Curve-fitting, Validation

6.100 LECTURE 9

SPRING 2026

Announcements

- Pset 2 checkoff due Wed 3/4
- Pset 3 is out, due next Mon 3/9
- Add Date is Fri 3/6
  - talk with your advisor if you need to make changes to registration

Exam 1 scores

- Rough ranges for letter grades in announcement
- Treat Exam 1 more like a diagnostic
- Final grade is largely influenced by performance over all three exams
  - later exams will focus more on later material
  - but still assume knowledge from earlier

- Other factors, not primary but also non-negligible
  - psets, checkoffs, exercises
  - participation in class
  - interaction in office hours

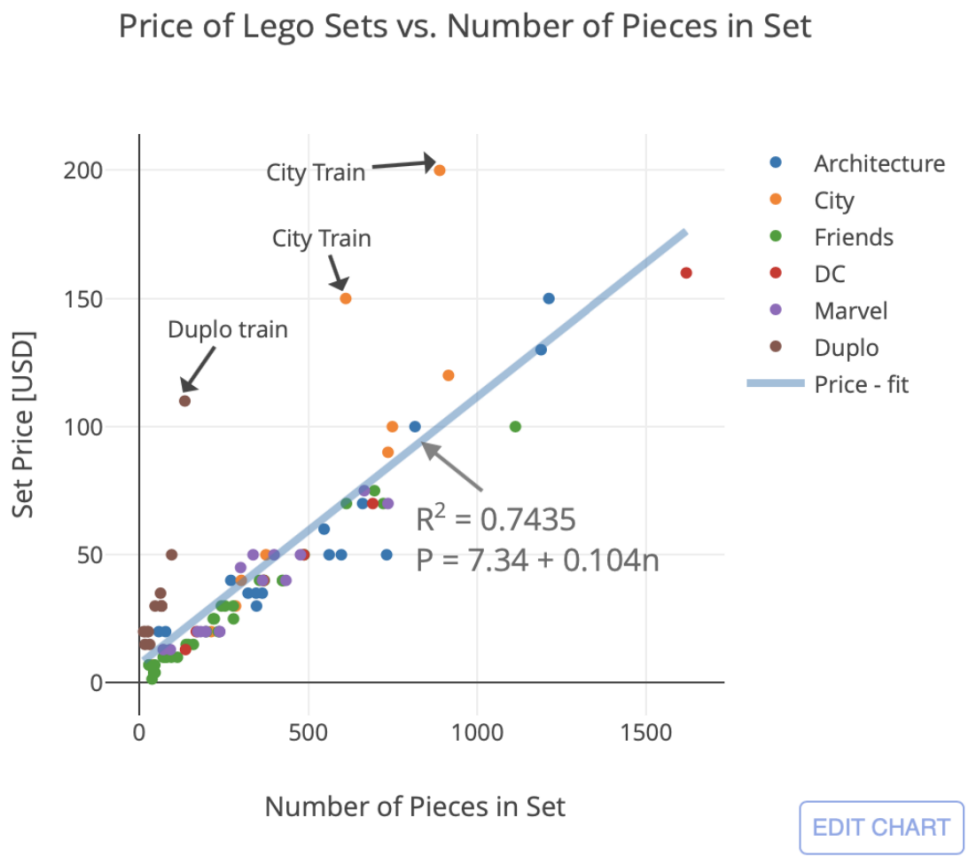
Pset submission tips

- Treat psets as learning opportunity
  - don’t stress about points
- Psets are designed to take about 1.5 weeks each
  - reduced frequency of having to turn something in
  - but also need to start early
- Make submissons along the way
  - we count only last submission before your deadline
  - submitting helps you verify partial credit
  - also shows us you’re making progress

Optimization problems

Curve-fitting

- Distributions alone lack the power to explain relationships between random variables



https://www.wired.com/2014/08/lego-cost

Linear regression

- Given a set of (x, y) data points
  - $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Want to model a predictive relationship between them
  - i.e., predict $\boldsymbol{y}$ value associated with any $\boldsymbol{x}$
- Simplest relationship is $\boldsymbol{y} = \boldsymbol{a} \cdot \boldsymbol{x} + \boldsymbol{b}$
  - specifies a linear model
- The task is to find parameters $\boldsymbol{a}$ and $\boldsymbol{b}$ that "best fit" the data
  - $\boldsymbol{a}$ indicates upward or downward trend and how much
- Can we frame this as an optimization problem?

Linear regression as optimization

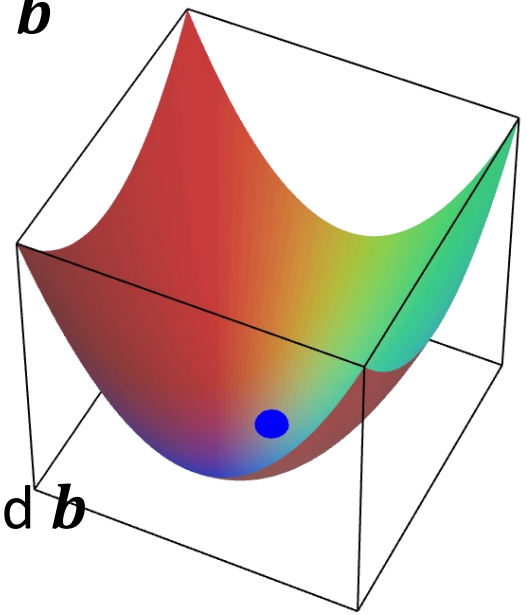
- Consider the **sum of all squared errors (SSE)**

$$\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \sum_{i=1}^{n}(a \cdot x_i + b - y_i)^2$$

- The average of this is the **mean squared error (MSE)**
- This is a quadratic expression in terms of a and b
  - the x_i and y_i values are known, so they are constants
  - the number of data points n is also a constant
  - a and b are **continuous variables** in our objective function
- There exist efficient (even closed-form) solutions for the optimal a and b that minimize the SSE (or MSE)
  - available in **numpy,** will show next lecture
  - Pset 3 begins by asking you to implement exhaustive enumeration and bisection approaches

Solving linear regression exactly

- **Goal:** minimize the SSE
- SSE is quadratic expression in terms of $\boldsymbol{a}$ and $\boldsymbol{b}$
  - every slice at an $\boldsymbol{a}$ value is a 1-d quadratic polynomial in terms of $\boldsymbol{b}$
  - and vice versa
- At the true minimum, both slices are minimized
  - take partial derivatives with respect to $\boldsymbol{a}$ and $\boldsymbol{b}$
  - set them both to 0
  - now we have two linear equalities in terms of $\boldsymbol{a}$ and $\boldsymbol{b}$
  - unique solution guaranteed (under most conditions)



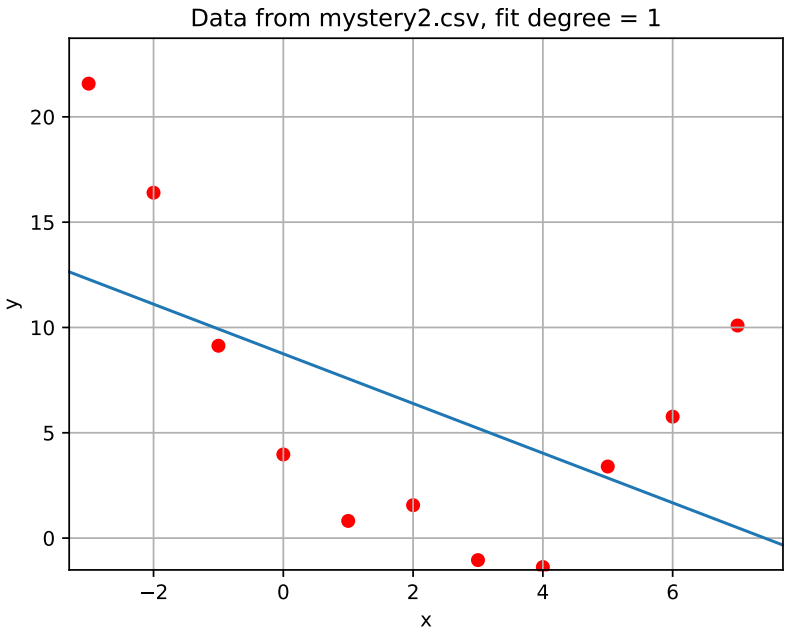
Fitting higher-order polynomials

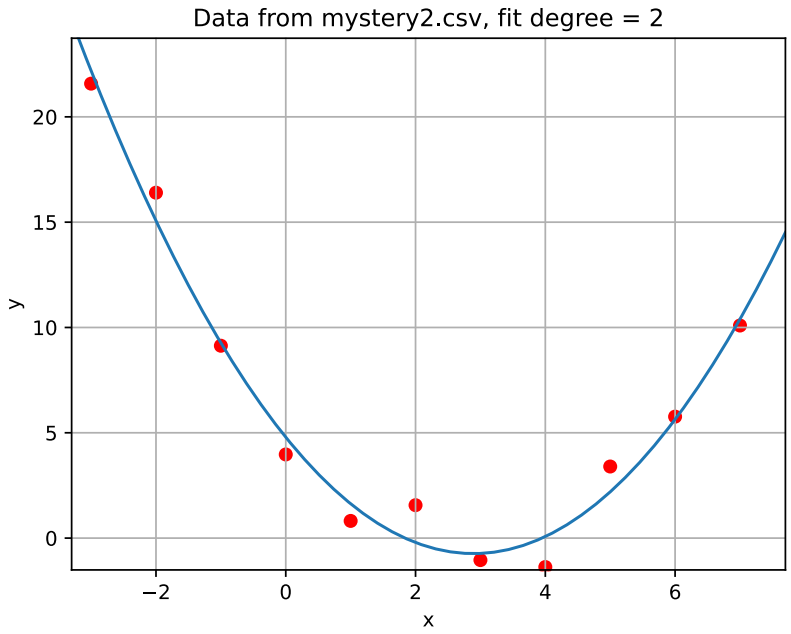
- Can generalize linear regression to fit higher-order polynomials
  - e.g., quadratic model: $\boldsymbol{y = ax^2 + bx + c}$
  - SSE is still quadratic in terms of three variables $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$
  - take three partial derivatives, solve for three variables
    - could also generalize exhaustive search or bisection to 3-D space
- Some sources call this **polynomial regression**
- Others still call it **linear regression** because the model is linear ***in terms of the coefficients***
- **`numpy.polyfit(x_vals, y_vals)`**
  → **`list`** of coefficients
  - e.g., outputs a quadratic polynomial model as **`[a, b, c]`**

Evaluating goodness of fit

Evaluating a fit

- **Two questions**
  - How well does a line/curve fit the data?
  - What degree polynomial would be an appropriate model?





Evaluating a fit

- **Two questions**
  - How well does a line/curve fit the data?
  - What degree polynomial would be an appropriate model?
- **Idea 1:** use SSE
  - after all, curve was supposed to minimize SSE
  - **issue:** SSE increases with number of points
  - if points come from the same underlying experiment/process, getting more data shouldn't appreciably affect the model

Evaluating a fit

- **Two questions**
  - How well does a line/curve fit the data?
  - What degree polynomial would be an appropriate model?
- **Idea 2:** use MSE
  - normalize against number of points, i.e., average squared error per point
  - **issue:** if data scales by a factor, MSE scales by factor squared

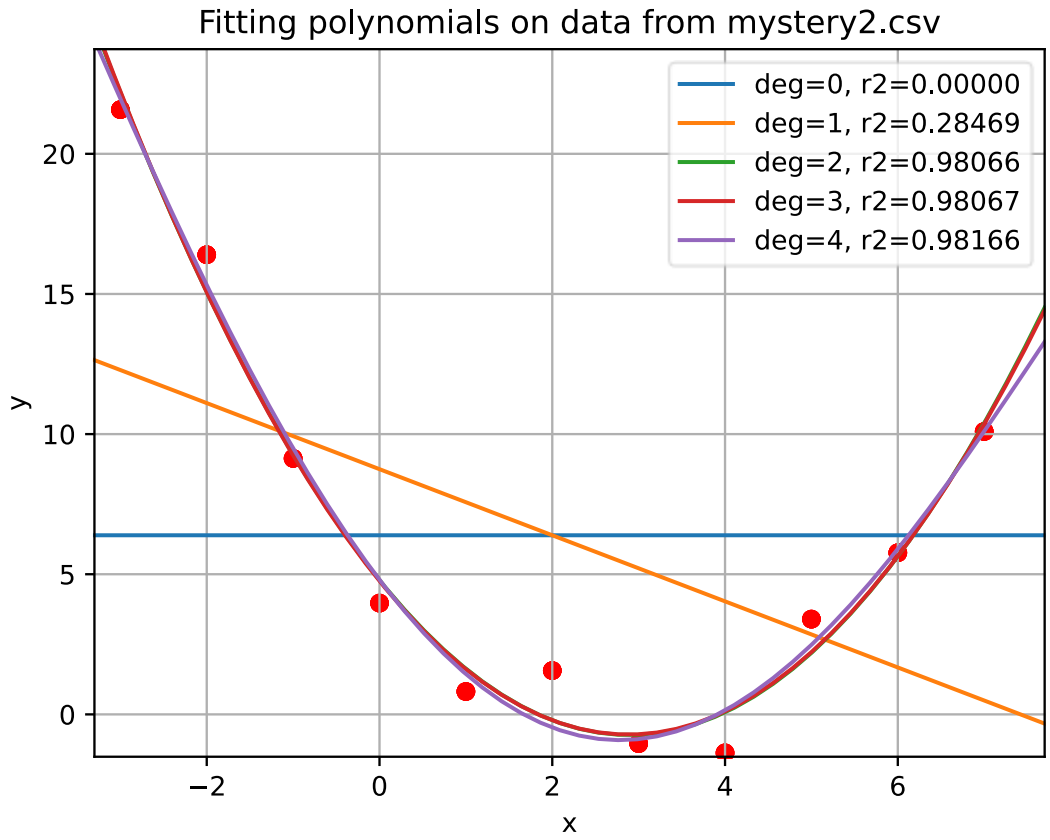
Evaluating a fit

- **Two questions**
  - How well does a line/curve fit the data?
  - What degree polynomial would be an appropriate model?
- **Idea 3:** normalize MSE against a "default" square error
  - **property:** the mean of the $\boldsymbol{y}$ values is the best-fit 0-degree polynomial
  - i.e., $\boldsymbol{\mu_y}$ minimizes the MSE when fitting a constant line to the data, that MSE is the variance $\boldsymbol{\sigma_y^2}$
  - **interpretation: MSE / $\boldsymbol{\sigma_y^2}$** measures what portion of the variance in the data is not explained by the model
  - hence, **coefficient of determination =** $\boldsymbol{r^2 = 1 - \left(\frac{MSE}{\sigma_y^2}\right)}$ measures how much the variance ***is*** explained by the model

$$\boldsymbol{r^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \mu_y)^2}}$$

Training and validation

Does r^2 increase with polynomial degree?



What to evaluate r^2 on?

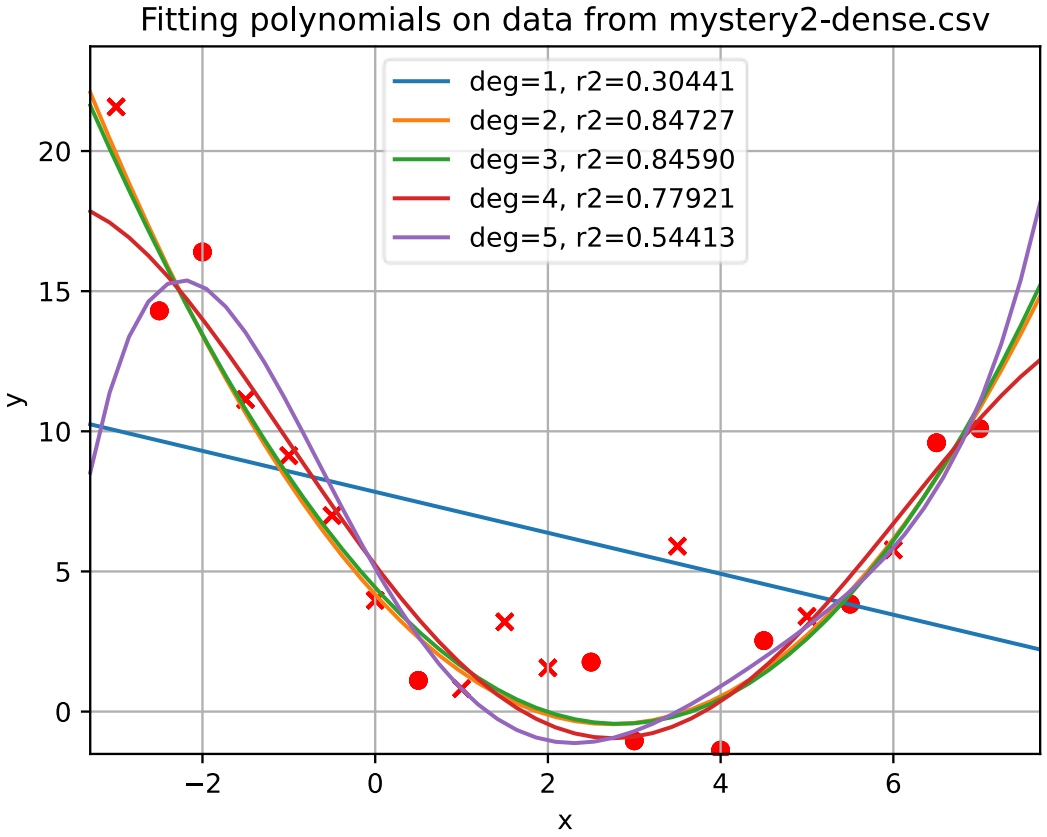
- **Property:** larger degree polynomials are more capable of modeling complexity in the data
- **Issue:** larger degree polynomials are more capable of ***modeling noise*** in the data
- In fact, polynomial of degree $\boldsymbol{n}$ can be fitted to exactly $\boldsymbol{n+1}$ points
  - degree $\boldsymbol{n}$ polynomial is linear in terms of $\boldsymbol{n+1}$ coefficiencts
  - so only need $\boldsymbol{n+1}$ points to form equations for unique solution

Training and validation

- **Insight:** if all the data comes from the same experiment/process, then a similar curve should fit any subset of the data
- **Idea:** split the data into a training set and a validation set
  - fit a model/curve to the training set only
  - evaluate the model (using $\boldsymbol{r^2}$) on the validation set only
  - if the model is modeling too much complexity in the training set, it shouldn't carry over to the validation set, and the $\boldsymbol{r^2}$ will be lower

Training and validation

- **Insight:** if all the data comes from the same experiment/process, then a similar curve should fit any subset of the data
- **Idea:** split the data into a training set and a validation set
  - fit a model/curve to the training set only
  - evaluate the model (using r^2) on the validation set only



How to split data

- Various strategies/techniques/theories for splitting data
  - depends on how heterogeneous/"mixed" your data is
- **Idea 1:** pick every other data point for training
- **Idea 2:** randomly select data points for training
  - in case there's any regularity / periodicity in the data
- **Idea 3:** k-fold cross validation
  - in case you get unlucky with your random selection
  - implement on Pset 3

A quick word on randomness and seeds

- A stochastic program, by definition, gives a different result each time
  - e.g., random splits will yield different r^2 values on validation data, and hence, possibly different best fit degree
- For development/debugging purposes, often advantageous to have deterministic behavior
  - **`random.seed(`*`number`*`)`** sets an internal state for the Python's random number generator
  - results in all subsequent random outputs being deterministic
  - however, still hard to predict if you don't know / aren't considering how Python actually generates random numbers

More info about random number generation

- Actually pretty non-trivial to generate truly random numbers
- Common strategy is to use the **Mersenne Twister**
  - outputs a sequence of numbers between 0 and $2^{32}-1$
  - given initial state, sequence is deterministic and repeats every $2^{19937}-1$ steps
  - for most pratical purposes, output looks random ***(pseudo-random)***
  - setting **`random.seed()`** determines the initial state
- Documentation
  - https://docs.python.org/3/library/random.html
  - https://docs.python.org/3/library/random.html#random.seed
  - https://en.wikipedia.org/wiki/Mersenne_Twister

Next time

- **Lecture 10, Wed 3/4**
  - revisit sample mean and CLT
  - making valid statistical claims
- This week's material rounds out Pset 3