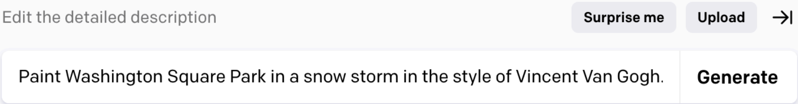
CS-GY 6923: Lecture 1
Introduction to Machine Learning

NYU Tandon School of Engineering, Prof. Christopher Musco

Who has tried ChatGPT? DALLE? Imagen?



Who ~~has~~ hasn't tried ChatGPT? Github Copilot? DALLE? Midjourney?

Can you write a bash script to resize all images in a directory to be less than 100kb?

Yes, here is an example bash script that uses the `mogrify` command from the `imagemagick` package to resize all images in a directory to be less than 100KB:

Copy code

```
#!/bin/bash

# specify the directory containing the images
dir="path/to/images"

# resize all images in the directory
mogrify -path $dir -resize x800 -quality 50 -define jpeg:extent=100kb
-format jpg $dir/*
```

This script takes a directory specified by the variable `dir` and uses the `mogrify` command to resize all images in the directory with the following options:

May look back on 2015-now as a the dawn of true artificial intelligence.

Times Topics: Artificial Intelligence

News about artificial intelligence, including commentary and archival articles published in The New York Times.

4h ago

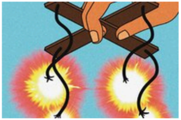
BUSINESS

A.I. Isn't Magic, but Can It Be 'Agentic'?

So-called agentic artificial intelligence doesn't require a human to tell it what to do. At least, that's the idea.

By Erin Griffith

PRINT EDITION
September 8, 2024



Aug. 30

TECHNOLOGY

How Do You Change a Chatbot's Mind?

When I set out to improve my tainted reputation with chatbots, I discovered a new world of A.I. manipulation.

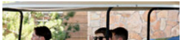
By Kevin Roose

PRINT EDITION How to Change a Chatbot's Mind | September 6, 2024, Page B1

Sept. 3

TECHNOLOGY

OpenAI, Still Haunted by Its Chaotic Past, Is



Other developments in recent years:

- Human-level image classification and understanding.
- Near perfect machine translation.
- Human level game play in very complex games (Go, Starcraft).
- Machine learning as a central tool in science.

What other technologies have caught people's eye?

Give you a foundation to understand the main ideas in modern machine learning.

We will do so through a combination of:

- Hands on implementation.
    - Demos and take-home labs using `Python` and `Jupyter notebooks`. 20% of grade
    - We will use `Google Colab` as the primary programming environment.
- Theoretical exploration.
    - Written problem sets. 20%
    - Midterm and final exam. 25% of grade each.

Goals of theoretical component:

1. Build experience with the most important mathematical tools used in machine learning, including probability, statistics, and linear algebra. This experience will prepare you for more advanced coursework in ML, or research.
2. Be able to understand contemporary research in machine learning, including papers from NeurIPS, ICML, ICLR, and other major machine learning venues.
3. Learn how theoretical analysis can help explain the performance of machine learning algorithms and lead to the design of entirely new methods.

Goals of hands-on component:

1. Reinforce theory learned in class, and make sure you understand algorithms described by implementing them.
2. Learn how to view and formulate real world problems in the language of machine learning.
3. Gain experience applying the most popular and successful machine learning algorithms to thse problems.

MORE ADVANCED/FOCUSED CLASSES AT TANDON

- CS-GY 6953: **Deep Learning** (Prof. Chinmay Hegde)
- CS-GY 6943: **Artificial Intelligence for Games** (Prof. Julian Togelius)
- ECE-GY 9163: **Machine Learning for Cybersecurity** (Prof. Siddharth Garg)
- ECE-GY 7143: **Advanced Machine Learning** (Prof. Anna Chromanska)
- CS-GY 6763: **Algorithmic Machine Learning and Data Science** (me)
- Keep your eyes out for special topics courses.

All class information can be found at:

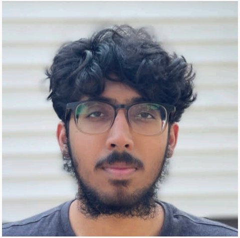
`www.chrismusco.com/machinelearning2024_grad`

Make sure you will be around for midterm (10/18) and final (12/20).

1. Make sure you are signed into and follow Ed discussion, which will be used for all classroom communication (no email). Now integrated into Brightspace.
2. Don't hesitate to ask me or the TAs for help.[1]

Prajjwal Bhattarai

Marc Chiu



Usaid Malik

[1] Fill out office hours poll on Ed!

Collaboration: Students may discuss problem set problems and coding assignments. However, you must write their solutions independently.

We have a zero-tolerance policy for copied solutions. Do not let other students copy off your work or risk a zero on the assignment.

AI tools: You can use AI tools like ChatGPT as you wish, just make sure they do not impede your learning.

Class participation accounts for 10% of your grade. It's easy to get a perfect score:

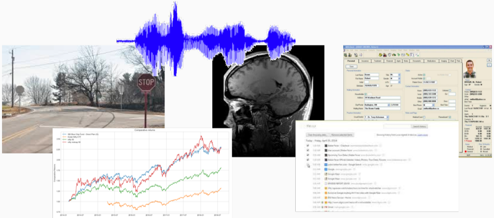
- Ask and answer questions in lecture.
- Post questions or responses to other students on Ed. Or other things you find interesting.
- Participate in professor or TA office hours.

THE PREDICTION PROBLEM

BASIC GOAL

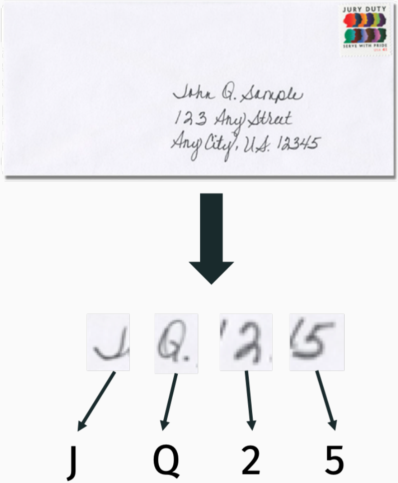
Goal: Develop algorithms to make predictions based on data.

- **Input:** A single piece of data (an image, audio file, patient healthcare record, MRI scan, beginning of a sentence).



- **Output:** A prediction (this image is a stop sign, this stock will go up 10% next quarter, this song is in French, the next word in the sentence is "tomorrow").

Optical character recognition (OCR): Decide if a handwritten character is an $a, b, \ldots, z, 0, 1, \ldots, 9, \ldots$.



Optical character recognition (OCR): Decide if a handwritten character is an $a, b, \ldots, z, 0, 1, \ldots, 9, \ldots$.

Applications:

- Automatic mail sorting.
- Text search in handwritten documents.
- Digitizing scanned books.
- License plate detection for tolls.
- Precursor to translation from images.

How would you write an **code** to distinguish these digits?

Suppose you just want to distinguish <u>between a 1 and a 7</u>.

Reasonable approach: A number which contains one vertical line is a 1, if it contains one vertical and one horizontal line, it's a 7.

```
def count_vert_lines(image):
...

def count_horiz_lines(image):
...

def classify(image):
...
    nv = count_vert_lines(image)
    nh = count_vert_lines(image)

    if (nv == 1) and (nh == 1):
        return '7'
    elif (nv == 1) and (nh == 0):
        return '1'
    elif ...
```

This rule breaks down in practice:

Even fixes/modifications of the rule tend to be brittle... Maybe you could get 80% accuracy, but not nearly good enough.

CHALLENGE OF EXPERT SYSTEMS

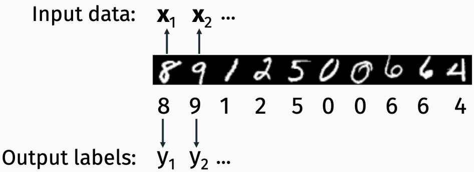
Rule based systems, also called Expert Systems were the dominant approach to artificial intelligence in the 1970s and 1980s. Still widely used (e.g., in Amazon Alexa and similar systems).

Major limitation: While human's are very good at many tasks,

- It's often hard to encode why humans make decisions in simple programmable logic.
- We think in abstract concepts with no mathematical definitions (how exactly do you define a line? how do you define a curve? straight line?)

Focus on what humans do well: solving the task at hand!

Step 1: Collect and label many input/output pairs $(\mathbf{x}_i, y_i)$. For our digit images, we have each $\mathbf{x}_i \in \mathbb{R}^{28\times 28}$ and $y_i \in \{0, 1, \ldots, 9\}$.



This is called the training dataset.

Step 2: Learn from the examples we have.

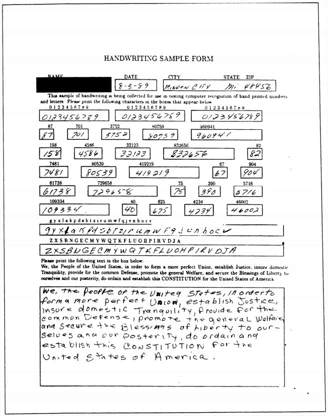
- Have the computer automatically find some function $f(\mathbf{x})$ such that $f(\mathbf{x}_i) = y_i$ for most $(\mathbf{x}_i, y_i)$ in our training data set (by searching over many possible functions).

Think of f as any crazy equation, or an arbitrary program:

$$f(\mathbf{x}) = 10 \cdot x[1,1] - 6 \cdot x[3,45] \cdot x[9,99] + 5 \cdot \text{mean}(\mathbf{x}) + \ldots$$

This approach of learning a function from labeled data is called **supervised learning.**

National Institute for Standards and Technology collected a huge amount of handwritten digit data from census workers and high school students in the early 90s:



HANDWRITING SAMPLE FORM

NAME [redacted] DATE 8-3-89 CITY MINDEN CITY STATE MI ZIP 48456

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below

0123456789 0123456789 0123456789

87 701 3752 80759 960941

158 4586 32123 832656 82

7481 80539 419219 67 904

61738 729658 75 390 5716

109334 40 625 4234 46002

gyxlakpdsbtzirumwfqjenhocv

ZXSBNGECMYWQTKFLUOHPIRVDJA

Please print the following text in the box below:

We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America

This is called the NIST dataset, and was used to create the famous MNIST handwritten digit dataset.

Since the 1990s machine learning have overtaken expert systems as the dominant approach to artificial intelligence.

- Current methods achieve .17% error rate for OCR on benchmark datasets (MNIST).[2]
- Very successful on other problems as well. The big break through for supervised learning in the 2010s was image classification.

[2]Not because of overfitting! See: *Cold Case: The Lost MNIST Digits* by Chhavi Yadav + Léon Bottou.

CENTRAL QUESTIONS IN SUPERVISED MACHINE LEARNING

Once we have the basic supervised machine learning setup, many very difficult questions remain:

- How do we parameterize a class of functions f to search?
- How do we efficiently find a good function in the class?
- How do we ensure that an $f(\mathbf{x})$ which works well on our training data will generalize to perform well on future data?
- How do we deal with imperfect data (noise, outliers, incorrect training labels)?

Recall that in the supervised learning setup every input $\mathbf{x}_i$ in our training dataset comes with a desired output y_i (typically generated by a human, or some other process).

Types of supervised earning:

- **Classification** – predict a discrete class label.
- **Regression** – predict a continuous value.
  - Dependent variable, response variable, target variable, lots of different names for y_i.

Another example of supervised classification: **Face Detection**.

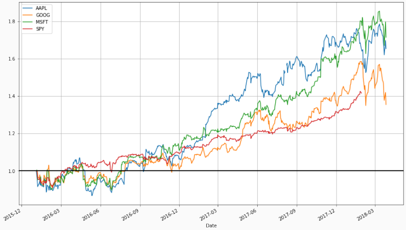
Each input data example $\mathbf{x}_i$ is an image. Each output y_i is 1 if the image contains a face, 0 otherwise.

- Harder than digit recognition, but we now have essentially perfect methods (used in nearly all digital cameras, phones, etc.)

Other examples of supervised classification:

- Object detection (Input: image, Output: dog or cat)
- Spam detection (Input: email text, Output: spam or not)
- Medical diagnosis (Input: patient data, Output: disease condition or not)
- Credit decision making (Input: financial data, Output: offer loan or not)

Example of supervised regression: **Stock Price Prediction**.



Each input $\mathbf{x}$ is a vector of metrics about a company (sales volume, PE ratio, earning reports, historical price data).

Each output y_i is the **price of the stock** 3 months in the future.

Other examples of supervised regression:

- Home price prediction (Inputs: square footage, zip code, number of bathrooms, Output: Price)
- Car price prediction (Inputs: make, model, year, miles driven, Output: Price)
- Weather prediction (Inputs: weather data at nearby stations, Output: tomorrows temperature)
- Robotics/Control (Inputs: information about environment and current position at time t, Output: estimate of position at time $t+1$)

Later in the class we will talk about other frameworks:

- **Unsupervised learning** (no labels or response variable)
  - Important for representation learning and generative ML.
- **Self-supervised learning**.
  - Taking over the world. What Language Models like the GPT models are based on.

Focus less in this class on:

- **Reinforcement learning**
  - Game playing.
- **Active-learning.**
  - The learning algorithms can request labels.

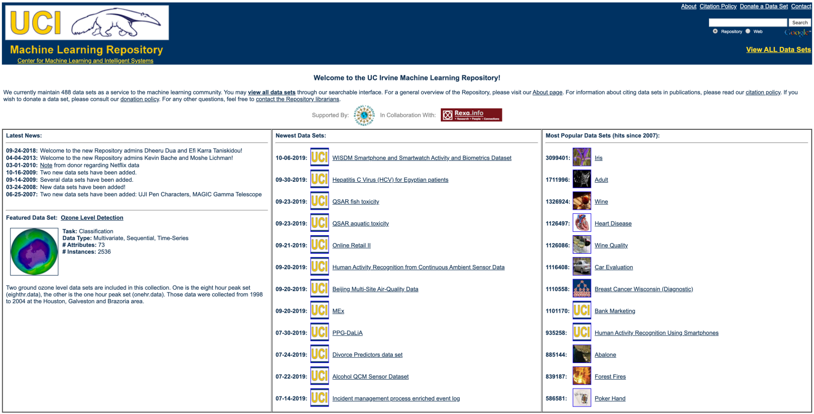
Types of supervised learning:

- **Classification** – predict a discrete class label.
- **Regression** – predict a continuous value.
    - Dependent variable, response variable, target variable, lots of different names for y_i.

Motivating example: Predict the highway miles per gallon (MPG) of a car given quantitative information about its engine. Demo in `demo_auto_mpg.ipynb`.

What factors might matter?

Data set available from the UCI Machine Learning Repository: https://archive.ics.uci.edu/.



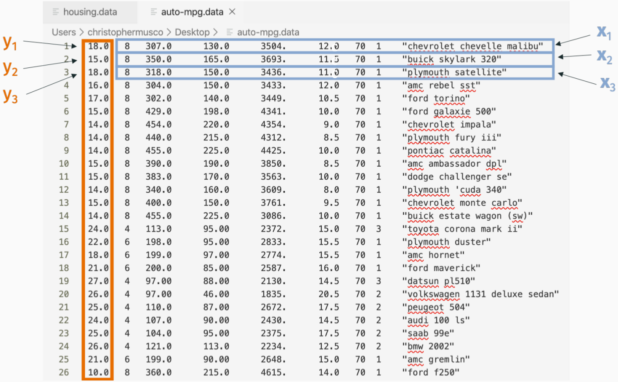
Datasets from UCI (and many other places) comes as tab, space, or comma delimited files.

housing.data   auto-mpg.data

Users > christophermusco > Desktop > auto-mpg.data

```
18.0   8   307.0      130.0      3504.      12.0   70  1	"chevrolet chevelle malibu"
15.0   8   350.0      165.0      3693.      11.5   70  1	"buick skylark 320"
18.0   8   318.0      150.0      3436.      11.0   70  1	"plymouth satellite"
16.0   8   304.0      150.0      3433.      12.0   70  1	"amc rebel sst"
17.0   8   302.0      140.0      3449.      10.5   70  1	"ford torino"
15.0   8   429.0      198.0      4341.      10.0   70  1	"ford galaxie 500"
14.0   8   454.0      220.0      4354.       9.0   70  1	"chevrolet impala"
14.0   8   440.0      215.0      4312.       8.5   70  1	"plymouth fury iii"
14.0   8   455.0      225.0      4425.      10.0   70  1	"pontiac catalina"
15.0   8   390.0      190.0      3850.       8.5   70  1	"amc ambassador dpl"
15.0   8   383.0      170.0      3563.      10.0   70  1	"dodge challenger se"
14.0   8   340.0      160.0      3609.       8.0   70  1	"plymouth 'cuda 340"
15.0   8   400.0      150.0      3761.       9.5   70  1	"chevrolet monte carlo"
14.0   8   455.0      225.0      3086.      10.0   70  1	"buick estate wagon (sw)"
24.0   4   113.0      95.00      2372.      15.0   70  3	"toyota corona mark ii"
22.0   6   198.0      95.00      2833.      15.5   70  1	"plymouth duster"
18.0   6   199.0      97.00      2774.      15.5   70  1	"amc hornet"
21.0   6   200.0      85.00      2587.      16.0   70  1	"ford maverick"
27.0   4   97.00      88.00      2130.      14.5   70  3	"datsun pl510"
26.0   4   97.00      46.00      1835.      20.5   70  2	"volkswagen 1131 deluxe sedan"
25.0   4   110.0      87.00      2672.      17.5   70  2	"peugeot 504"
24.0   4   107.0      90.00      2430.      14.5   70  2	"audi 100 ls"
25.0   4   104.0      95.00      2375.      17.5   70  2	"saab 99e"
26.0   4   121.0      113.0      2234.      12.5   70  2	"bmw 2002"
21.0   6   199.0      90.00      2648.      15.0   70  1	"amc gremlin"
10.0   8   360.0      215.0      4615.      14.0   70  1	"ford f250"
```

Check dataset description to know what each column means.

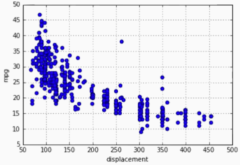


1	18.0	8	307.0	130.0	3504.	12.0	70	1	"chevrolet chevelle malibu"
2	15.0	8	350.0	165.0	3693.	11.5	70	1	"buick skylark 320"
3	18.0	8	318.0	150.0	3436.	11.0	70	1	"plymouth satellite"
4	16.0	8	304.0	150.0	3433.	12.0	70	1	"amc rebel sst"
5	17.0	8	302.0	140.0	3449.	10.5	70	1	"ford torino"
6	15.0	8	429.0	198.0	4341.	10.0	70	1	"ford galaxie 500"
7	14.0	8	454.0	220.0	4354.	9.0	70	1	"chevrolet impala"
8	14.0	8	440.0	215.0	4312.	8.5	70	1	"plymouth fury iii"
9	14.0	8	455.0	225.0	4425.	10.0	70	1	"pontiac catalina"
10	15.0	8	390.0	190.0	3850.	8.5	70	1	"amc ambassador dpl"
11	15.0	8	383.0	170.0	3563.	10.0	70	1	"dodge challenger se"
12	14.0	8	340.0	160.0	3609.	8.0	70	1	"plymouth 'cuda 340"
13	15.0	8	400.0	150.0	3761.	9.5	70	1	"chevrolet monte carlo"
14	14.0	8	455.0	225.0	3086.	10.0	70	1	"buick estate wagon (sw)"
15	24.0	4	113.0	95.00	2372.	15.0	70	3	"toyota corona mark ii"
16	22.0	6	198.0	95.00	2833.	15.5	70	1	"plymouth duster"
17	18.0	6	199.0	97.00	2774.	15.5	70	1	"amc hornet"
18	21.0	6	200.0	85.00	2587.	16.0	70	1	"ford maverick"
19	27.0	4	97.00	88.00	2130.	14.5	70	3	"datsun pl510"
20	26.0	4	97.00	46.00	1835.	20.5	70	2	"volkswagen 1131 deluxe sedan"
21	25.0	4	110.0	87.00	2672.	17.5	70	2	"peugeot 504"
22	24.0	4	107.0	90.00	2430.	14.5	70	2	"audi 100 ls"
23	25.0	4	104.0	95.00	2375.	17.5	70	2	"saab 99e"
24	26.0	4	121.0	113.0	2234.	12.5	70	2	"bmw 2002"
25	21.0	6	199.0	90.00	2648.	15.0	70	1	"amc gremlin"
26	10.0	8	360.0	215.0	4615.	14.0	70	1	"ford f250"

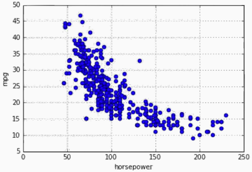
'mpg', 'cylinders','displacement', 'horsepower', 'weight', 'acceleration', 'model year', 'origin', 'car name'

- Use `pandas` for reading data from delimited files. Stores data in a type of table called a "data frame" but this is just a wrapper around a `numpy` array.
- Use `matplotlib` for initial exploration.

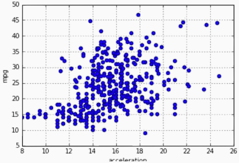
mpg



Displacement



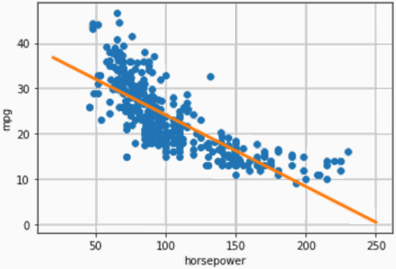
Horsepower



Acceleration

SIMPLE LINEAR REGRESSION

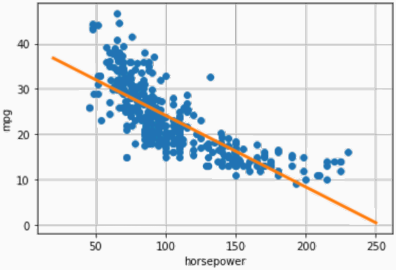
Linear regression from a Machine Learning (not a Statistics) perspective. Our first supervised machine learning model.



Only focus on <u>one predictive variable</u> at a time (e.g. horsepower). This is why it's called <u>simple</u> linear regression.

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)



- **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 knobs which parameterize our class of models.
- **Loss Function $L(\boldsymbol{\theta})$:** Measure of how well a model fits our data. Often some function of $f_{\boldsymbol{\theta}}(x_1) - y_1, \ldots, f_{\boldsymbol{\theta}}(x_n) - y_n$

Common Goal: Choose parameters $\boldsymbol{\theta}^*$ which minimize the Loss Function:

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

Choosing $\boldsymbol{\theta}^*$ based on minimizing the empirical error on our training data is called Empirical Risk Minimization. It is by far the most common approach to solving supervised learning problems.

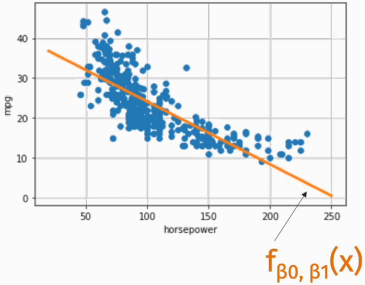
General Supervised Learning

- Model: $f_{\boldsymbol{\theta}}(x)$
- Model Parameters: $\boldsymbol{\theta}$
- Loss Function: $L(\boldsymbol{\theta})$

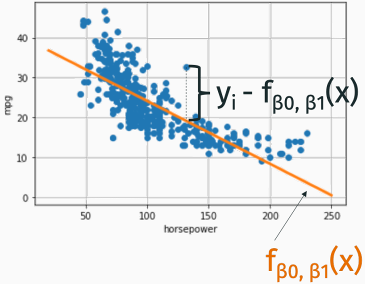
Linear Regression

- Model:
- Model Parameters:
- Loss Function:

What is a natural **loss function** for linear regression?

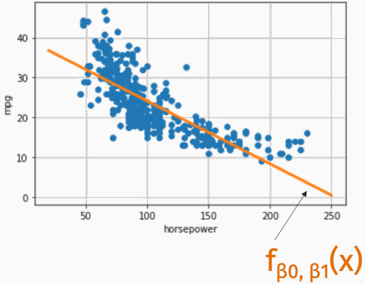


Typical choices are a function of $y_1 - f_{\beta_0,\beta_1}(x_1), \ldots, y_n - f_{\beta_0,\beta_1}(x_n)$



- ℓ_2/**Squared Loss:** $L(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - f_{\beta_0,\beta_1}(x_i))^2$.
- ℓ_1/**Least absolute deviation:** $L(\beta_0, \beta_1) = \sum_{i=1}^{n} |y_i - f_{\beta_0,\beta_1}(x_i)|$.
- ℓ_∞ **Loss** $L(\beta_0, \beta_1) = \max_{i \in 1,\ldots,n} |y_i - f_{\beta_0,\beta_1}(x_i)|$.

We're going to start with the Squared Loss/Sum-of-Squares Loss. Also called "Residual Sum-of-Squares (RSS)"



- Relatively robust to outliers.
- Simple to define, leads to simple algorithms for finding β_0, β_1
- Theoretically justified from classical statistics related to assumptions about Gaussian noise. Will discuss later in the course.

General Supervised Learning

- Model: $f_{\boldsymbol{\theta}}(x)$
- Model Parameters: $\boldsymbol{\theta}$
- Loss Function: $L(\boldsymbol{\theta})$

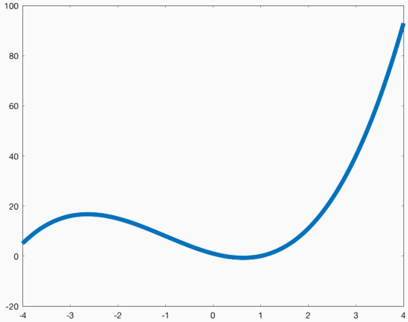
Linear Regression

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

This is the entire job of any Supervised Learning Algorithm.

Univariate function:



$$x^3 + 3 \cdot x^2 - 5 \cdot x + 1$$

- Find all places where <u>derivative</u> $f'(x) = 0$ and check which has the smallest value.

Multivariate function: $L(\beta_0, \beta_1)$

- Find values of β_0, β_1 where <u>all</u> partial derivatives equal 0.
- $\frac{\partial L}{\partial \beta_0} = 0$ and $\frac{\partial L}{\partial \beta_1} = 0$.

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

- Find values of β_0, β_1 where all partial derivatives equal 0.
- $\frac{\partial L}{\partial \beta_0} = 0$ and $\frac{\partial L}{\partial \beta_1} = 0$.

Some definitions:

- Let $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. $\bar{y}$ is the mean of y.
- Let $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. $\bar{x}$ is the mean of x.
- Let $\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$. σ_y^2 is the variance of y.
- Let $\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$. σ_x^2 is the variance of x.
- Let $\sigma_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$. σ_{xy} is the covariance.

Claim: $L(\beta_0, \beta_1)$ is minimized at:

- $\beta_1^* = \sigma_{xy}/\sigma_x^2$
- $\beta_0^* = \bar{y} - \beta_1\bar{x}$

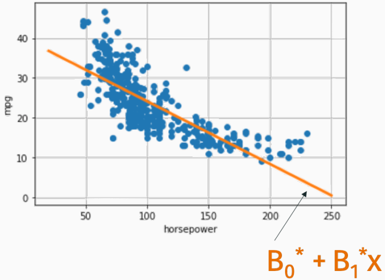
Loss function: $L(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$

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

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

Takeaways:

- Minimizing functions exactly is sometimes easy with calculus, but not always! We will learn much more general tools (like gradient descent).
- Simple closed form formula for optimal parameters β_0^* and β_1^* for squared-loss!



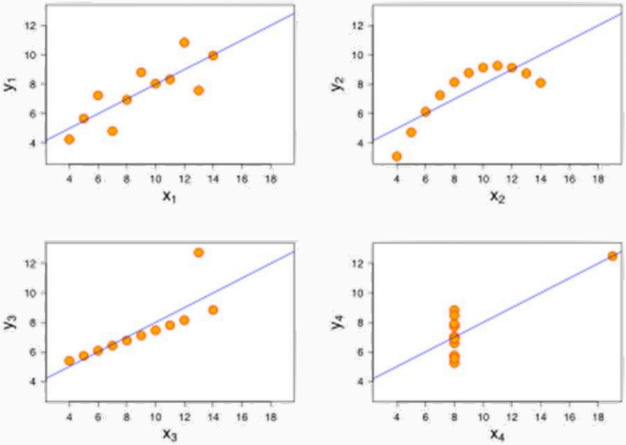
Let $L(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$.

$$R^2 = 1 - \frac{L(\beta_0, \beta_1)}{n\sigma_y^2}$$

is exactly the R^2 value ("coefficient of determination") you may remember from statistics.

The smaller the loss, the closer R^2 is to 1, which means we have a better regression fit.

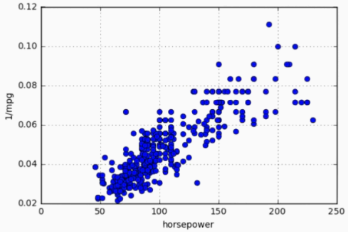
Many reasons you might get a poor regression fit:

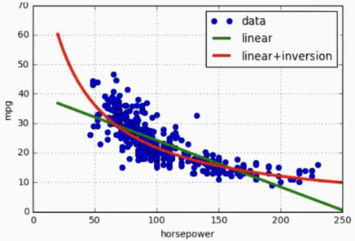


Some of these are fixable!

- Remove outliers, use more robust loss function.
- Non-linear model transformation.

Fit the model $\frac{1}{\text{mpg}} \approx \beta_0 + \beta_1 \cdot \text{horsepower}$.





Much better fit, same exact learning algorithm!

MULTIPLE LINEAR REGRESSION

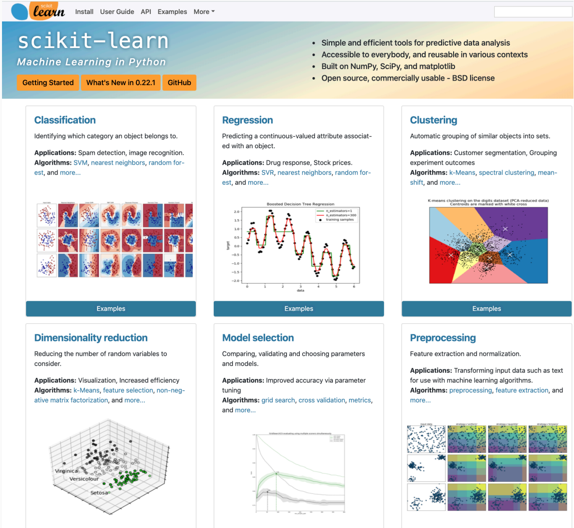
Predict target y using multiple features, simultaneously.

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

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

Demo in `demo_diabetes.ipynb`.

Introducing **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:

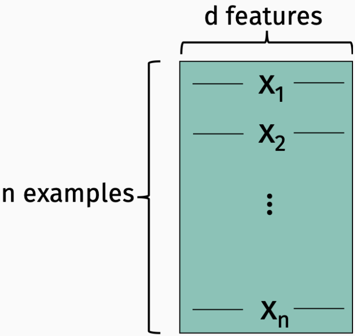
- `datasets` module contains a number of pre-loaded datasets. Saves time over downloading and importing with `pandas`.
- `linear_model` can be used to solve Multiple Linear Regression. A bit overkill for this simple model, but gives you an idea of `sklearn`'s general structure.

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



Now is 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 $\mathbf{d}$.
- $\mathbf{XD}$ scales the columns of $\mathbf{X}$. $\mathbf{DX}$ scales the rows.

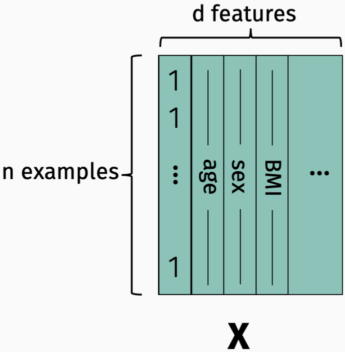
You also need to be comfortable working with matrices in `numpy` . Go through the `demo_numpy_matrices.ipynb` slowly.

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:

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

Multiple Linear Regression Model:

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

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

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

Multiple Linear Regression Model:

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

In this case, β_1 serves as the "intercept" parameter.

Use as much linear algebra notation as possible!

- Model Parameters:
- Model:
- Loss Function:

Linear Least-Squares Regression.

- Model Parameters:

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

- Model:

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

- Loss Function:

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

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

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

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

The vector of partial derivatives of $L(\boldsymbol{\beta})$ is called the <u>gradient</u> of $L(\boldsymbol{\beta})$, denoted by $\nabla L(\boldsymbol{\beta})$.

Loss function:

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

Gradient:

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

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

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

Loss function:

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

Loss function:

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

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

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

- 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.

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

$$
\begin{aligned}
x_1 &= [42, 4, 104, \texttt{hybrid}, \texttt{ford}] \\
x_2 &= [18, 8, 307, \texttt{gas}, \texttt{bmw}] \\
x_2 &= [31, 4, 150, \texttt{gas}, \texttt{honda}] \\
&\vdots
\end{aligned}
$$

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

$$
\begin{aligned}
x_1 &= [42, 4, 104, \texttt{hybrid}, \texttt{ford}] \\
x_2 &= [18, 8, 307, \texttt{gas}, \texttt{bmw}] \\
x_2 &= [31, 4, 150, \texttt{gas}, \texttt{honda}] \\
&\vdots
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= [42, 4, 104, \texttt{1}, \texttt{ford}] \\
x_2 &= [18, 8, 307, \texttt{0}, \texttt{bmw}] \\
x_2 &= [31, 4, 150, \texttt{0}, \texttt{honda}] \\
&\vdots
\end{aligned}
$$

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?**

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

Better approach: <u>One Hot Encoding.</u>

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

- 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.