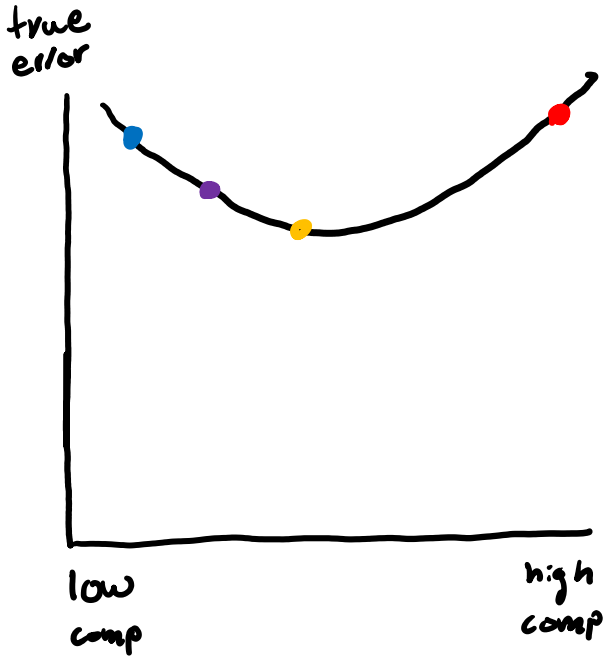


True Error

What happens to true error as we increase model complexity?

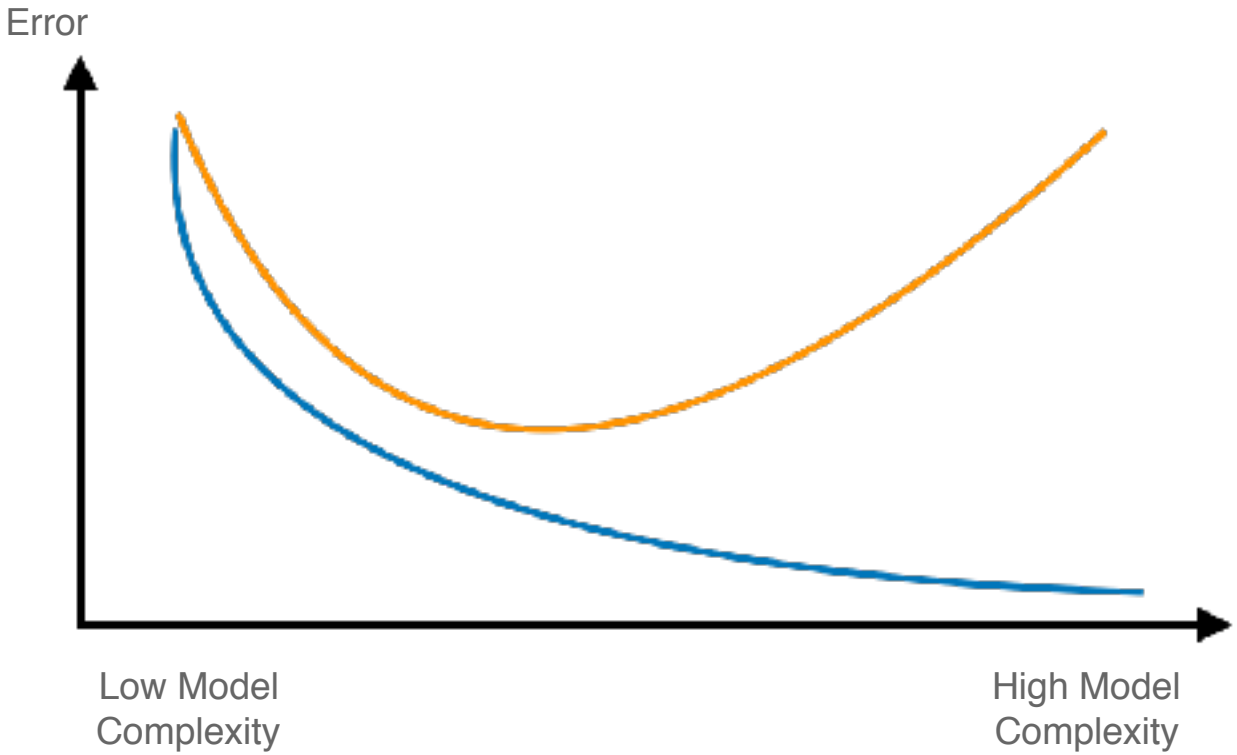
- Start with the simplest model (a constant function)
- End with a very high degree polynomial





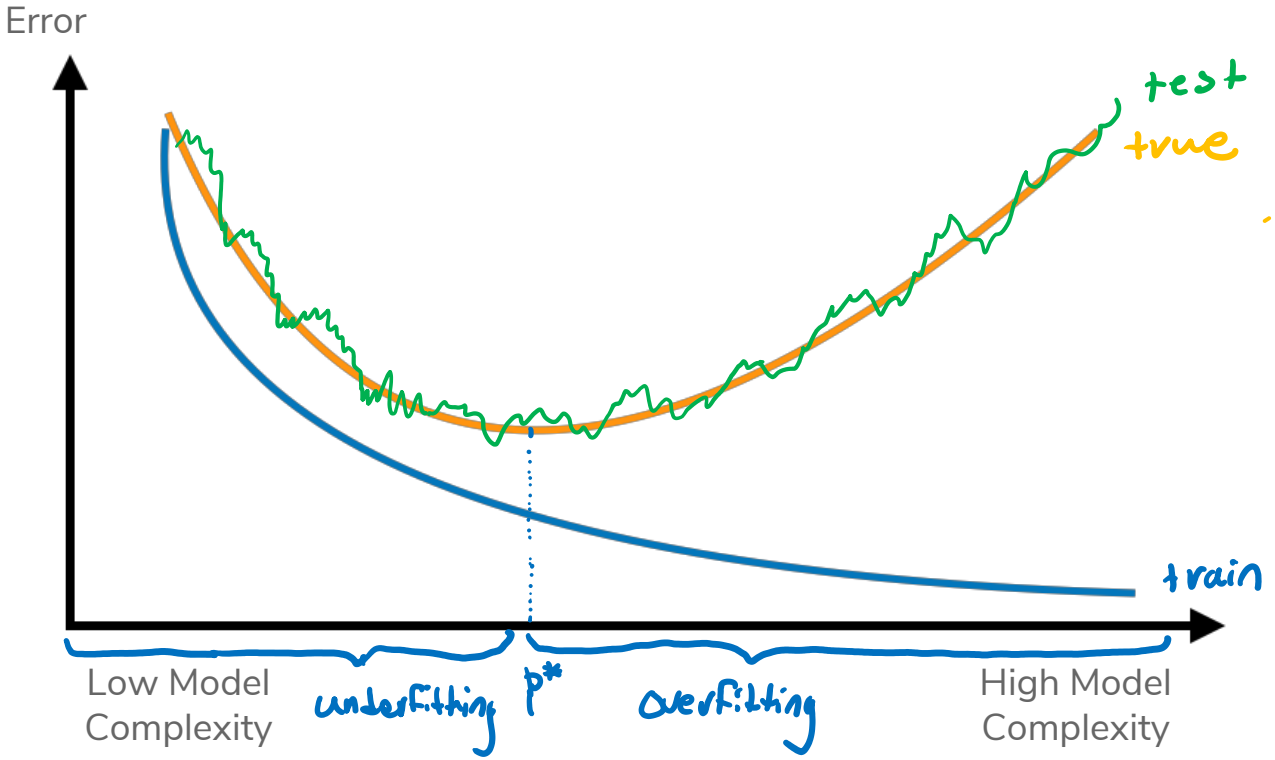
Train/True Error

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



Train/True Error

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

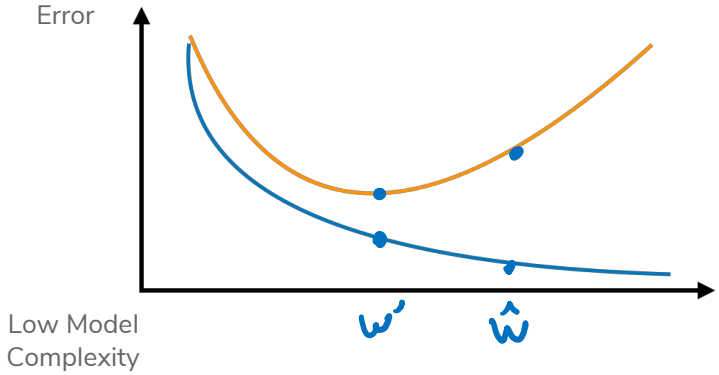


Overfitting

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

Overfitting happens when, you train a predictor $\hat{w}$, but there exists another predictor w' from that model that has the following properties

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



Bias-Variance Tradeoff

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 / Overfitting are a result of certain types of errors

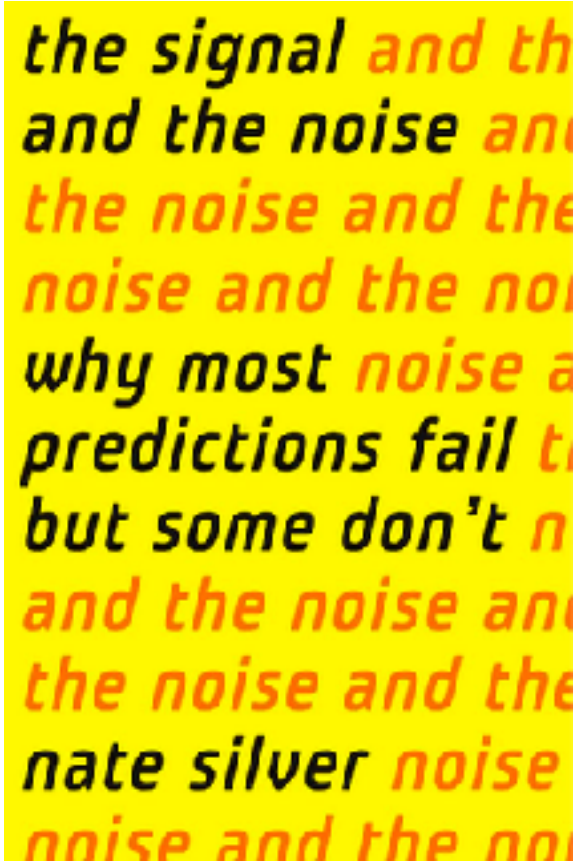
Signal vs. Noise

Learning from data relies on balancing two aspects of our data

- **Signal**
- **Noise**

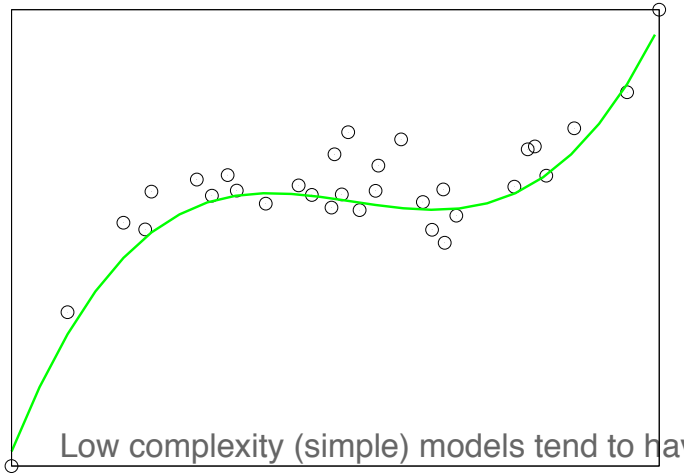
Complex models make it easier to fit too closely to the noise

Simple models have trouble picking up the signal



Bias

A model that is too simple fails to fit the signal. In some sense, this signifies a fundamental limitation of the model we are using to fail to fit the signal. We call this type of error **bias**.



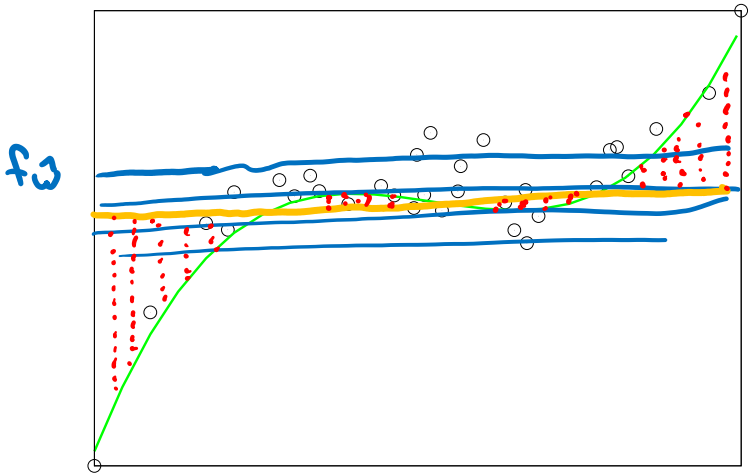
Low complexity (simple) models tend to have high bias.*

Bias

Notation: $\hat{f} = f_{\hat{w}}$ read model f with parameters $\hat{w}$

A model that is too simple fails to fit the signal. In some sense, this signifies a fundamental limitation of the model we are using to fail to fit the signal. We call this type of error **bias**.

Bias $E[|f(x) - \overline{f_{\hat{w}}(x)}|]$

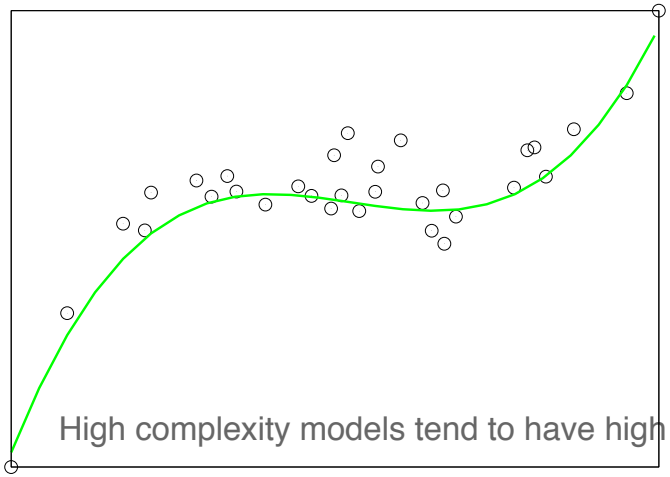


English: Expected diff between true fun. and "average" predictor

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

Variance

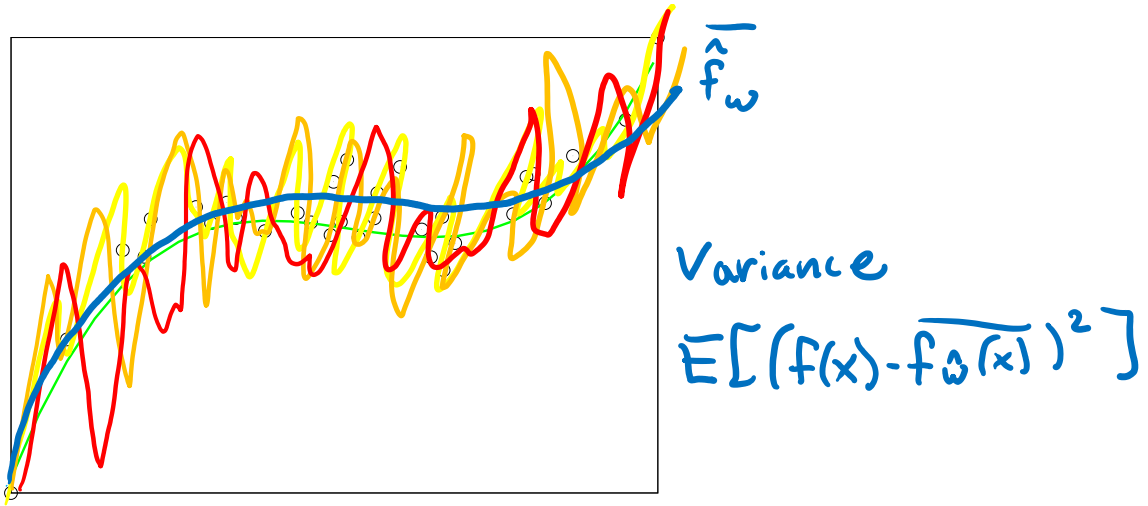
A model that is too complicated for the task overly fits to the noise. The flexibility of the complicated model makes it capable of memorizing answers rather than learning general patterns. This contributes to the error as **variance**.



High complexity models tend to have high variance.*

Variance

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



High complexity models tend to have high variance.*

Bias-Variance Tradeoff

It turns out that bias and variance live on a spectrum, increasing one tends to decrease the other

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

In the case for squared error with regression

$$Error = Bias^2 + Variance + Noise$$

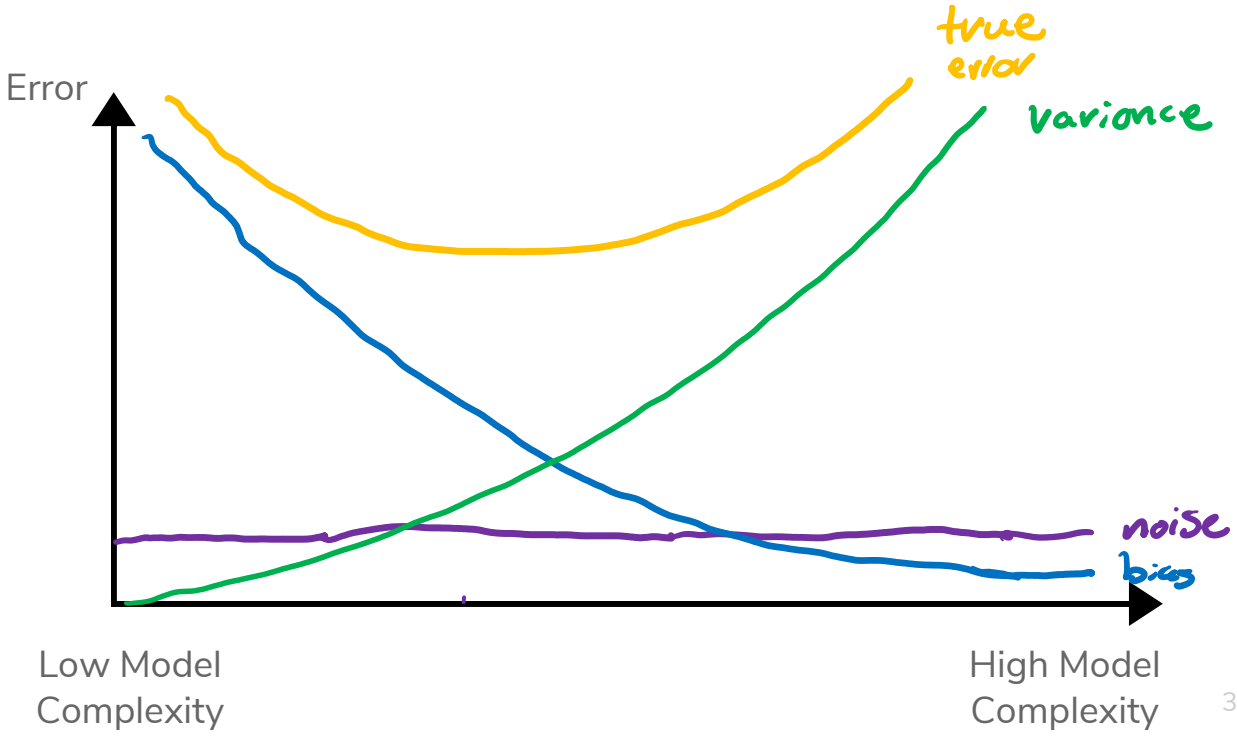
Noise comes from the regression model (ϵ_i) and is impossible to avoid!

Brain Break

Bias-Variance Tradeoff

Visually, this looks like the following!

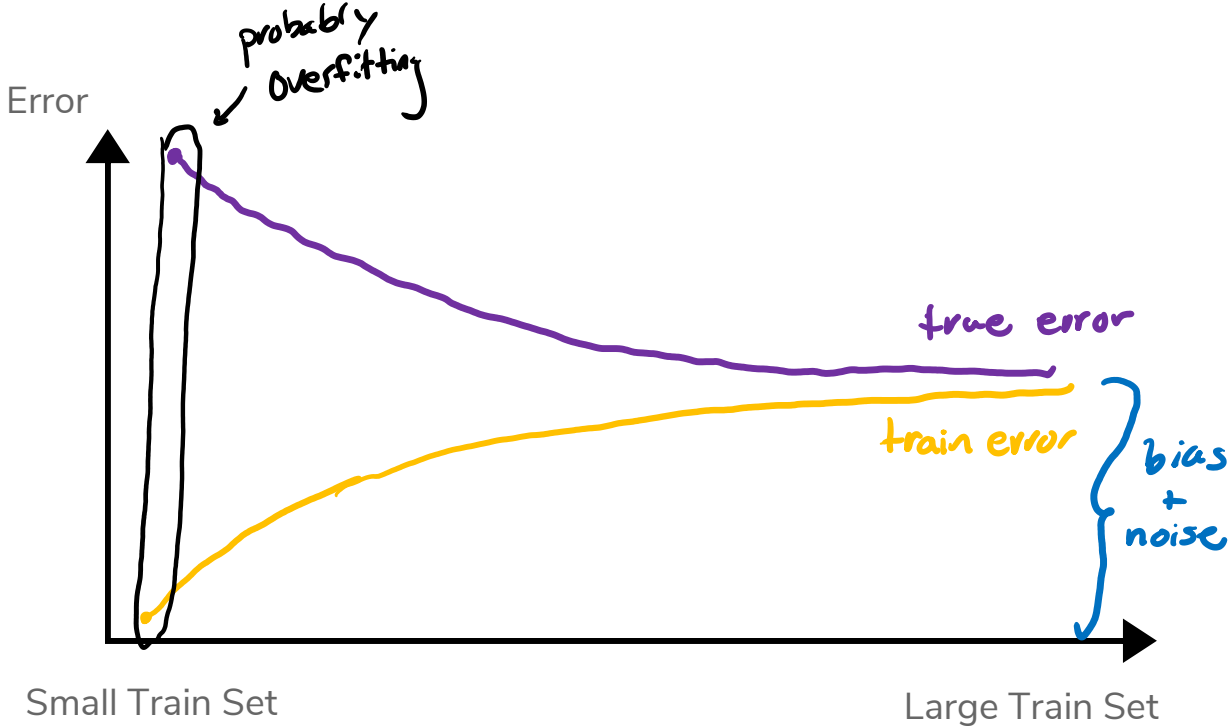
$$Error = Bias^2 + Variance + Noise$$



Dataset Size

p = 13

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



Poll Everywhere

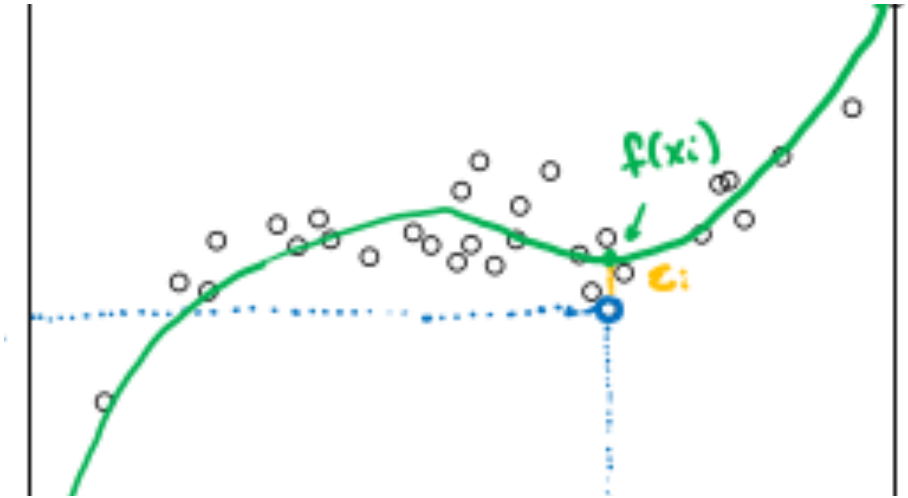
Think

1 min

pollev.com/
karthikmohan088

POLYNOMIAL FITTING

Let’s take the housing prices data set. Consider two ML models - One with a polynomial fitting of degree p = 1 and another with degree p = 8. Check all statements that are true.



Poll Everywhere

Think

1 min

pollev.com/
karthikmohan088

POLYNOMIAL FITTING

Let's take the housing prices data set. Consider two ML models - One with a polynomial fitting of degree p = 1 and another with degree p = 8. Check all statements that are true.

p=1 would have higher bias

p=8 is a more complex model

p=1 would have lower variance than p=8

p=8 would be more prone to overfit

Question: What might be a sweet spot for p then?

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

Think

1 min

pollev.com/
karthikmohan088

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

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

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, but now we don't have a good estimate of the true error anymore.
- 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.

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

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

Validation Set

The process generally goes

```
train, validation, test = split_data(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

The process generally goes

```
train, validation, test = split_data(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! ☹️

Cross Validation

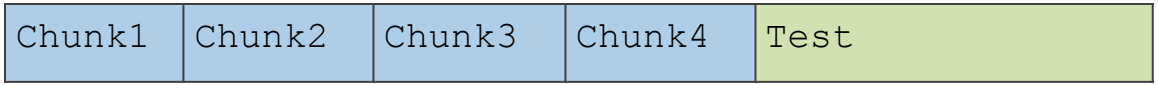
In the pre-lecture videos for next week, we will introduce another way to perform validation called **cross-validation**.

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 chunks.



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

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

Don’t have to actually get rid of any training data!

Cons

Can be a bit slow. For each model complexity, trains models!

For best results, need to make really big

- Theoretical best estimator is to use
  - Called "Leave One Out Cross Validation”
- In practice, people use to

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