

# **CS-GY 6923: Lecture 2**

## **Multiple Linear Regression + Feature Transformations + Model Selection**

---

NYU Tandon School of Engineering, Akbar Rafiey

- First lab assignment `lab1.ipynb` due **Wednesday, by midnight.**
- Lab 02 will be posted this weekend.

# Recap: supervised learning

## Training Dataset:

- Given input pairs  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ .
- Each  $\mathbf{x}_i$  is an input data vector (the predictor).
- Each  $y_i$  is a (continuous) output variable (the target).

## Objective:

- Have the computer automatically find some function  $f(\mathbf{x})$  such that  $f(\mathbf{x}_i)$  is close to  $y_i$  for the input data.

**Standard approach:** Convert the supervised learning problem to a multi-variable optimization problem.

# Supervised learning definitions

**What are the three components needed to setup a supervised learning problem?**

- **Model**  $f_{\theta}(x)$ : Class of equations or programs which map input  $x$  to predicted output. We want  $f_{\theta}(x_i) \approx y_i$  for training inputs.
- **Model Parameters**  $\theta$ : Vector of numbers. These are numerical nobs which parameterize our class of models.
- **Loss Function**  $L(\theta)$ : Measure of how well a model fits our data. Typically some function of  $f_{\theta}(x_1) - y_1, \dots, f_{\theta}(x_n) - y_n$

**Empirical Risk Minimization:** Choose parameters  $\theta^*$  which minimize the Loss Function:

$$\theta^* = \arg \min_{\theta} L(\theta)$$

## Simple Linear Regression

- Model:  $f_{\beta_0, \beta_1}(x) = \beta_0 + \beta_1 \cdot x$
- Model Parameters:  $\beta_0, \beta_1$
- Loss Function:  $L(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - f_{\beta_0, \beta_1}(x_i))^2$

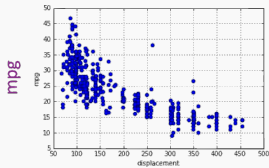
**Goal:** Choose  $\beta_0, \beta_1$  to minimize

$$L(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2.$$

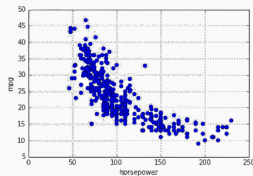
Simple closed form solution:  $\beta_1 = \sigma_{xy} / \sigma_x^2, \beta_0 = \bar{y} - \beta_1 \bar{x}$ . **How did we solve for this solution?**

# Example from last class

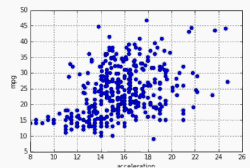
Predict miles per gallon of a vehicle given information about its engine/make/age/etc.



Displacement



Horsepower



Acceleration

# Multiple linear regression

## More common goal

Predict target  $y$  using multiple features, simultaneously.


**Motivating example:** Predict diabetes progression in patients after 1 year based on health metrics. (Measured via numerical score.)

**Features:** Age, sex, average blood pressure, six blood serum measurements (e.g. cholesterol, lipid levels, iron, etc.)

Demo in `demo_diabetes.ipynb` (Demo 3).



## Introducing Scikit Learn.

[Install](#) [User Guide](#) [API](#) [Examples](#) [More](#)

### scikit-learn

Machine Learning in Python

[Getting Started](#) [What's New in 0.22.1](#) [GitHub](#)

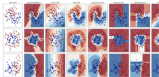
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

#### Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...



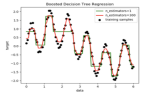
Examples

#### Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...




Examples

#### Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...



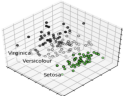
Examples

#### Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** k-Means, feature selection, non-negative matrix factorization, and more...




Examples

#### Model selection

Comparing, validating and choosing parameters and models.

**Applications:** Improved accuracy via parameter tuning

**Algorithms:** grid search, cross validation, metrics, and more...



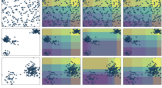
Examples

#### Preprocessing

Feature extraction and normalization.

**Applications:** Transforming input data such as text for use with machine learning algorithms.

**Algorithms:** preprocessing, feature extraction, and more...



Examples



## Pros:

- One of the most popular “traditional” ML libraries.
- Many built in models for regression, classification, dimensionality reduction, etc.
- Easy to use, works with ‘numpy’, ‘scipy’, other libraries we use.
- Great for rapid prototyping, testing models.

## Cons:

- Everything is very “black-box”: difficult to debug, understand why models aren’t working, speed up code, etc.

## Modules used:

- `datasets` module contains a number of pre-loaded datasets. Saves time over downloading and importing with `pandas`.
- `linear_model` can be used to solve Multiple Linear Regression. A bit overkill for this simple model, but gives you an idea of `sklearn`'s general structure.

# The data matrix

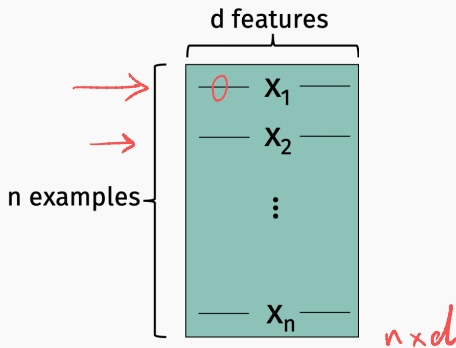
## Target variable:

- Scalars  $y_1, \dots, y_n$  for  $n$  data examples (a.k.a. samples).

## Predictor variables:

- $d$  dimensional vectors  $\mathbf{x}_1, \dots, \mathbf{x}_n$  for  $n$  data examples and  $d$  features

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$



Now it the time to review your linear algebra!

## Notation:

- Let  $\mathbf{X}$  be an  $n \times d$  matrix. Written  $\mathbf{X} \in \mathbb{R}^{n \times d}$ .
- $\mathbf{x}_i$  is the  $i^{\text{th}}$  row of the matrix.
- $\mathbf{x}^{(j)}$  is the  $j^{\text{th}}$  column.
- $x_{ij}$  is the  $i, j$  entry.
- For a vector  $\mathbf{y}$ ,  $y_i$  is the  $i^{\text{th}}$  entry.
- $\mathbf{X}^T$  is the matrix transpose.
- $\mathbf{y}^T$  is a vector transpose.

$$\mathbf{X} \in \mathbb{R}^{n \times d} \rightarrow \mathbf{X}^T \in \mathbb{R}^{d \times n}$$

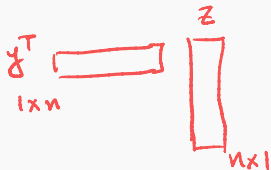
# Linear algebra review

$$\begin{aligned} \mathbf{y} &= [0 \ 0 \ \dots \ 1] \\ \mathbf{z} &= [0 \ 1 \ \dots \ 1] \end{aligned}$$

## Things to remember:

- Matrix multiplication. If we multiply  $\mathbf{X} \in \mathbb{R}^{n \times d}$  by  $\mathbf{Y} \in \mathbb{R}^{d \times k}$  we get  $\mathbf{XY} = \mathbf{Z} \in \mathbb{R}^{n \times k}$ .
- Inner product/dot product.  $\langle \mathbf{y}, \mathbf{z} \rangle = \sum_{i=1}^n y_i z_i$ .
- $\langle \mathbf{y}, \mathbf{z} \rangle = \mathbf{y}^T \mathbf{z} = \mathbf{z}^T \mathbf{y}$ .
- Euclidean norm:  $\|\mathbf{y}\|_2 = \sqrt{\mathbf{y}^T \mathbf{y}}$ .
- $(\mathbf{XY})^T = \mathbf{Y}^T \mathbf{X}^T$ .

$$= \sqrt{\sum_{i=1}^n y_i^2}$$



$$\mathbf{y} = [1, 2, -1] \quad \|\mathbf{y}\|_2 = \sqrt{1^2 + 2^2 + (-1)^2} = \sqrt{1+4+1} = \sqrt{6}$$

$$\mathbf{I} = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}_{n \times n}$$

## Things to remember:

- Identity matrix is denoted as  $\mathbf{I}$ .
- “Most” square matrices have an inverse: i.e. if  $\mathbf{Z} \in \mathbb{R}^{n \times n}$ , there is a matrix  $\mathbf{Z}^{-1}$  such that  $\mathbf{Z}^{-1}\mathbf{Z} = \mathbf{Z}\mathbf{Z}^{-1} = \mathbf{I}$ .
- Let  $\mathbf{D} = \text{diag}(\mathbf{d})$  be a diagonal matrix containing the entries in  $\mathbf{d}$ .
- $\mathbf{XD}$  scales the columns of  $\mathbf{X}$ .  $\mathbf{DX}$  scales the rows.

$$\mathbf{X} = \begin{bmatrix} \vdots \\ x^{(j)} \\ \vdots \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} d_1 & & 0 \\ & d_2 & \\ 0 & & \ddots \\ & & & d_n \end{bmatrix}$$

$$\Rightarrow \mathbf{XD} = \begin{bmatrix} \vdots \\ d_j \cdot x^{(j)} \\ \vdots \end{bmatrix}$$

You also need to be comfortable working with matrices in `numpy` .  
Go through the `demo_numpy_matrices.ipynb` (Demo 1) slowly.



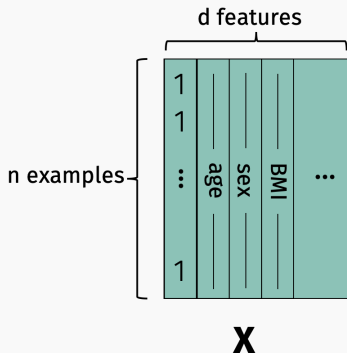
# The data matrix

## Target variable:

- Scalars  $y_1, \dots, y_n$  for  $n$  data examples (a.k.a. samples).

## Predictor variables:

- $d$  dimensional vectors  $\mathbf{x}_1, \dots, \mathbf{x}_n$  for  $n$  data examples and  $d$  features



# Multiple linear regression

Data matrix indexing:

$$f_{\beta_0, \beta_1}(x) = \beta_0 + \beta_1 x$$


$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & \dots & \vdots \\ x_{i1} & x_{i2} & \dots & x_{id} \\ \vdots & \vdots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

Multiple Linear Regression Model:

Predict  $y_i \approx \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_d x_{id}$

The rate at which diabetes progresses depends on many factors, with each factor having a different magnitude effect.

# Multiple linear regression

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} = \begin{bmatrix} 1 & x_{12} & \dots & x_{1d} \\ 1 & x_{22} & \dots & x_{2d} \\ 1 & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n2} & \dots & x_{nd} \end{bmatrix}$$


## Multiple Linear Regression Model:

Predict  $y_i \approx \beta_1 + \beta_2 x_{i2} + \dots + \beta_d x_{id}$



In this case,  $\beta_1$  serves as the “intercept” parameter.

# Multiple linear regression

## Multiple Linear Regression Model:

Predict  $y_i \approx \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_d x_{id}$

### Data matrix:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} = \begin{bmatrix} 1 & x_{12} & \dots & x_{1d} \\ 1 & x_{22} & \dots & x_{2d} \\ 1 & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n2} & \dots & x_{nd} \end{bmatrix} \rightarrow \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \end{bmatrix}$$

### Linear algebraic form:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \approx \begin{bmatrix} \langle x_{1, \cdot}, \beta \rangle \\ \langle x_{2, \cdot}, \beta \rangle \\ \vdots \\ \langle x_{n, \cdot}, \beta \rangle \end{bmatrix}$$

$$y_i \approx \langle \mathbf{x}_i, \beta \rangle$$

$$\mathbf{y} \approx \mathbf{X}\beta$$

$$y_3 \approx \beta_1 + \beta_2 x_{32} + \dots + \beta_d x_{3d}$$

# Multiple linear regression

## Linear Least-Squares Regression.

- Model Parameters:

$$\beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \end{bmatrix}$$

- Model:

$$f_{\beta}(\mathbf{x}) = \langle \mathbf{x}, \beta \rangle$$

- Loss Function:

$$\begin{aligned} L(\beta) &= \sum_{i=1}^n |y_i - \langle \mathbf{x}_i, \beta \rangle|^2 \\ &= \|\mathbf{y} - \mathbf{X}\beta\|_2^2 \end{aligned}$$

$$\begin{aligned} \|\mathbf{z}\|_2^2 &= \\ &= \sum_{i=1}^n z_i^2 \\ \|\mathbf{z}\|_2^2 &= \sum_{i=1}^n z_i^2 \end{aligned}$$

# Linear algebraic form of loss function

$$L(\beta) = \|y - X\beta\|_2^2$$

$$y = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_n \end{bmatrix}_{n \times 1} \quad X\beta = \begin{matrix} \nearrow n \times d \\ \searrow d \times 1 \end{matrix} \begin{bmatrix} X \\ \end{bmatrix}_{n \times d} \begin{bmatrix} \beta \\ \end{bmatrix}_{d \times 1} = \begin{bmatrix} \langle x_1, \beta \rangle \\ \langle x_2, \beta \rangle \\ \vdots \\ \langle x_n, \beta \rangle \end{bmatrix}_{n \times 1}$$

$$\underline{y - X\beta} = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \end{bmatrix}_{n \times 1} - \begin{bmatrix} \langle x_1, \beta \rangle \\ \vdots \\ \langle x_n, \beta \rangle \end{bmatrix}_{n \times 1} = \begin{bmatrix} \delta_1 - \langle x_1, \beta \rangle \\ \vdots \\ \delta_n - \langle x_n, \beta \rangle \end{bmatrix}_{n \times 1} = \begin{bmatrix} 3 - 2.5 \\ \vdots \\ 5 - 5.5 \\ \vdots \\ -0.5 \end{bmatrix}$$

$$\|y - X\beta\|_2^2 = \sum_{i=1}^n (\delta_i - \langle x_i, \beta \rangle)^2$$

# Loss minimization

**Machine learning goal:** minimize the loss function

$$L(\boldsymbol{\beta}) : \mathbb{R}^d \rightarrow \mathbb{R}.$$

Find possible optima by determining for which  $\boldsymbol{\beta}^T = [\beta_1, \dots, \beta_d]$  all the **gradient** equals **0**. I.e. when do we have:

$$\nabla L(\boldsymbol{\beta}) = \begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

# Gradient

Loss function:

$$L(\beta) = \|\mathbf{y} - \mathbf{X}\beta\|_2^2$$

Gradient:

$$-2 \cdot \mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta)$$

Can check that this is equal to 0 if  $\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ . There are no other options, so this must be the minimum.

$$\begin{aligned} -2 \mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta) = 0 &\Rightarrow \mathbf{X}^T \mathbf{y} - \mathbf{X}^T \mathbf{X} \beta = 0 \\ \Rightarrow \mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X} \beta &\Rightarrow (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \underbrace{(\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{X})}_{= I} \beta \end{aligned}$$



## Single variable warmup

What is the derivative of:  $f(x) = x^2$ ?

$$\lim_{\delta \rightarrow 0} \frac{f(x+\delta) - f(x)}{\delta} = \lim_{\delta \rightarrow 0} \frac{(x+\delta)^2 - x^2}{\delta}$$

$$\begin{aligned} \Rightarrow \lim_{\delta \rightarrow 0} \frac{\cancel{x^2} + \cancel{\delta^2} + 2x\delta - \cancel{x^2}}{\cancel{\delta}} &= \lim_{\delta \rightarrow 0} \delta + 2x \\ &= 2x \end{aligned}$$

# Gradient

Loss function:  $L(\beta) = \|y - X\beta\|_2^2$

$$\lim_{\delta \rightarrow 0} \frac{L(\beta + \delta e_1) - L(\beta)}{\delta} = \lim_{\delta \rightarrow 0} \frac{\|y - X(\beta + \delta e_1)\|_2^2 - \|y - X\beta\|_2^2}{\delta}$$

$$= \lim_{\delta \rightarrow 0} \frac{\|y - X\beta - \delta X \cdot e_1\|_2^2 - \|y - X\beta\|_2^2}{\delta}$$

$$= \lim_{\delta \rightarrow 0} \frac{\|y - X\beta\|_2^2 + \delta \|X^{(1)}\|_2^2 - 2(X^{(1)})^T (y - X\beta) - \|y - X\beta\|_2^2}{\delta}$$

$$= 2(X^{(1)})^T (y - X\beta) = \frac{\partial L(\beta)}{\partial \beta_1}$$

$$\begin{aligned} \|a - b\|_2^2 &= \\ \|a\|_2^2 + \|b\|_2^2 &- 2a^T b \end{aligned}$$

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$e_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$e_3 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

## Multiple linear regression solution

Take away: simple form for the gradient means that multiple linear regression models are easy and efficient to optimize.

$$\beta^* = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- Often the “go to” first regression method. Throw your data in a matrix and see what happens.
- Serve as the basis for much richer classes of models.

## Multiple linear regression solution

Need to compute  $\beta^* = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ .

- Main cost is computing  $(\mathbf{X}^T \mathbf{X})^{-1}$  which takes  $O(nd^2)$  time.
- Can solve slightly faster using the method `numpy.linalg.lstsq`, which is running an algorithm based on QR decomposition.
- For larger problems, can solve much faster using an *iterative methods* like `scipy.sparse.linalg.lsqr`.

Will learn more about iterative methods when we study Gradient Descent.

## Encoding data as a numerical matrix

It is not always immediately clear how to do this! One of the first issue we run into is categorical data:

$$\mathbf{x}_1 = [42, 4, 104, \text{hybrid}, \text{ford}]$$
$$\mathbf{x}_2 = [18, 8, 307, \text{gas}, \text{bmw}]$$
$$\mathbf{x}_2 = [31, 4, 150, \text{gas}, \text{honda}]$$
$$\vdots$$

## Encoding data as a numerical matrix

Binary data is easy to deal with – pick one category to be 0, one to be 1. The choice doesn't matter – it will not impact the overall loss of the model

$$\mathbf{x}_1 = [42, 4, 104, \text{hybrid}, \text{ford}]$$
$$\mathbf{x}_2 = [18, 8, 307, \text{gas}, \text{bmw}]$$
$$\mathbf{x}_2 = [31, 4, 150, \text{gas}, \text{honda}]$$
$$\vdots$$
$$\mathbf{x}_1 = [42, 4, 104, 1, \text{ford}]$$
$$\mathbf{x}_2 = [18, 8, 307, 0, \text{bmw}]$$
$$\mathbf{x}_2 = [31, 4, 150, 0, \text{honda}]$$
$$\vdots$$

## Dealing with categorical variables

What about a categorical predictor variable for car make with more than 2 options: e.g. Ford, BMW, Honda. **How would you encode as a numerical column?**

|               |   |          |
|---------------|---|----------|
| [ ford ]      | → | [ 0 ]    |
| [ ford ]      |   | [ 0 ]    |
| [ honda ]     |   | [ 1 ]    |
| [ bmw ]       |   | [ 2 ]    |
| [ honda ]     |   | [ 1 ]    |
| [ ford ]      |   | [ 0 ]    |
| <i>Nissan</i> |   | <i>5</i> |

# One hot encoding

Better approach: One Hot Encoding.

$$\begin{array}{l} \left[ \begin{array}{l} \text{ford} \\ \text{ford} \\ \text{honda} \\ \text{bmw} \\ \text{honda} \\ \text{ford} \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array} \right] \\ \text{Nissan} \quad \quad \quad 0 \quad 0 \quad 0 \quad 1 \end{array}$$

- Create a separate feature for every category, which is 1 when the variable is in that category, zero otherwise.
- Not too hard to do by hand, but you can also use library functions like `sklearn.preprocessing.OneHotEncoder`.

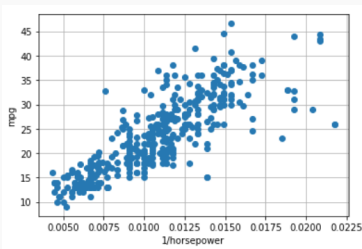
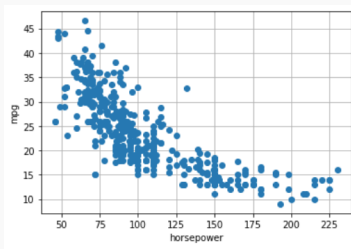
**Avoids adding inadvertent linear relationships.**



# Transformed linear models

## Example from last time

Instead of fitting the model  $\text{mpg} \approx \beta_0 + \beta_1 \cdot \text{horsepower}$ , fit the model  $\text{mpg} \approx \beta_0 + \beta_1 \cdot 1/\text{horsepower}$ .



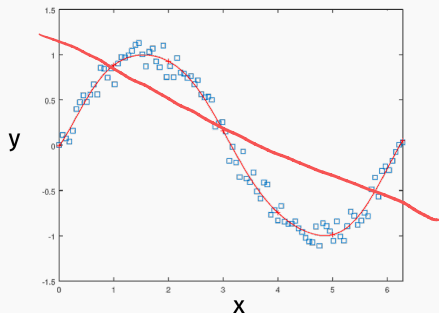
**How would you know to make such a transformation?**

**Better approach:** Choose a more flexible non-linear model class.

# Transformed linear models

Suppose we have singular variate data examples  $(x, y)$ . We could fit the non-linear polynomial model:

$$y \approx \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3.$$



**Claim:** This can be done using an algorithm for multivariate regression!

## Transformed linear models

Transform into a multiple linear regression problem:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix} \quad \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$

What is the output of  $\mathbf{X}\beta$ ?

$$\langle x_i, \beta \rangle = \beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3$$

## Transformed linear models

More generally, have each column  $j$  is generated by a different basis function  $\phi_j(x)$ . Could have:

- $\phi_j(x) = x^q$
- $\phi_j(x) = \sin(x)$
- $\phi_j(x) = \cos(10x)$
- $\phi_j(x) = 1/x$

$$X = \begin{bmatrix} 1 & x_1 & \sin(x_1) \\ 1 & x_2 & \sin(x_2) \\ \vdots & \vdots & \vdots \\ 1 & x_n & \sin(x_n) \end{bmatrix} \quad \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

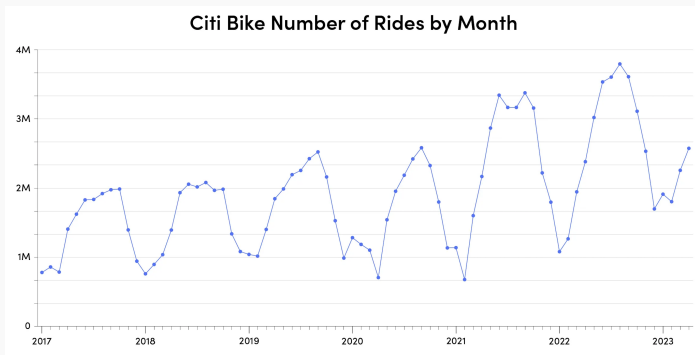
$$\langle X, \beta \rangle = \beta_1 + \beta_2 x_2 + \beta_3 \sin(x_3)$$

When might you want to include sines and cosines?

# Transformed linear models

When might you want to include sines and cosines?

Time series data (figure from Lyft website):



There is usually not much harm in including irrelevant variable transformation.

# Multinomial model

Transformations can also be for multivariate data.

**Example:** Multinomial model.

- Given a dataset with target  $y$  and predictors  $x, z$ .
- For inputs  $(x_1, z_1), \dots, (x_n, z_n)$  construct the data matrix:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & z_1 & z_1^2 & x_1 z_1 \\ 1 & x_2 & x_2^2 & z_2 & z_2^2 & x_2 z_2 \\ \vdots & \vdots & & \vdots & & \\ 1 & x_n & x_n^2 & z_n & z_n^2 & x_n z_n \end{bmatrix} \rightarrow \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_6 \end{bmatrix}$$

- Captures non-linear interaction between  $x$  and  $z$ .

$$\beta_1 + \dots + \beta_6 x_1 z_1$$

**Remainder of lecture:** Learn about model selection, test/train paradigm, and cross-validation through a simple example.

Check out Demo 4.

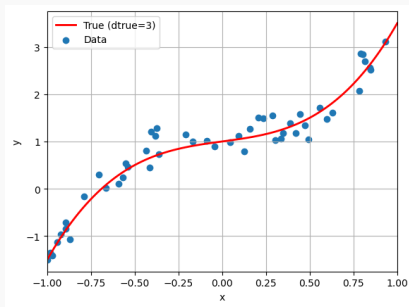


# Fitting a polynomial

## Simple experiment:

$$L(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

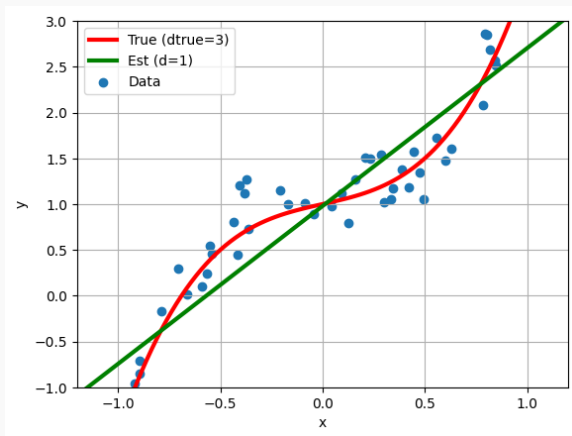
- Randomly select data points  $x_1, \dots, x_n \in [-1, 1]$ .
- Choose a degree 3 polynomial  $p(x)$ .
- Create some fake data:  $y_i = p(x_i) + \eta$  where  $\eta$  is a random number (e.g random Gaussian).



# Fitting a polynomial

## Simple experiment:

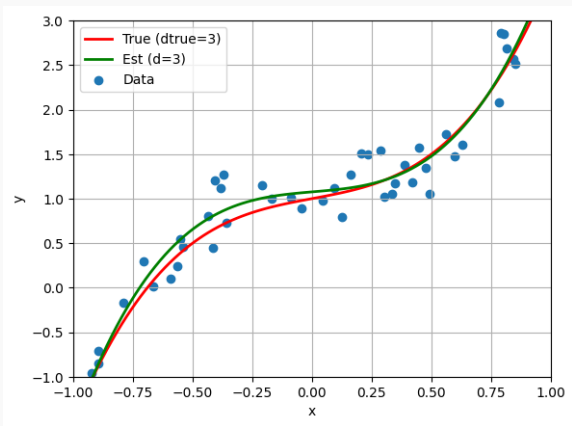
- Use multiple linear regression to fit a line (degree 1 polynomial). This model seems underfit.



# Fitting a polynomial

## Simple experiment:

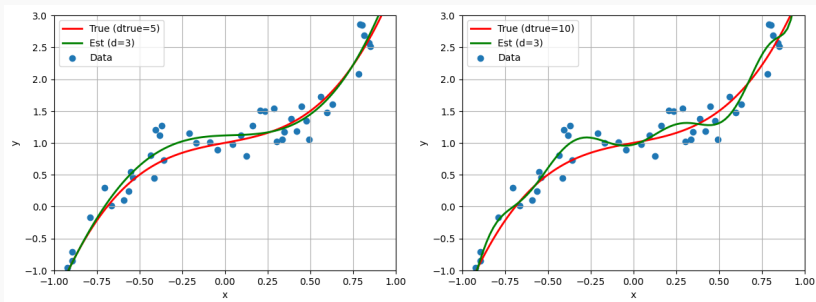
- Use multiple linear regression to fit a degree 3 polynomial. Almost perfectly captures the true function!



# Fitting a polynomial

## What if we fit a higher degree polynomial?

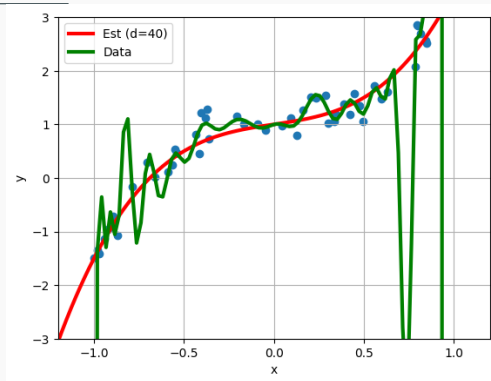
- Fit degree 5 polynomial under squared loss.
- Fit degree 10 polynomial under squared loss.



# Fitting a polynomial

## Even higher?

- Fit degree 40 polynomial under squared loss. This model seems **overfit**.



The model “overreacts” to minor variations in the data, which can lead to some bad behavior..

## Quick aside on numerical issues

In the demo we have you use `numpy.polynomial.polynomial`. However, as we discussed early, we can use multiple linear regression instead by constructing the data matrix:

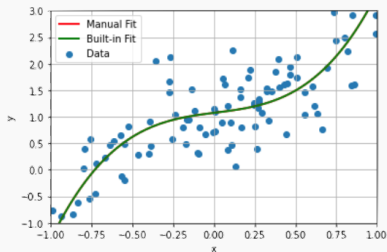
$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

Then find polynomial coefficients as  $\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ .

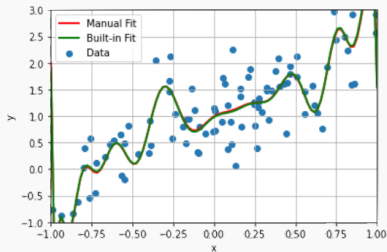
# Quick aside on numerical issues

```
# built in function
beta_hat = poly.polyfit(xdat,ydat,d)

# manual fit using naive multivariate regression
X = np.zeros([len(xdat),d+1])
for i in range(d+1):
    X[:,i] = xdat**i
my_beta = np.linalg.inv(np.transpose(X)@X)@np.transpose(X)@ydat
```



Degree 3

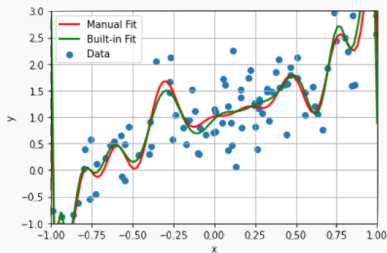


Degree 22

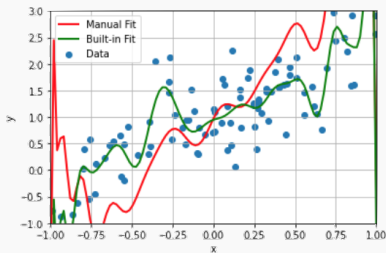
## Quick aside on numerical issues

```
# built in function
beta_hat = poly.polyfit(xdat,ydat,d)

# manual fit using naive multivariate regression
X = np.zeros([len(xdat),d+1])
for i in range(d+1):
    X[:,i] = xdat**i
my_beta = np.linalg.inv(np.transpose(X)@X)@np.transpose(X)@ydat
```



Degree 23



Degree 30

Has to do with numerical roundoff error. Scipy still uses linear regression, but with extra “tricks” to avoid numerical issues.



## Quick aside on numerical issues

- Your computer can easily deal with both very large and very small numbers. Underflow and overflow are extremely unlikely to be issues in floating point arithmetic.
- The issue is when you compute using numbers of very differing magnitude.

```
print(.3*10**-34 + 10**-36 - 10**-36)
```

```
3e-35
```

```
print(.3*10**-34 + 10 - 10)
```

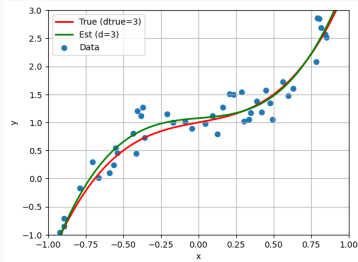
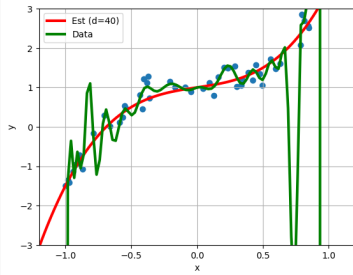
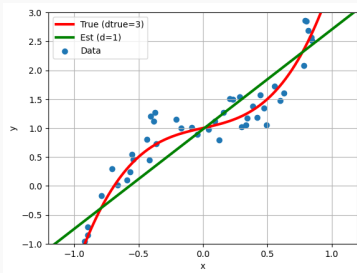
```
0.0
```

## Quick aside on numerical issues

Recall that we chose each  $x_i \in [-1, 1]$  uniformly at random.

$$\mathbf{X} = \begin{matrix} \longrightarrow & \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix} & x_1^4 & \dots & x_1^{30} \end{matrix}$$

# Back to the problem at hand

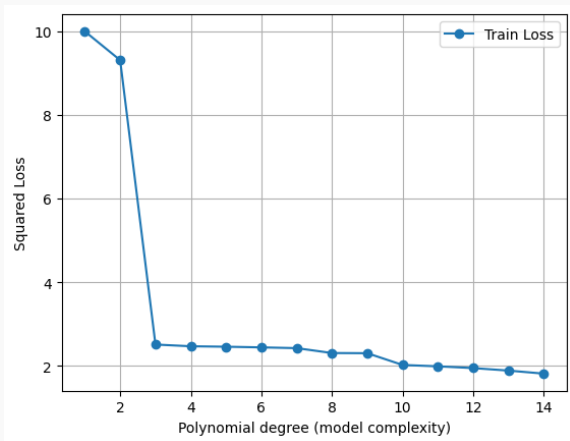


Underfit, overfit, just right.

For high-dimensional data, we cannot produce such easy to read plots. How can we automatically detect when we have “underfit” or “overfit” to choose the right model?

## Model complexity vs. loss

Typically, the more **complex** our model, the better our loss:



For transformed linear models, this is formally true: more feature transformations leads to lower loss.

## Model selection

Consider  $\mathbf{X} \in \mathbb{R}^{n \times d}$  and  $\bar{\mathbf{X}} = [\mathbf{X}, \mathbf{z}] \in \mathbb{R}^{n \times d+1}$  with one additional column appended on.

**Claim:**

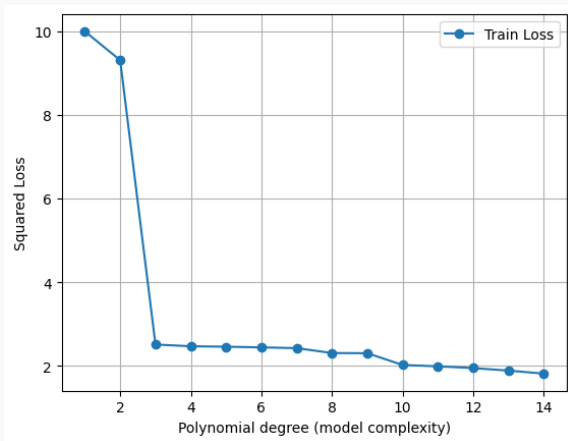
$$\min_{\bar{\beta} \in \mathbb{R}^{d+1}} \|\bar{\mathbf{X}}\bar{\beta} - \mathbf{y}\|_2^2 \leq \min_{\beta \in \mathbb{R}^d} \|\mathbf{X}\beta - \mathbf{y}\|_2^2.$$

$$\beta^* = \arg \min_{\beta} \|\mathbf{X}\beta - \mathbf{y}\|_2^2$$

$$\beta' = \begin{bmatrix} \beta^* \\ 0 \end{bmatrix} \Rightarrow \|\bar{\mathbf{X}}\beta' - \mathbf{y}\|_2^2 = \|\mathbf{X}\beta^* - \mathbf{y}\|_2^2$$

# Model selection

The more **complex** our model class the better our loss:

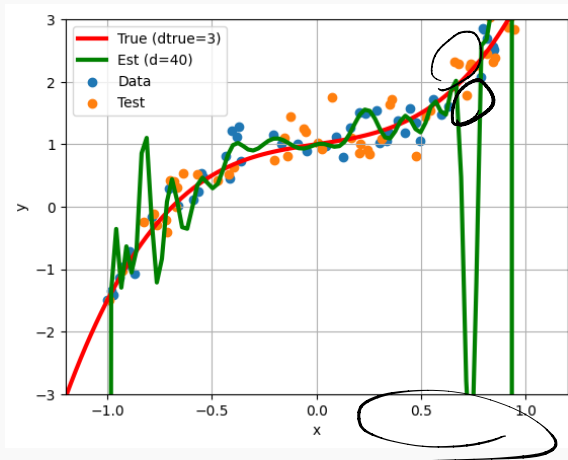


So training loss alone is not usually a good metric for model selection.

# Model selection

**Problem:** Small loss does not imply generalization.

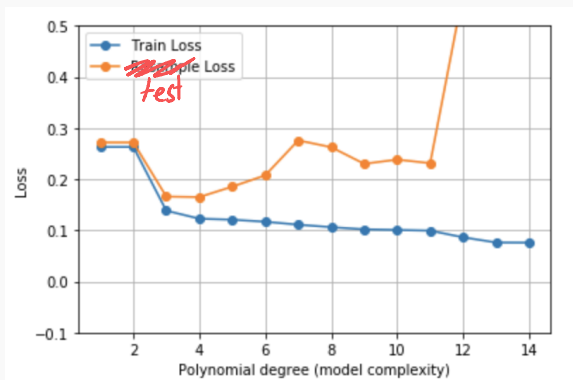
**Generalization:** How well do we do on new data.





# Model selection

**Solution:** Directly test model on “new data”.



- **Train loss** decreases as model complexity grows.
- **Test loss** “turns around” once our model gets too complex. Minimized around degree 3 – 4.

# Train-test paradigm

**More reasonable approach:** Evaluate model on fresh test data which was not used during training.

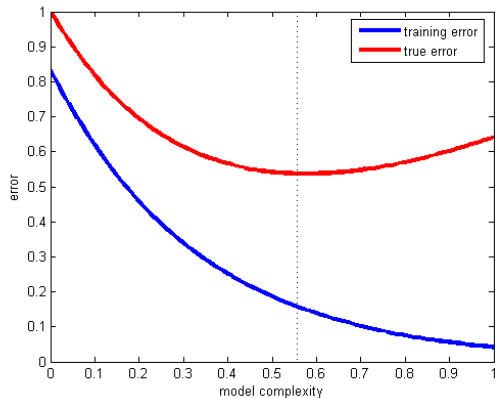
## Test/train split:

- Given data set  $(\mathbf{X}, \mathbf{y})$ , split into two sets  $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$  and  $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$ .
- Train  $q$  models  $f^{(1)}, \dots, f^{(q)}$  of varying complexity by finding parameters which minimize the loss on  $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$ .
- Evaluate loss of each trained model on  $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$ .
- Pick model with lowest test loss.

Sometimes you will see the term **validation set** instead of test set. Sometimes there will be both: use validation set for choosing the model, and test set for getting a final performance measure.

# The fundamental curve of ML

The above trend is fairly representative of what we tend to see across the board:



## Generalization error

If the test loss remains low, we say that the model **generalizes**.  
Test loss is often called **generalization error**.

**Typical train-test split:** 90-70% / 10-30%. Trade-off between optimization of model parameters and better estimate of model performance.

# K-fold cross validation



- Randomly divide data in  $K$  parts.
  - Typical choice:  $K = 5$  or  $K = 10$ .
- Use  $K - 1$  parts for training, 1 for test.
- For each model, compute test loss  $L_{TS}$  for each "fold".
- Choose model with best average loss.
- Retrain best model on entire dataset.

Is there any disadvantage to choosing  $K$  larger?

Is “test error” the end goal though? Don’t we care about “future” error?

**Intuition:** Models which perform better on the test set will **generalize** better to future data.

**Goal:** Introduce a little bit of formalism to better understand what this means. What is “future” data?

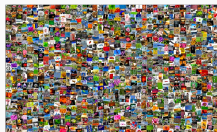
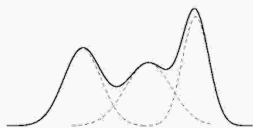
# Statistical learning model

## Statistical Learning Model:

- Assume each data example is randomly drawn from some distribution  $(\mathbf{x}, y) \sim \mathcal{D}$ .

E.g.  $x_1, \dots, x_d$  are Gaussian random variables with parameters

$$\mu_1, \sigma_1, \dots, \mu_d, \sigma_d.$$



This is not really a simplifying assumption! The distribution could be arbitrarily complicated.



## Statistical Learning Model:

- Assume each data example is randomly drawn from some distribution  $(\mathbf{x}, y) \sim \mathcal{D}$ .
- Define the **Risk** of a model/parameters:

$$R(f, \theta) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [L(f(\mathbf{x}, \theta), y)]$$

here  $L$  is our loss function (e.g.  $L(z, y) = |z - y|$  or  $L(z, y) = (z - y)^2$ ).

**Ultimate Goal:** Find model  $f \in \{f^{(1)}, \dots, f^{(q)}\}$  and parameter vector  $\theta$  to minimize the  $R(f, \theta)$ .

- **(Population) Risk:**

$$R(f, \theta) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [L(f(\mathbf{x}, \theta), y)]$$

- **Empirical Risk:** Draw  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n) \sim \mathcal{D}$

$$R_E(f, \theta) = \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i, \theta), y_i)$$

For any fixed model  $f$  and parameters  $\theta$ ,

$$\mathbb{E}[R_E(f, \theta)] = R(f, \theta).$$

Only true if  $f$  and  $\theta$  are chosen *without looking at the data used to compute the empirical risk*.

## Model selection

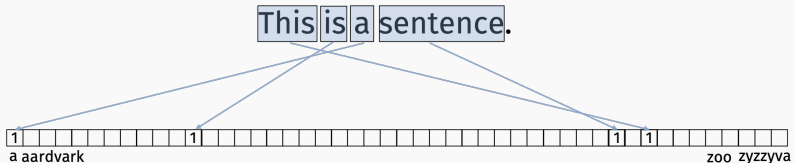
- Train  $q$  models  $(f^{(1)}, \theta_1^*), \dots, (f^{(q)}, \theta_q^*)$ .
- For each model, compute empirical risk  $R_E(f^{(i)}, \theta_i^*)$  using test data.
- Since we assume our original dataset was drawn independently from  $\mathcal{D}$ , so is the random test subset.

No matter how our models were trained or how complex they are,  $R_E(f^{(i)}, \theta_i^*)$  is an unbiased estimate of the true risk  $R(f^{(i)}, \theta_i^*)$  for every  $i$ . Can use it to distinguish between models.

# Model selection example

## bag-of-words models and n-grams

Common way to represent documents (emails, webpages, books) as numerical data. The ultimate example of 1-hot encoding.

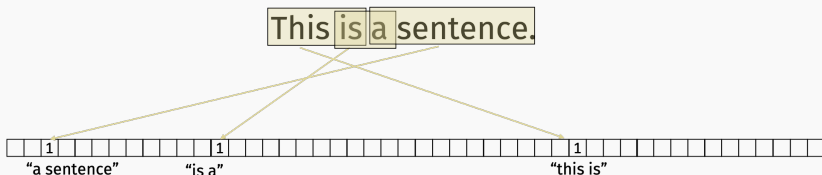


**bag-of-words**

# Model selection example

## bag-of-words models and n-grams

Common way to represent documents (emails, webpages, books) as numerical data. The ultimate example of 1-hot encoding.

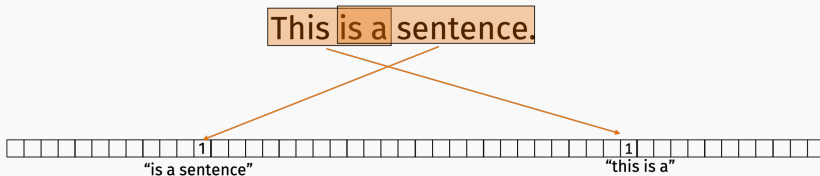


## bi-grams

# Model selection example

## bag-of-words models and n-grams

Common way to represent documents (emails, webpages, books) as numerical data. The ultimate example of 1-hot encoding.



**tri-grams**

# Model selection example

## Models of increasing order:

- Model  $f_{\theta_1}^{(1)}$ : spam filter that looks at **single words**.
- Model  $f_{\theta_2}^{(2)}$ : spam filter that looks at **bi-grams**.
- Model  $f_{\theta_3}^{(3)}$ : spam filter that looks at **tri-grams**.
- ...

“interest”

“low interest”

“low interest loan”

Increased length of **n-gram** means more expressive power.

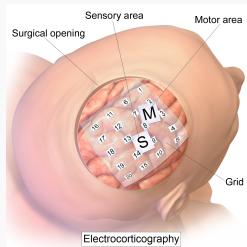
**Will be very relevant in our lab on generative language models!**



# Model selection example

## Electrocorticography ECoG (next lab):

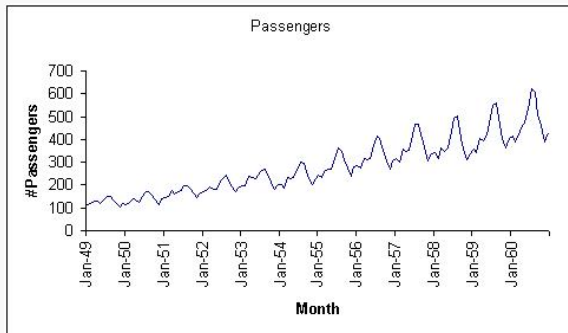
- Implant grid of electrodes on surface of the brain to measure electrical activity in different regions.



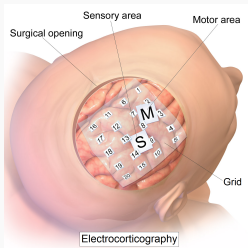
- Predict hand motion based on ECoG measurements.
- **Model order:** predict movement at time  $t$  using brain signals at time  $t, t - 1, \dots, t - q$  for varying values of  $q$ .

## Autoregressive model

Predicting time  $t$  based on a linear function of the signals at time  $t, t - 1, \dots, t - q$  is not the same as fitting a line to the time series. It's much more expressive.



## Electrocorticography ECoG lab:



**First lab where computation actually matters (solving regression problems with  $\sim 40k$  examples,  $\sim 1500$  features)**

Makes sense to test and debug code using a subset of the data.





**Slight caveat: The train-test paradigm is typically not how machine learning or scientific discovery works in practice!**

## Typical workflow:


- Train a class of models.
- Test.
- Adjust class of models.
- Test.
- Adjust class of models.
- Cont...

Final model implicitly depends on test set because performance on the test set guided how we changed our model.

## Popularity of ML benchmarks and competitions leads to adaptivity at a massive scale.

| 11 Active Competitions  |   |                            |
|---|---|----------------------------|
|  | <b>Deepfake Detection Challenge</b><br>Identify videos with facial or voice manipulations<br><small>Featured · Code Competition · 2 months to go · video data, online video</small>                           | \$1,000,000<br>1,595 teams |
|  | <b>Google QUEST Q&amp;A Labeling</b><br>Improving automated understanding of complex question answer content<br><small>Featured · Code Competition · 19 hours to go · text data, nlp</small>                  | \$25,000<br>1,559 teams    |
|  | <b>Real or Not? NLP with Disaster Tweets</b><br>Predict which Tweets are about real disasters and which ones are not<br><small>Getting Started · Ongoing · text data, binary classification</small>           | \$10,000<br>2,657 teams    |
|  | <b>Bengali.AI Handwritten Grapheme Classification</b><br>Classify the components of handwritten Bengali<br><small>Research · Code Competition · a month to go · multiclass classification, image data</small> | \$10,000<br>1,194 teams    |

Kaggle (various competitions)



14,197,122 images, 21841 synsets indexed

[Explore](#) [Download](#) [Challenges](#) [Publications](#) [Updates](#) [About](#)

Not logged in. [Login](#) | [Signup](#)

Imagenet (image classification and categorization)

**Is adaptivity a problem? Does it lead to over-fitting? How much? How can we prevent it?** All current research. Related to the problem of “p-value hacking” in science.

REPORT

## The reusable holdout: Preserving validity in adaptive data analysis

Cynthia Dwork<sup>1,\*</sup>, Vitaly Feldman<sup>2,\*</sup>, Moritz Hardt<sup>3,\*</sup>, Toniann Pitassi<sup>4,\*</sup>, Omer Reingold<sup>5,\*</sup>, Aaron Roth<sup>6,\*</sup>

+ See all authors and affiliations

Science 07 Aug 2015:  
Vol. 349, Issue 6248, pp. 636-638  
DOI: 10.1126/science.aaa9375

### Do ImageNet Classifiers Generalize to ImageNet?

Benjamin Recht\*  
UC Berkeley

Rebecca Roelofs  
UC Berkeley

Ludwig Schmidt  
UC Berkeley

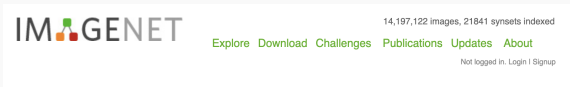
Vaishal Shankar  
UC Berkeley

#### Abstract

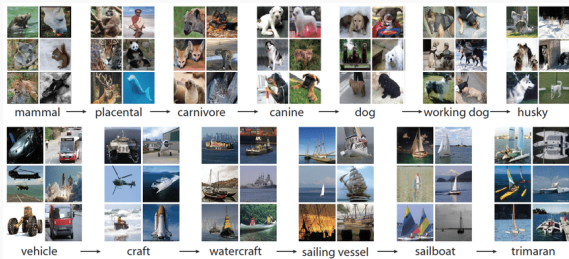
We build new test sets for the CIFAR-10 and ImageNet datasets. Both benchmarks have been the focus of intense research for almost a decade, raising the danger of overfitting to excessively re-used test sets. By closely following the original dataset creation processes, we test to what extent current classification models generalize to new data. We evaluate a broad range of models and find accuracy drops of 3% – 15% on CIFAR-10 and 11% – 14% on ImageNet. However, accuracy gains on the original test sets translate to larger gains on the new test sets. Our results suggest that the accuracy drops are not caused by adaptivity, but by the models’ inability to generalize to slightly “harder” images than those found in the original test sets.

12 Jun 2019

# Imagenet dataset

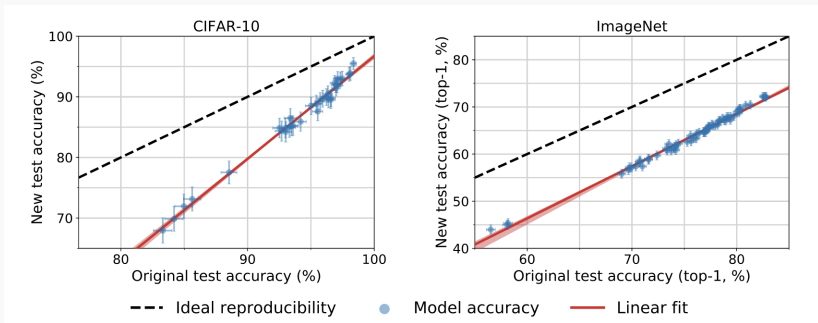


Collected by Fei-Fei Li's group at Stanford in 2006ish and labeled using Amazon Mechanical Turk.



We now have neural network models that can solve these classification problems with  $> 95\%$  accuracy.

## Do ImageNet Classifiers Generalized to ImageNet?



Interestingly, when comparing popular vision models on “fresh” data, while performance dropped across the board, the relative rank of model performance did not change significantly.