CS-GY 6923: Lecture 2
Multiple Linear Regression + Feature Transformations + Model Selection

NYU Tandon School of Engineering, Prof. Christopher Musco

COURSE ADMIN

- First lab assignment `lab_housing_partial.ipynb` due **tonight, by midnight.**
- First written assignment due **next Thursday, by midnight.**
    - 10% extra credit if you use LaTeX or Markdown to typeset your assignment.

The problem set is challenging. I expect working through the problems to be one of the major ways you master material for the course. Please try to get started ASAP so that you can take advantage of office hours next week if needed.

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^{th} row of the matrix.
- $\mathbf{x}^{(j)}$ is the j^{th} column.
- x_{ij} is the i, j entry.
- For a vector $\mathbf{y}$, y_i is the i^{th} entry.
- $\mathbf{X}^T$ is the matrix transpose.
- $\mathbf{y}^T$ is a vector transpose.

Things to remember:

- Matrix multiplication. If I multiply $\mathbf{X} \in \mathbb{R}^{n \times d}$ by $\mathbf{Y} \in \mathbb{R}^{d \times k}$ I 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: $\|\mathbf{y}\|_2 = \sqrt{\mathbf{y}^T \mathbf{y}}$.
- $(\mathbf{XY})^T = \mathbf{Y}^T \mathbf{X}^T$.

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 $\mathbf{d}$.
- $\mathbf{X}\mathbf{D}$ scales the columns of $\mathbf{X}$. $\mathbf{D}\mathbf{X}$ scales the rows.

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

REMINDER: SUPERVISED REGRESSION

Training Dataset:

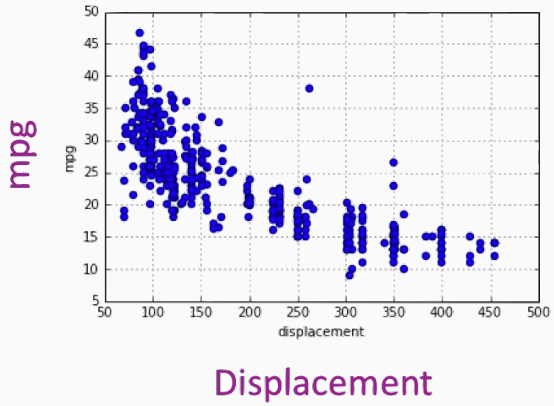
- Given input pairs $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_i$ is an input data vector (the predictor).
- Each y_i is a continuous output variable (the target).

Objective:

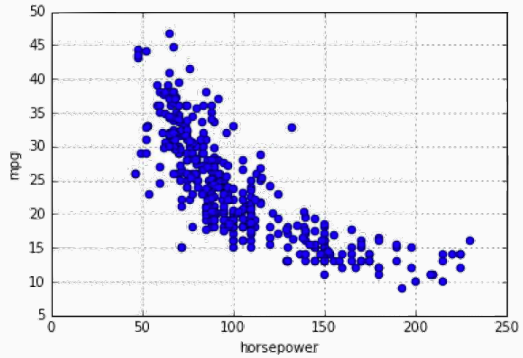
- Have the computer automatically find some function $f(\mathbf{x})$ such that $f(\mathbf{x}_i)$ is close to y_i for the input data.

EXAMPLE FROM LAST CLASS

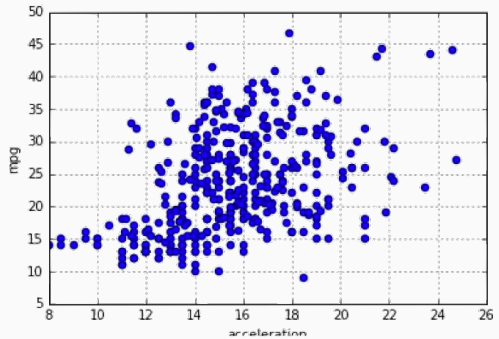
Predict miles per gallon of a vehicle given information about its engine/make/age/etc.



Displacement



Horsepower

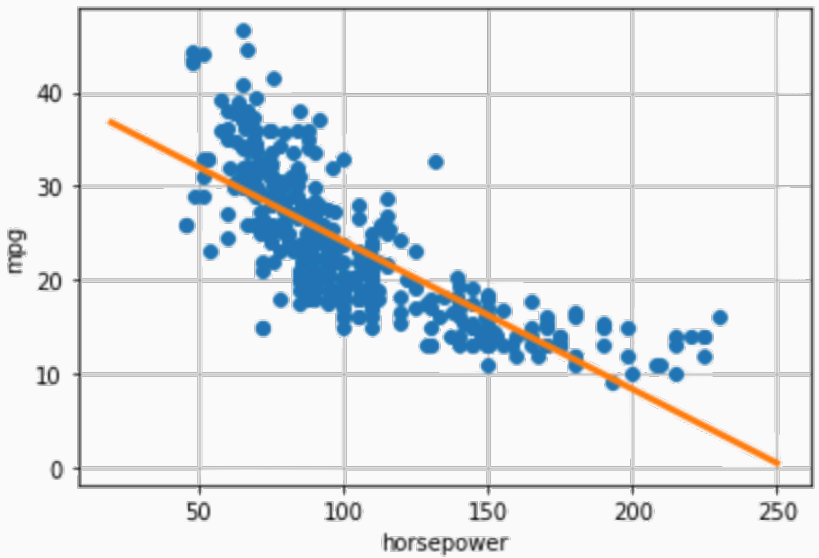


Acceleration

EXAMPLE FROM LAST CLASS

Dataset:

- $x_1, \ldots, x_n \in \mathbb{R}$ (horsepowers of n cars – this is the predictor/independent variable)
- $y_1, \ldots, y_n \in \mathbb{R}$ (MPG – this is the response/dependent variable)



SUPERVISED LEARNING DEFINITIONS

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

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

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

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

Simple Linear Regression

- Model: $f_{\beta_0,\beta_1}(x) = \beta_0 + \beta_1 \cdot x$
- Model Parameters: β_0, β_1
- Loss Function: $L(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - f_{\beta_0,\beta_1}(x_i))^2$

Goal: Choose β_0, β_1 to minimize
$L(\beta_0, \beta_1) = \sum_{i=1}^{n} |y_i - \beta_0 - \beta_1 x_i|^2$.

MULTIPLE LINEAR REGRESSION

Predict target *y* using multiple features, simultaneously.

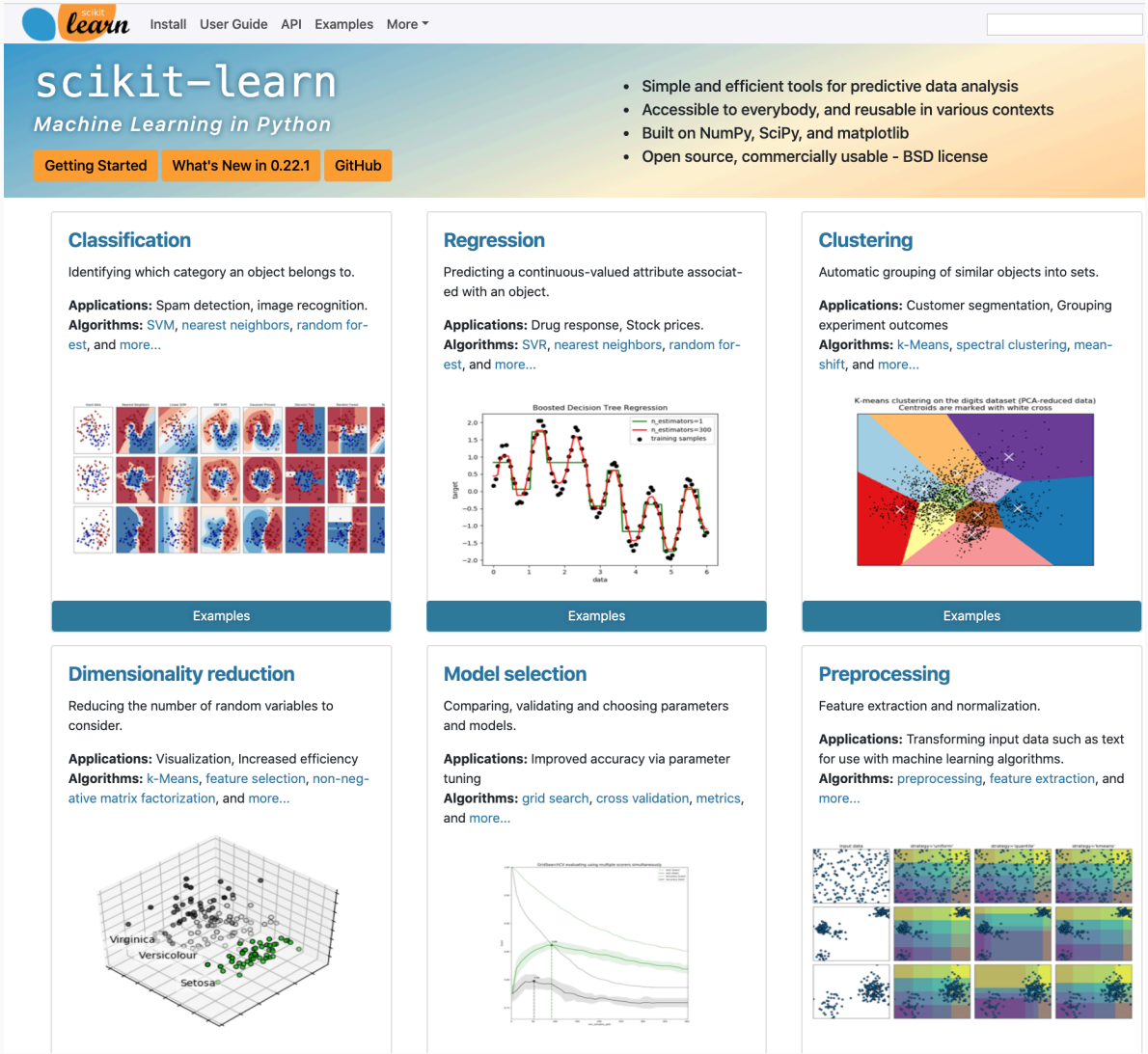
Motivating example: Predict diabetes progression in patients after 1 year based on health metrics. (Measured via numerical score.)

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

Demo in `demo_diabetes.ipynb`.

LIBRARIES FOR THIS DEMO

Introducing **Scikit Learn**.



SCIKIT LEARN

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:

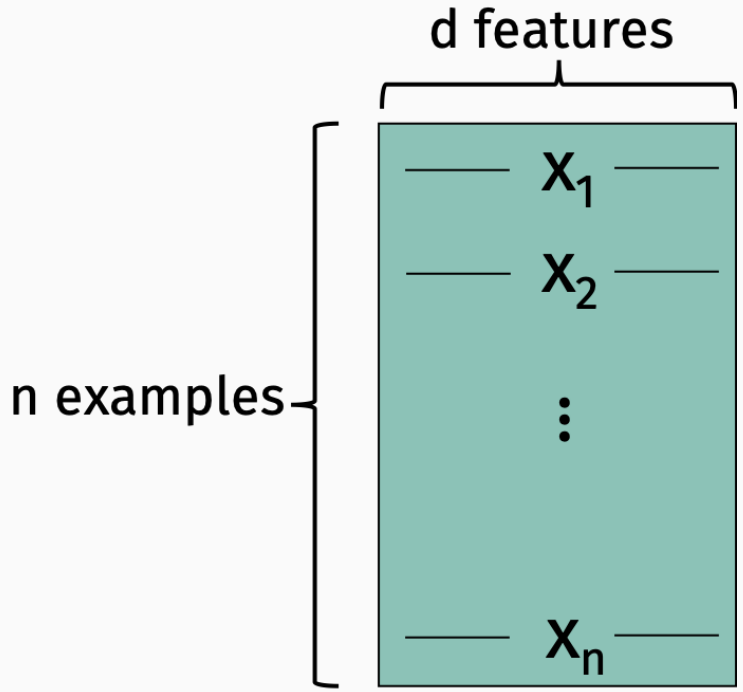
- `datasets` module contains a number of pre-loaded datasets. Saves time over downloading and importing with `pandas`.
- `linear_model` can be used to solve Multiple Linear Regression. A bit overkill for this simple model, but gives you an idea of `sklearn`'s general structure.

Target variable:

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

Predictor variables:

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

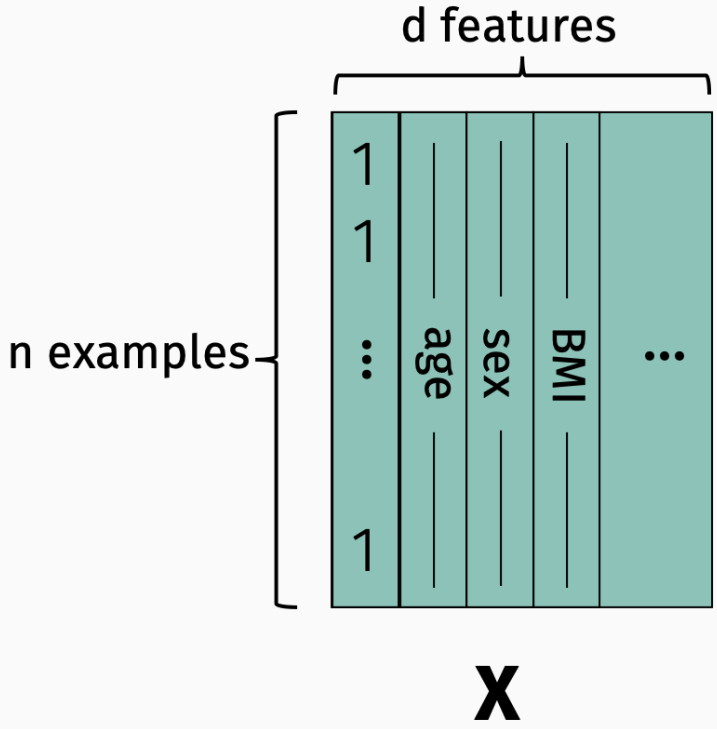


Target variable:

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

Predictor variables:

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



Data matrix indexing:

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

Multiple Linear Regression Model:

$$\text{Predict} \qquad y_i \approx \beta_0 + \beta_1 \mathbf{x}_{i1} + \beta_2 \mathbf{x}_{i2} + \ldots + \beta_d \mathbf{x}_{id}$$

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

Assume first columns contains all 1's. If it doesn't append on a column of all 1's.

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

Multiple Linear Regression Model:

$$\text{Predict} \qquad y_i \approx \beta_1 \mathbf{x}_{i1} + \beta_2 \mathbf{x}_{i2} + \ldots + \beta_d \mathbf{x}_{id}$$

Use as much linear algebra notation as possible!

- Model:

- Model Parameters:

- Loss Function:

Linear Least-Squares Regression.

- Model:

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

- Model Parameters:

$$\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_d]$$

- Loss Function:

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

LINEAR ALGEBRAIC FORM OF LOSS FUNCTION

Machine learning goal: minimize the loss function $L(\boldsymbol{\beta}) : \mathbb{R}^d \to \mathbb{R}$.

Find optimum by determining for which $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]$ all partial derivatives are 0. I.e. when do we have:

$$\begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \ldots \\ 0 \end{bmatrix}$$

THE ALL IMPORTANT GRADIENT

For any function $L(\boldsymbol{\beta}) : \mathbb{R}^d \to \mathbb{R}$, the gradient $\nabla L(\beta)$ is a function from $\mathbb{R}^d \to \mathbb{R}^d$ defined:

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

The gradient of the loss function is a central tool in machine learning. We will use it again and again.

Loss function:

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

Gradient:

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

Find optimum by determining for which $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]$ the gradient is 0. I.e. when do we have:

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

Goal: minimize the loss function $L(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$.

$$\begin{aligned}\nabla L(\boldsymbol{\beta}) &= -2 \cdot \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ &= 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^T\mathbf{y} = 0\end{aligned}$$

Solve for optimal $\boldsymbol{\beta}^*$:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}^* = \mathbf{X}^T\mathbf{y}$$

$$\boldsymbol{\beta}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

MULTIPLE LINEAR REGRESSION SOLUTION

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

- 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.

GRADIENT WARMUP

Function:

$$f(\mathbf{z}) = \mathbf{a}^T\mathbf{z} \text{ for some fixed vector } \mathbf{a} \in \mathbb{R}^d$$

Gradient:

Function:

$$f(\mathbf{z}) = \|\mathbf{z}\|_2^2$$

Gradient:

Loss function:

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

TEST YOUR INTUITION

Example from book: What is the sign of β_1 when we run a simple linear regression using the following predictors for number of sales in a particular market as a function of:

- Amount of TV advertising in that market:
- Amount of print advertising in that market:

INTERACTING VARIABLES

What is the sign of the corresponding β's when we run a multiple linear regression using the following predictors together:

- Amount of TV advertising in that market: Positive
- Amount of print advertising in that market: Negative, close to zero

Can you explain this? Try to think of your own example of a regression problem where this phenomenon might show up.

DEALING WITH CATEGORICAL VARIABLES

The sex variable in the diabetes problem was binary. We encoded it as 2 numbers – e.g. (0,1), (-1,1), (1,2).

Suppose we go back to the MPG prediction problem. What if we had 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?**

$$\begin{bmatrix} \texttt{ford} \\ \texttt{ford} \\ \texttt{honda} \\ \texttt{bmw} \\ \texttt{honda} \\ \texttt{ford} \end{bmatrix} \rightarrow \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

ONE HOT ENCODING

Better approach: One Hot Encoding.

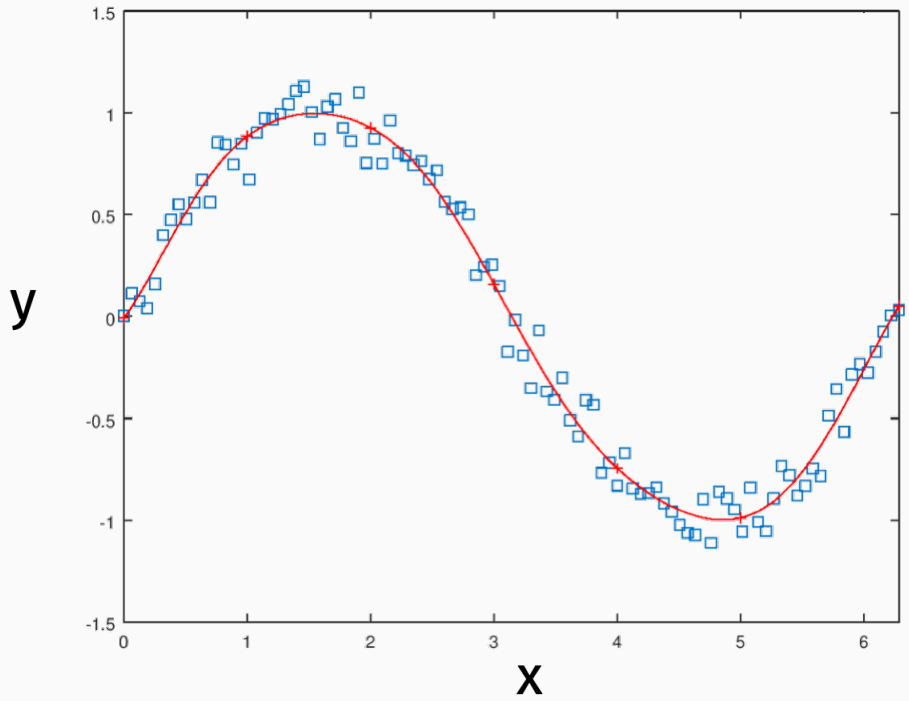
$$\begin{bmatrix} \texttt{ford} \\ \texttt{ford} \\ \texttt{honda} \\ \texttt{bmw} \\ \texttt{honda} \\ \texttt{ford} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

- 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.

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

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



Transform into a multiple linear regression problem:

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

Each column j is generated by a different basis function $\phi_j(x)$.

Could have:

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

Transformations can also be for multivariate data.

Example: Multinomial model.

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

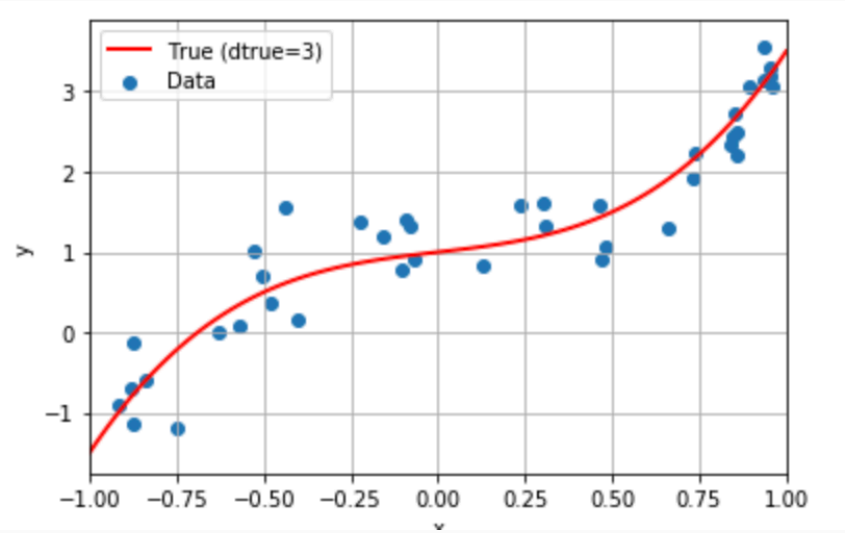
$$\begin{bmatrix} 1 & x_1 & x_1^2 & z_1 & z_1^2 & x_1 z_1 \\ 1 & x_2 & x_2^2 & z_2 & z_2^2 & x_2 z_2 \\ \vdots & \vdots & & \vdots & & \\ 1 & x_n & x_n^2 & z_n & z_n^2 & x_n z_n \end{bmatrix}$$

- Captures non-linear interaction between x and y.

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

Simple experiment:

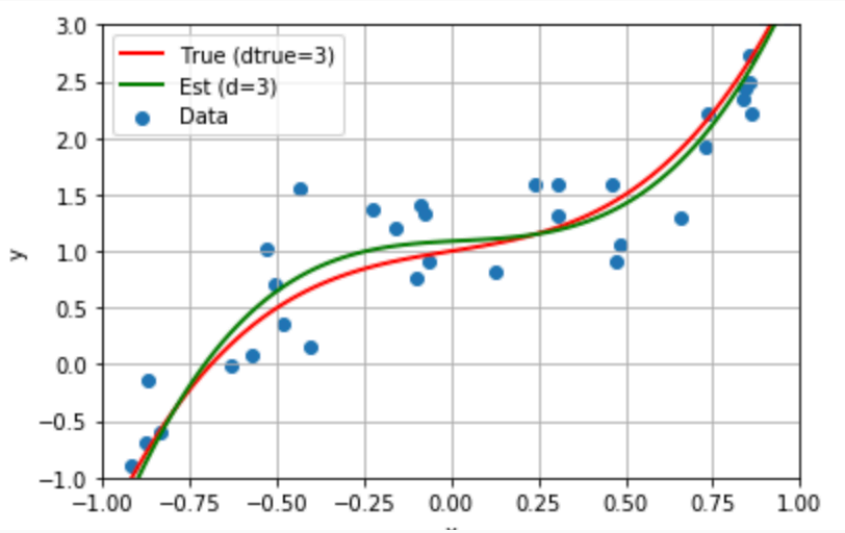
- Randomly select data points $x_1, \ldots, x_n \in [-1, 1]$.
- Choose a degree 3 polynomial $p(x)$.
- Create some fake data: $y_i = p(x_i) + \eta$ where η is a random number (e.g random Gaussian).



FITTING A POLYNOMIAL

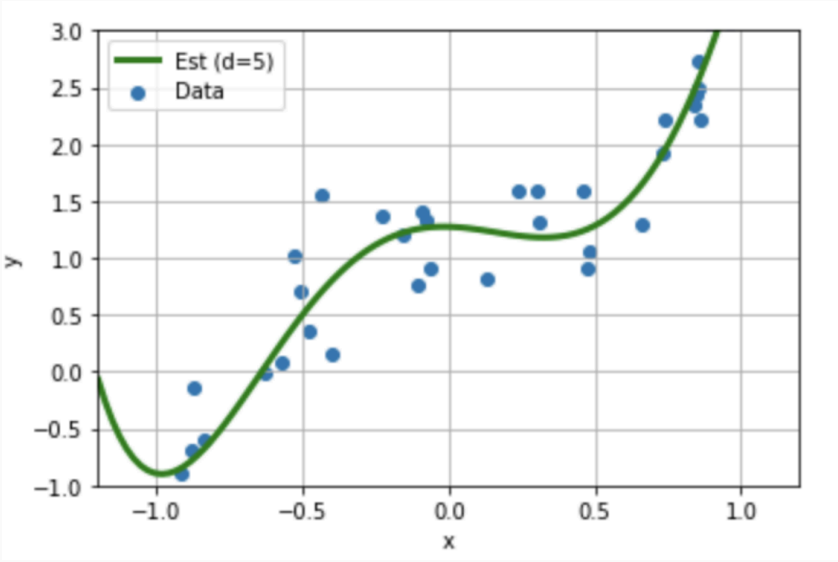
Simple experiment:

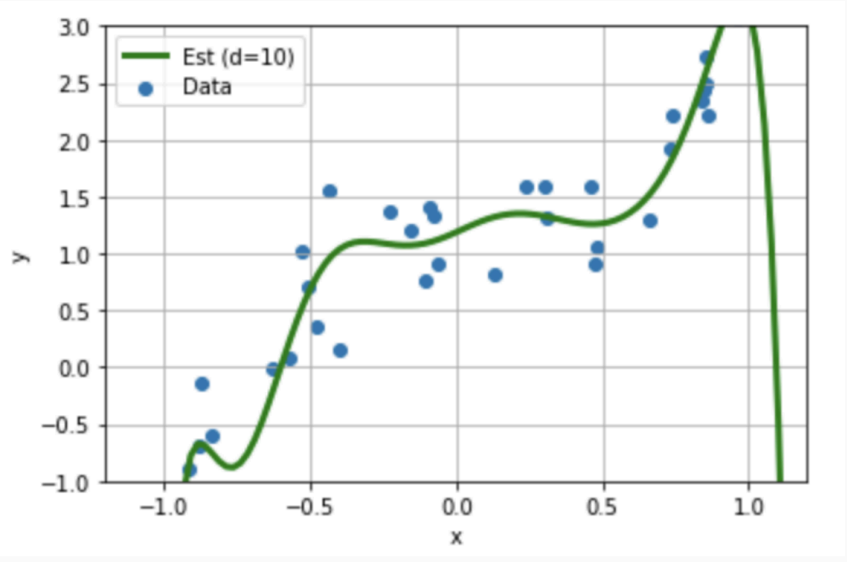
- Use multiple linear regression to fit a degree 3 polynomial.



What if we fit a higher degree polynomial?

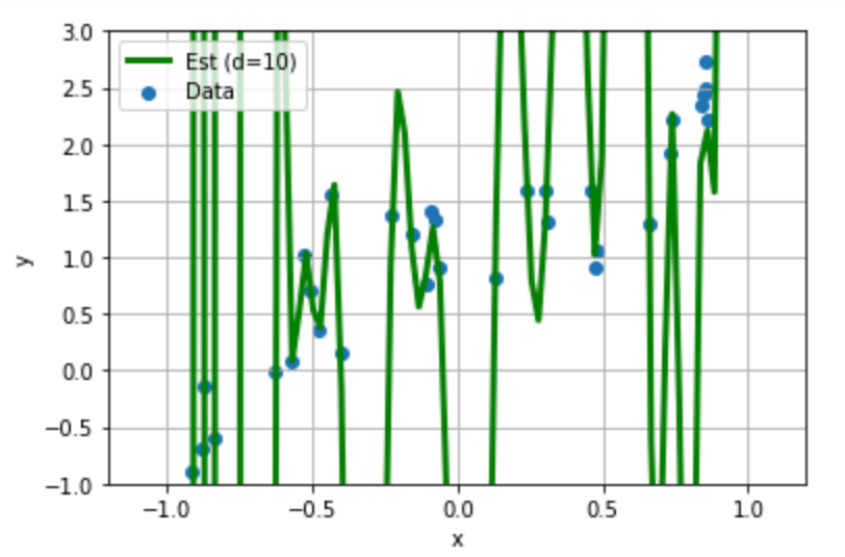
- Fit degree 5 polynomial under squared loss.
- Fit degree 10 polynomial under squared loss.



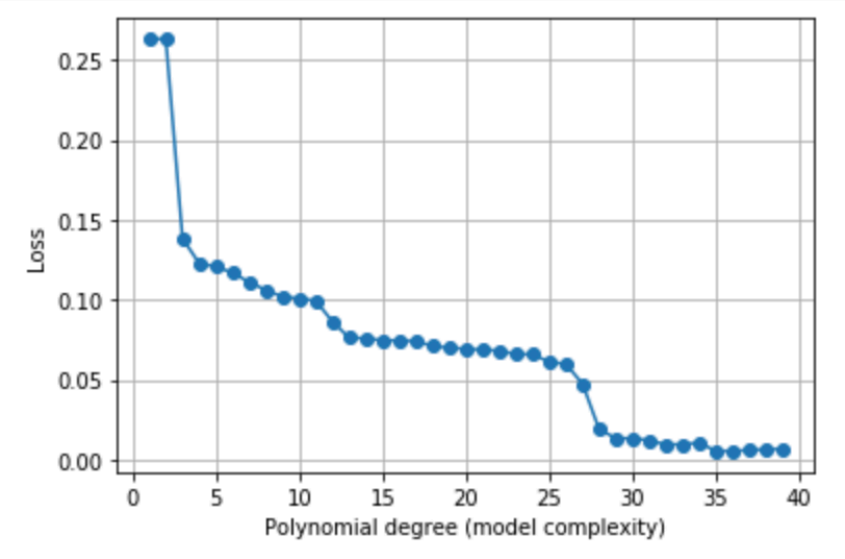


Even higher?

- Fit degree 40 polynomial under squared loss.



The more **complex** our model class (i.e. the higher degree we allow) the better our loss:

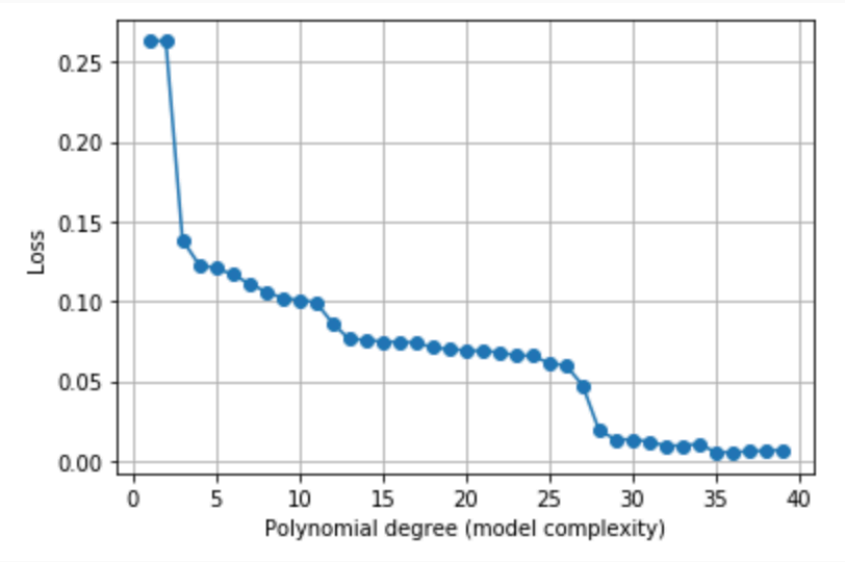


Is our model getting better and better?

Given the raw data, how do we know which model to choose?

Degree 3? Degree 5? Degree 40?

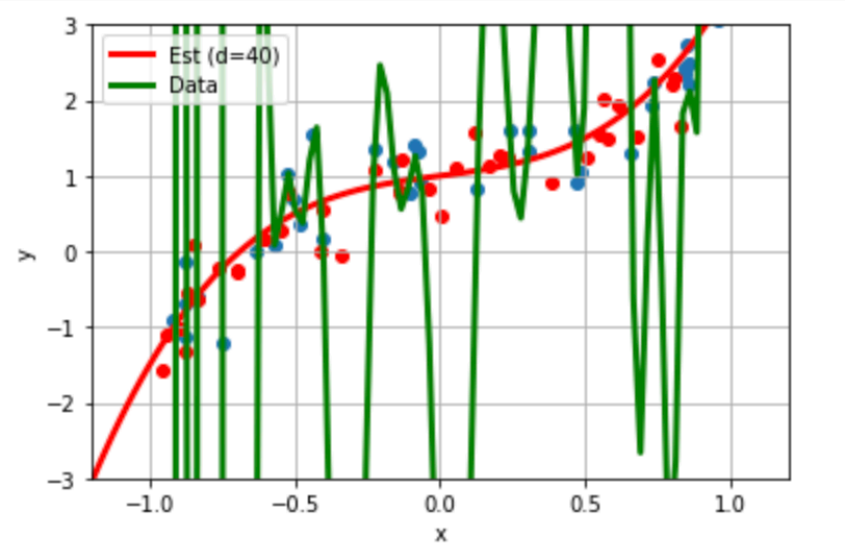
The more **complex** our model class the better our loss:



So training loss alone is not usually a good metric for model selection. Small loss does not imply generalization.

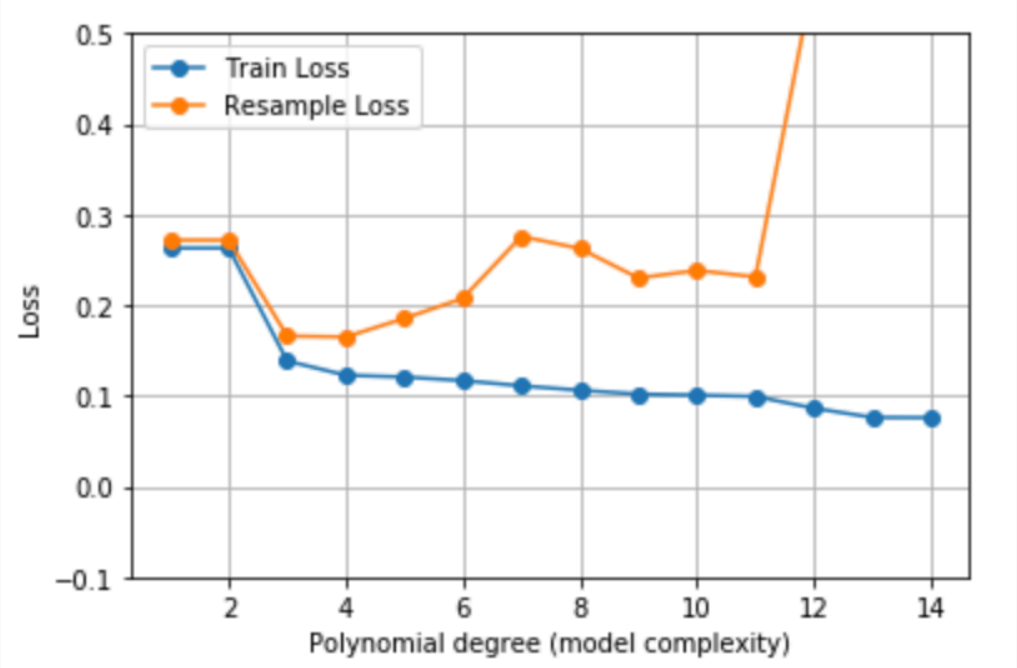
Problem: Loss alone is not informative for choosing model.

For more complex models, we get smaller loss on the training data, but don't expect to perform well on "new" data:



In other words, the model does not generalize.

Solution: Directly test model on "new data".



- Loss continues to decrease as model complexity grows.
- Performance on new data "turns around" once our model gets too complex. Minimized around degree 4.

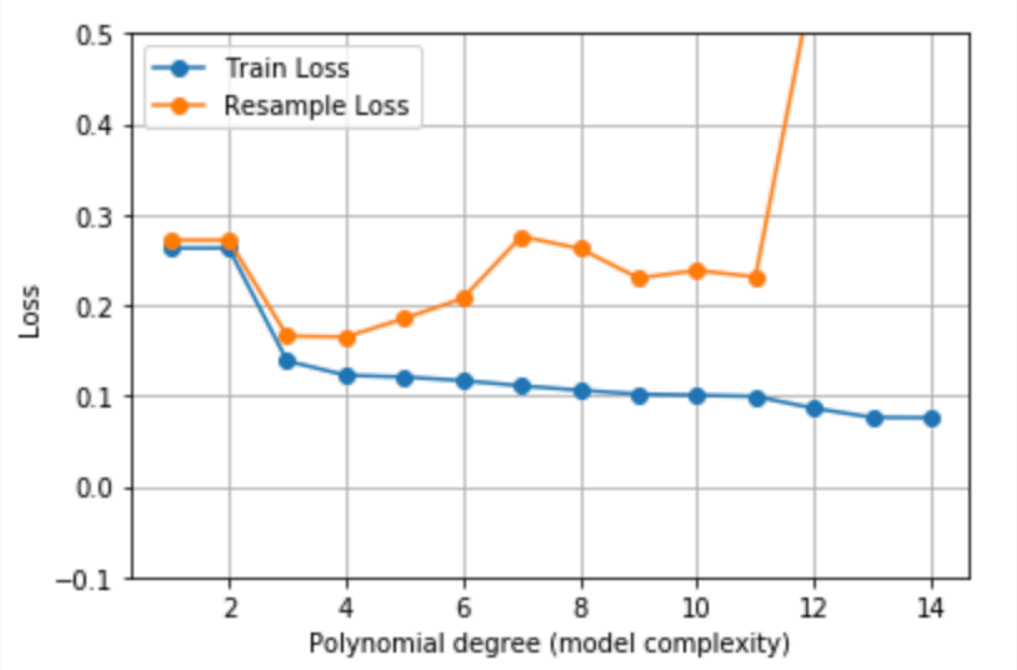
Better approach: Evaluate model on fresh test data which was not used during training.

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 $f^{(1)}, \ldots, f^{(q)}$ by finding parameters which minimize the loss on $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$.
- Evaluate loss of each trained model on $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$.

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.

TRAIN-TEST PARADIGM



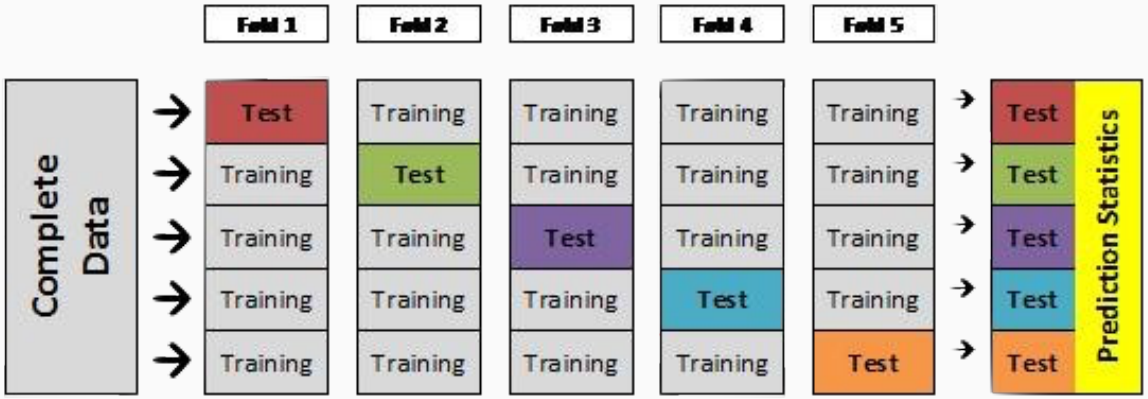
- **Train loss** continues to decrease as model complexity grows.
- **Test loss** "turns around" once our model gets too complex. Minimized around degree 3 – 4.

GENERALIZATION ERROR

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

Typical train-test split: 70-90% / 10-30%. Trade-off between 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.

Leave-one-out cross validation: take $K = n$, where n is our total number of samples.

Is there any disadvantage to choosing K larger?

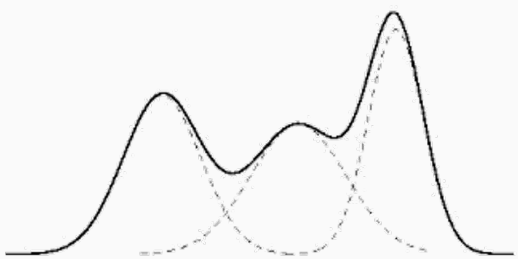
TRAIN-TEST INTUITION

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:

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



✓ ✓ ✓

This is not a simplifying assumptions! 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:

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

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

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

- **(Population) Risk**:

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

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

$$R_E(f, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L\left(f(\mathbf{x}, \boldsymbol{\theta}), y\right)$$

Minimizing training loss is the same as minimizing the empirical risk on the training data.

Often called empirical risk minimization.

For any fixed model f and parameters $\boldsymbol{\theta}$,

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

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

- Train q models $(f^{(1)}, \boldsymbol{\theta}_1^*), \ldots, (f^{(q)}, \boldsymbol{\theta}_q^*)$.
- For each model, compute empirical risk $R_E(f^{(i)}, \boldsymbol{\theta}_i^*)$ 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, $R_E(f^{(i)}, \boldsymbol{\theta}_i^*)$ is an unbiased estimate of the true risk $R(f^{(i)}, \boldsymbol{\theta}_i^*)$ for every i. Can use it to distinguish between models.

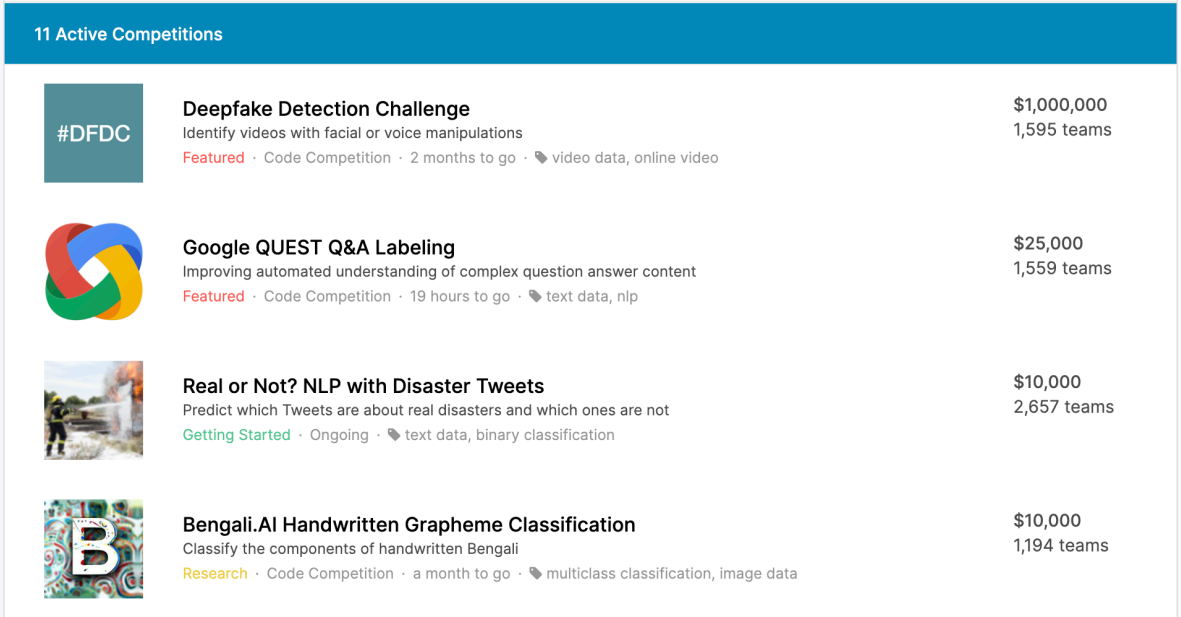
Slight caveat: This is typically not how machine learning or scientific discover works in practice!

Typical workflow:

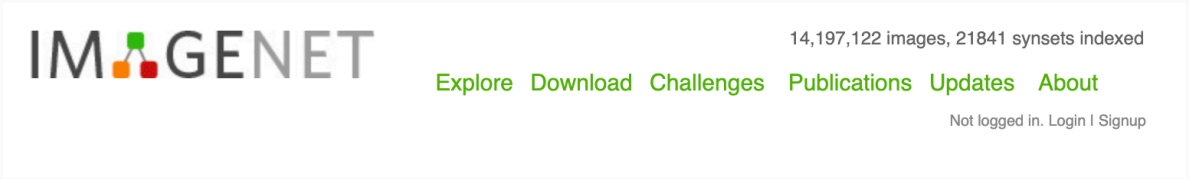
- Train a class of models.
- Test.
- Adjust class of models.
- Test.
- Adjust class of models.
- Cont...

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.



Kaggle (various competitions)



Imagenet (image classification and categorization)

ADAPTIVE DATA ANALYSIS

Is adaptivity a problem? Does it lead to over-fitting? How much? How can we prevent it? All current research.

REPORT

The reusable holdout: Preserving validity in adaptive data analysis

Cynthia Dwork[1,*], Vitaly Feldman[2,*], Moritz Hardt[3,*], Toniann Pitassi[4,*], Omer Reingold[5,*], Aaron Roth[6,*]
+ 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
Vaishaal Shankar UC Berkeley

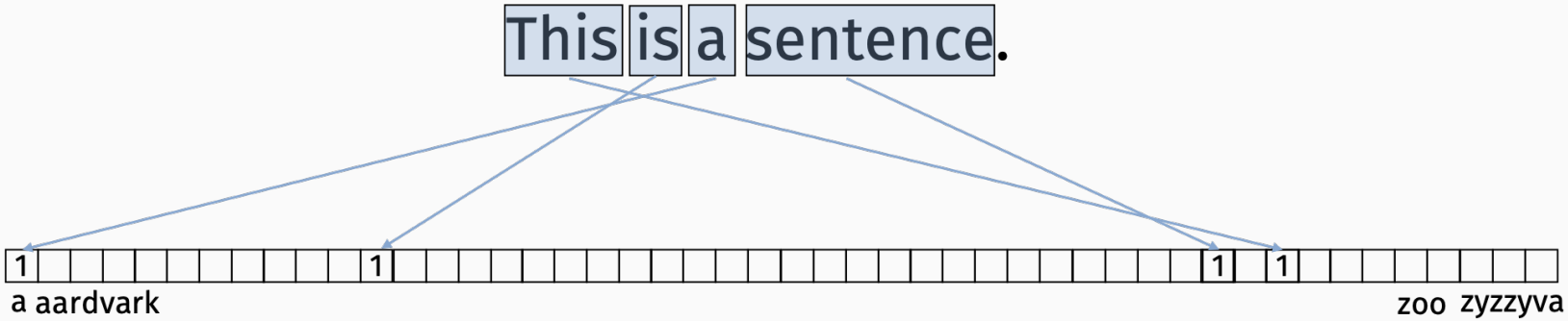
12 Jun 2019

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.

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

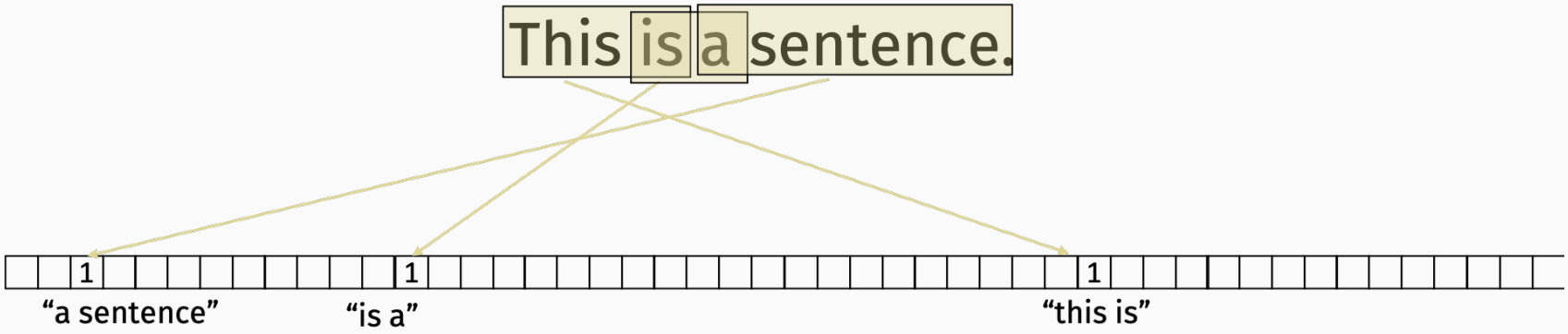
Common way to represent documents (emails, webpages, books) as numerical data. The ultimate example of 1-hot encoding.



bag-of-words

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

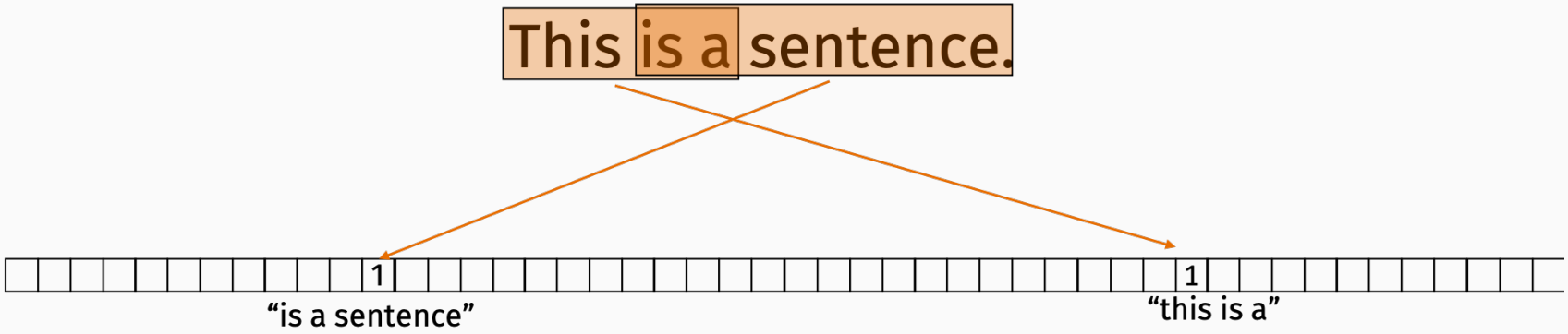
Common way to represent documents (emails, webpages, books) as numerical data. The ultimate example of 1-hot encoding.



bi-grams

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

Models of increasing order:

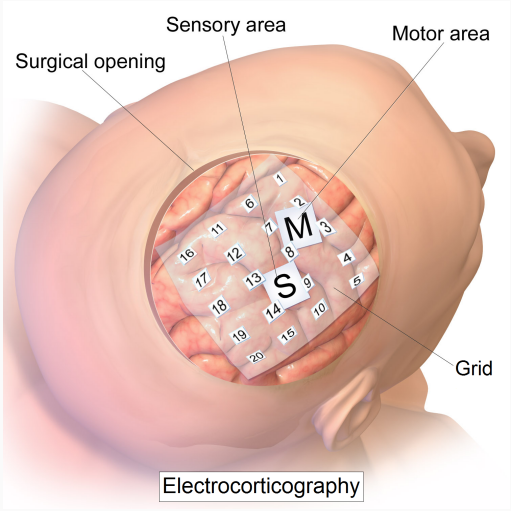
- Model $f^{(1)}_{\boldsymbol{\theta}_1}$: spam filter that looks at **single words**.
- Model $f^{(2)}_{\boldsymbol{\theta}_2}$: spam filter that looks at **bi-grams**.
- Model $f^{(3)}_{\boldsymbol{\theta}_3}$: spam filter that looks at **tri-grams**.
- ...

"interest" "low interest" "low interest loan"

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

Electrocorticography ECoG (upcoming 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 $t, t-1, \ldots, t-q$ for varying values of q.