CSE/STAT 416

Assessing Performance

Vinitra Swamy
University of Washington
June 24, 2020

Announcements

- Slides + annotations can be found on the course website
- Jupyter Notebooks (Local or Colab)
  - notebook autograder
- Section tomorrow will give you practice writing some of the code for HW1
  - HW1 will be released shortly after lecture

Linear Regression Model

Assume the data is produced by a line.

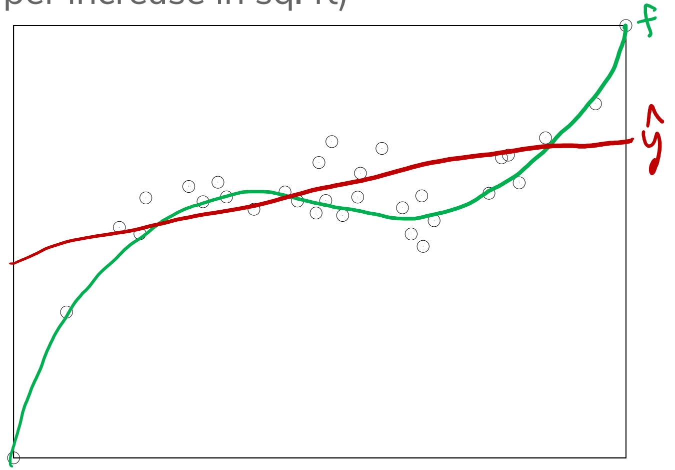
$$y_i = w_0 + w_1 x_i + \epsilon_i$$

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

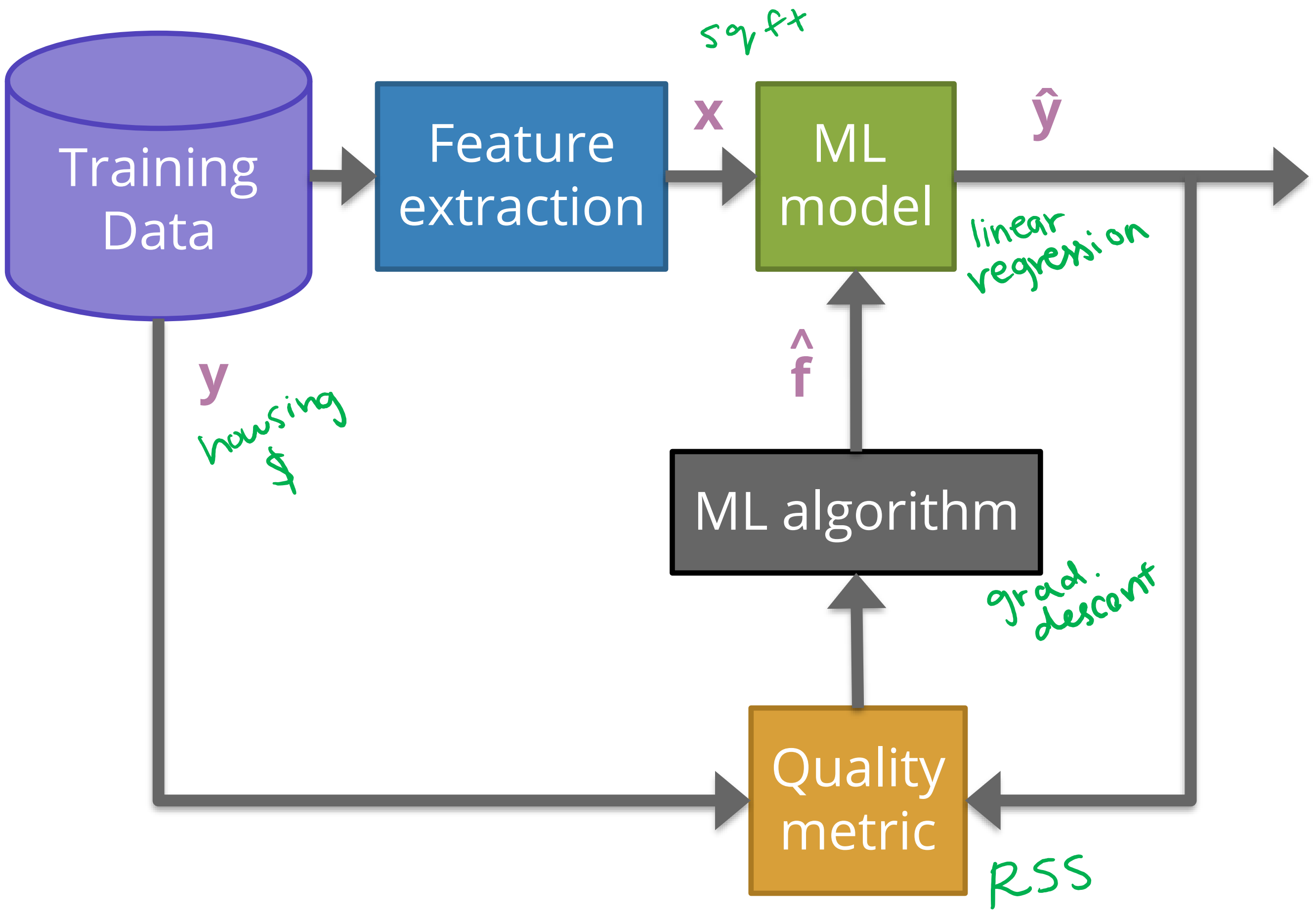
- w_0 is the intercept ($ of the land with no house)
- w_1 is the slope ($ increase per increase in sq. ft)

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

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



ML Pipeline


Training Data
Feature extraction
ML model
ML algorithm
Quality metric
x
sq ft
ŷ
linear regression
f̂
y
housing $
grad. descent
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: $x_i = (x_i[1], x_i[2], \dots, x_i[d])$

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

Dataset

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

Feature Extraction

$h(x)\colon \mathbb{R}^d \to \mathbb{R}^D$

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

Regression Model

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

Quality Metric

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

Predictor

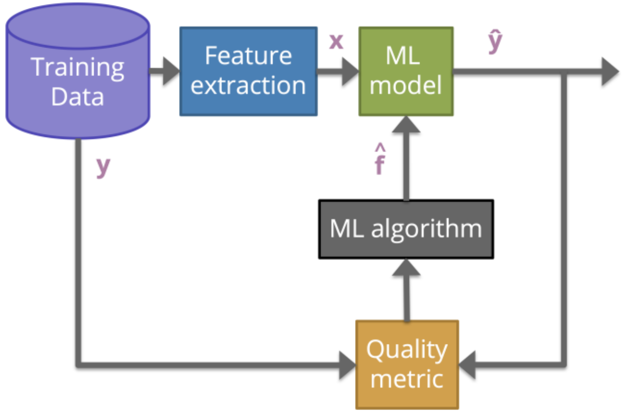
$$\hat{w} = \min_{w} RSS(w)$$

ML Algorithm

Optimized using Gradient Descent

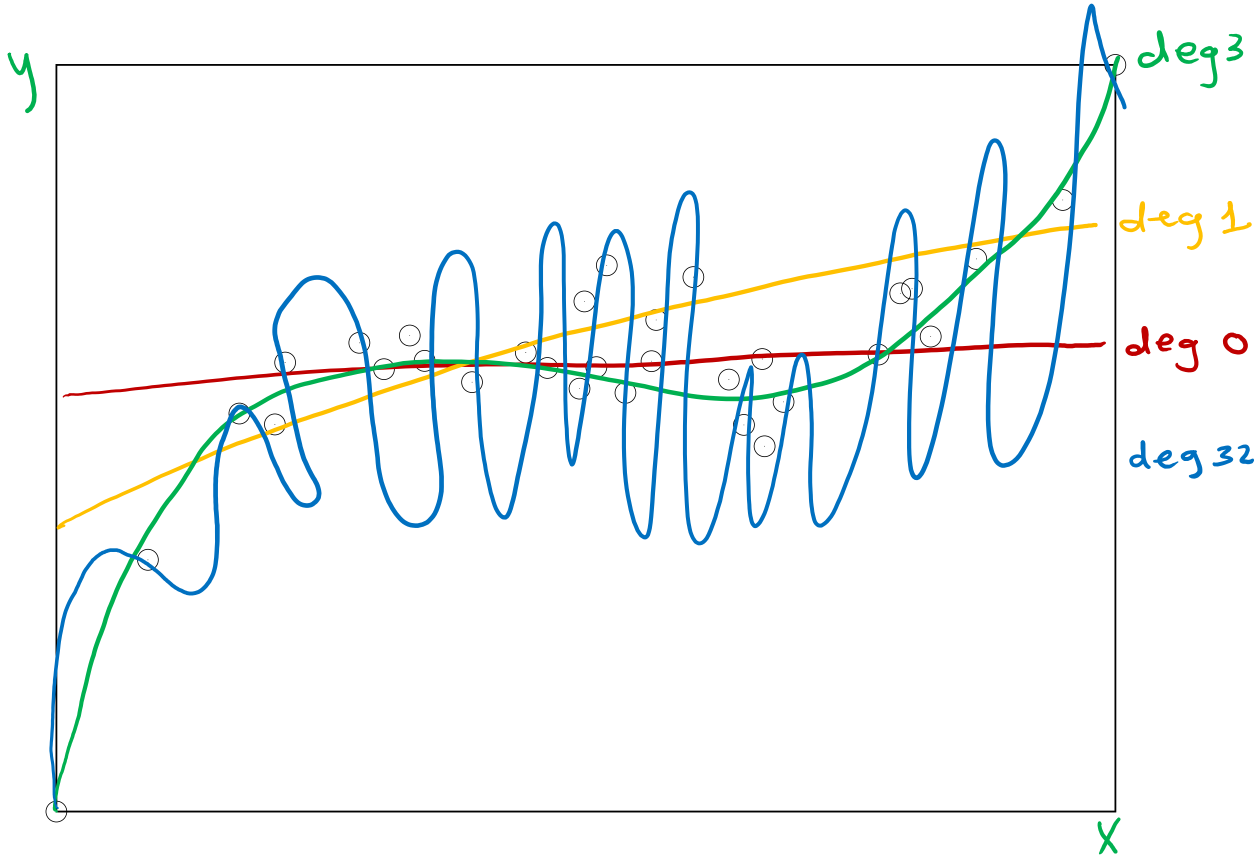
Prediction

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



Assessing Performance

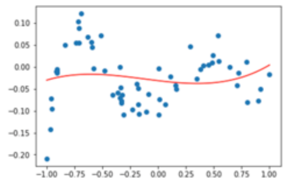
Polynomial Regression

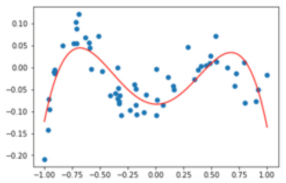


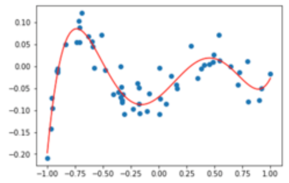
How do we decide what the right choice of p is?

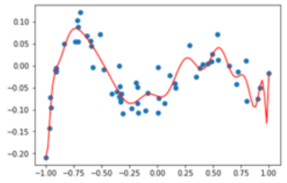
Polynomial Regression

Consider using different degree polynomials on the same dataset


$p = 3$


$p = 4$


$p = 5$


$p = 20$

Which one has a lower RSS on this dataset?

$p = 20$

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

A model that best minimizes RSS on the data it learned will **memorize** the data, not **generalize** well.

- Practice exam vs. actual exam

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

Future Performance

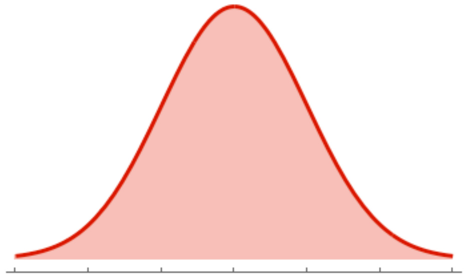
general loss function: $L(y, \hat{f}(x))$

$\rightarrow$ so far, we have used RSS: $L(y, \hat{f}(x)) = (y - \hat{f}(x))^2$

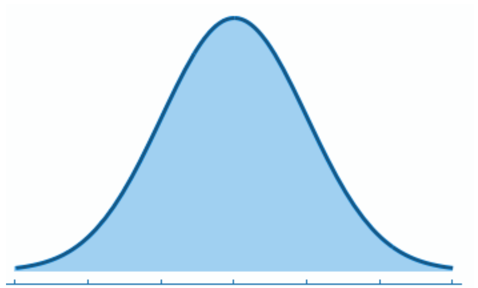
What we care about is how well the model will do in the future.

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

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



Sq. Ft.



Price | Sq. Ft.

True Error

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

expected loss over all possible x, y pairs

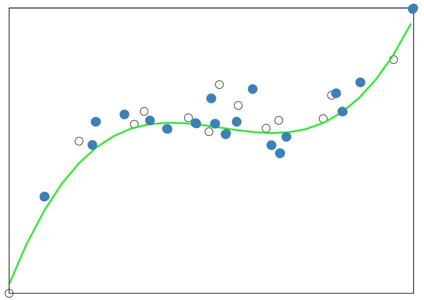
Model Assessment

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

Test error: error on the test set

$$RSS_{test}(\hat{w}) = \sum_{i \in Test} \left(y_i - \underbrace{f_{\hat{w}}(x_i)}\right)^2 \longrightarrow f_{\hat{w}} \text{ is the same as } \hat{f}$$

~ model learned w/ parameters $\hat{w}$ ~

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

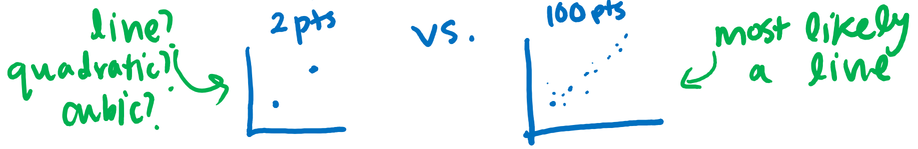
Train/Test Split

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

Bigger test set $\Rightarrow$ better estimate of the error

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)

model will not learn as well



In practice people generally do train:test as either

- 80:20
- 90:10

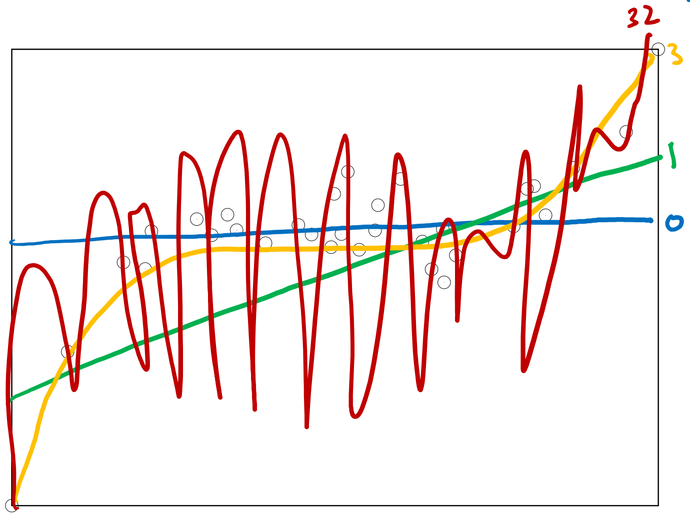
Should be split randomly!

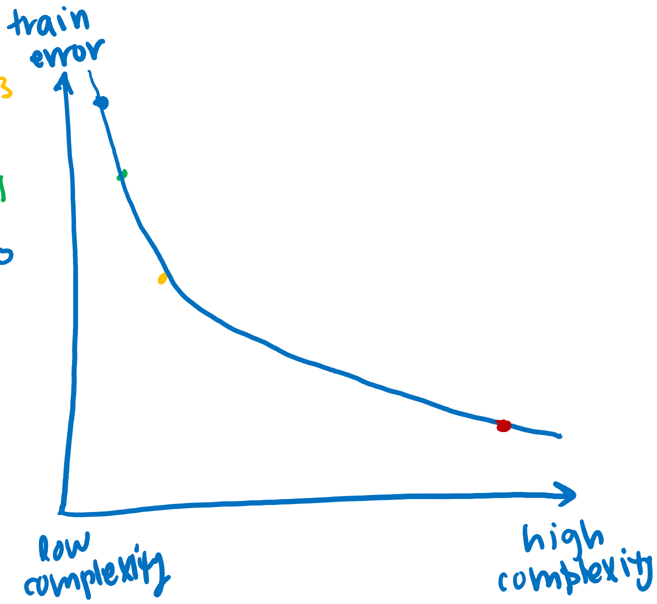
Important: Never train your model on data in the test set!

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

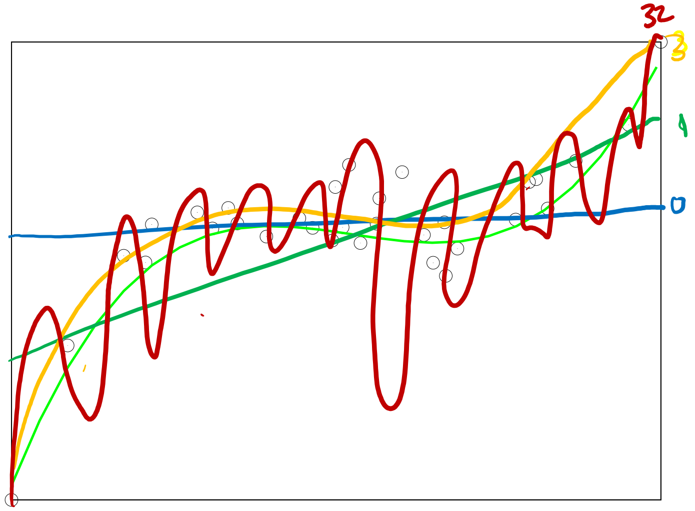


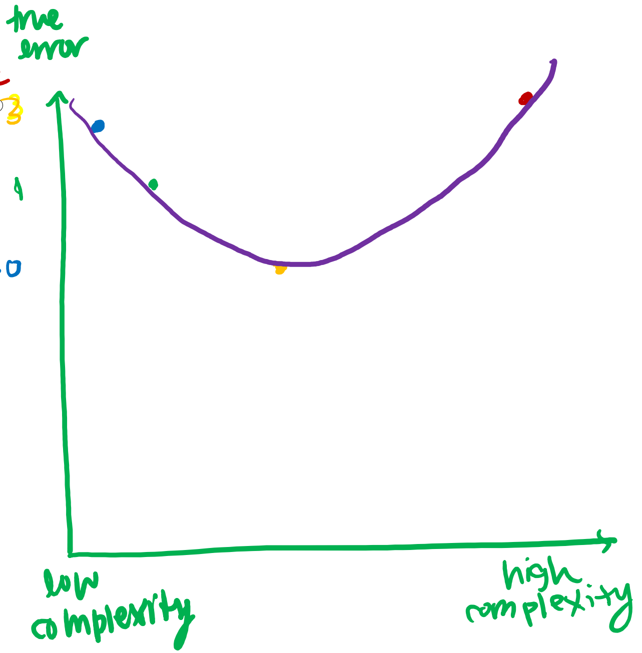


True Error

What happens to true error as we increase model complexity?

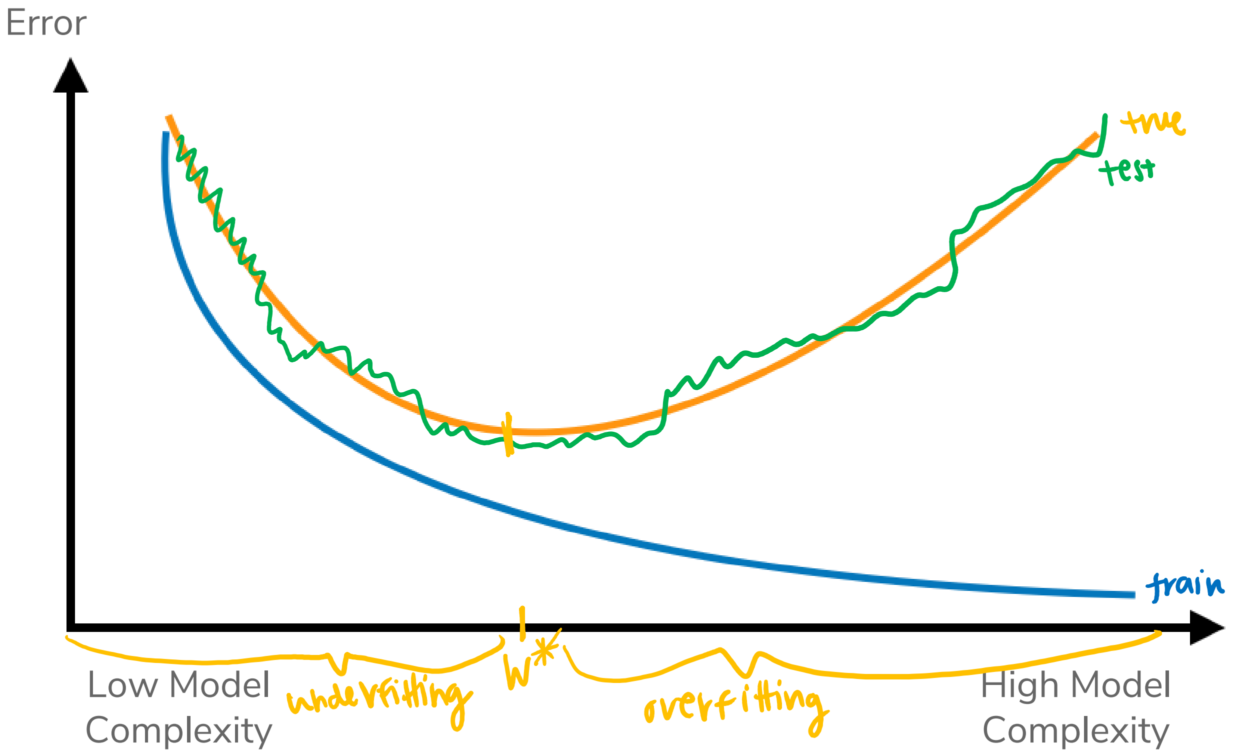
- Start with the simplest model (a constant function)
- End with a very high degree polynomial





Train/True Error

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

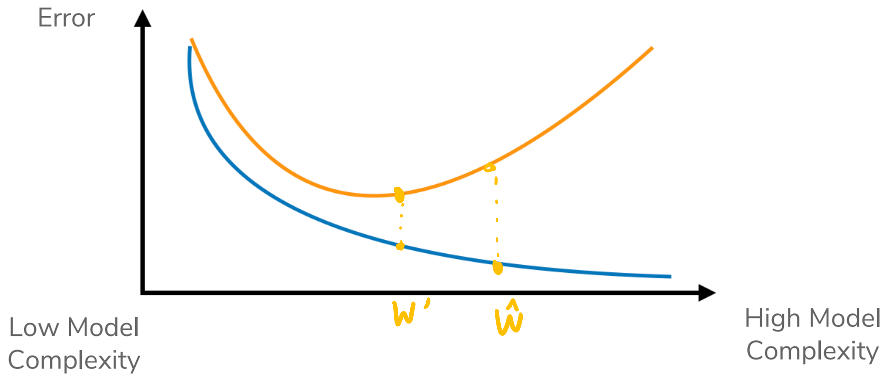


Overfitting

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

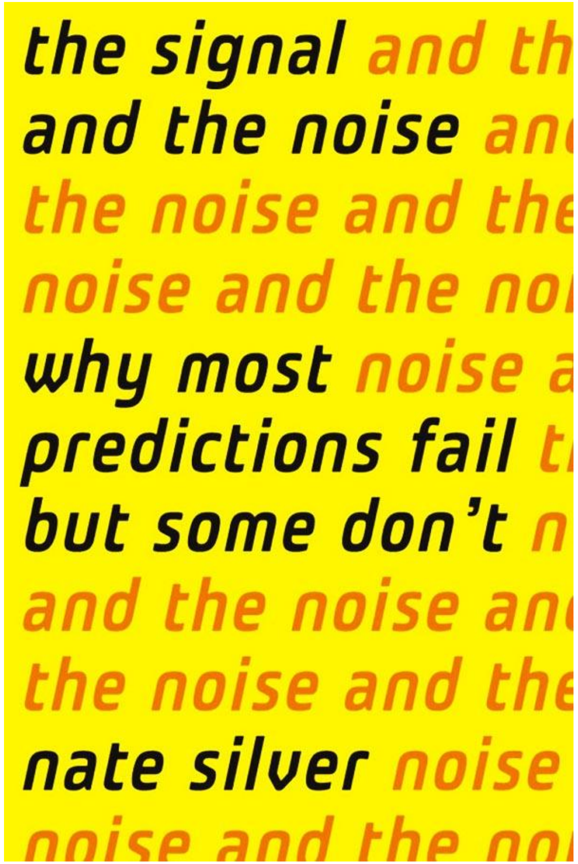
Signal vs. Noise

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

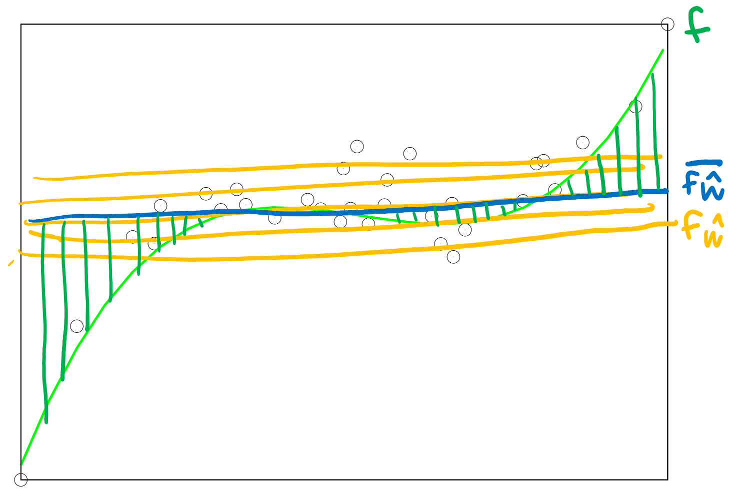
Simple models have trouble picking up the signal



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**.

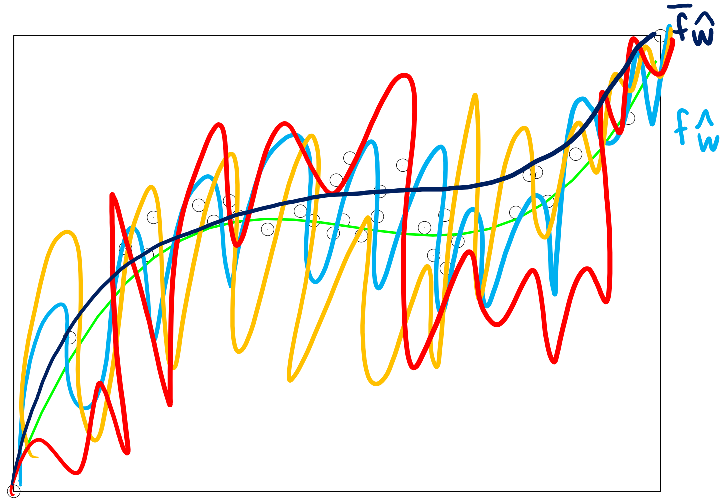
$$\text{Bias} = E\left[\left|\underbrace{f(x)}_{\text{true function}} - \underbrace{\overline{f_{\hat{w}}}(x)}_{\text{average predictor}}\right|\right]$$



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**.



variance =

$$E\left[\left(\underbrace{\bar{f}_{\hat{w}}(x)}_{\text{avg predictor}} - \underbrace{f_{\hat{w}}(x)}_{\text{specific predictor}}\right)^2\right]$$

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

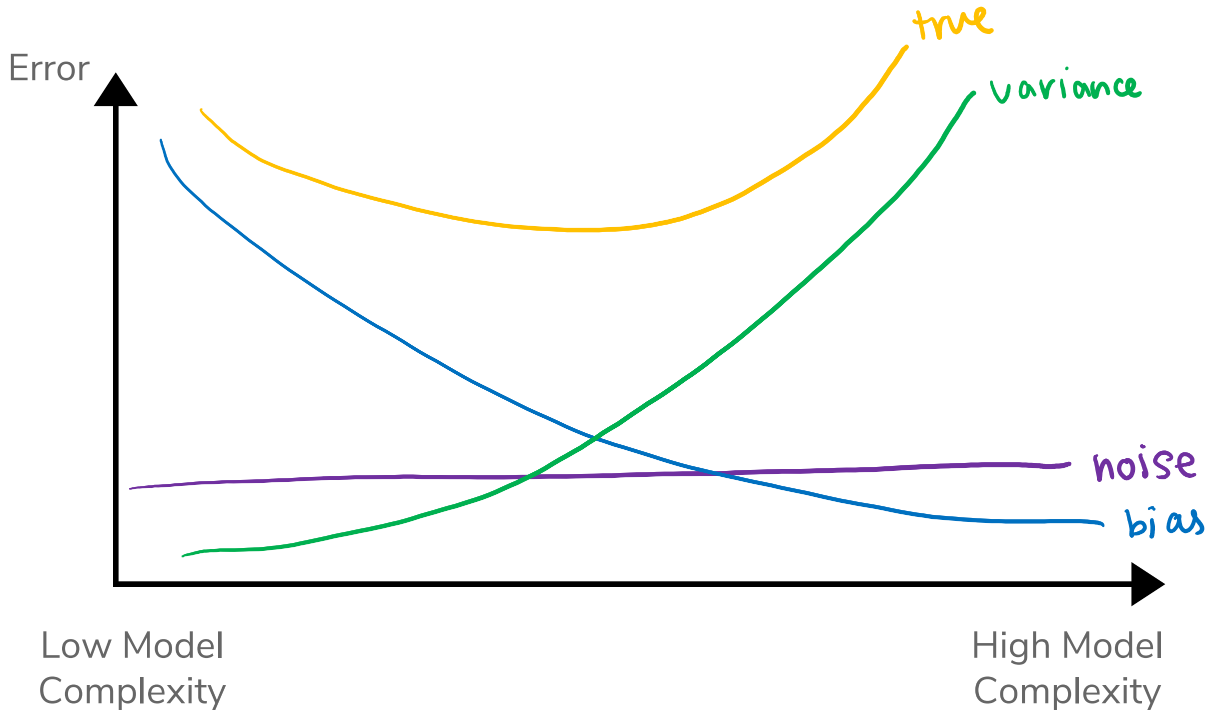
$$Error = Bias^2 + Variance + Noise$$

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

Bias-Variance Tradeoff

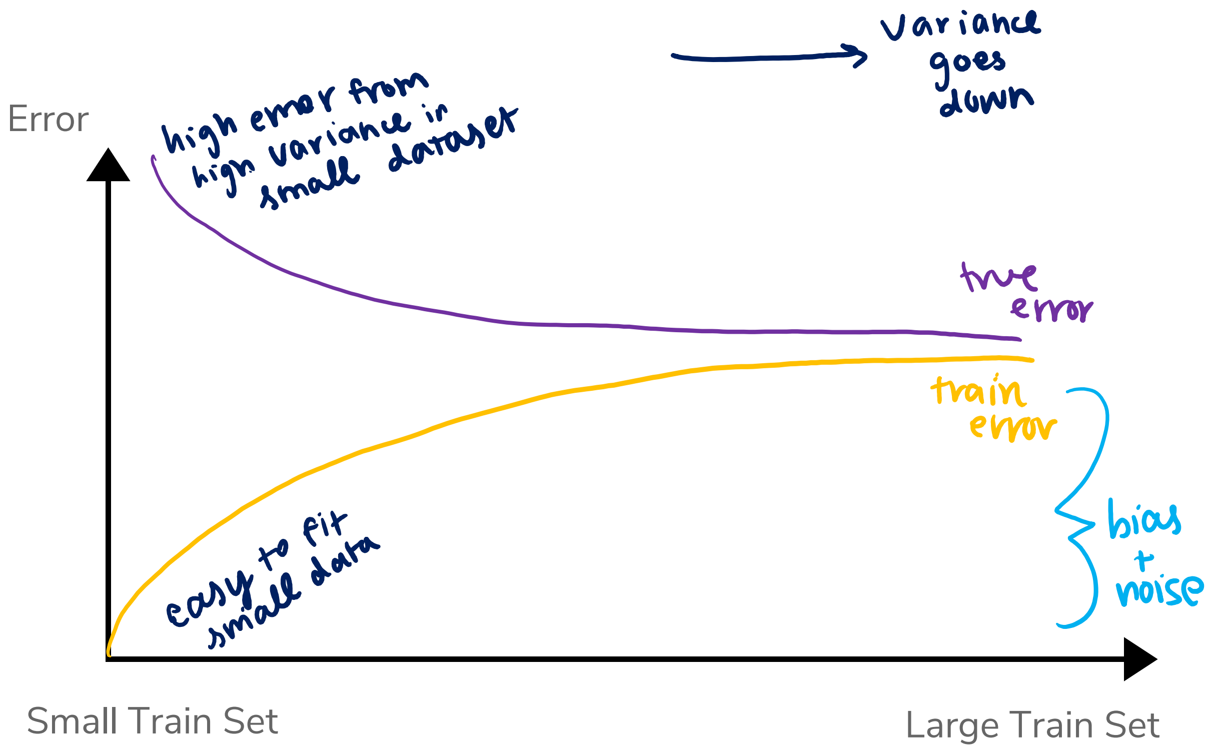
Visually, this looks like the following!

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

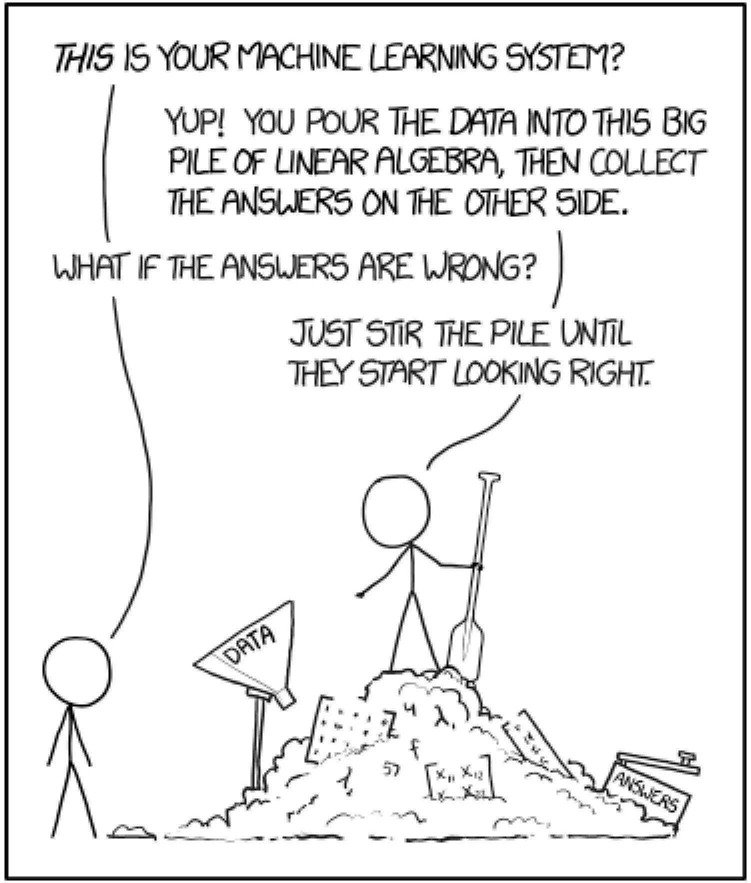


Dataset Size

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



Brain Break


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Credit: https://xkcd.com/1838/

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/cse416

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 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.

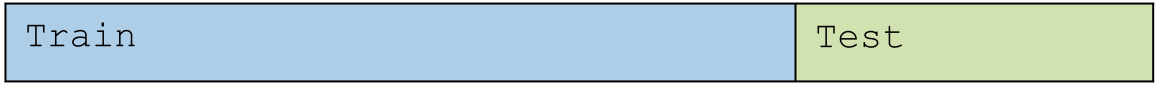
Choosing Complexity

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

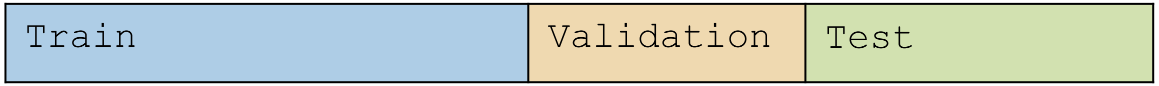
- Using a validation set
- Doing cross validation

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

Pros

Easy to describe and implement

Pretty fast

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

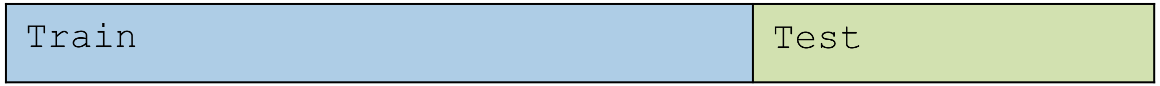
Cons

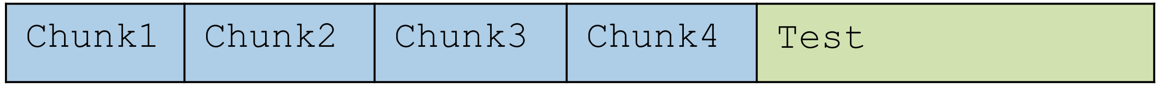
Have to sacrifice even more training data! ☹

Cross-Validation

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

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 k models!

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

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