

Bias/Variance and Cross-Validation

Moses Marsh



- Review: Linear Regression
- Overfitting and Underfitting
- The Bias/Variance Tradeoff
- Cross-Validation
- K-fold Cross-Validation
- Subset Selection of Predictors

One goal is to make accurate **predictions** on future (unseen) data.

1. Define a business goal.

e.g. make Tesla cars the most dependable vehicles on the market

2. Collect training data.

e.g. Tesla cars' event logs + historical record of parts replaced

3. Train a model.

e.g. **features:** event statistics, **target:** time until failure

4. Deploy the model.

e.g. monitor cars' events in real time, send mechanics to replace parts that will soon fail

Questions!

Review: Linear Regression

We assume the world is built on linear relationships. Under that assumption, we can model the relationship between *features* and a *target* like this:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon$$

The diagram illustrates the components of the linear regression equation $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon$. Arrows of different colors point from descriptive text to specific terms in the equation:

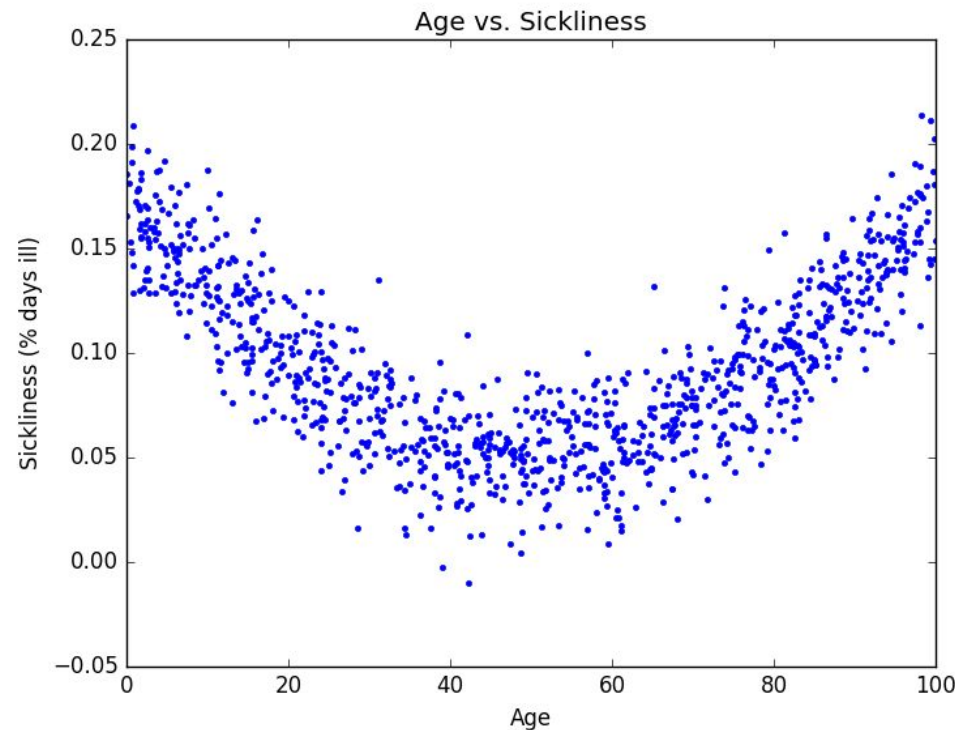
- A yellow arrow points from "target (aka, outcome)" to Y .
- Four blue arrows point from "model parameters (aka, coefficients)" to $\beta_0, \beta_1, \beta_2,$ and β_p .
- Three green arrows point from "features (aka, predictors)" to $X_1, X_2,$ and X_p .
- A red arrow points from "sampling error" to ϵ .

We can make linear regression non-linear by inserting extra “interaction” features or higher-order features.

Example:

$$Y = \beta_0 + \beta_1 * \text{age}$$

$$Y = \beta_0 + \beta_1 * \text{age} + \beta_2 * \text{age}^2$$



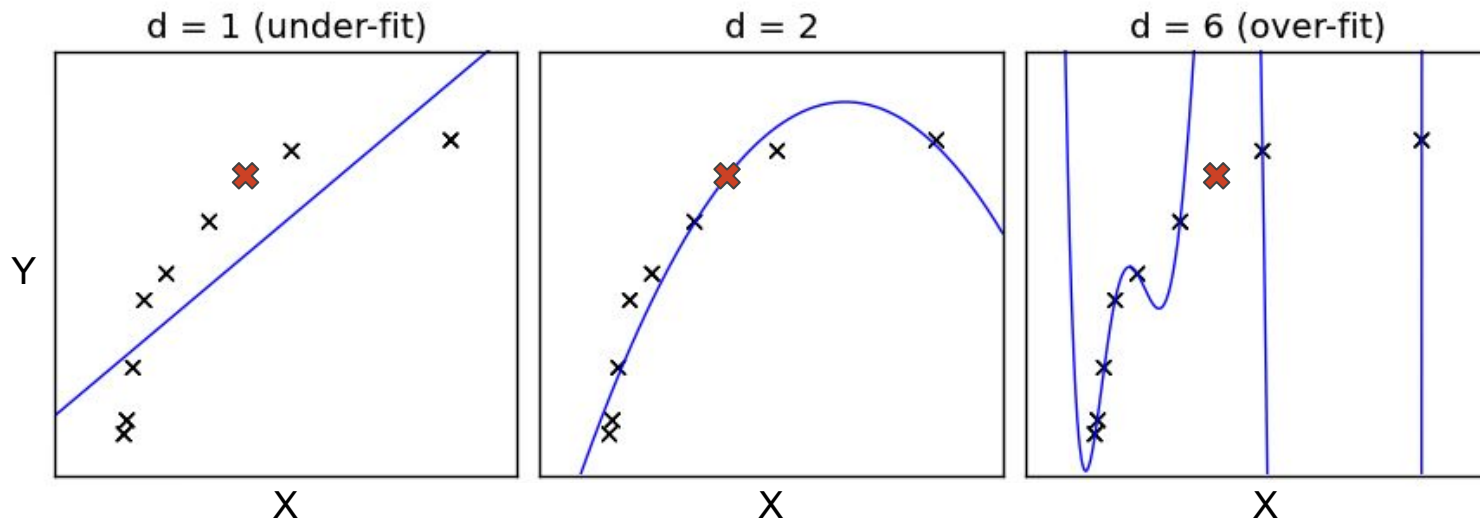
We *could* just keep inserting interaction features until $R^2 = 1$.

Boom. I solved data science. Here's my idea:

```
def train_super_awesome_perfect_model (X, y):  
    while True:  
        model = LinearRegression()  
        model.fit(X, y)  
        if calculate_r2(model, X, y) >= 0.999:  
            return model  
        else:  
            X = insert_random_interaction_feature(X)
```

Why is this a
bad idea?

Oh the woes of overfitting...



What's bad about the first model?

What's bad about the second model?

What's bad about the third model?

Underfitting and Overfitting

Underfitting: The model doesn't fully capture the relationship between predictors and the target. The model has *not* learned the data's signal.

→ What should we do if our model underfits the data? (assume using lin. reg.)

Overfitting: The model has tried to capture the sampling error. The model has learned the data's signal *and* the noise.

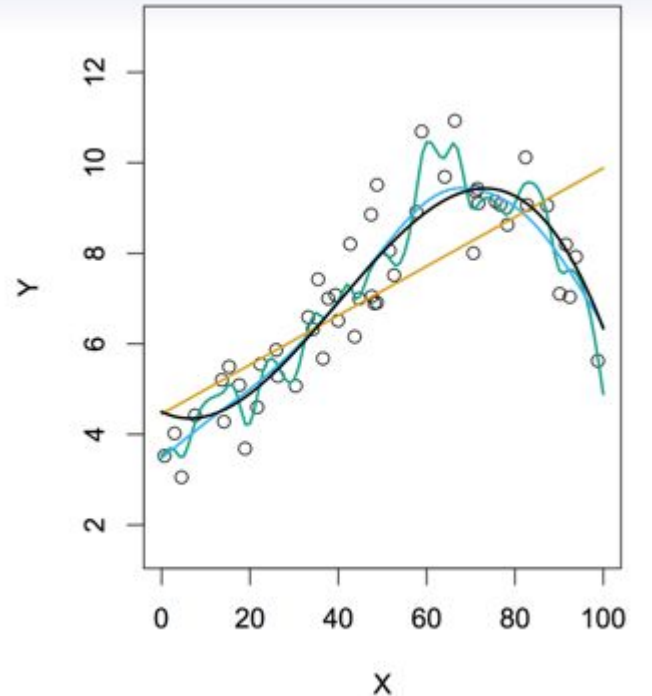
→ What should we do if our model overfits the data? (harder... any guesses?)

The Bias/Variance Tradeoff

Boardwork... build intuition...

Let's get an intuitive feel for the bias and the variance of a model... we'll see more math on the next slide.

Note: **Bias** and **Variance** are terms you will use A TON as a data scientist! Exciting times!



We assume the true predictor/target relationship is given by an unknown function plus some sampling error:

$$Y = f(X) + \epsilon$$

We estimate the true (unknown) function by fitting a model over the training set.

$$\hat{Y} = \hat{f}(X)$$

Let's evaluate this model using a test observation $(\mathbf{x}_o, \mathbf{y}_o)$ drawn from the population. What is the model's expected squared prediction error on this test observation?

$$E[(y_o - \hat{f}(x_o))^2] = \dots$$

Our model's expected squared prediction error will depend on (1) the variability of \mathbf{y}_0 and (2) the variability of the training set used to train our model. We can break this into three pieces:

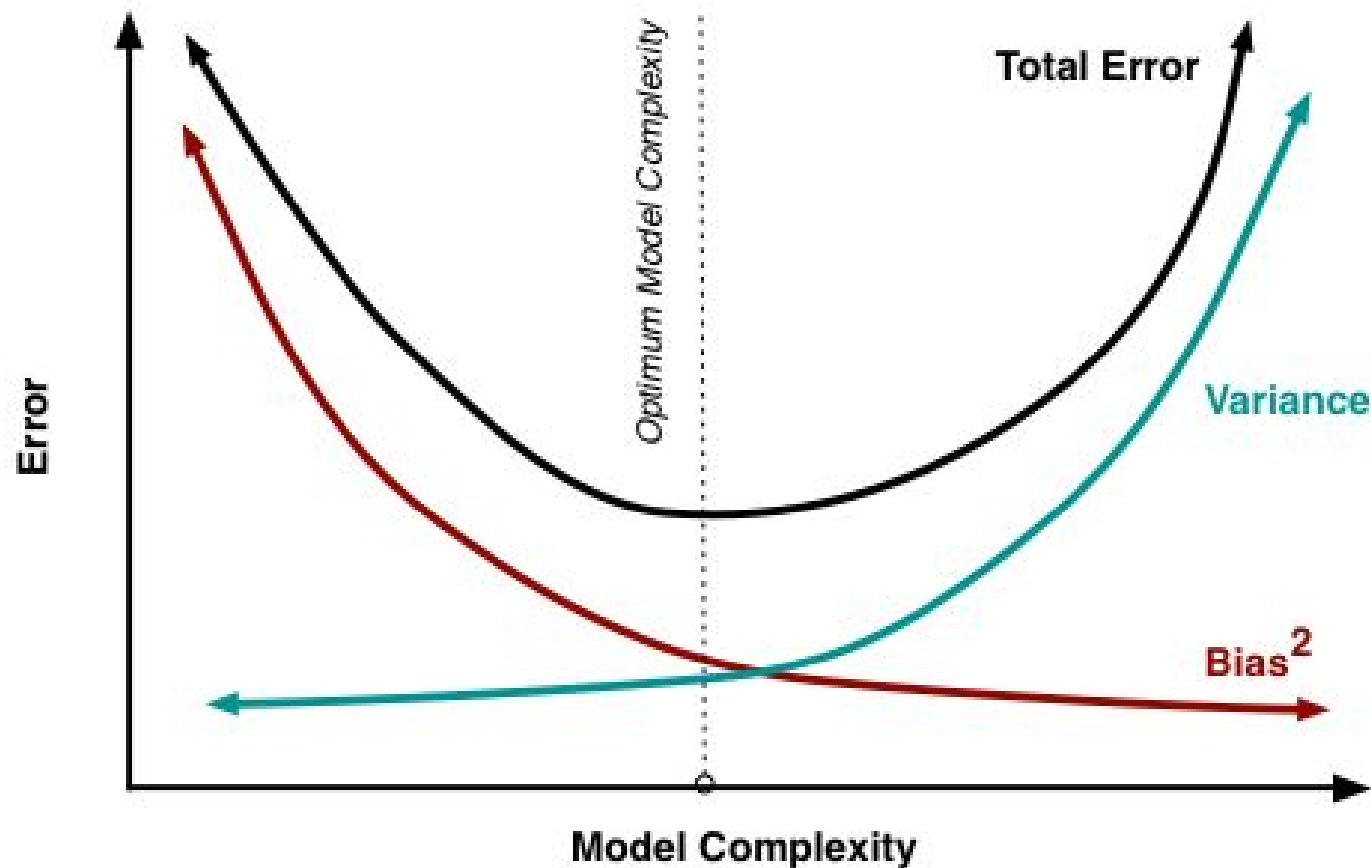
$$E[(y_o - \hat{f}(x_0))^2] = \dots = \text{Var}(\hat{f}(x_0)) + \text{Bias}^2(\hat{f}(x_0)) + \text{Var}(\epsilon)$$

The variance of our model's prediction of \mathbf{x}_0 over all possible training sets

The difference between the true target and our model's average prediction over all possible training sets

The variance of the irreducible error.

$$\text{Bias}(\hat{f}(x_0)) = E[\hat{f}(x_0)] - f(x_0)$$



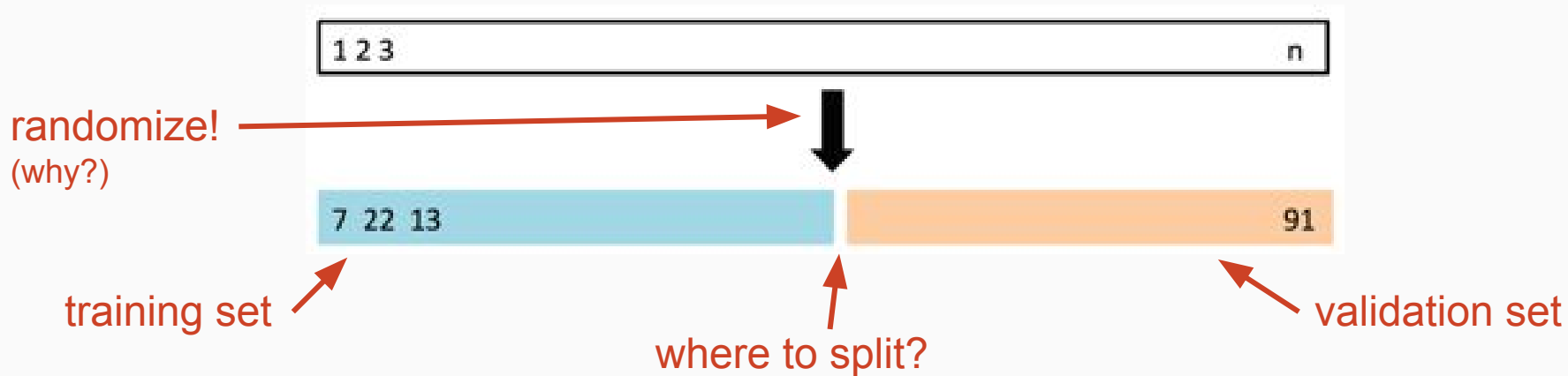
How is the **bias/variance tradeoff** related to **underfitting** and **overfitting**?

How can we find the best tradeoff point? I.e. The optimum model complexity

Cross-Validation

Main idea: **Don't use all your data for training.**

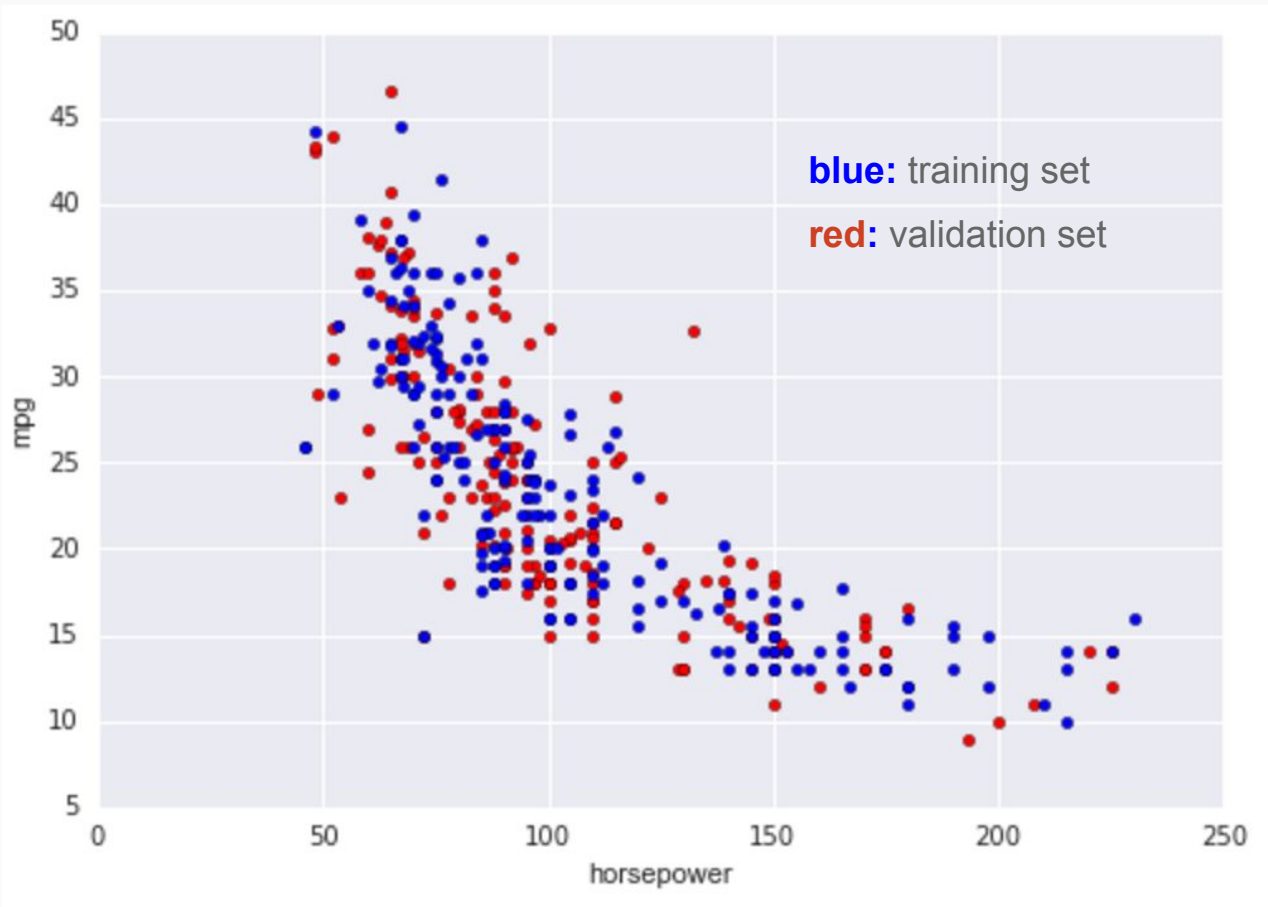
Instead: **Split your data into a “training set” and a “validation set”.**



Cross-Validation

1. Split your data into training/validation sets.
70/30 or 90/10 splits are commonly used
2. Use the training set to train several models of varying complexity.
e.g. linear regression (w/ and w/out interaction features), neural nets, decision trees, etc.
(we'll talk about hyperparameter tuning, grid search, and feature engineering later)
3. Evaluate each model using the validation set.
calculate R^2 , MSE, accuracy, or whatever you think is best
4. Keep the model that performs best over the **validation** set.

Let's predict MPG from horsepower



Cross-Validation Example



You will see this shape all the time!

You will wrestle with the bias/variance tradeoff constantly...

E.g. linear regression w/ varying degree of polynomial

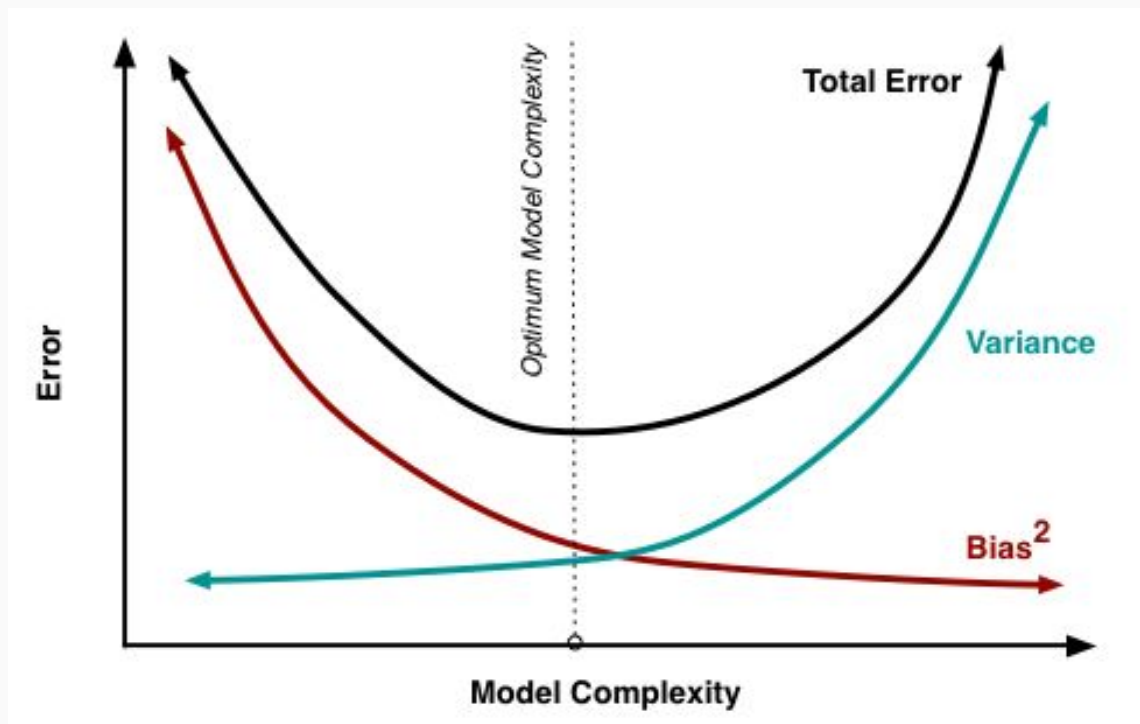
Recall our goal: Making accurate future predictions

Fitting the training set perfectly is *easy*.

How?

Fitting future (unseen) data is *not easy*.

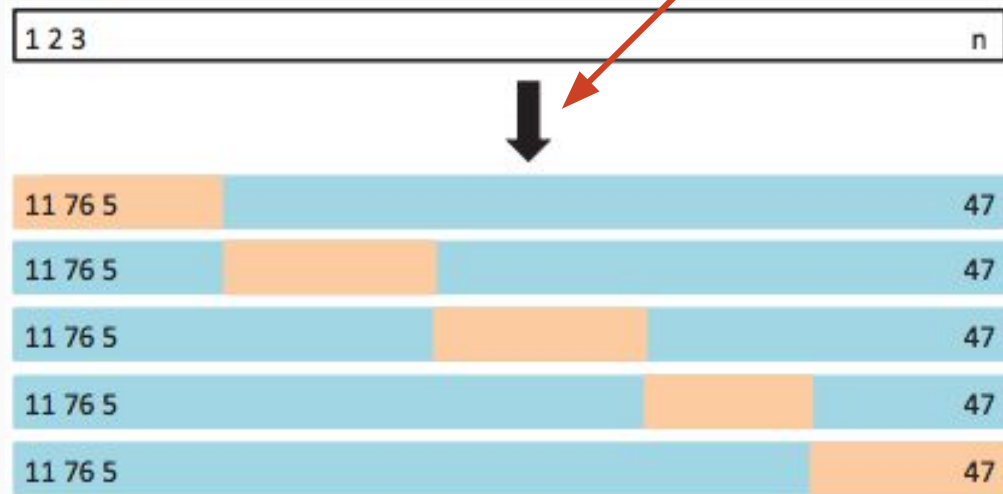
Cross validation helps us choose a model that performs well on unseen data.



1. Split the dataset into k “folds”.
2. Train using $(k-1)$ folds. Validate using the one “leave out” fold. Record a validation metric such as RSS or accuracy.
3. Train k models, leaving out a different fold for each one.
4. Average the validation results.

Commonly, $k=5$ or $k=10$

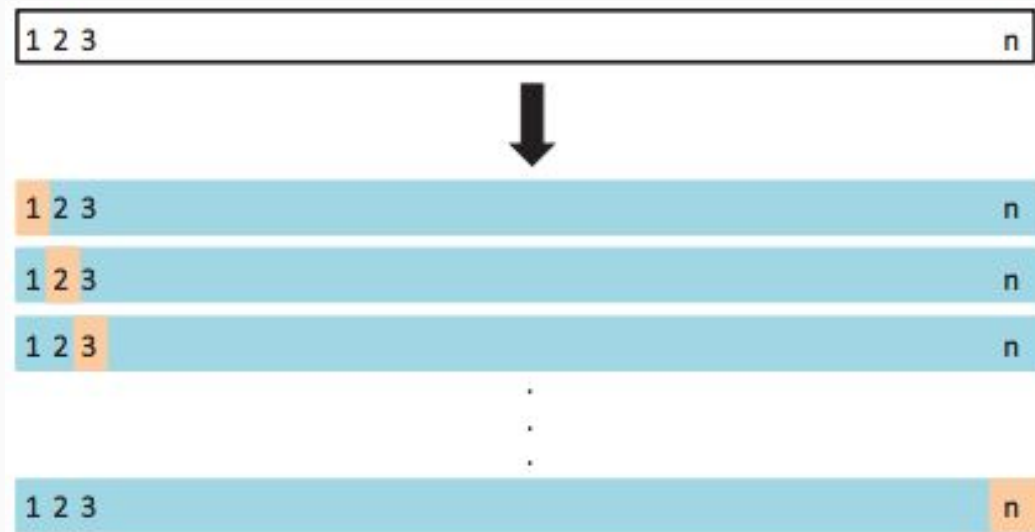
randomize!



Assume we have n training examples.

A special case of k -fold CV is when $k=n$. This is called *leave-one-out cross-validation*.

Useful (only) if you have a tiny dataset where you can't afford a large validation set.

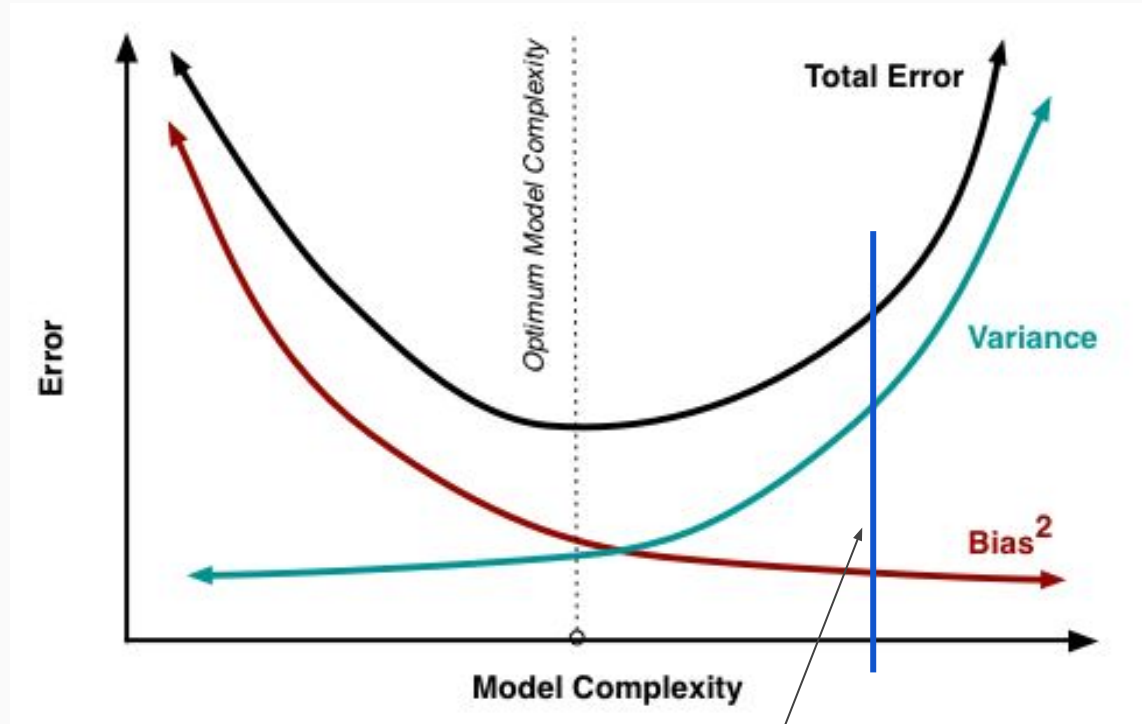


Overfitting in high dimensions is easy, even with simple models.

If our data has high dimensionality (many many predictors), then it becomes easy to overfit the data.

This is one result of the so-called **Curse of Dimensionality** (look it up).

Even linear regression might be too complex of a model for high dimensional data (and the smaller the dataset, the worse this problem is).



Linear regression in high dimensions

You have a few options.

1. Get more data... (not usually possible/practical)
2. **Subset Selection:** keep only a subset of your predictors (i.e, dimensions)
3. **Regularization:** restrict your model's parameter space
4. **Dimensionality Reduction:** project the data into a lower dimensional space

Subset Selection

Best subset: Try every model. Every possible combination of p predictors

- Computationally intensive. 2^p possible subsets of p predictors
- High chance of finding a “good” model by random chance.
... A sort-of monkeys-Shakespeare situation ...

Stepwise: Iteratively pick predictors to be in/out of the final model.

- Forward, backward, forward-backward strategies

$M_0 \rightarrow M_1 \rightarrow M_2 \rightarrow \dots \rightarrow M_p$

+1 predictor with
smallest RSS \Leftrightarrow
largest R^2

+1 predictor with
smallest RSS \Leftrightarrow
largest R^2

Now we have p candidate models

Are RSS and R^2 good ways to decide amongst the resulting $(p+1)$ candidates?

Answer: Don't use RSS or R^2 for this part.
Use Mallows's C_p , or AIC, or BIC, or Adjusted R^2 .

... or just use cross-validation with any error measurement.

$$C_p = \frac{1}{n}(RSS + \underline{2p}\hat{\sigma}^2)$$

Mallow's C_p

p is the total # of parameters

$\hat{\sigma}^2$ is an estimate of the variance of the error, ε

$$AIC = -2\log L + 2 \cdot \underline{p}$$

L is the maximized value of the likelihood function for the model estimated

$$BIC = \frac{1}{n}(RSS + \log(n)\underline{p}\hat{\sigma}^2)$$

This is C_p , except 2 is replaced by $\log(n)$.
 $\log(n) > 2$ for $n > 7$, so BIC generally exacts a heavier penalty for more variables

$$\text{Adjusted } R^2 = 1 - \frac{RSS/(n - \underline{p} - 1)}{TSS/(n - 1)}$$

Similar to R^2 , but pays price for more variables

Side Note: Can show AIC and Mallow's C_p are equivalent for linear case

OLS Regression Results

```

=====
Dep. Variable:          y      R-squared:          0.933
Model:                  OLS    Adj. R-squared:       0.928
Method:                 Least Squares    F-statistic:         211.8
Date:                   Mon, 03 Nov 2014    Prob (F-statistic):   6.30e-27
Time:                   14:45:06    Log-Likelihood:      -34.438
No. Observations:       50    AIC:                 76.88
Df Residuals:           46    BIC:                 84.52
Df Model:                3
Covariance Type:        nonrobust
=====

```

| | coef | std err | t | P> t | [95.0% Conf. Int.] |
|-------|---------|---------|--------|-------|--------------------|
| x1 | 0.4687 | 0.026 | 17.751 | 0.000 | 0.416 0.522 |
| x2 | 0.4836 | 0.104 | 4.659 | 0.000 | 0.275 0.693 |
| x3 | -0.0174 | 0.002 | -7.507 | 0.000 | -0.022 -0.013 |
| const | 5.2058 | 0.171 | 30.405 | 0.000 | 4.861 5.550 |

```

=====
Omnibus:                0.655    Durbin-Watson:         2.896
Prob(Omnibus):          0.721    Jarque-Bera (JB):      0.360
Skew:                   0.207    Prob(JB):              0.835
Kurtosis:               3.026    Cond. No.               221.
=====

```


Regularized Linear Regression

Moses Marsh



- Shortcomings of Ordinary Linear Regression
- Ridge Regression
- Lasso Regression
- When to use each!

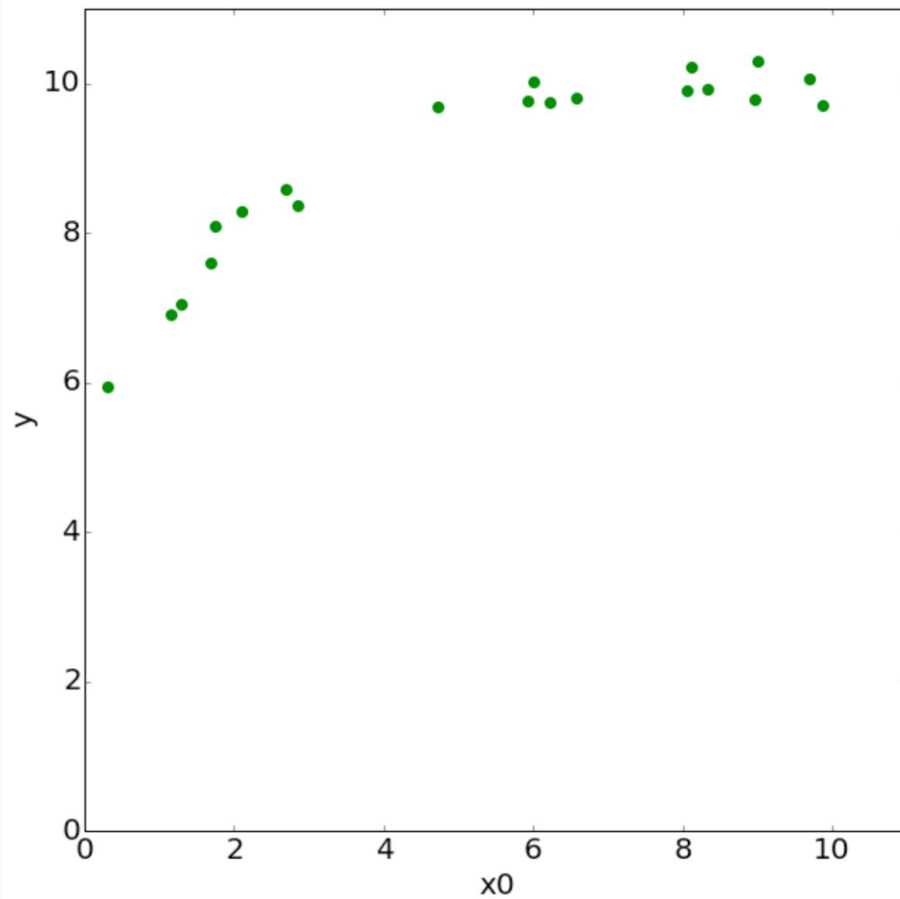
Linear Regression Example

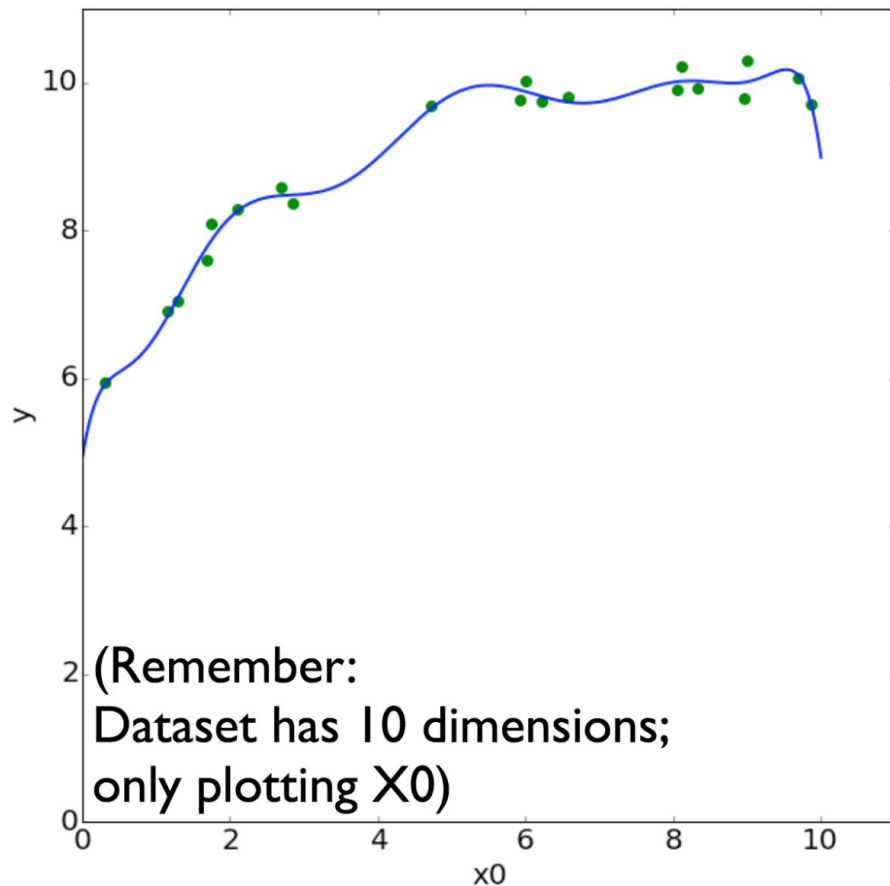
Data: 20 examples x 10 features

Predict: y

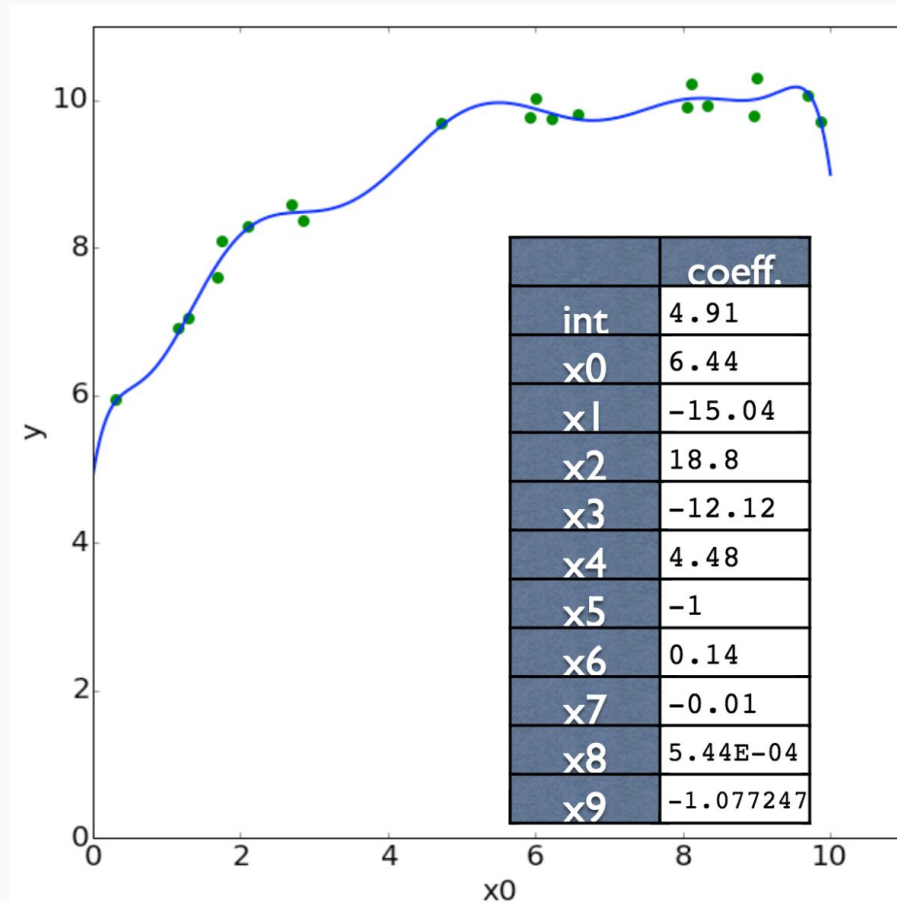
| y | x_0 | x_1 | x_2 | x_3 | ... |
|------|-------|-------|--------|--------|-----|
| 9.92 | 8.33 | 69.39 | 578.00 | 4815.4 | ... |
| 8.58 | 2.69 | 7.26 | 19.54 | 52.64 | ... |
| 8.07 | 1.75 | 3.06 | 5.35 | 9.36 | ... |
| 8.29 | 2.11 | 4.46 | 9.41 | 19.86 | ... |
| ... | ... | ... | ... | ... | ... |

Linear Regression Example (x0 vs y)

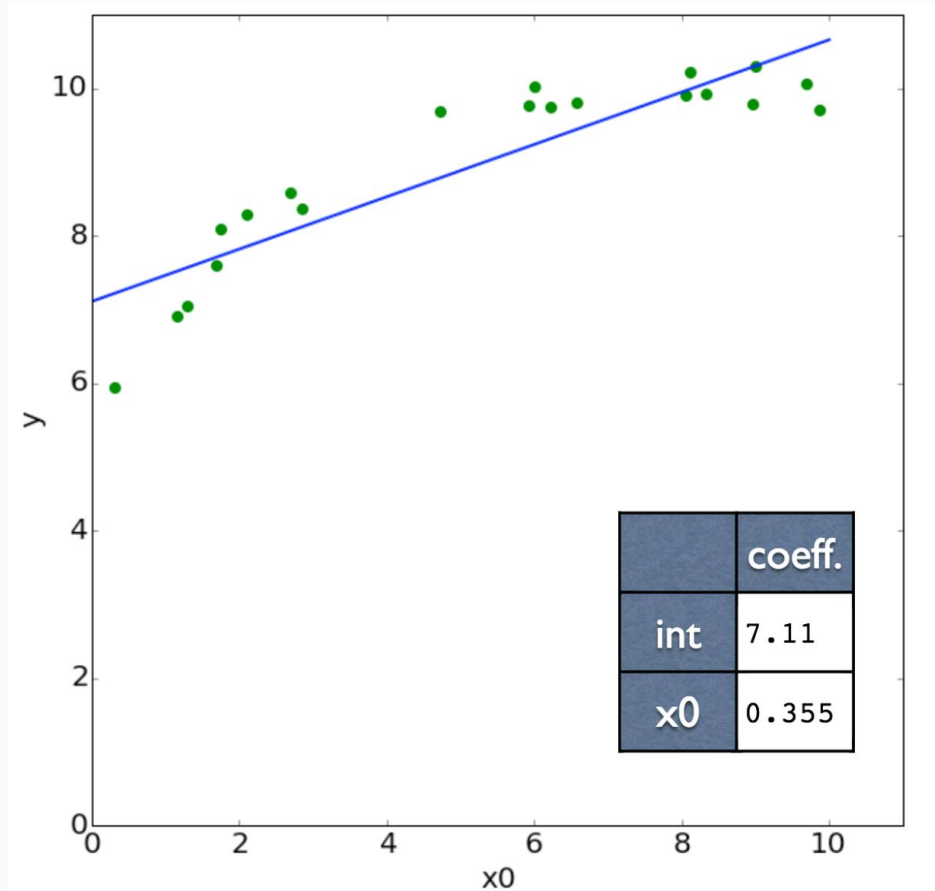




Linear Regression Example (x0 vs y, model over all features)



Linear Regression Example (x0 vs y, model over only x0 features)



In high dimensions, data is (usually) sparse

Linear regression can have high variance (i.e. tends to overfit) on high dimensional data...

We'd like to restrict ("normalize", or "regularize") the model so that it has less variance.

Take the 20 example x 10 feature dataset as an example... when we fit over all features, the complexity of the model grew dramatically.

(and keep in mind, some datasets have thousands of features)

Linear Regression (another review)

We model the world as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon$$

We estimate the model parameters by minimizing:

$$\sum_{i=1}^N (y_i - \hat{\beta}_0 - \sum_{j=1}^p x_{ij} \hat{\beta}_j)^2$$

Ridge Regression

(Linear Regression w/ Tikhonov (L2) Regularization)

We model the world as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon$$

← (same as before)

We estimate the model parameters by minimizing:

$$\sum_{i=1}^N (y_i - \hat{\beta}_0 - \sum_{j=1}^p x_{ij} \hat{\beta}_j)^2 + \underbrace{\lambda \sum_{i=1}^p \hat{\beta}_i^2}_{\text{(new term!)}}$$

← (the “regularization” parameter)

← (new term!)

Ridge Regression

$$\sum_{i=1}^N (y_i - \hat{\beta}_0 - \sum_{j=1}^p x_{ij} \hat{\beta}_j)^2 + \lambda \sum_{i=1}^p \hat{\beta}_i^2$$

What if we set the lambda equal to zero?

What does the new term accomplish?

What happens to a features whose corresponding coefficient value (beta) is zero?

Ridge Regression

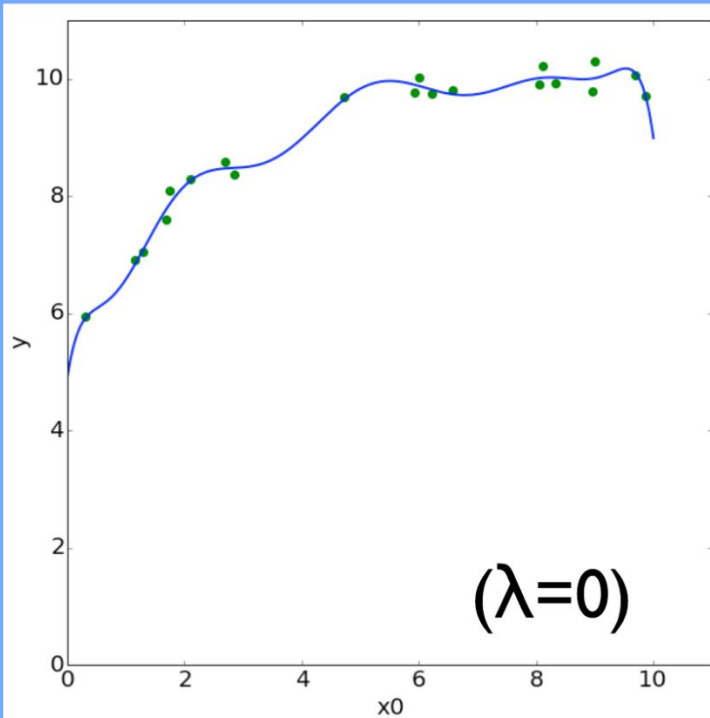
$$\sum_{i=1}^N (y_i - \hat{\beta}_0 - \sum_{j=1}^p x_{ij} \hat{\beta}_j)^2 + \lambda \sum_{i=1}^p \hat{\beta}_i^2$$

Notice, we do not penalize β_0 .

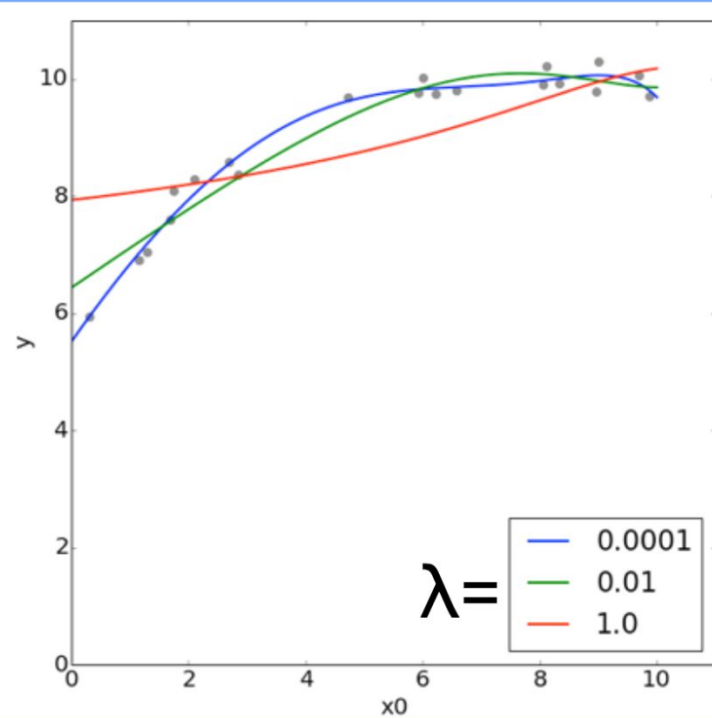
Changing lambda changes the amount that large coefficients are penalized.

Increasing lambda increases the model's bias and decreases its variance. ← this is cool!

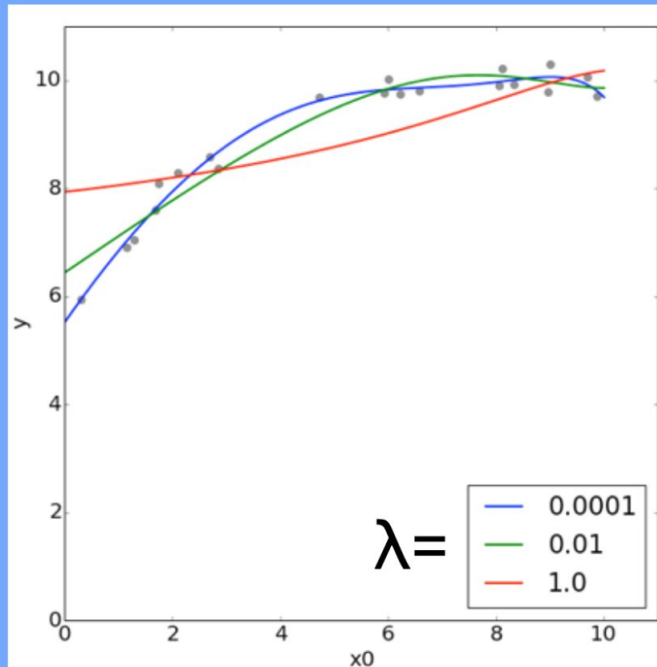
Linear Regression



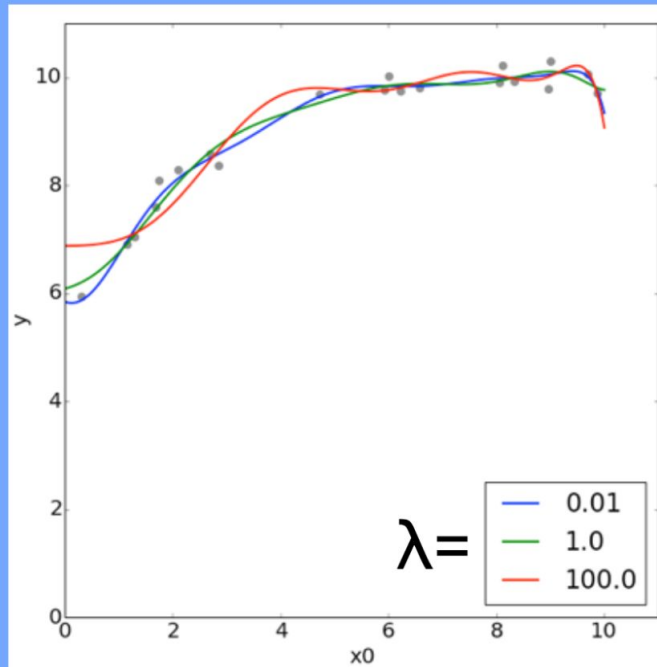
Ridge Regression



Normalized Data



Non-Normalized Data



Single value for λ assumes features are on the same scale!!

LASSO Regression

(Linear Regression w/ LASSO (L1) Regularization)

We model the world as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon$$

← (same as before)

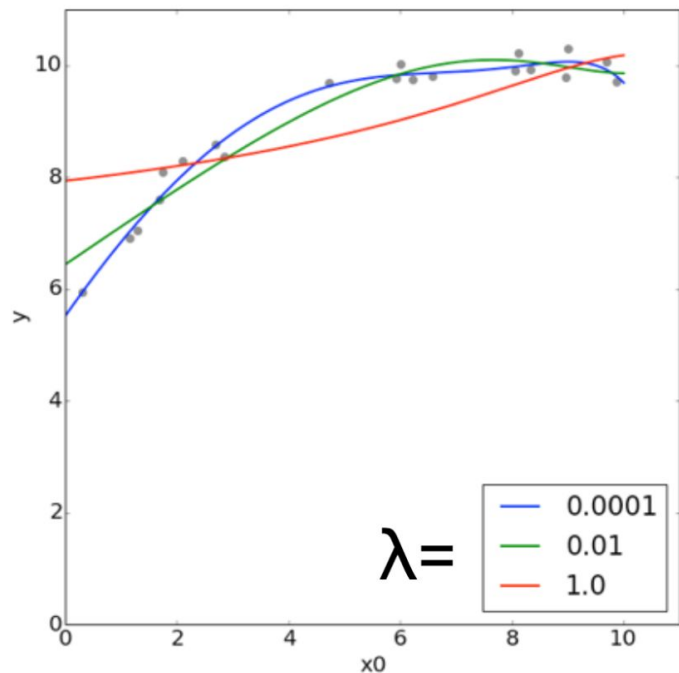
We estimate the model parameters to minimizing:

$$\sum_{i=1}^N (y_i - \hat{\beta}_0 - \sum_{j=1}^p x_{ij} \hat{\beta}_j)^2 + \lambda \sum_{i=1}^p |\hat{\beta}_i|$$

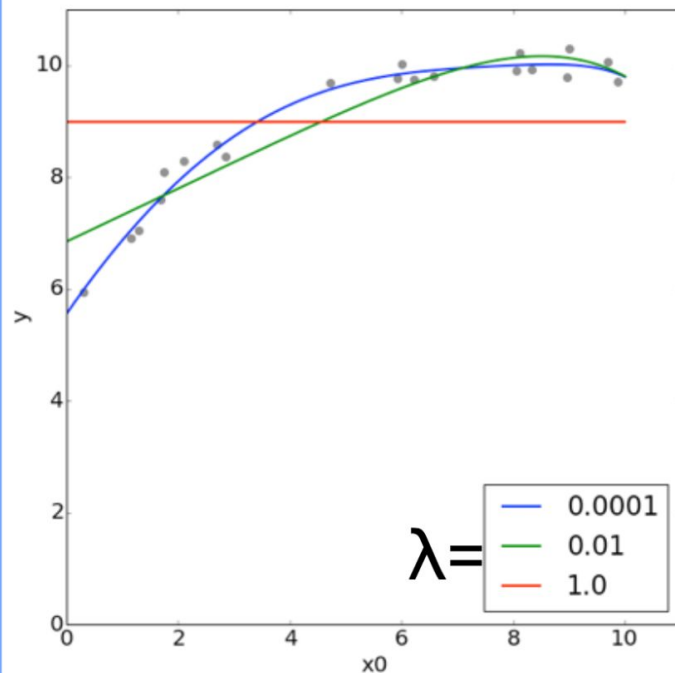
(the “regularization” parameter)

(absolute value instead of squared)

Ridge Regression



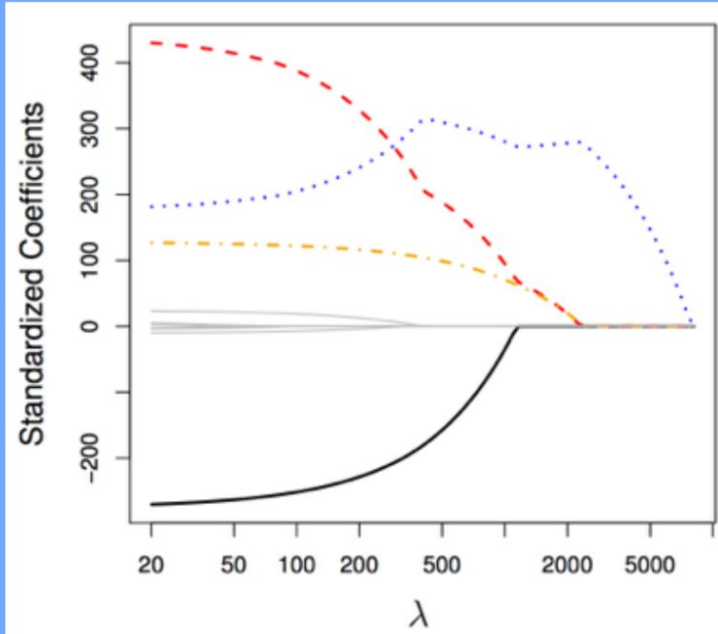
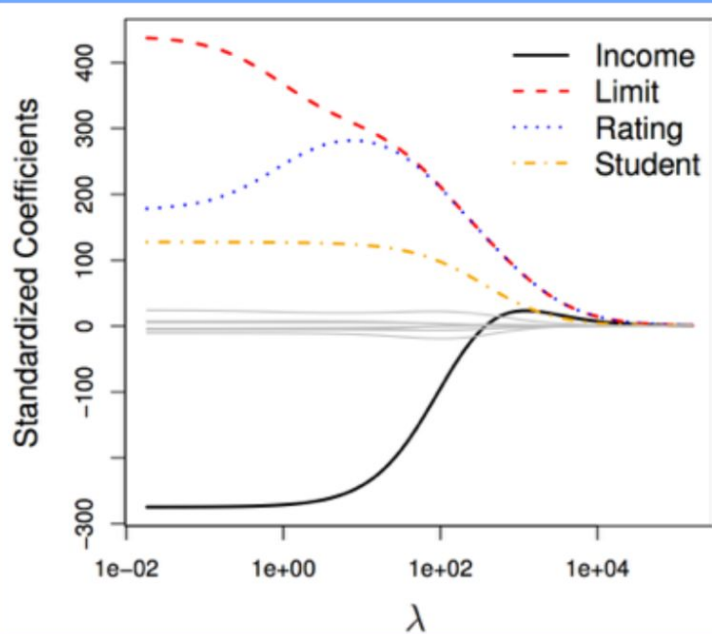
Lasso Regression



Ridge

vs.

Lasso

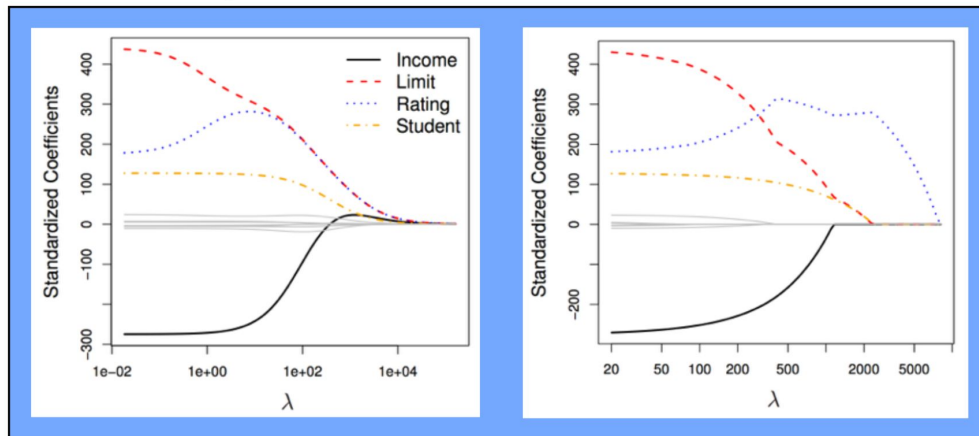


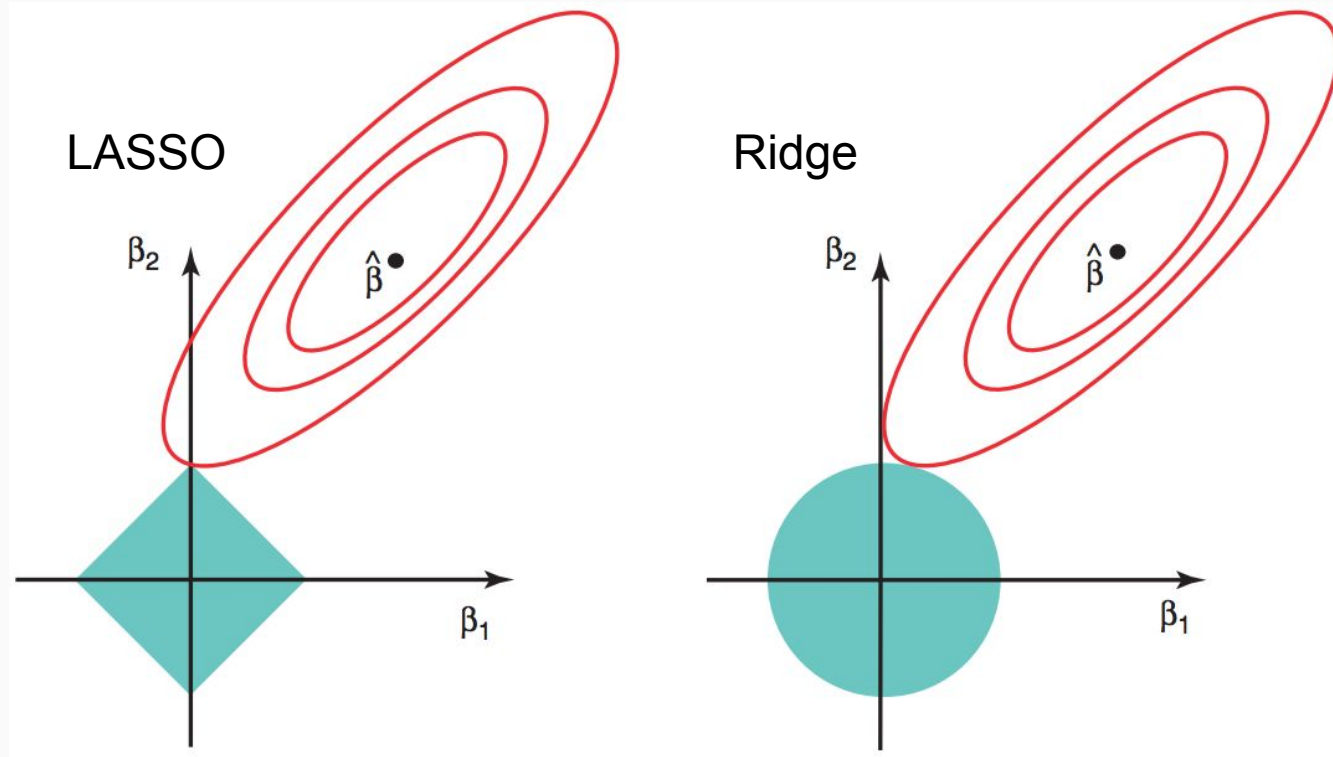
- Ridge forces parameters to be small + Ridge is computationally easier because it is differentiable
- Lasso tends to set coefficients exactly equal to zero
 - This is useful as a sort-of “automatic feature selection” mechanism,
 - leads to “sparse” models, and
 - serves a similar purpose to stepwise features selection

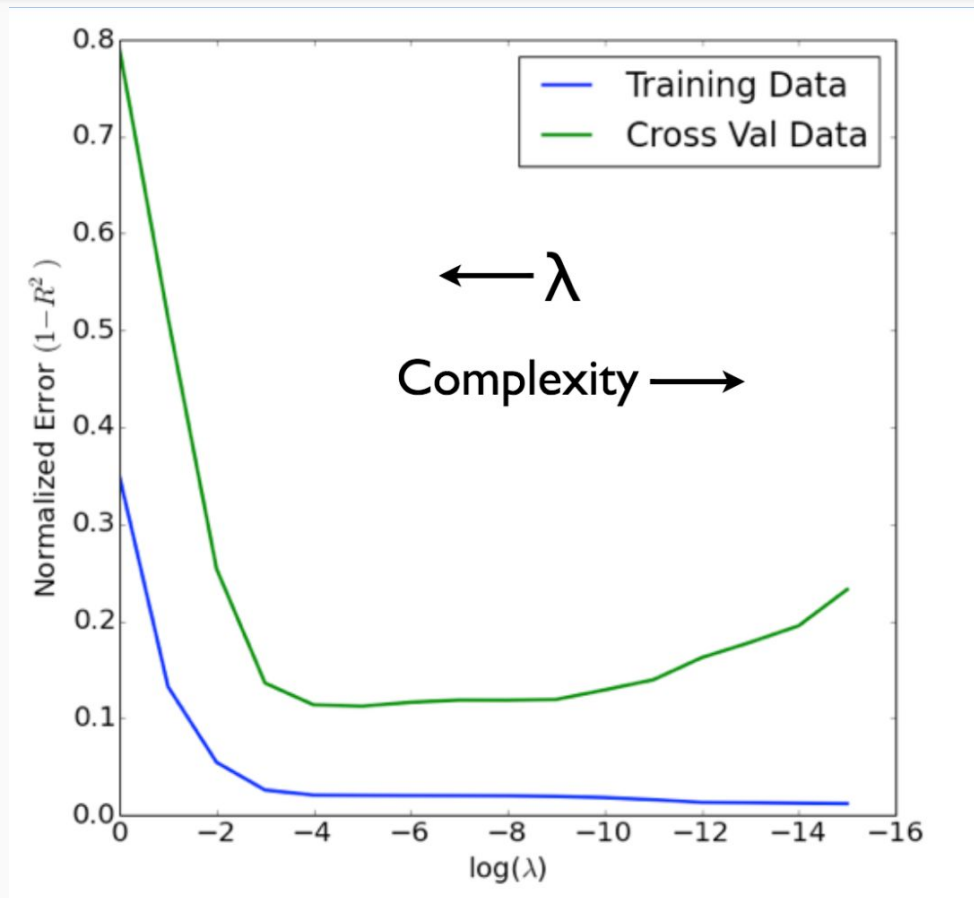
Which is better depends on your dataset!

True sparse models will benefit from lasso; true dense models will benefit from ridge.

Ridge vs. Lasso







scikit-learn

Classes:

- `sklearn.linear_model.LinearRegression(...)`
- `sklearn.linear_model.Ridge(alpha=my_alpha, ...)`
- `sklearn.linear_model.Lasso(alpha=my_alpha, ...)`

All have these methods:

- `fit(X, y)`
- `predict(X)`
- `score(X, y)`