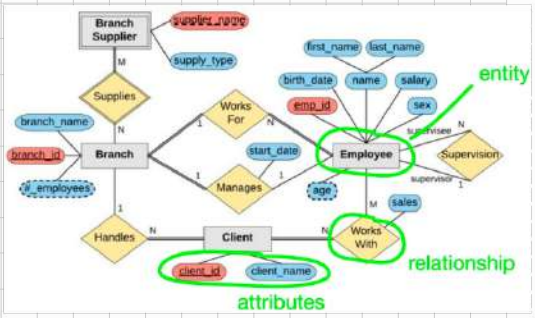
Lecture 2: Handling data:

1) Data Model:

- A data model specifies how we think about the world => which aspects do we care about!
- Modelled with an entity-relationship diagram. Abstract!
- Connect and interact with each other and shows modelling thought.
- Abstract: not tells us how to keep data but a general overview!



Models:

1.1) Flat Model: ▶ Simple: one type of entity => all w/ same attributes. ▶ No relations => only a text file. ▶ Like a mess => we're the ones to figure it out. ▶ CSV, log files.

1.2) Relational model: ▶ Entities w/ relationships. ▶ Ubiquitous (→ everywhere): MySQL, PostgreSQL, oracle, DB2, SQLite, ... ▶ use it may times every day.

SQL: Structured Query Language:

▶ Declarative Language for core data manipulations => think in what you want, not how to compute it.

Declarative (e.g., SQL): • you don't specify how • but only what you want.

```
SELECT * from dogs
INNER JOIN owners
WHERE dogs.owner_id = owners.id
```

→ just join them => we only write what we want and it is done.

Entity →

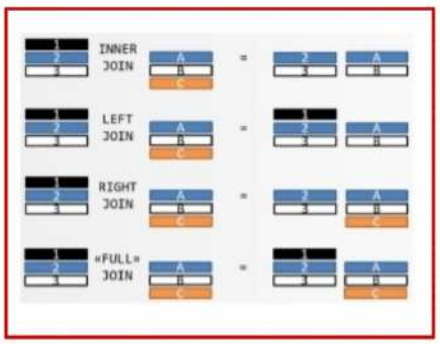
id	name
1	Bush
2	Trump
3	Obama

→ Relations.

president	successor
1	3
3	2

Operations:

- Key operations:
  - Select (!), update, delete
  - Unique keys
  - Joins (inner, left outer, right outer, full)
  - Sorting
  - Aggregation (group by, count, min, max, avg, etc.)



+ Pandas is lightweight and fast
+ Natively Python, i.e., full SQL expressiveness plus the expressiveness of Python, especially for function evaluation
+ Integration with plotting functions like Matplotlib

- In Pandas, tables must fit into memory
- No post-load indexing functionality: indices are built when a table is created
- No transactions, journaling, etc. (matters for parallel applications)
- Large, complex joins are slower

SQL And Pandas:

▶ More Functional than SQL => Data Frame

3. Document model: ▶ The world is a hierarchy of entities!

SCHEMAS FOR XML AND JSON:

A **schema** specifies the **structure** and **types** of a data repository, e.g., the types of each column in a table. It may also specify constraints **within** or **between** data fields. Traditional databases are **schema-on-write**. You cannot load data into a table without a schema.

Newer "NoSQL" data stores are schema-on-read or schema-less → you can defer applying a schema until you read the data or avoid schemas all together.

- **Schema-on-write:** SQL
- **Schema-on- read:** XML. XML schemas can be applied later to interpret XML data by specifying data types.

XML: Hierarchical

- **Document Object Model (DOM):** XML encodes DOM, a widely used data structure. DOM is tree structured.
- **Queries:** XML schema allows a DB to interpret the data when running queries, e.g., to do **arithmetic** or **range queries** on numerical values. XQuery is a standard for querying XML data with or without schemas.

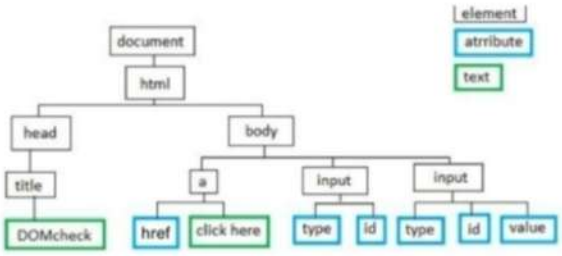
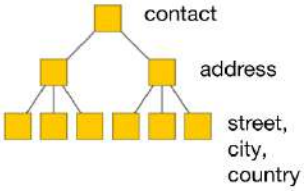


Figure 1: XML and DOM

JSON: Hierarchical

(JavaScript Object Notation) by contrast is a schemaless data description language (schema support was added later). But JSON is becoming more and more shemafull!

- Same expressivity as XML: hierarchical objects
- But JSON directly reflects how object is stored in target
- More lightweight, but may be more painful, too: transformations on JSON data are procedural in the target language (not *declarative* as in XQuery)



XML vs JSON:

XML:

- Separation between schema and data.
- Data can be represented and stored without schema (as strings).
- More verbose (but not true after compression or in DB).
- Standard query/transformation languages: XQuery, XSLT, etc.

JSON:

- Schema rarely used but can be.
- Data without schema uses type inference (string, int, float).
- Transformation/ingestion rely on code (e.g., JavaScript, Python

- XML format:

Like contact list
Hierarchical info.

```
<contact>
  <id>656</id>
  <firstname>Chuck</firstname>
  <lastname>Smith</lastname>
  <phone>(123) 555-0178</phone>
  <phone>(890) 555-0133</phone>
  <address>
   <street>Rue de l'Ale 8</street>
   <city>Lausanne</city>
   <zip>1007</zip>
   <country>CH</country>
  </address>
</contact>
```

- JSON format:

```
contact: {
  id: 656,
  firstname: "Chuck",
  lastname: "Smith",
  phones: ["(123) 555-0178",
           "(890) 555-0133"],
  address: {
   street: "Rue de l'Ale 8",
   city: "Lausanne",
   zip: 1007,
   country: "CH"
  }
}
```

Processing JSON and XML:

- The DOM tree is an easy object to work with: all the data in the object is accessible by links (edges in the DOM tree).
- The problem is that I might not care about most of the data, and I might **not be able to fit the DOM tree for a large object in RAM**

Event-driven sparsing: SAX

A SAX parser finds all the **open/close-tag events** in an XML documents and **does call-backs to user code**.

- (+) User code can respond to only a subset of events corresponding to the tags it is interested in.
- (+) User code can correctly compute aggregates from the data rather than create a record for each tag.
- (+) User code can implement flexible error recovery strategies for ill-formed XML.
- (-) User code must implement a state machine to keep track of "where it is" in the DOM tree.

JSON: Most JSON parsers construct the DOM tree directly. But there are a few SAX-style parsers (e.g Jackson and JSON-simple).

4) Network Model: ▶ Network 8x...

FORMATS:

Tabular data:

What is a table?

- A **table** is a collection of **rows** (or, alternatively, **columns**)
- Each row has an **index** and each column has a **name**
- A **cell** is specified by an (index, name) pair
- A cell may or may not have a **value**
- Schema = column types and names. Often stored as text files in CSV or TSV format.

Example, meteorological measurements: time series table

Sensors usually output data in the form of time series, collated in a tabular format. Yet, a system dealing with sensor data should:

- support both long-term (**trend**) and short-term (**real-time**) queries.
- have **low latency** but also efficient, **real-time indexing** for longer-term queries.
- support triggers (**alerts**) for a variety of conditions.

The complexity of a data format does not determine the complexity of the system required to properly handle it.

Binary formats: ▶ Parsing: converting strings to data types. ▶ Expensive! => Can be avoided by using binary format => data is stored as is => no conversion to strings.

- They are often the key to performance, **avoiding expensive parsing** ("deserialization")
- Modern binary formats support nested structures, various levels of schema enforcement, **compression**, etc.
- Python pickle, Java Serializable, Protocol Buffers (Google), Avro (supports schema evolution), Parquet (column-oriented, first-class citizen in Spark), etc.

→ Consider converting to one of these formats at the beginning of the processing pipeline. Especially with big data.

Data Sources:

Structured: Data that has predefined structures. e.g. tables, spreadsheets, or relational databases.
Unstructured Data: Data with no predefined structure, comes in any size or form, cannot be easily stored in tables. e.g. blobs of text, images, audio
Quantitative Data: Numerical. e.g. height, weight
Categorical Data: Data that can be labeled or divided into groups. e.g. race, sex, hair color.
Big Data: Massive datasets, or data that contains greater *variety* arriving in increasing *volumes* and with ever-higher *velocity* (3 Vs). Cannot fit in the memory of a single machine.

▶ Semistructured data too: mainly servers or website logs => "self-describing" structure!

▶ Websites like Wikipedia have a lot of data.
- REST API.
- XML dumps with wiki markup, SQL database dumps.
- Probs: Unicode, size, recency, etc.

▶ Make our life easier => ▶ Find projects to help you.
▶ Use more structured versions (like Wikidata => accesible as Json or RDF).

HTML => Plenty of downloadble data w/ it! => ▶ We will need a crawler or spider.

HTML AND REST:

HTML is everywhere. Useful tools:

- **Beautiful Soup:** a Python API for handling real HTML. DOM or SAX interfaces.
- **Requests:** an elegant and simple HTTP library for Python, built for human beings.
- **Scrapy**: an open-source framework to build Web crawlers.

REST: Representational State Transfer → We don't accept Wvally screen-scrapings => We Rest to have standards set or acceptance/ demands!

Stateless client/server protocol. Principles:

- Each message in the protocol contains all the information needed by the receiver to understand and/or process it. This constraint attempts to "*keep things simple*" and avoids needless complexity.
- Set of uniquely addressable resources requires universal syntax for resource identification (e.g. URI)
- Set of well-defined operations that can be applied to all resources. The primary methods are **POST, GET, PUT, DELETE** (Modify it. / Obtain info)
- The use of hypermedia in query results: resources are typically stored in a structured data format that supports hypermedia links, such as XHTML or JSON

We request via HTTP and receive a text file as a return!
▶ (Ain't be too Thin=) We need to see what we're doing!

3) DATA Wrangling:

1. Handling Data:

▶ Clean, merge. Adopt, Evaluate usability.
≈ 50-80% of your time.

DATA WRANGLING:

- Goal: extract and standardize the raw data
- Strategy: combine automation with interactive visualizations to aid in cleaning
- Outcome: improve efficiency and scale of data importing

Data preparation: Problems: ▶ Missing data, incorrect data, inconsistent representation of data => around 3/4 require human intervention => Trade off: Cleaning, oversanitizing.

- Deal with uncertain data (can arise from measurement errors, wrong sampling strategies, etc.)
- Parse/transform data (with the techniques we saw during the first hour) to obtain meaningful records for your specific analysis

We Vizualization and sampling to see where we have data problems!

2. V is for variety:

Data Preparation Overview:

- We need to **extract** data from the **source(s)**
- We need to **transform** data at into manageable format
- We need to **load** data into the **sink**

What we can get: structured data (with clear schema → e.g. application databases), semi-structured data (csv, self-describing → e.g. server logs) and unstructured data (Wikipedia, images and video).

Quiz: First lecture:

1) Visualization useful to highlist connectivity patterns in network data { matrix view.

2) Advantage of binary:
- Less expensive Parsing
- More efficient storage of num vals.

3) About flat data models:
1) can be represented in plain text.
2) More difficult to update in contrast to relational model.
3) No need to use delimiter.
4) simple relation.

4) => Inner join: only common joints of the dataset! What is common between TL on monty and the next one!

Lecture 3: Visualizing data

3. Data Visualization

VISUALIZATION FOR DATA EXPLORATION:

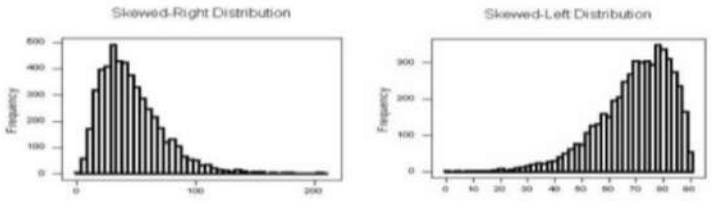


Figure 2: Histograms

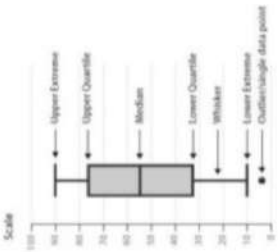
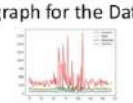


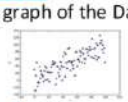
Figure 3: Boxplot

- useful for analysis, communication and decision making.
- Static visualization: great for data exploration, developed for a while.
- Interactive visualization: more and more common when delivering results! => key new enabler and also during exploration. Chart selection:

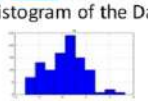
df.plot()
Plot a line graph for the DataFrame.



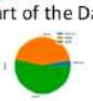
df.plot.scatter(x='w', y='h')
Plot a scatter graph of the DataFrame.



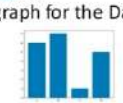
df.plot.hist()
Plot a histogram of the DataFrame.



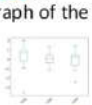
df.plot.pie()
Plot a pie chart of the DataFrame.



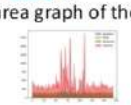
df.plot.bar()
Plot a line graph for the DataFrame.



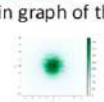
df.plot.boxplot()
Plot a scatter graph of the DataFrame.



df.plot.area()
Plot an area graph of the DataFrame.



df.plot.hexbin()
Plot a hexbin graph of the DataFrame.



df.plot(subplots=True)
Separate into different graphs for each column in the DataFrame.

df.plot(title="Graph of A against B")
Sets the title of the graph.

df.plot(cumulative=True)
Creates a cumulative plot

df.plot(bins=30)
Set the number of bins into which data is grouped (histograms)

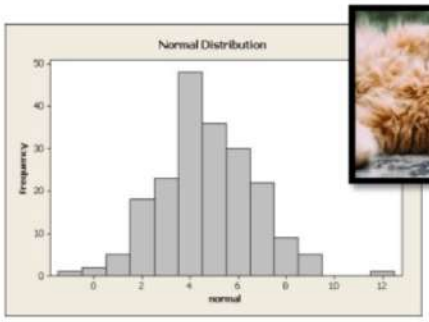
df.plot(stacked=True)
Stacks the data for the columns on top of each other. (bar, barh and area only)

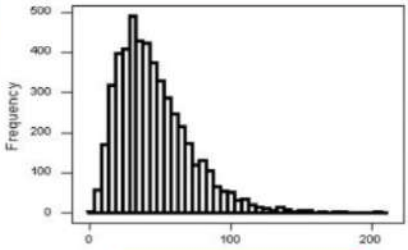
df.plot(alpha=0.5)
Sets the transparency of the plot to 50%.

df.plot(subplots=True, title=['col1', 'col2', 'col3'])
Arguments can be combined for more flexibility when graphing, this would plot a separate line graph for of column of a 3-columned DataFrame. The first string in the list of titles applies to the graph of the left-most column.

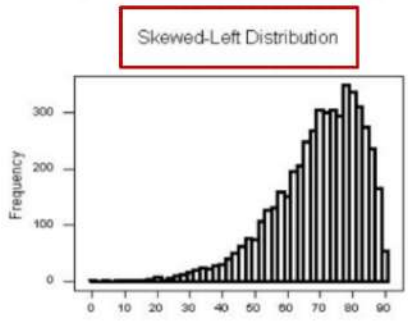
1) Histograms: ▶ Tells a lot about a single variable, discrete or continous.

Histograms can tell you a lot about a single variable, discrete or continuous





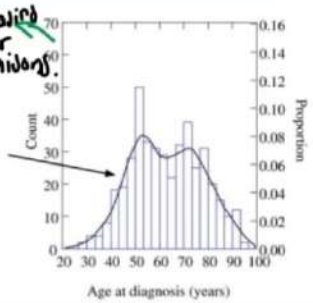
not normally distributed



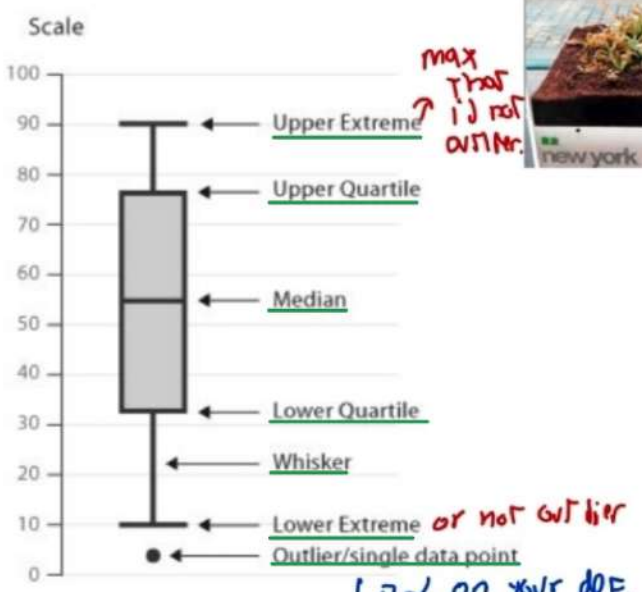
Easy to recognize skewed distributions!

⊕ visually pleasing + better for comparisons.

Smoothed histogram (a.k.a. kernel density estimate)



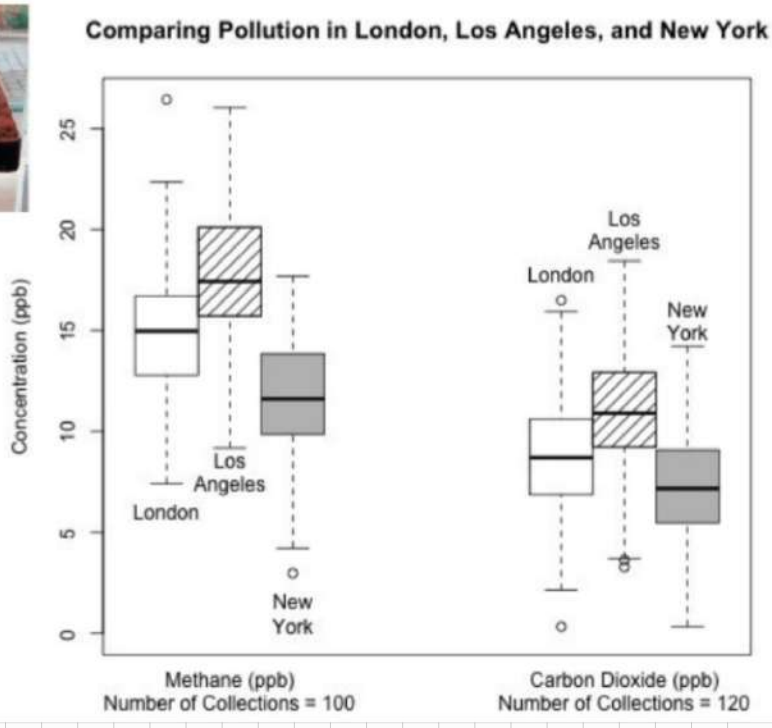
2) Box plots:



max that is not outlier.

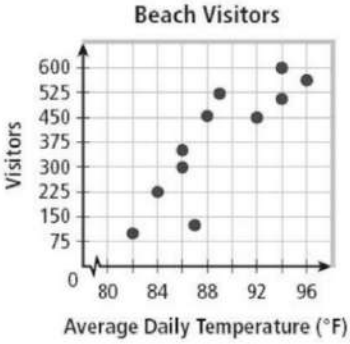
or not outlier

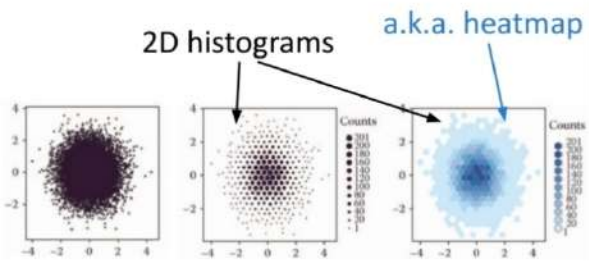
Up on your def.



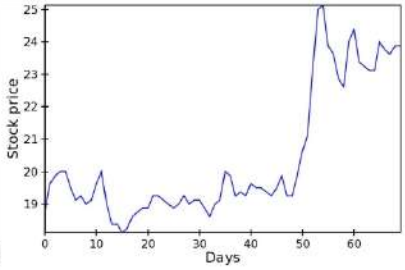
3) Two Variables: Scatter Plots:

Scatter plots quickly expose the relationships between two variables

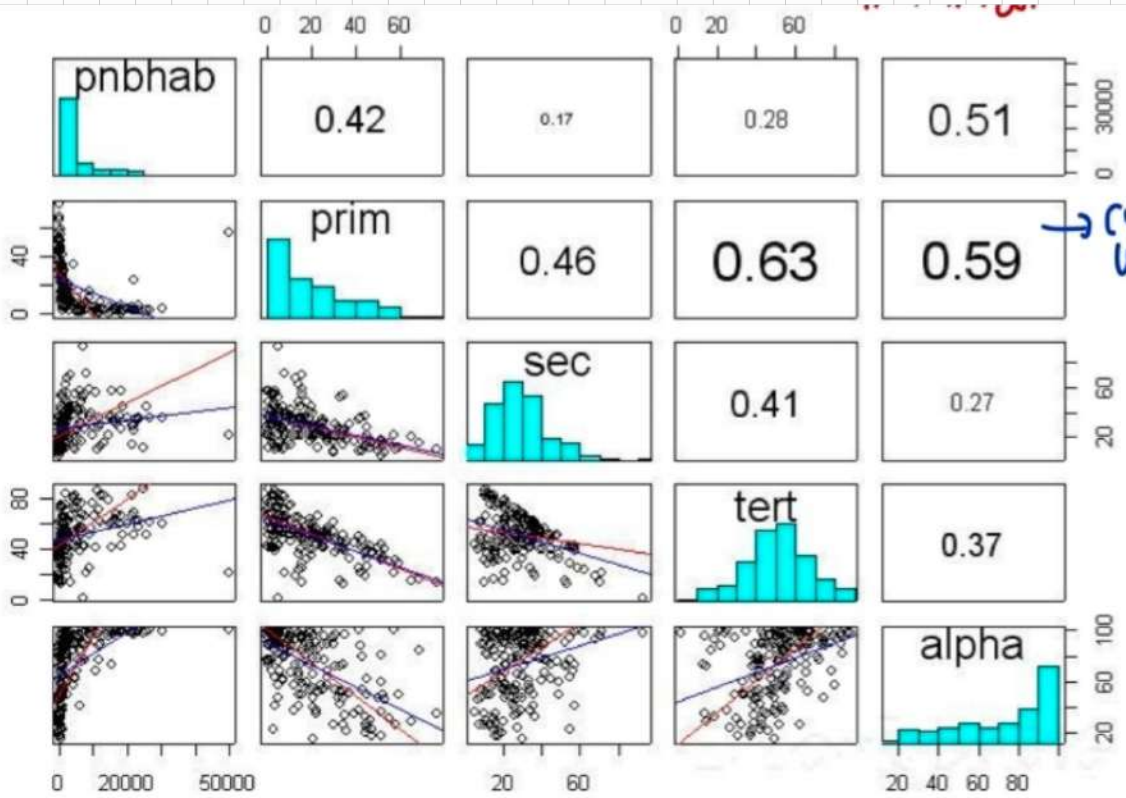




4) Two variables: line plots: If relationship is functional For instance:



5) Two variables: scatter plot matrix: ▶ Scatter plot of combination of Things.



→ Correlation => Be space wisely.

1 Basic version (very close already)

Python

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

cols = ["pnbhab", "prim", "sec", "tert", "alpha"]

sns.pairplot(
    df[cols],
    diag_kind="hist",
    plot_kws={"alpha": 0.6, "s": 20}
)

plt.show()
```

(1)

Python — Copier le code

```
g = sns.PairGrid(df[cols])

# lower triangle → scatter
g.map_lower(sns.scatterplot, alpha=0.6, s=20)

# diagonal → histograms
g.map_diag(sns.histplot, bins=10, color="turquoise")

# upper triangle → correlations
g.map_upper(corr_coefs)

plt.show()
```

(3)

2 Add correlation numbers in the upper triangle (like your image)

We define a small helper function to write the correlation coefficient.

Python — Copier le code

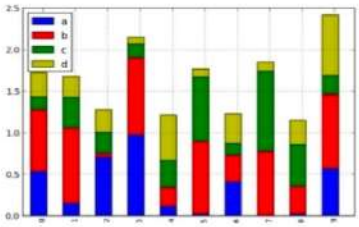
```
import numpy as np

def corr_coefs(x, y, **kws):
    r = np.corrcoef(x, y)[0, 1]
    ax = plt.gca()
    ax.annotate(
        f"{r:.2f}",
        xy=(0.5, 0.5),
        xycoords=ax.transAxes,
        ha="center",
        va="center",
        fontsize=14
    )
    ax.set_axis_off()
```

(2)

6) >2 Vars: Stacked plots:

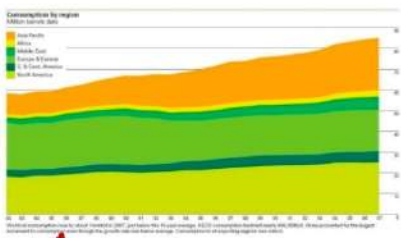
▶ Stack index, height, color.

Stack variable and color variables categorical, height variable continuous:



i)

Color variable categorical, stack and height variables continuous:



ii)

A) Example "long" data

i)

Python — Copier le code

```
np.random.seed(1)

long = pd.DataFrame({
    "x": np.repeat(list("ABCDEFGHIJ"), 4),
    "stack": list("abcd") * 10,
    "value": np.random.uniform(0.05, 1.0, 40)
})
long.head()
```

B) Convert to wide + plot stacked bar

Python — Copier le code

```
wide = long.pivot_table(index="x", columns="stack", values="value", aggfunc="sum").fillna(0)

ax = wide.plot(kind="bar", stacked=True, figsize=(9,4))
ax.set_xlabel("x")
ax.set_ylabel("value")
ax.set_title("Stacked bar (categorical stack)")
plt.tight_layout()
plt.show()
```

ii)

3) Stacked area (x continuous, categories stacked)

Typical case: **time (continuous) + category (color) + value**.

A) Example time-series by category

Python — Copier le code

```
np.random.seed(2)

dates = pd.date_range("2024-01-01", periods=40, freq="W")
cats = ["Asia Pacific", "Africa", "Middle East", "Europe", "Americas"]

ts_long = pd.DataFrame({
    "date": np.repeat(dates, len(cats)),
    "region": cats * len(dates),
    "value": np.random.uniform(10, 50, len(dates)*len(cats))
})

# add smooth-ish trend per region (optional)
region_effect = {c: i*3 for i, c in enumerate(cats)}
ts_long["value"] = ts_long["value"] + ts_long["region"].map(region_effect) +
np.linspace(0, 15, len(ts_long))
ts_long.head()
```

B) Pivot wide then area plot stacked

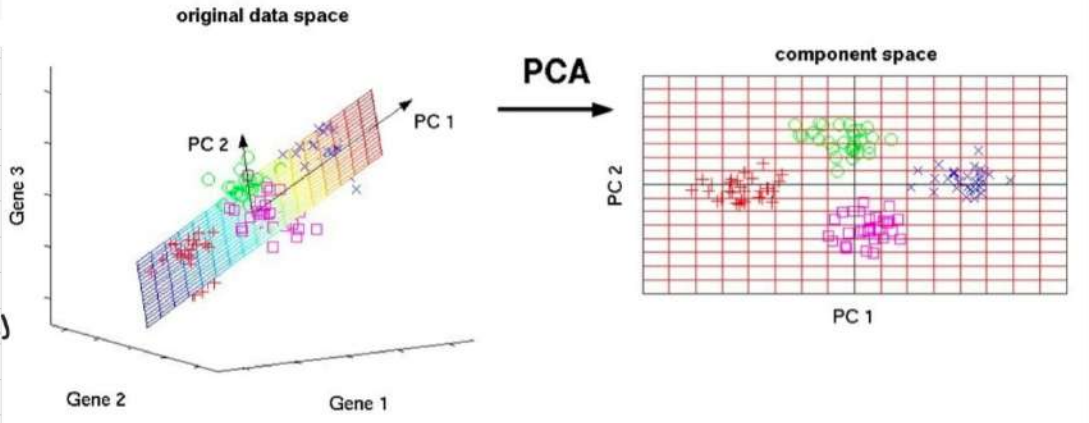
Python — Copier le code

```
ts_wide = ts_long.pivot_table(index="date", columns="region", values="value",
aggfunc="sum").fillna(0)

ax = ts_wide.plot.area(stacked=True, figsize=(10,4))
ax.set_xlabel("date")
ax.set_ylabel("value")
ax.set_title("Stacked area (time continuous)")
plt.tight_layout()
plt.show()
```

That's the right plot style in your second image.

7) Dimensionality reduction:

- For example => PCA allow visualization of high-dim (continuous) data in 2D principal components!
- Principal comps: highest variation of data => orthogonal.

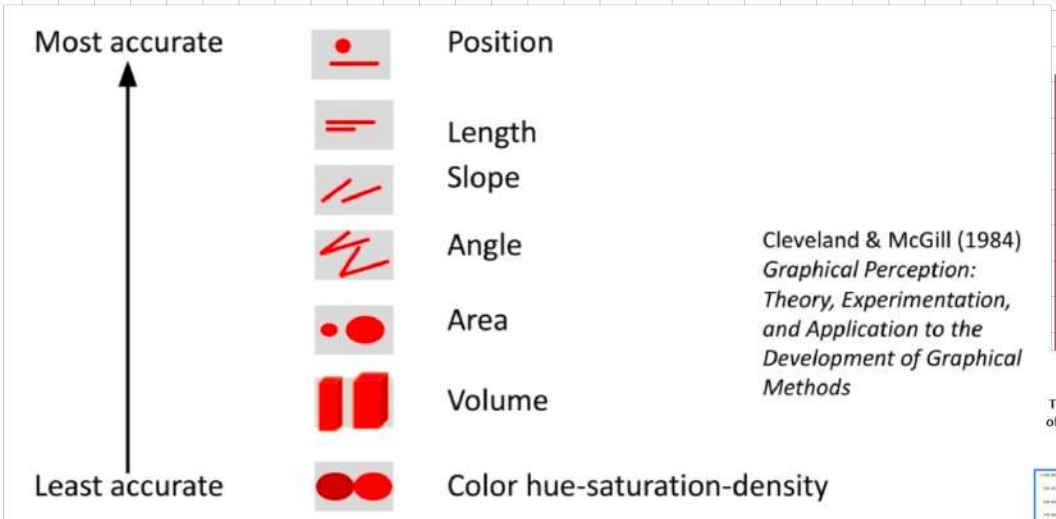


Part ②: Principles and best practices:

Perception of magnitudes:

Webers law: ▶ Weber's Law: $\Delta I / I = k$ ⇒ I: intensity, ΔI: Increase from I to notice a difference; k: constant ⇒ required ΔI ↗ on I! ⇒ If I↑, need big ΔI!

⇒ perceived in discrete (multiplicative) steps; not additive.

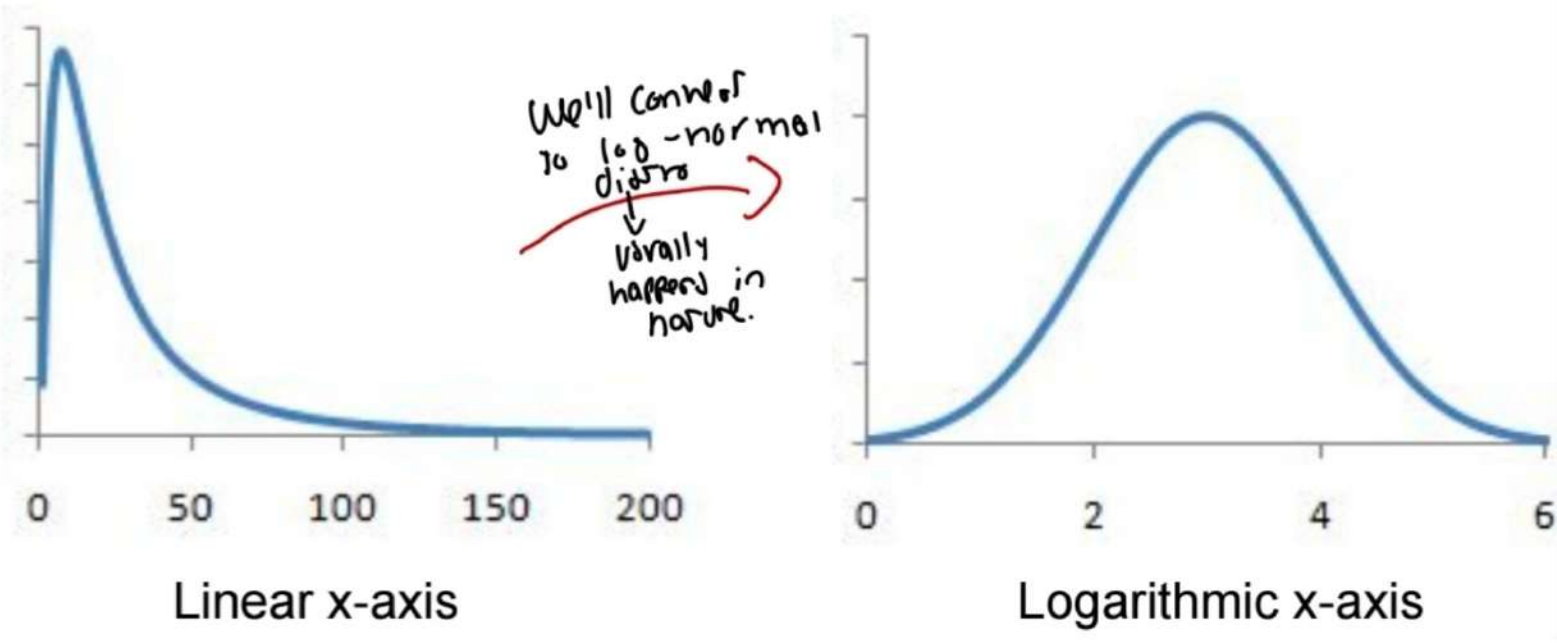


Properly choose axes:

▶ When having big values using logarithmic scales can be better than just using linear ones!

Visualizing heavy-tailed distributions

⌣) mean will be way higher than median



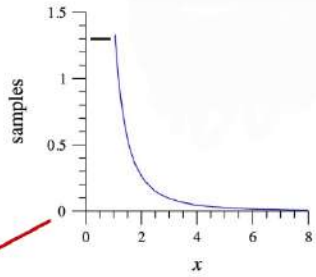
Heavy-tailed data: power laws : $P(x) = C x^{-\alpha}$: For linear:

Plotted on logarithmic axes for better visualisations. Makes it easier to spot from an exponential distribution. For an exponential distribution to be a straight line we need linear x axis and log y axes.

- Very very large values are rare, "but not very rare"
- Many natural phenomena are power laws (e.g., # of friends)
- For dealing with them, need to know some tricks:
  - E.g., straight line on log-log axes: $y = Cx^{-\alpha} \leftrightarrow log(y) = log(C) - \alpha log(x)$
  - Complementary cumulative distributive function (CDF) : $P(x) := Pr\{X >= x\}$
  - CCDF of power law is also a power law (with exponent $\alpha - 1$)

But for a power law ⇒ log - log

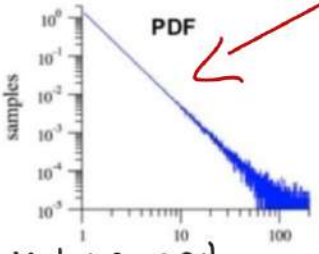
monotonically decreasing!

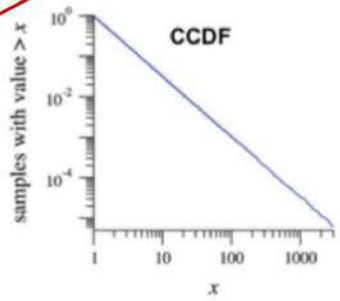


▶ For small α ⇒ mean and var = ∞ ⇒ use median!

Huge numbers will make it explode!

need >2 and >3 for convergence!





Smart trick for plotting CCDF of any distribution:

- x-axis: data sorted in ascending order
- y-axis: (n:1)/n (where n is number of data points)

$$P(x) = C\int_x^\infty x'^{-\alpha}\,dx' = \frac{C}{\alpha - 1}x^{-(\alpha-1)}.$$

⇒ Things are extreme ⇒ huge values of mean and var appear and this extreme events can happen!

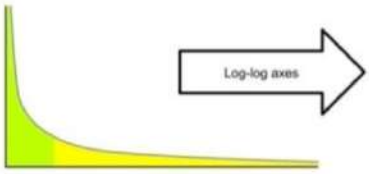
⇒ We can also use log scales to show more data than what we could w/ linear axis!

Heavy-tailed distributions:

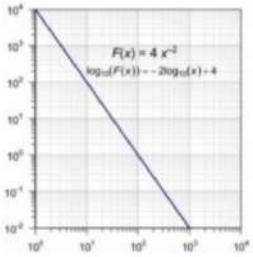
Point of the heavy tail is that most of the information is in the tail

E.g. power laws: $f(x) = ax^{-k}$

- Very very large values are rare, "but not very rare"
- For k <= 2: infinite mean
- For k <= 3: infinite variance
- Don't report mean/variance for power-law-distributed data!
- Use robust statistics (e.g., median, "80/20 rule", etc.)



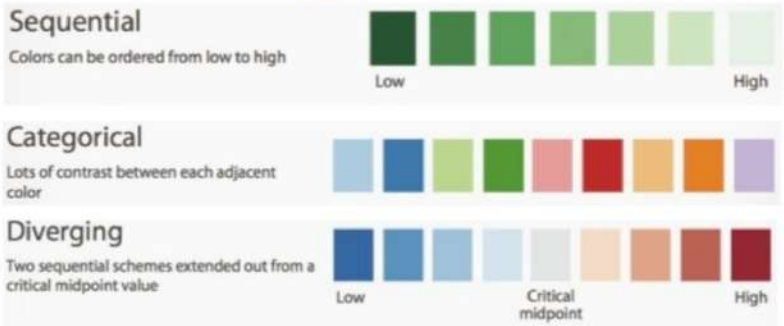
Log-log axes



⇒ Show uncertainty!!! ⇒ otherwise non meaningful!

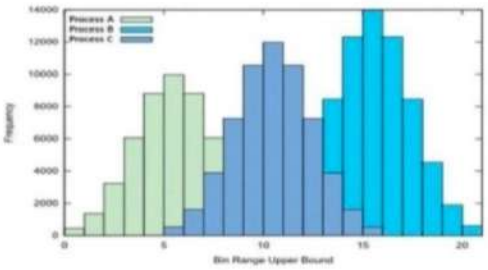
Colours:

- Choose colours based on the information you want to convey. Sequential, diverging, categorical
- Use online resources to find the best colour schemes ▶ Color blind Palettes.



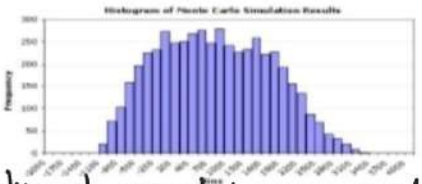
▶ Use visual contrast when needed! ⇒ Don't confuse readers w/ them!

Multimodal data:



- Two or more distinct peaks in a histogram.
- Suggests two or more distinct populations of samples.
- Often arise from binary factors such as gender or political views.
- But don't guess! Explore further by using, e.g., colour and a histogram of multiple populations → Stratify the data and plot them separately.

Weird data:



- Don't guess! Trace through the data pipeline to find where the strangeness comes from. Usually it's a processing bug.
- If data looks ok, take a picture and save it for later. Then periodically compare new data with old whenever there is a pipeline update.
- Always try to have a theory of what the data should look like.

▶ Don't discard "weird" looking data! ⇒ maybe it is ground for a new passionate discovery! ⇒ verify pipeline and then look for explanation.

Lecture 4: Describing data:

DESCRIPTIVE STATISTICS:

▶ Median: value that splits distro in half.
▶ 25%, 50% (median), 75% (percentiles).

Mean, variance and normal distribution:

The mean of a set of values is the average over the values. Variance is a measure of the width of a distribution. Specifically, the variance is the mean squared deviation of points from the mean. The standard deviation is the square root of variance. The normal distribution is completely characterized by mean and std.

Robust:

▶ Micro-average: standard-average of points. ▶ Group-average: calculate average for each ≠ group.
▶ Macro-average: Average of group averages.

A statistic is said to be robust if it is not sensitive to outliers

- Min, max, mean, std are not robust
- Median, quartiles (and others) are robust (robust because adding one data point can only shift everything by one data point)

Generalized Means:

▶ Transform into a different space ($F(x)$) ⇒ take mean and come back $F^{-1}(x)$.

⇒ $F^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} F(x_i)\right)$

- $F(x) = x^2$, $F^{-1}(x) = \sqrt{x}$ — "root mean sq"
- $F(x) = x$, $F^{-1}(x) = x$ — "normal mean"
- $F(x) = \log(x)$, $F^{-1}(x) = \exp(x)$ — "geometric mean"
- $F(x) = 1/x$, $F^{-1}(x) = 1/x$ — "harmonic"

↑ Effects of big adj values.

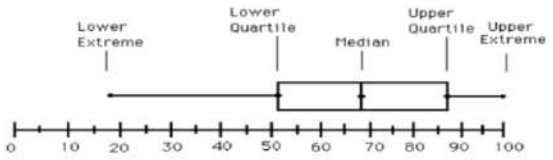
Mean	Formula	Equivalent transform $f(x)$	Best used when…	Typical examples	Sensitivity
Arithmetic mean	$\frac{1}{n}\sum x_i$	$f(x) = x$	Data are **additive**, noise is **Gaussian**, scale is linear	Heights, temperatures, sensor noise	**Very sensitive** to outliers
Root Mean Square (RMS)	$\sqrt{\frac{1}{n}\sum x_i^2}$	$f(x) = x^2$	You care about **magnitude / energy / power**	Signal amplitude, voltage, RMSE	**Strongly sensitive** to large values
Geometric mean	$\exp\left(\frac{1}{n}\sum \log x_i\right)$	$f(x) = \log x$	Data are **multiplicative**, **heavy-tailed**, span **orders of magnitude**	Growth rates, fold-changes, reaction rates	**Robust** to large outliers
Harmonic mean	$\left(\frac{1}{n}\sum \frac{1}{x_i}\right)^{-1}$	$f(x) = 1/x$	Averaging **rates**, **ratios**, or **bottlenecks**	Speed, throughput, F1 score	Dominated by **small values**

Distributions:

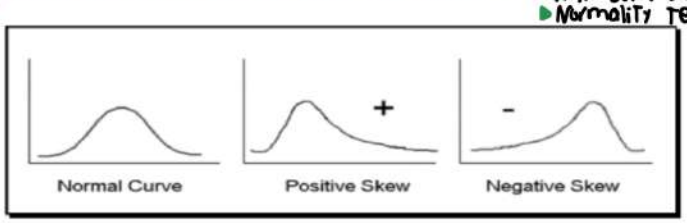
- **Poisson:** the distribution of counts that occur at a certain "rate"; e.g., number of visits to a given website in a fixed time interval.
- **Exponential:** the time interval between two such events.
- **Binomial/multinomial:** The number of counts of events (e.g., coin flips = heads) out of *n* trials.
- **Power-law/Zipf/Pareto/Yule:** e.g., frequencies of different terms in a document; city size

Visual inspection for ruling out certain distributions: e.g., when it's asymmetric, the data cannot be normal. The histogram gives even more information.

(+) Statistical tests to confirm afterwards:
- Goodness of Fit.
- Kolmogorov-Smirnoff test
- Normality tests.

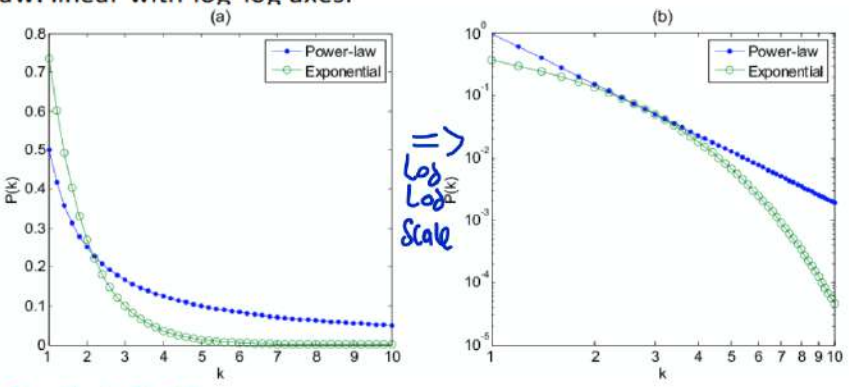


Box plot



(Smoothed) histogram

Recognizing a power law: linear with log-log axes.



=> Log Log Scale

Quantification of Uncertainty:

Sampling and Uncertainty:

- Datasets are samples from an underlying distribution.
- We are most interested in measures on the entire population, but we have access only to a sample of it. That makes measurement hard:
  - Sample measurements have variance between samples
  - Sample measurements have bias: systematic variation from the population value.

} Finite samples have uncertainty that we need to report!!!

Sampling is tricky:

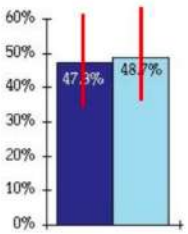
- Sometimes need to subsample for computational efficiency:
  - Uniformly at random
  - Stratified sampling (e.g., equal number per quantile)
  - Importance sampling
- Sometimes need to subsample during data collection:
  - E.g., Google Flu Trends: only Google users → Careful: bias!

Most of the time **uniformly** is the way to go but with weird distributions we might do stratified.

Quantifying uncertainty:

Even a complete dataset is a sample → all plots should have error bars!

Error bars:



Can represent many things (→ always explain; always question):

- Standard deviation
- Standard error of the mean: sd/sqrt(n)
- Confidence intervals: how precise we think our estimates are.
  - Parametric (ugh!)
  - Non-parametric (yay!): bootstrap resampling

Use **standard deviation** when we just want to show the variation in the data. Though if we want to make a claim about how good our estimates are, we need **confidence intervals**. If we want to make a claim that the mean we computed is a good estimation of the data, then we need to do confidence intervals.

Hypothesis testing:

Misconceptions: ⚠ None of these are true! ⚠

1. *If P = .05, the null hypothesis has only a 5% chance of being true.*
2. *A nonsignificant difference (eg, P ≥.05) means there is no difference between groups.*
3. *A statistically significant finding is clinically important.*
4. *Studies with P values on opposite sides of .05 are conflicting.*
5. *Studies with the same P value provide the same evidence against the null hypothesis.*
6. *P = .05 means that we have observed data that would occur only 5% of the time under the null hypothesis.*
7. *P = .05 and P ≤.05 mean the same thing.*
8. *P values are properly written as inequalities (eg, "P ≤.02" when P = .015)*
9. *P = .05 means that if you reject the null hypothesis, the probability of a type I error is only 5%.*
10. *With a P = .05 threshold for significance, the chance of a type I error will be 5%.*
11. *You should use a one-sided P value when you don't care about a result in one direction, or a difference in that direction is impossible.*
12. *A scientific conclusion or treatment policy should be based on whether or not the P value is significant.*

Hypothesis testing:

Reasoning:

- Flip a coin 100 times; 40 heads; "Is the coin fair?" → Simplest explanation
- Null hypothesis: "yes"; alternative hypothesis: "no" → "How likely would I be to see 40 or fewer heads if the null hypothesis were true?" → P. Value!
- If this probability is large, the null hypothesis suffices (and is thus not rejected)
- Otherwise, keep experimenting → It doesn't mean that the coin is not fair! { Just that it is unlikely.
- **Note:** we do not prove the alternative hypothesis. We work by making the null hypothesis seem insufficient.
- **Idea:** gain (weak and indirect) support for a hypothesis H_A by **ruling out a null hypothesis H_0** → gain (weak and indirect) support for a hypothesis H_A **(the coin is biased)** by means of disproving a null hypothesis H_0 **(the coin is fair).** We inspect a test statistic:
- A test statistic is some measurement we can make on the data which is likely to be large under H_A but small under H_0. E.g. The number of heads after *k* coin tosses (one sided) or the difference between number of heads and *k*/2 (two-sided)
- **Note:** tests can be either one-tailed or two-tailed. Here, a two-tailed test is convenient because it treats very large and very small counts of heads the same way.

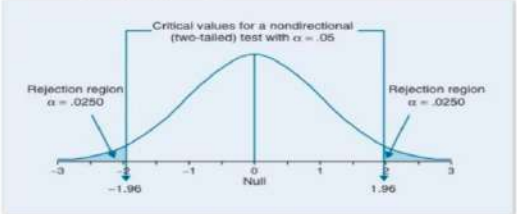
Coin example (cont'd)

- **Null hypothesis H_0:** "the coin is fair", i.e., "probability of heads = 0.5"
- **Test statistic s:** abs(50 - # of empirically observed heads after 100 coin tosses)
- $Pr(S \mid H_0)$: probability distribution of test statistic, assuming that H_0 is true

- Decision rule: **reject H_0 if $Pr(S \geq s \mid H_0) < \alpha$,** ⇒ null is not enough to explain data.
  i.e., if the probability of deviating from 50 heads at least as much as empirically observed is small
  - $Pr(S \geq s \mid H_0)$ = **"p-value"**
  - α = **"significance level"** ↗ decide before!
- α controls "false-rejection rate" (probability of rejecting H_0 although it is true)
  - You as the data analyst choose α (common values: 5%, 1%, 0.5%, 0.1%)
  - Higher α → higher false-rejection rate → we make rejecting null more likely!

Another example:

- Two samples *a* and *b*, normally distributed, from *A* and *B*.
- Null hypothesis H_0: mean(*A*) = mean(*B*)
- test statistic is: *s* = mean(*a*) − mean(*b*).
- Under H_0, *s* has mean zero and is normally distributed because the sum of two independent, normally distributed variables is also normally distributed.
- But it is "large" if the two means are different.
- We reject H_0 if $Pr(S > s \mid H_0) < \alpha$, i.e., if the probability of a statistic value **at least as large as** *s* is small.
- $Pr(S > s \mid H_0)$ is the infamous **p-value;** α is called **significance level**
- α is a suitable "small" probability, say 0.05 and directly controls the false-positive rate (probability of rejecting H_0 although it is true). The higher α, the higher false-positive rate. As we make α smaller, the false-negative rate increases (probability of not rejecting H_0 although it is false)
- Common values for α: 0.05, 0.02, 0.01, 0.005, 0.001



There are many statistical tests that we can do ↓

- Question
- Data type
- Sample size
- Same pop / ≠ pop
- Parametric assumptions ?

p-values:

Standard method to report significant results.

▶ Probability of observing the data at least as extreme as I observed under the null.

▶ Large p =) even under null =) very likely data came from it.

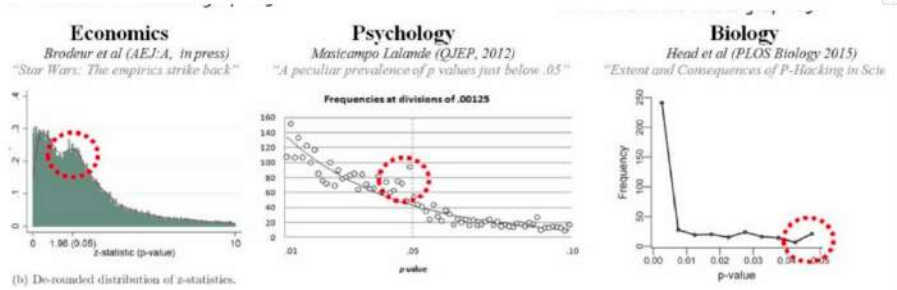
- Large p means that even under a null hypothesis your data would be quite likely. This tells you nothing about the alternative hypothesis.
- Historically, not meant as a method for formally deciding whether a hypothesis is true or not. Rather, an informal tool for assessing a particular result.
- Low p-value means: the simple null hypothesis doesn't explain the data, so keep looking for other explanations!
- p = 0.05 means: if you repeat experiment 20 times, you'll see extreme data even under null hypothesis → you might have "lucked out" → look at the y-axis, not just the p-value!

5 Intuition summary (one sentence)

> A p-value of 0.05 means that **even if nothing is going on**, results that look "significant" will naturally appear **about 1 in 20 times**, so you must inspect the **size and context of the effect**, not just the p-value.

p-value hacking: Don't do this!!!

1. Stop collecting data once $p<.05$
2. Analyze many measures, but report only those with $p<.05$.
3. Collect and analyze many conditions, but only report those with $p<.05$.
4. Use covariates to get $p<.05$.
5. Exclude participants to get $p<.05$.
6. Transform the data to get $p<.05$.

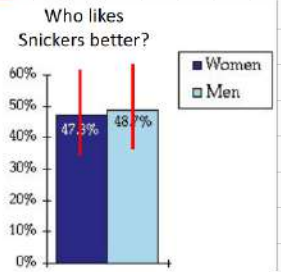


Alternative approach: Bayes factors

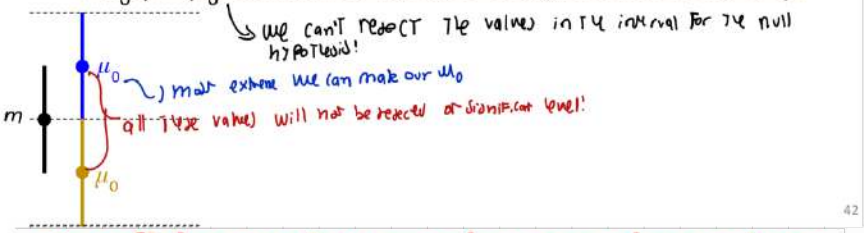
$$\frac{\text{Prob(Data, under } H_0)}{\text{Prob(Data, under } H_A)}$$

Confidence Intervals

- Confidence interval (CI) = a range of estimates for the parameter of interest (e.g., mean) that seems reasonable given the observed data
- Confidence level γ ⇒ "γ CI" (often γ = 95% ⇒ 95% CI)



- μ: true value of parameter of interest
- m: empirical estimate of parameter of interest
- CIs and hypothesis testing are tightly connected:
  - γ CI contains those values μ_0 for which the null hypothesis "$H_0: \mu = \mu_0$" cannot be rejected at significance level 1−γ

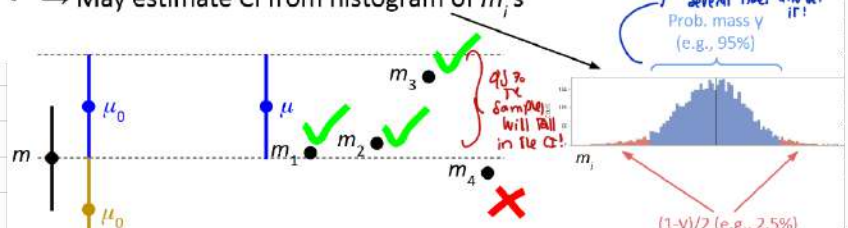


How to compute confidence intervals?

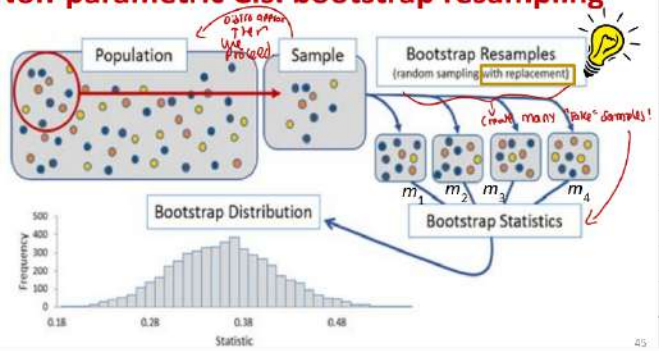
- **Parametric** methods assume that the test statistic follows a known (typically Normal) distribution
  → Need to verify that this is actually true! Ugh…
- **Non-parametric** methods make no assumptions about the distribution of the test statistic. They instead work by sampling the empirical data.
  → Yay!

Confidence intervals: another view

- If we were to repeat the data collection $N \to \infty$ independent times, we'd obtain N estimates of μ: $m_1, \ldots, m_N$
- Average of m_i's will approach the true μ (by law of large numbers)
- For a fraction γ of the N repetitions, m_i lies within the γ CI around μ
- → May estimate CI from histogram of m_i's



Non-parametric CIs: bootstrap resampling

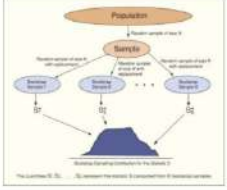


Bootstrap resampling:

Idea : start with one dataset and simulate new populations from the dataset that we have. We pick uniformly at random samples (they can be resampled). For these resampled datasets we compute the parameters (mean, etc). We repeat that many times → this gives us a distribution of our parameters which is a good approximation of the original distribution and can do a confidence interval.

RELATING TWO VARIABLES:

Pearson's correlation coefficient:



Multiple hypothesis testing:

- If you perform experiments over and over, you're bound to find something
- Significance level must be adjusted down when performing multiple hypothesis tests!

Family-wise error rate corrections:

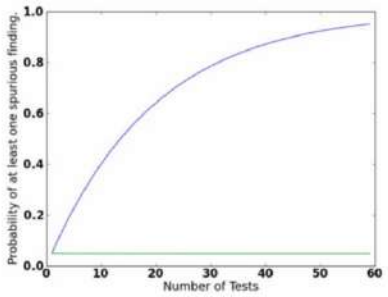
- Bonferroni correction: $\alpha_c = \alpha/k$
- Sidak correction: $\alpha_c = 1 - (1-\alpha)^{1/k}$

P(detecting an effect when there is none) = $\alpha = 0.05$

P(detecting no effect when there is none) = $1 - \alpha$

P(detecting no effect when there is none, on every experiment) = $(1-\alpha)^k$

P(detecting an effect when there is none on at least one experiment) = $1 - (1-\alpha)^k$



$\alpha = 0.05$

"Familywise Error Rate"

Family-wise error rate corrections

Bonferroni Correction

- Just divide by the number of hypotheses

$$\alpha_c = \frac{\alpha}{k}$$

Šidák Correction

- Asserts independence

we need to correct our P values!

$$\alpha = 1 - (1-\alpha_c)^k$$

$$\alpha_c = 1 - (1-\alpha)^{\frac{1}{k}}$$

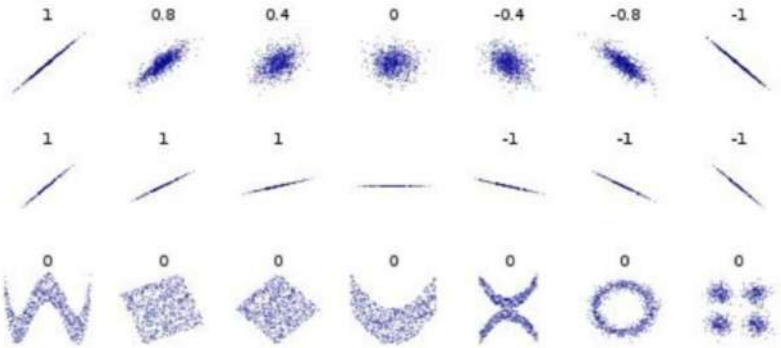
Relating Two Variables:

RELATING TWO VARIABLES:

Pearson's correlation coefficient:

Amount of linear dependence:

- +1: if positively linearly dependant
- -1: if negatively linearly dependant
- 0: no linear dependence



Attention: Pearson correlation coefficient does not take into account the slope. Only when there is no slope it's not defined. Captures only linear dependencies (last row).

Anscombe's quartet: By looking at only some specific statistics (mean, etc), we cannot distinguish these four datasets. Illustrates the importance of looking at a set of data graphically before starting to analyse.

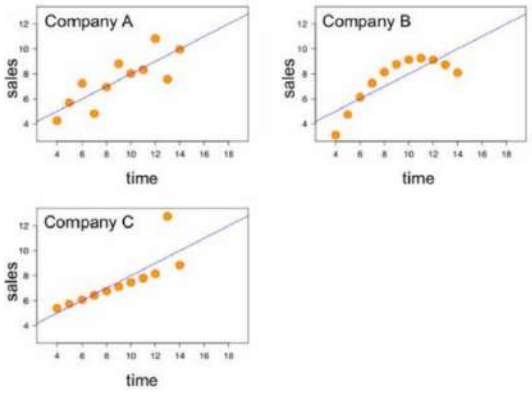


Figure 4: Anscombe's quartet

Simpson's paradox:

- When a trend appears in different groups of data but disappears or reverses when these groups are combined → beware of aggregates!
- E.g. women tended to apply to competitive departments with low rates of admission

Question 1

Incorrect

Mark 0.00 out of 1.00

Flag question

Which of the following statements are true? Assume you are working with non-negative data. Check all that apply.

- a. For symmetrical distributions, median is always the same as mean
- b. For heavy-tailed distribution, median can be smaller than mean ✓
- c. For skewed distribution, median is smaller than mean
- d. For skewed distribution, median can be smaller, bigger or equal to the mean ✗

The correct answers are: For heavy-tailed distribution, median can be smaller than mean, For symmetrical distributions, median is always the same as mean

Symmetrical : mean = med

Heavy tail, right skewed: mean < med

left skewed : mean > med

Lecture 5: Regression for disentangling data:

- Given n data pairs (X_i, y_i); X_i = k-dim vect of features and y_i: outcome.
- Find optimal $\beta = (\beta_1, \dots, \beta_k)$ for: $y_i = X_i \beta + \epsilon_i$ → error that should be as ↓ as poss. (scalar product!)
- Usually $X_{i1} = 1 \Rightarrow \beta_1$ = constant intercept.

Example: one predictor: $y \approx \beta_1 + \beta_2 X$ (β_1 → intercept, β_2 → slope)

Criterion: LSE: ⇒ ▶ Have ϵ_i as small as possible ⇒ Find β s.t. $\sum_i^n (y_i - X_i\vec{\beta})^2$ ↓ { ▶ Differentiable ▶ Both directions (outl. ea.) ▶ Min ↓ }

Use: ▶ Prediction ▶ Causal modeling ▶ Descriptive data analysis: compare outcome across subgroups of data!

Regression as comparison of average outcomes:

Example: Moms education and kids scores:

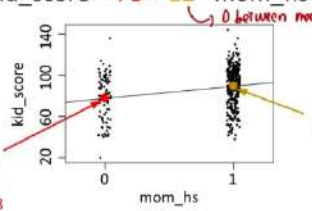
- X_i = mom_hs = "Did mother finish high school?" ∈ {0, 1} (No, Yes) — binary
- y_i = kid_score = child's score on cognitive test ∈ [0, 140]

$$y_i = \beta_1 + \beta_2 X_i + \epsilon_i$$

(↳ Fisher Error!)

kid_score = 78 + 12 · mom_hs + error

(↳ 0 between mean of two groups.)



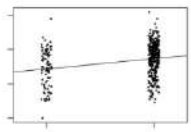
mean kid_score for moms who didn't finish high school: 78

mean kid_score for moms who finished high school: 78 + 12 = 90

So why not just compute both means and compare them?

$$y_i = \beta_1 + \beta_2 X_i + \epsilon_i$$

- **Intercept** β_1: mean outcome for data points i with $X_i = 0$
- **Slope** β_2: difference in mean outcomes between data points with $X_i = 1$ and data points with $X_i = 0$
- **Reason:** means minimize least-squares criterion:
  $\sum_{i=1}^n (y_i - m)^2$ is minimized w.r.t. m when
  $-2 \sum_{i=1}^n (y_i - m) = 0$, i.e., when $m = (1/n) \sum_{i=1}^n y_i$
  (⇒ rearrive)



↳ means are ile solution!

14

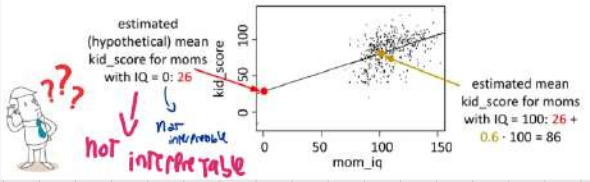
▶ Sometimes means are just not interpretable by their own:

Example: continous:

- X_i = mom_iq = mother's IQ score ∈ [70, 140]
- y_i = kid_score = child's score on cognitive test ∈ [0, 140]

$$y_i = \beta_1 + \beta_2 X_i + \epsilon_i$$

kid_score = 26 + 0.6 · mom_iq + error



$$y_i = \beta_1 + \beta_2 X_i + \epsilon_i$$

- **Intercept** β_1: estimated mean outcome for data points i with $X_i = 0$ (↳ no "exact" cuz it doesn't exist)
- **Slope** β_2: difference in estimated mean outcomes between data points whose X_i's differ by 1
- Why "estimated"? → e.g.,

data may not be linear so that's why we say estimated.

- NB: for binary predictor, we got "exact" instead of "estimated"

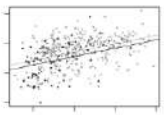
17

Example with multiple predictors:

- (X_{i1} = 1 = constant)
- X_{i2} = mom_hs = "Did mother finish high school?" ∈ {0, 1} (No, Yes)
- X_{i3} = mom_iq = mother's IQ score ∈ [70, 140]
- y_i = kid_score = child's score on cognitive test ∈ [0, 140]

$$y_i = \beta_1 + \beta_2 X_{i2} + \beta_3 X_{i3} + \epsilon_i$$

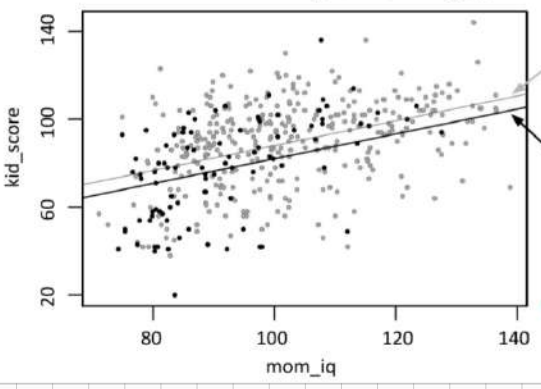
kid_score = 26 + 6 · mom_hs + 0.6 · mom_iq + error



kid_score = 26 + 6 · mom_hs + 0.6 · mom_iq + error

(↗ single model.)

(2 Features {Bin, Cont} — no meaningful no if =0 for moms.)



Single model.

▶ Both have same slope but ≠ intercept

19

19/53

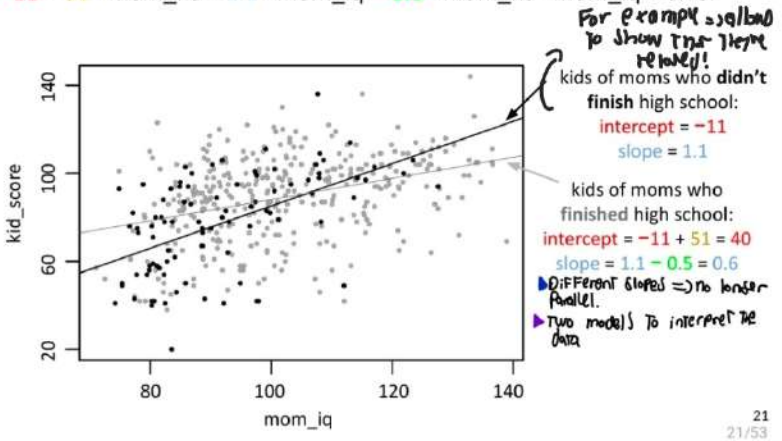
Example with interaction of predictors: ▶ Allows to visualize relation between ≠ parameters that simple relations don't show!

- X_{i2} = mom_hs = "Did mother finish high school?" ∈ {No, Yes} = {0, 1}
- X_{i3} = mom_iq = mother's IQ score ∈ [70, 140]
- y_i = kid_score = child's score on cognitive test ∈ [0, 140]

$$y_i = \beta_1 + \beta_2 X_{i2} + \beta_3 X_{i3} + \beta_4 X_{i2} X_{i3} + \epsilon_i$$

kid_score = −11 + 51 · mom_hs + 1.1 · mom_iq − 0.5 · mom_hs · mom_iq + err

kid_score = −11 + 51 · mom_hs + 1.1 · mom_iq − 0.5 · mom_hs · mom_iq + error



21/53

Example:

	Mom drives Mercedes	Mom doesn't drive Mercedes
Mom finished high school	avg kid_score 90	avg kid_score 90
Mom didn't finish high school	avg kid_score 78	avg kid_score 78

	Mom drives Mercedes	Mom doesn't drive Mercedes
Mom finished high school	990 women	10 women
Mom didn't finish high school	10 women	990 women

23

- Mean kid_score for Mercedes drivers: **0.99 · 90 + 0.01 · 78 ≈ 90**
- Mean kid_score for non-Mercedes drivers: **0.01 · 90 + 0.99 · 78 ≈ 78**
- But really driving Mercedes makes no difference (for fixed high-school predictor)!
- Root of evil: **correlation** between finishing high school and driving Mercedes
- **Regression** to the rescue: **kid_score = 78 + 12 · mom_hs + 0 · mercedes + error**

↳ It plays no role ⇒ data is enlarge in correlation but this relationship

Mercedes already given in Hs! ⇒ Aha!

	Mercedes	No Mercedes
Mom finished high school	mean kid_score 90	mean kid_score 90
Mom didn't finish high school	mean kid_score 78	mean kid_score 78

	Mercedes	No Mercedes
Mom finished high school	990 women	10 women
Mom didn't finish high school	10 women	990 women

25
25/53

Quantifying uncertainty:

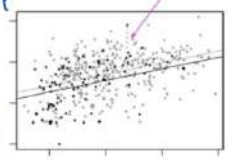
▶ We get a p-value: (proba of estimating such extreme coefficient if true coeff = 0).

+ Residual standard error + R-squared:

- **Residual** for data point *i*: estimation error on data point *i*:

$$r_i = y_i - X_i\hat{\beta}$$

↳ error pt in making



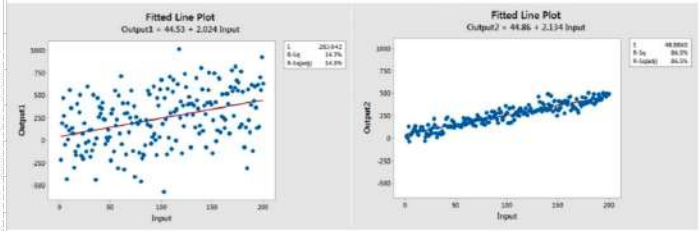
- Mean of residuals = 0 (total overestimation = total underestimation) ↗ same error above and below
- Variance of residuals = avg squared distance of predicted value from observed value = "unexplained variance" ↗ standard deviation = RSE
- Fraction of variance explained by the model:

$$R^2 = 1 - \hat{\sigma}^2/s_y^2$$

↑ better ✓ — Variance of outcomes y (how much my results vary) — Variance remaining after I fit model ⇒ 0 = perfect; if perfect = 0.

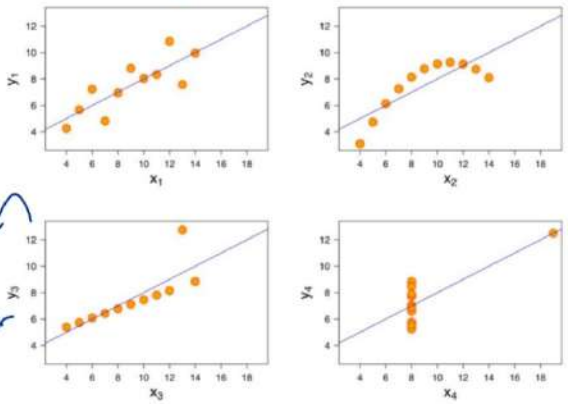
29

$$R^2 = 1 - \hat{\sigma}^2/s_y^2$$



R^2 = 0.147 R^2 = 0.865

31



Even if they are should be better

R^2 = 0.67 everywhere! ↗ very useful but not all!

R^2 is useful but doesn't tell us totally all the story!

Assumptions for reg. model:

1) Validity { a) outcome should show phenomenon of interest ⇒ predictors are imp.
b) All relevant predictors.
c) Good generalization.

2) Linearity: ▸ In predictors! Not really in inputs! ⇒ $y = \sum \beta \cdot \phi(x)$! ⇒ may transformation possible!

Let's use in practice!

3) Independence of errors in data points. 4) Equal variance of errors. 5) Normality of errors)

Transformations of Predictors and Outcomes:

1) Linear Transformations:

- When we apply **linear transformations** to predictors, the model remains "equally good":
  - The fitted coefficients may change, but predicted outcomes and model fit (R^2) won't change
- For instance,

$$\begin{aligned} \text{earnings} &= -61000 + 51 \cdot \text{height (in millimeters)} + \text{error} \\ \text{earnings} &= -61000 + 81000000 \cdot \text{height (in miles)} + \text{error} \end{aligned}$$

R^2 remain same but we only change Final unit.

After mean-centering of predictors, …

… you have a convenient interpretation of coefficients β_j of main predictors (i.e., non-interaction predictors):

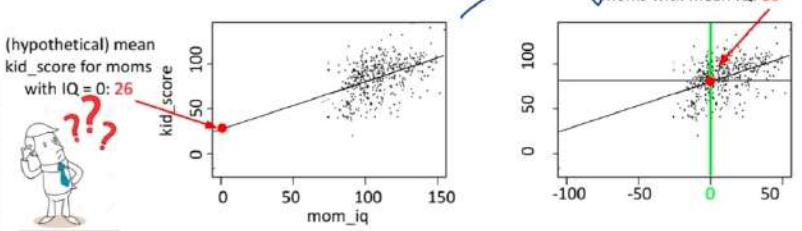
- $j = 1$ (i.e., intercept):
  - Estimated mean outcome when each predictor has its mean value
- $j > 1$:
  - Model w/o interactions: estimated mean increase in outcome y for each unit increase in X_{ij}
  - Model with interactions: estimated mean increase in outcome y for each unit increase in X_{ij} **when each other predictor has its mean value**

2) Mean centering of predictors

Mean-centering of predictors

- Compute the mean value of a predictor over all data points, and subtract it from each value of that predictor:
  $X_{ij} \leftarrow X_{ij} - \text{mean}(X_{1j}, \ldots, X_{nj})$
- ⇒ the predictor X_{ij} now has mean 0

Ordering and interpretability! directly read from it.



40

3) Standardization via z-scores:

- First **mean-center** all predictors, then **divide them by their standard deviations:**
  $X_{ij} \leftarrow [X_{ij} - \text{mean}(X_{1j}, \ldots, X_{nj})] \,/\, \text{sd}(X_{1j}, \ldots, X_{nj})$
- Resulting values are called "**z-scores**"
- All predictors now have the same units: distance (in terms of standard deviations) from the mean

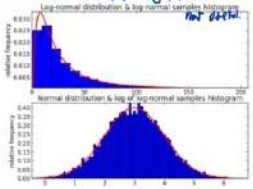
now we can compare "how many stds" from mean we are so for all we can easily compare them.

- This lets us compare coefficients for predictors with previously incomparable units of measurement, e.g., IQ score vs. earnings in Swiss francs vs. height in centimeters

4) Logarithmic outcomes:

Logarithmic outcomes

Power law better for log + mean curve! real data!



- **Practical:** makes sense if the outcome y follows a heavy-tailed distribution
- Only works for non-negative outcomes
- **Theoretical:** turns an additive model into a **multiplicative model**:

$$\log y_i = b_0 + b_1 X_{i1} + b_2 X_{i2} + \cdots + \epsilon_i$$

Exponentiating both sides yields

$$\begin{aligned} y_i &= e^{b_0 + b_1 X_{i1} + b_2 X_{i2} + \cdots + \epsilon_i} \\ &= B_0 \cdot B_1^{X_{i1}} \cdot B_2^{X_{i2}} \cdots E_i \end{aligned}$$

same after positive transform.

43 43/59

Logarithmic outcomes: Interpreting coefficients

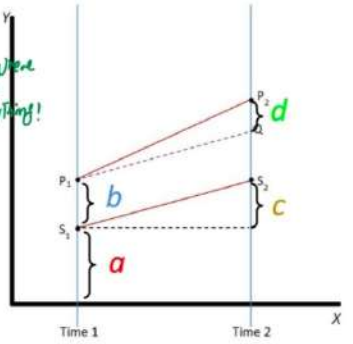
$$\begin{aligned} y_i &= e^{b_0 + b_1 X_{i1} + b_2 X_{i2} + \cdots + \epsilon_i} \\ &= B_0 \cdot B_1^{X_{i1}} \cdot B_2^{X_{i2}} \cdots E_i \end{aligned}$$

- An **additive** increase of 1 in predictor $X_{\cdot 1}$ is associated with a **multiplicative** increase of $B_1 := \exp(b_1)$ in the outcome
- If $b_1 \approx 0$, we can immediately interpret b_1 (without needing to exponentiate it first to get B_1!) as the **relative increase** in outcomes, since $\exp(b_1) \approx 1 + b_1$ } useful for interpretation
- E.g., $b_1 = 0.05 \Rightarrow B_1 = \exp(b_1) \approx 1.05$
  ⇒ "+1 in predictor $X_{\cdot 1}$" is associated with "+5% in outcome"

Beyond linear reg: ▸ Logistic reg: binary outcomes.
▸ Poisson reg: non-negative integer outcomes (e.g counts).

Beyond comparing means; or, A taste of causality: "Difference in differences" (2)

- Elegant linear model with binary predictors:
  $y_{it} = a + b \cdot \text{treated}_i + c \cdot \text{time2}_t + d \cdot (\text{treated}_i \cdot \text{time2}_t) + \text{error}_i$
- d = treatment effect
- All of this with one single regression!
- You get quantification of uncertainty (significance) for free!

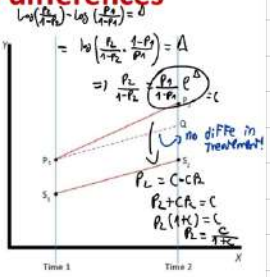


48

Example of Causality:

Beyond comparing means; or, A taste of causality: "Difference in differences"

- Two groups: *P*, *S*
- At time 2, group *P* receives a **treatment**, group *S* doesn't
- Question: Did the treatment have an **effect?** If so, how large was it?
- *P* and *S* don't start out the same at time 1
- There is a temporal "baseline effect" even w/o treatment



47

QCMs:

Question 4
Incorrect
Mark 0.00 out of 1.00
Flag question

You build a linear regression model to predict website traffic y based on advertising spending x. After assessing the data, you decide to log-transform the outcome and model log(y) instead of y. When evaluating the model, you note the coefficient for x is 0.005. Which interpretation of this coefficient is correct?

a. The 0.5 coefficient means advertising spending has no relationship with website traffic.
b. The transformation implies that traffic is now modeled as a logarithmic, not linear, function of advertising spending ✗
c. A $1 increase in advertising spending predicts a 0.5 absolute increase in y.
d. A $1 increase in advertising spending predicts a 0.5% increase in website traffic y.

The correct answer is: A $1 increase in advertising spending predicts a 0.5% increase in website traffic y.

One-line rule to remember

- log(y) ~ x → coefficient ≈ percent change in y per unit x
- y ~ log(x) → coefficient ≈ absolute change in y per % change in x

If you want, I can give you a cheat table for all 4 common log models — super useful for exams.

Regression Interpretation Cheat Sheet

Model	Equation	Interpretation of β
Linear–Linear	$y = \beta_0 + \beta_1 x$	+1 unit in x → β_1 units change in y
Log–Linear	$\log(y) = \beta_0 + \beta_1 x$	+1 unit in x → ≈ 100·β_1% change in y
Linear–Log	$y = \beta_0 + \beta_1 \log(x)$	+1% in x → β_1 / 100 units change in y
Log–Log	$\log(y) = \beta_0 + \beta_1 \log(x)$	+1% in x → β_1% change in y

Question 2
Partially correct
Mark 0.67 out of 1.00
Flag question

Which of the following statements is correct? Check all that apply.

a. Intercept represents the predicted value of the outcome when all predictor variables are set to zero ✓
b. If we have negative predictors, we should not perform linear regression
c. Low R^2 score indicates that the predictors have no statistically significant correlation with the outcome
d. If a predictor has a positive linear regression coefficient, that means that an increase in the predictor is associated with an increase in the outcome ✗

The correct answer is: Intercept represents the predicted value of the outcome when all predictor variables are set to zero

No CVE Prove could be big! => No stat sig!

Lecture 6: Causal analysis of observational data:

▶ Clarify ≠ in observational and experimental studies

Dr. Bob's smoking cure:

- I claim to have developed a medicine that helps you quit smoking
- I ask all smokers: "Do you want to try my medicine?"
- Smokers = treated smokers ∪ untreated ("control") smokers
- Fraction of successful quitters is higher in the treated group
- I conclude: "My medicine helps you quit smoking! Buy it!"
- Do you believe me? → No! there may be confounders in his experiment!

Causal diagram:

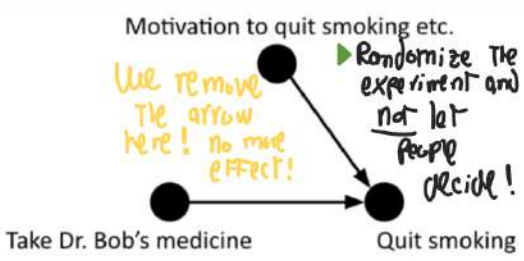
- Claim: there is a causal link between taking the medicine and quitting smoking.
- Semantic of arrows: cause and effect → wiggling the quantity on the start point changes the quantity on the end point, ceteris paribus.
- In presence of confounder, you are measuring a mix of the effects of treatment and confounders; don't know what part of effect comes from which
- Association between treatment and outcome could be there even without any causal link
- Confounder: confounds the treatment and motivation → measure a mix between treatment and confounder in the outcome

"Confounder" = which affects both treatment and outcome!

So no arrow here :)

Motivation to quit smoking etc.

Ideal setting as a causal diagram:



Now there is no link between motivation and the treatment. If the motivation in a random group of people does not differ, then we really do measure the effect.

- Ideally (not possible): for any person, observe their quitting behavior in two parallel worlds:
  - (1) They take Bob's medicine
  - (2) They don't take Bob's medicine
  - (everything else being the same)

▶ Ideal: probability of receiving treatment is the same for everyone => treatment and control groups are indistinguishable.

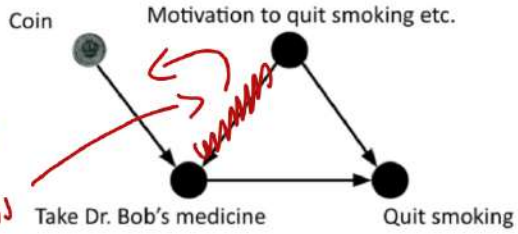
- To approximate this ideal parallel-worlds situation:
  - Want to control treatment ourselves (as researchers), observe outcome
  - In doing so, want to make sure motivation (and in fact any other potential confounders) has no influence = deleting all arrows leading to treatment. In practice: randomize

We need to make sure that the people who take and ho don't take the medicine are completely indistinguishably from each-other → Idea of *randomized*: flip a coin for each person on whether they take the treatment or not

RANDOMIZED CONTROLLED EXPERIMENT:

- Two experimental conditions:
  - Treatment (e.g., medicine)
  - Control (e.g., placebo → because it's a better control to keep everything as close as possible. Giving them, something compared to nothing is closer to take a medicine).
- Assignment of participants to conditions is random. Probability of receiving treatment same for everyone
- Treatment and control groups are indistinguishable. E.g., determination to quit smoking is not systematically higher in the treated group

Causal diagram:



Randomization takes arrow away, as confounds cannot influence decision to treat anymore. Coin is now the dictator (money rules...). More explicit: replace bad arrow with an arrow from the coin outcome, which is not affected by anything else => Basically: The randomness replaces this

Limits:

E.g. Do seat belts save lives? Experiment:

- Flip coin at birth to assign to treatment (always wear seat belt for entire life) or control (never wear seat belt)
- Measure fraction of traffic deaths in each group

Randomized experiments aren't always feasible:

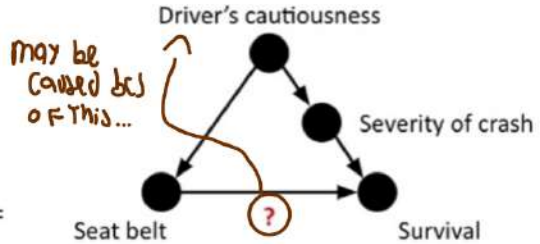
- Unethical (see above), expensive (because we need to set up this whole infrastructure), fundamentally impossible (e.g., do earthquakes decrease life spans?)
- Most modern "big data" is "found data" → just fundamentally observational data, you cannot go back for example to elections and recreate the environment
- Sometimes, observational studies are even better suited

ALTERNATIVE: OBSERVATIONAL STUDIES

- Fundamentally different from experiment:
  - Researcher can't control who goes to which condition
  - Researcher is merely an observer, not a tinkerer
  - Much less problematic w.r.t. ethics, price, feasibility but much more problematic w.r.t. validity of conclusions
- All advantages of randomized experiment are gone:
  - Subjects self-select to be treated
  - Treatment assignment and response may be caused by same hidden correlate (a.k.a. confounders; e.g., resolve to quit smoking)

→ Interestingly, the case of smoking vs lung cancer created a lot of methodology we are going to cover

Example: seat belts revisited

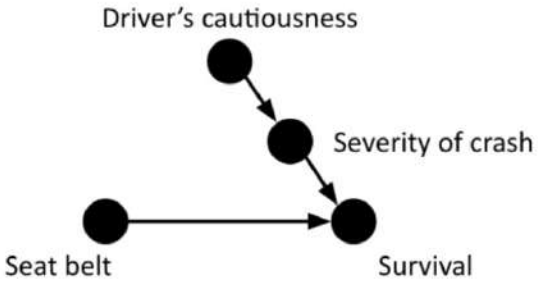


Recall: experiment infeasible because unethical (cannot force people not to wear seatbelts). Observational study:

- Dataset: all traffic accidents in a given time span
- Two treatment conditions: treated = seat-belt wearers and control = non-seat-belt wearers
- Compare fraction dead in treated vs. control
- What problems do you see? Possible confounds → willingness to take risks, driving off in a hurry (forget seat belt, drive fast), being drunk (forget seatbelt, drive dangerously), old cars might lack both seat belts and airbags

▶ Be more cautious, position where wear, Age, alt ...

Matched observational study:



- Consider only particular subset of accident cars:
  - 2 people in car: driver + passenger
  - Exactly one of them died in accident
  - Exactly one of them wearing seat belt at time of accident (i.e., 1 treated + 1 control per car)
- As before: compare fraction dead in treated vs. control
- New: everything else is controlled for, incl. type of car, speed, severity of accident → everything in the control group and treated group are now comparable and controlled
- Fundamental concept: **matching** (paired study)
- Smart sub-selection of data points ("adjustment") → Driver's cautiousness does not change fraction of belted people in any car

	Driver Passenger	Not Belted Belted	Belted Not Belted
Driver Died	Passenger Survived	189	153
Driver Survived	Passenger Died	111	363

- What could still happen? Different confounds (and mediators) are possible; e.g., position of seat-belt wearer
  - Being driver could increase prob of wearing seatbelt (e.g. because less likely to be drunk) and increase prob of dying (because of steering wheel) → this could cancel out true positive effect of belt
  - Passenger more likely to be drunk, therefore less likely to be belted, but also more likely to die → potentially spurious correlation of belt and death

Natural experiments:

This is called a nature experiment. Because nature decides who wears a seatbelt and not.

- Not researcher, but nature, "flips a coin" to decide treatment assignment
- Rosenbaum: "When investigators are especially proud, having found unusual circumstances in which treatment assignment, though not random, seems unusually haphazard, they may speak of a 'natural experiment.'" E.g., seat-belt study: ceteris paribus, who wears the belt is (nearly) haphazard
- Point 3: "nearly": not fully met by seat-belt study (but circumvented by table analysis)
- To make it even clearer:
  - imagine both people in car are passengers (two passengers in back; or self-driving cars)
  - Twin studies with adoption → e.g. distinguish between nature vs nurture
- How could this still fail? Maybe certain group of people (e.g., men) less likely to wear belt and more vulnerable to impacts of head on glass...

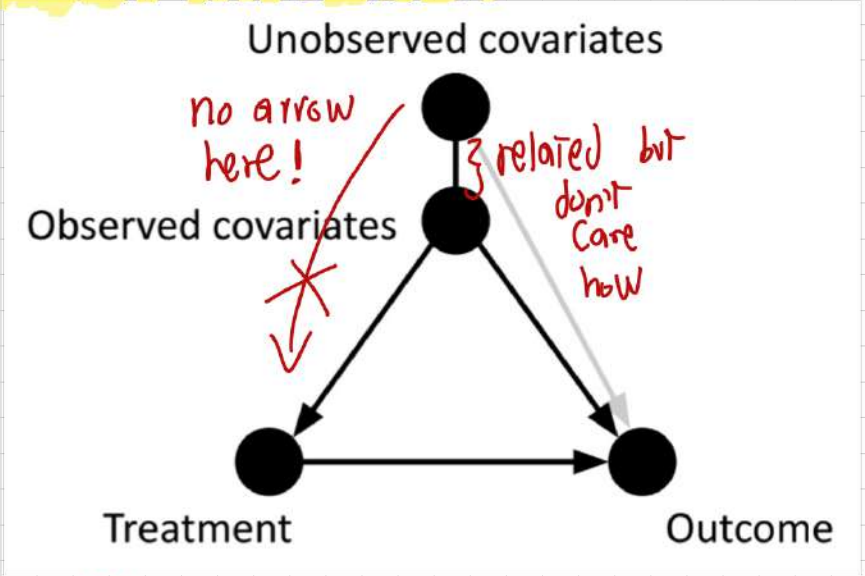
Nature didn't flip a coin for me – should I just go home and weep?

▶ We compare outcome of having a treated and control very similar (mean diff or regression analysis).

- Fundamental concept: matching
- Idea: Pair up 2 "similar" people: 1 treated, 1 control. Ideal (rather: Utopian): "similar" := "identical"
- Also sufficient (phew!): "similar":= equal probability to receive treatment (given the state of the world before the study) → condition that we need to unpack more below. The point is that for example for the smoking cure, the probability of being treated was different for the people in the treatment group vs control because people self-selected. We need to have people to have the same probability to end up here. The matching is similar to the seat-belt study but here we do it after the data was collected. In the seatbelt study the matching happened before it happened as people decided to go into the same car. But the principle of matching is still the same.

Two problems: 1) Unobserved Covariates: Usually, hidden Factors Can make'em s.t 2 People are not identical.
2) Combinatorial Explosion: look for People That are identical.

=> We will Adress problem ① by ignoring it! => Naive Model => only observed Covariates determine Treatement assignement.



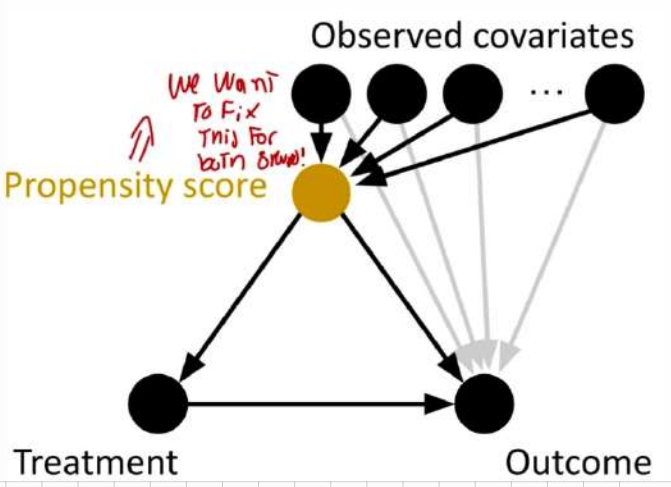
If Naive Model was True:

▶ If no problem 1... => just match two subjects w/ identical observed Covariates (1 Tre / 1 cont) and call it a day ☺.

▶ We still have though problem 2: no two people are identical and thus finding them is impossible.

▶ But do we really need identical people? ---> Let's try to mimic RCT and what a coin does.

Propensity score:



- Compress the (potentially many) observed covariates into a single number: the probability to receive the treatment (a.k.a. **propensity score**):

  Pr(subject is treated | observed covariates)

- Can be estimated from the data
  - E.g., via logistic regression (see next lecture)
  - Input: observed covariates
  - Output: treatment indicator (1 if treated, 0 if control)

34/5

Balancing:

- Fact: all subjects (treated and control) with equal propensity score (PS) p have equal distribution of observed covariates **x**:

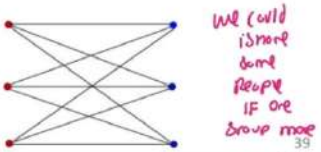
  $\Pr(\mathbf{x} \mid \text{treated} = 1, \text{PS} = p) = \Pr(\mathbf{x} \mid \text{treated} = 0, \text{PS} = p)$

- Subjects in a matched pair might not have equal **x**, but treated and control groups will have similar distributions of **x**

- There are many other methods for achieving balance
  - e.g., exact matching, Mahalanobis distance matching, coarsened exact matching, …
- You can mix and match methods (e.g., match exactly on gender, use propensity scores for other covariates)
- What eventually matters is whether you achieve balance
  - Regardless of *how you try to achieve* balance, you need to verify *that you managed to achieve* balance (p.t.o.)

Matching algorithms

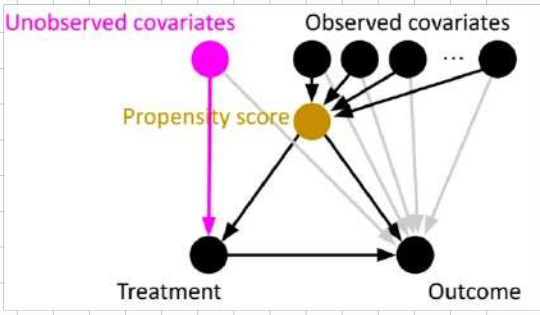
- Goal: Match subjects into pairs (1 treated, 1 control), with identical propensity scores within each pair
- Unlikely that 2 subjects have identical propensity scores
- → Use approximate matching (remember your algo class!)
- Bipartite graph: each subject connected to all other subjects
- Edge weights: absolute (or squared) difference of propensity scores (or other matching criterion)
- Find minimum matching, e.g., via Hungarian algorithm



39

What if naive model wasn't true?

▶ ⇒ Pr(Treated | obs covariates) ≠ Pr(Treated | all covariates)



Sensitivity analysis:

- **Idea:** Quantify the degree to which the naive model may be wrong without you having to change your (causal) conclusions
- Assume that treatment odds of identical-looking subjects (i.e., identical observed covariates **x**) may differ by up to a factor Γ
- Then reason in spirit of proof by contradiction: "To change the conclusions of my study, two identical-looking people (1 treated, 1 control) would have to have hugely different treatment odds (i.e., huge Γ). Common sense (or domain knowledge) suggests that this is not the case, so my conclusions stand."

The sensitivity analysis model

$$\frac{1}{\Gamma} \leq \frac{\pi_k/(1-\pi_k)}{\pi_\ell/(1-\pi_\ell)} \leq \Gamma \text{ whenever } \mathbf{x}_k = \mathbf{x}_\ell. \qquad \Gamma \geq 1.$$

π_ℓ: subject ℓ's (true) probability to treat

- Bounded odds ratio (OR)
  - Reason for using OR:
    OR = Pr(k treated | **either k or ℓ treated**) / Pr(ℓ treated | **either k or ℓ treated**)
- Sensitivity $\Gamma = 1 \rightarrow$ naive model is true
- Sensitivity $\Gamma = 2 \rightarrow$ subject with same observed covariates **x** up to twice as likely to be the one to receive treatment
- Sensitivity $\Gamma = \infty \rightarrow$ void statement (a.k.a. tautology)

$\Gamma \to \infty$ more from different

Example: smoking and lung cancer

- Under naive model: matching on observed covariates gives a very small p-value for the null hypothesis that smoking does not increase lung cancer risk (using an appropriate hypothesis test), i.e., data hard to explain w/o a causal effect
- Tobacco lobby: "The naive model isn't true! There may be hidden (e.g., genetic) correlates that increase both the probability to enjoy smoking and the probability of lung cancer. They, not smoking, cause cancer!"

Example: smoking and lung cancer

- Under sensitivity analysis model, increasing sensitivity Γ increases the p-value for null hypothesis
- Anti-tobacco lobby: "But making $p > 0.05$ would require $\Gamma > 6$; i.e., the odds of being a smoker would need to be six times higher for one of two people with the exact same observed features (age, gender, education, income, ...). It's unlikely that any unobserved covariate would have such a large effect on smoking habits. So smoking causes cancer!"

Two parts: mechanical vs. scientific

- Mechanical part:
  - Create pairs (1 treated + 1 control) with similar observed covariates (using exact or propensity-score matching)
- Scientific (i.e., fun) part:
  - Mitigate concerns that your findings might be caused by unobserved covariates, rather than treatment (e.g., using sensitivity analysis, ad-hoc arguments, natural experiments)

Lecture 7: Supervised Machine Learning:

5. Supervised Learning:

Supervised: We are given input/output samples (X, y) which we relate with a function y = f(X). We would like to "learn" f, and evaluate it on new data. Types:

- Classification: y is discrete (class labels) : Class
- Regression: y is continuous, e.g. linear/logistic regression : regre

Unsupervised: Given only samples X of the data, we compute a function f, such that y = f(X) is a "simpler" representation.

- Clustering: y is discrete : clusters
- y is continuous: Matrix factorization, topic modelling, unsupervised neural networks, word2vec, ... → dimensionality red

Supervised:
1) K-NN : k nearest neighbour
2) Tree-based models: decision tree, random forest.
3) Linear + Logistic : reg.

K Nearest Neighbours KNN:

Given a query item, find the k closest matches in a labelled dataset and return the most frequent label.

The data is the model:

- No training needed.
- Accuracy generally improves with more data.
- Matching is simple and fairly fast if data fits in memory.
- Usually need data in memory but can be run off disk.

Minimal configuration:

- Only parameter is k (number of neighbours)
- But two other choices are important:
  - Weighting of neighbours (e.g. inverse distance) → We can give a higher weight to things w/ more similarity!
  - Similarity metric : What is "close"?

k-NN flavours:

Classification:

- Model is y = f(X), y is from a discrete set (labels).
- Given X, compute y = majority vote of the k nearest neighbours. Can also use a weighted vote* of the neighbours.

Regression:

- Model is y = f(X), y is a real value.
- Given X, compute y = average value of the k nearest neighbours. Can also use a weighted average* of the neighbours.

Weighting function is usually the similarity.

k-NN distance measures and metrics:

This normalizes for length and compares angle at which vectors point.

Euclidean Distance: Simplest, fast to compute

$$d(x,y) = \|x - y\|$$

Cosine Distance: Good for documents, images, etc.

$$d(x,y) = 1 - \frac{x \cdot y}{\|x\|\|y\|}$$

Jaccard Distance: For set data:

(=0 if equal)

$$d(X,Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

Intersection / Union

Hamming Distance: For string data:

$$d(x,y) = \sum_{i=1}^{n} (x_i \neq y_i)$$

Manhattan Distance: Coordinate-wise distance

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

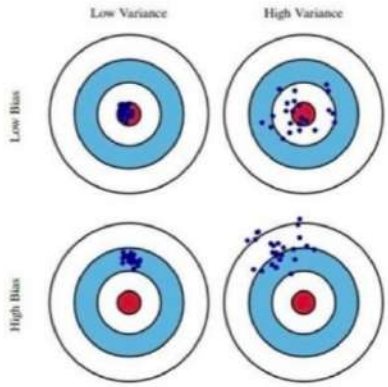
Edit Distance: for strings, especially genetic data.

Mahalanobis Distance: Normalized by the sample covariance matrix – unaffected by coordinate transformations.

Prediction from samples:

- Most datasets are **samples** from an **infinite population**.
- We are most interested in **models of the population**, but we have access only to a **sample** of it.
- For datasets consisting of (X, y): features X, label y. For the true model f: y = f(X) → we train on a training sample D and we denote the model as $f_D(X)$

↳ We want to make predictions on full sample though



Bias and variance:

Our data-generated model $f_D(X)$ is a statistical estimate of the true function f(x). Because of this, it's subject to bias and variance:

- Bias: if we train models $f_D(X)$ on many training sets D, bias is the expected difference between their predictions and the true y's → $Bias = E[f_D(X) - y]$ → How far off we are from the true value
- Variance: if we train models $f_D(X)$ on many training sets D, variance is the variance of the estimates.

$$Variance = E[(f_D(X) - E[f_D(X)])^2]$$ → How different are each estimates.

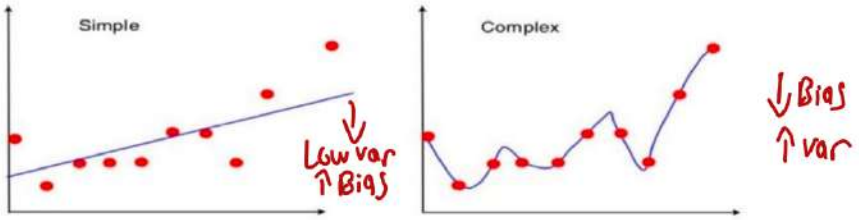
Low variance → data points clustered closely together while bias shifts the data points in on direction from the average.

Trade-off: There is usually a bias-variance trade-off caused by model complexity. → Increasing one will forcibly decrease the other.

- **Complex models** (many parameters) usually have lower bias, but higher variance.
- **Simple models** (few parameters) have higher bias, but lower variance.

⇒ We usually want a mid point!

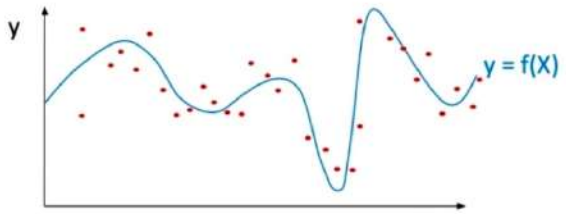
- E.g. A linear model can only fit a straight line. A high-degree polynomial can fit a complex curve. But the polynomial can fit the individual sample, rather than the population. Its shape can vary from sample to sample, so it has high variance.



- The total expected error is: $Error = Bias^2 + Variance$
- Because of the trade-off, we want to balance these two contributions. If variance strongly dominates, there is too much variation between models and causes **over-fitting**. If Bias strongly dominates, the models are not fitting the data well enough and causes **under-fitting**.

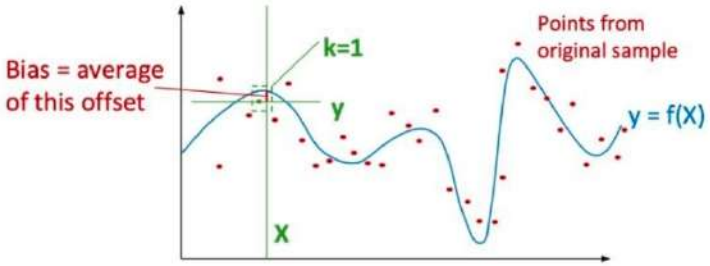
Choosing k for K-NN:

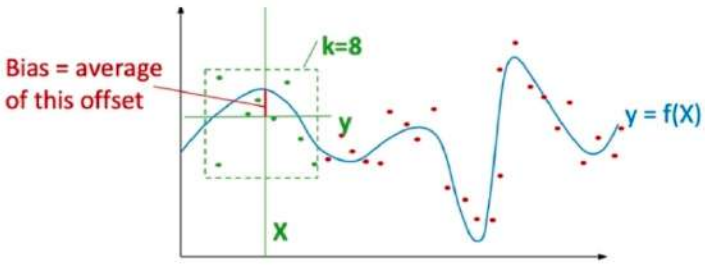
- Small k → low bias, high variance
- Large k → high bias, low variance



▶ We have that will be adding info from neighbours that don't have to do w/ bias ↑.

Assume the real data follows the blue curve, with some mean-zero additive noise. Red points are a data sample. For a small K, the bias is how much we're wrong in a certain direction. So, the proper x value we should have assigned minus what we actually assigned. For a larger value of K we are including more neighbours → the average is "pulled down" so the overall bias is going to be much higher. For variance, the smoothing effect of increasing k reduces the variance but increases the bias.





▶ Ummm... OK

Choosing k in practice:

- **Use leave-one-out cross-validation (LOO):** Break data into train and test subsets, e.g. 80-20 % random split.
- **Predict:** For each point in the training set, predict using the k nearest neighbours from the set of all *other* points in training set. Measure the error rate (classification) or squared error (regression).
- **Tune:** try different values of k, and use the one that gives minimum leave-one-out error
- **Evaluate:** test on the test set to measure performance.

⇒ The 80% is for model selection! ⇒ We iterate through a list of k and for all points in this 80% we make the classification using ≠ k and see which one performs better! ⇒ This k will then be used in an unseen data set to see how new it is!

Core idea of LOO for k-NN (intuition first)

For each data point x_i:

1. Pretend x_i is unseen
2. Predict y_i using its k nearest neighbors **among all other points**
3. Compare prediction to the true y_i

Repeat this for **every point** and average the error.

That's it.

This mirrors the real situation:

> "If I see a new point, I'll predict using nearby points I've already seen."

Curse of dimensionality:

The curse of dimensionality refers to a phenomena that occurs in high dimensions (100s to millions) that does not occur in low-dimensional (e.g. 3-dimensional) space. In particular, data in high dimensions are much sparser (less dense) than data in low dimensions. For kNN, that means there are fewer points that are very close in feature space (very similar), to the point we want to predict. From this perspective, it's surprising that kNN works at all in high dimensions. Luckily real data are not like random points in a high-dimensional cube. Instead they live in **dense clusters** and near **much lower-dimensional surfaces**. Also, points can be very "similar" even if their Euclidean distance is large. E.g. documents with the same few dominant words are likely to be on the same topic (→ use different distance).

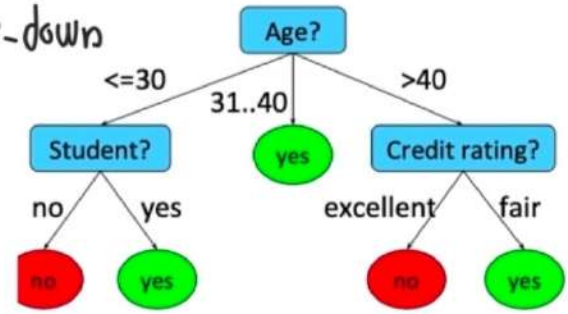
DECISION TREES:

▶ Problem: ▶ NP-Hard

▶ Heuristic: greedy Top-down tree construction + pruning.

Model: flow-chart-like tree structure

- Nodes are tests on a single attribute
- Branches are attribute values
- Leaves are marked with class labels



Induction:

Tree construction (top-down divide-and-conquer strategy)

- At the beginning, all training samples belong to the root
- Examples are partitioned recursively based on selected "most discriminative" attributes
- Discriminative power based on information gain (ID3/C4.5) or Gini impurity (CART)

Partitioning stops if:

- All samples belong to the same class → assign the class label to the leaf
- There are no attributes left → majority voting to assign the class label to the leaf

Attention: Need to know how to build a decision tree. Here done in **BFS** method.

Attribute selection:

At a given branch in the tree, the set of samples S to be classified has P positive and N negative samples. The amount of entropy in the set S is:

$$H(P, N) = -\frac{P}{P+N}\log_2\frac{P}{P+N} - \frac{N}{P+N}\log_2\frac{N}{P+N}$$

Note that:

- If P=0 (or N=0), H(P, N) = 0 → no uncertainty
- If P=N, H(P, N) = 1 → max uncertainty

(See slides on how attributes are selected).

Attribute selection

$H_S = H(9, 5) = 0.94$

Here the positive are yes (buys pc) and negative (=no) (doesn't buy!)

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

Age [<=30] H(2, 3) = 0.97 — Income [high] H(2, 2) = 1

Age [31...40] H(4, 0) = 0 — Income [med] H(4, 2) = 0.92

Age [>40] H(3, 2) = 0.97 — Income [low] H(3, 1) = 0.81

Student [yes] H(6, 1) = 0.59 — Rating [fair] H(6, 2) = 0.81

Student [no] H(3, 4) = 0.98 — Rating [exc] H(3, 3) = 1

Attribute selection

Attribute A partitions S into $S_1, S_2, \ldots S_v$

Entropy of attribute A is $H(A) = \sum_{i=1}^{v} \frac{P_i + N_i}{P+N} H(P_i, N_i)$

The *information gain* obtained by splitting S using A is

$Gain(A) = H(P, N) - H(A)$ (Before split → after split!)

Gain(Age) = 0.94 – 0.69 = 0.25 ← split on age

Gain(Income) = 0.94 – 0.91 = 0.03

Gain(Student) = 0.94 – 0.78 = 0.16

Gain(Rating) = 0.94 – 0.89 = 0.05

Attribute selection

$H_S = H(9, 5) = 0.94$

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

Lower entropy ⇒ we choose it before!

$H_{Age} = p([<=30]) \cdot H(2, 3) + p([31...40]) \cdot H(4, 0) + p([>40]) \cdot H(3, 2) =$
$= 5/14 \cdot 0.97 + 4/14 \cdot 0 + 5/14 \cdot 0.97 = 0.69$

$H_{Income} = p([high]) \cdot H(2, 2) + p([med]) \cdot H(4, 2) + p([low]) \cdot H(3, 1) =$
$= 4/14 \cdot 1 + 6/14 \cdot 0.92 + 4/14 \cdot 0.81 = 0.91$

$H_{Student} = p([yes]) \cdot H(6, 1) + p([no]) \cdot H(3, 4) = 7/14 \cdot 0.59 + 7/14 \cdot 0.98 = 0.78$

$H_{Rating} = p([fair]) \cdot H(6, 2) + p([exc]) \cdot H(3, 3) = 8/14 \cdot 0.81 + 6/14 \cdot 1 = 0.89$

We calculate the entropy for each decision and the one that has lowest ⇒ choose!

⇒ we recalculate the gain and reassign!

Pruning:

The construction phase does not filter out noise → **overfitting.** Many possible pruning strategies

- Stop partitioning a node when the corresponding number of samples assigned to a leaf goes below a threshold.
- Bottom-up cross validation: Build the full tree and replace nodes with leaves labelled with the majority class if classification accuracy on **validation set (!)** does not get worse this way.

Decision trees are just an example of classification algorithm an may be not the best one:

- Sensitive to small perturbation in the data
- Tend to overfit
- Non-incremental: Need to be re-trained from scratch if new training data becomes available
- As tree depth increases, bias decreases and variance generally increases. Why? (Hint: think about k-NN)

Not for K neighbors!!!

Bias decreases with tree depth

Variance increases with tree depth

Ensemble method:

Are like crowdsourced machine learning algorithms:

- Take a collection of simple or *weak* learners
- Combine their results to make a single, better learner

Types:

- Bagging: train learners in parallel on different samples of the data, then combine by voting (discrete output) or by averaging (continuous output).
- Stacking: combine model outputs using a second-stage learner like linear regression.
- Boosting: train learner again, but after filtering/weighting samples based on output of previous train/test runs.

Random Forests:

- Grow K trees on datasets sampled from the original dataset (size N) with replacement (bootstrap samples), p = number of features.
  - Draw K bootstrap samples of size N
  - Grow each Decision Tree, by selecting a random set of m out of p features at each node and choosing the best feature to split on.
  - Aggregate the predictions of the trees (most popular vote, or average) to produce the final class (example of bagging).
- Principles: we want to take a vote between different learners, so we don't want the models to be too similar. These two criteria ensure diversity in the individual trees. Draw K bootstrap samples of size N:
  - Each tree is trained on different data.
  - Grow a Decision Tree, by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
  - Corresponding nodes in different trees (usually) can't use the same feature to split.

Pros/Cons:

+ Very popular in practice, probably the most popular classifier for dense data (up to a few thousand features)
+ Easy to implement (simply train many normal decision trees)
+ Parallelizes easily ≈ Faster than boosting decision trees!
- Needs many passes over the data – at least the max depth of the trees (can be helped by boosted trees).

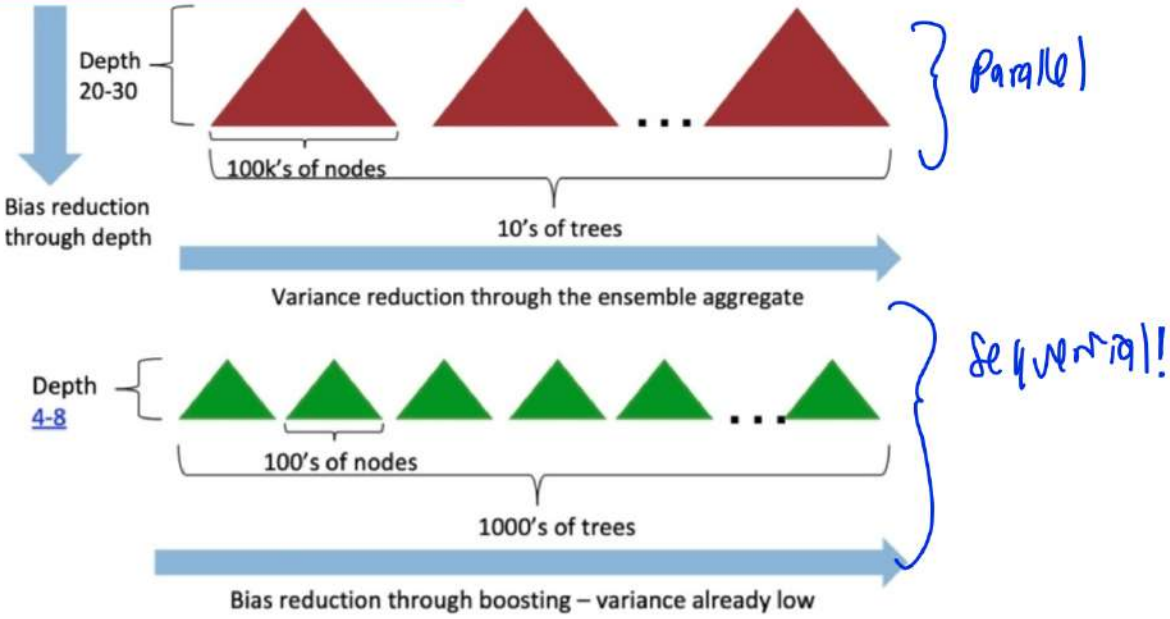
Boosted decision trees:

- A more recent alternative to Random Forests
- In contrast to RFs whose trees are trained independently, BDT trees are trained sequentially by boosting: Each tree is trained to predict error residuals of previous trees (→ bias reduction).
- Both RF and boosted trees can produce very high-quality models. Superiority of one method or the other is very dataset dependent.
- Resource requirements are very different as well, so it's actually non-trivial to compare the methods.

Random Forest vs Boosted Trees: Forest: big ass trees and around 10.

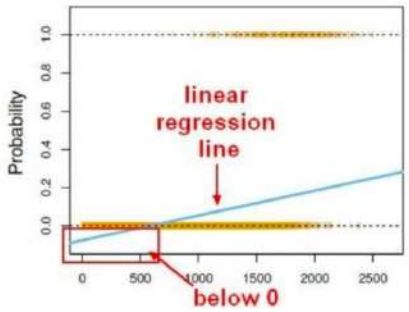
Geometry of the methods is very different. Random forest uses 10's of deep large trees while Boosted trees use 1000's of shallow small trees: → variance small by default!

- RF training embarrassingly parallel, can be very fast
- Evaluation of trees (runtime) also much faster for RFs



LINEAR AND LOGISTIC REGRESSION:

We want to find the "best" line (linear function y=f(X)) to explain the data. The most common measure of fit between the line and the data is the least-squares fit.



Modelling binary events:

E.g., X: student features; y: did student pass ADA?

- Desired output: $f(X)$ = probability of passing ADA, given feats X
- Problem with linear regression: $f(X)$ can be below 0 or above 1

Trick: don't deal with probabilities, which range from 0 to 1, but with log odds, which range from -inf to +inf

- Probability y ⇔ odds $y/(1-y)$ ⇔ *log odds* $log[y/(1-y)]$
- Model log odds as a linear function of X

Logistic regression:

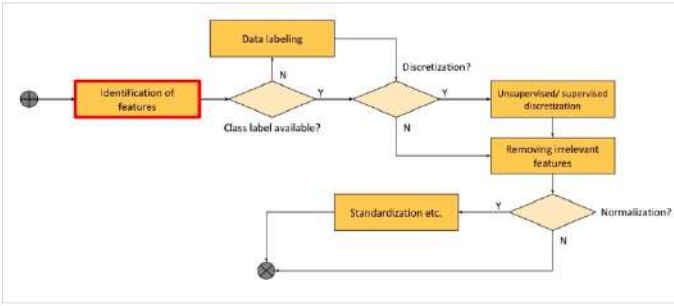
- Model log odds as a linear function of X : $\beta^T X = log[y/(1-y)]$
- Solve for y: $y = 1 / (1 + exp(-\beta TX))$
- Finding best model β via maximum likelihood:
  - Don't use square loss as in linear regression
  - Use cross-entropy loss instead

Overfitting:

The more features the better? **NO!** More features mean less bias, but more variance and can cause overfitting. Carefully selected features can improve model accuracy

- E.g., keep features that correlate with the label y
- Forward/backward feature selection
- Regularization (e.g., penalize norm of weight vector)

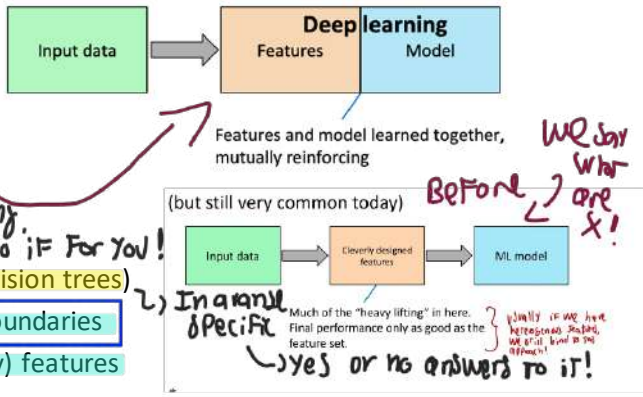
8. Applied Machine Learning:



DATA COLLECTION: : Collect data to train our models with!

Features:

- Continuous (e.g., height, temperature ...)
- Ordinal (e.g., "agree", "don't care", "disagree" ...)
- Categorical (e.g., color, gender ...)
- New features can be generated from **simple stats:** *Feature engineering* is considered a form of art; therefore, it is sometimes useful to find what other people did for similar problems
- Some classifiers require categorical features → **Discretization**
- Since 2012 usually that features, and model are learned together, mutually reinforcing.



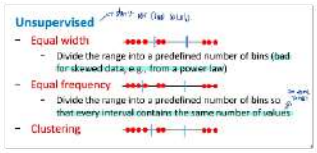
▶ Labels: (collection data =) easy => labelling; difficult and time consuming
▶ Usually w/ crowd sourcing: you pay people to do it for you!

Discretization:

- Some classifiers want discrete features (e.g., simplest kinds of decision trees)
- Discrete features let a linear classifier learn non-linear decision boundaries
- Certain feature selection methods require discrete (or even binary) features

Unsupervised:

- Equal width: Divide the range into a predefined number of bins (bad for skewed data, e.g., from a power law)
- Equal frequency: Divide the range into a predefined number of bins so that every interval contains the same number of values
- Clustering



Supervised:

- Start with very fine-grained discretization
- Test the hypothesis that membership in two adjacent intervals of a feature is independent of the class. If they are independent, they should be merged. Otherwise they should remain separate.

▶ Proceed recursively.

↳ X^2 test: we test if distribution of data doesn't change when merging => if change => merge!

Feature selection:

Reducing the number of N features to a subset of the best size M < N.

! 2^N possible subsets...

+ Efficient.
+ Interpretation.
+ ↓ overfitting.

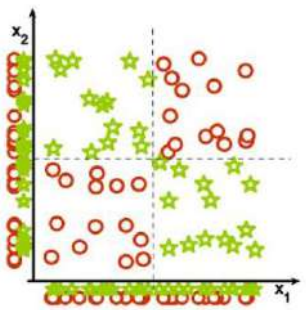
1) **Filtering as preprocessing: offline feature selection**

Rank features according to their individual predictive power; select the best ones

- Pros: Independent of the classifier (performed only once)
- Cons:
  - Independent of the classifier (ignore interaction with the classifier)
  - Assume features are independent

Ranking of features:

- Continuous features (and ideally labels): correlation coefficient (Pearson, linear rel.!) => to see if they're linearly correlated
- Categorical features and labels: mutual information. X^2 method: test whether feature is independent of label. Difference with respect to mutual information → the chi-square test checks the independence of the class and feature, without indicating the strength or direction of any existing relationship (you just get a significance, a.k.a. p-value) => to see if independence!



Beware: collectively relevant features may look individually irrelevant!

Iterative feature selection: online feature selection

- Forward feature selection: greedily add features; evaluate on validation dataset; stop when no improvement
  - Pros: Interact with the classifier + No feature-independence assumption
  - Cons: Computationally intensive
- Backward selection (a.k.a. ablation): greedily remove features; evaluate on validation dataset; stop when no improvement
  - Pros: Interact with the classifier + no feature-independence assumption
  - Cons: Computationally intensive

Feature normalization:

- Some classifiers do not manage well features with very different scales
- Features with large values dominate the others, and the classifier tends to over-optimize them
- Even single feature may span many orders of magnitude: City size (most cities small, some huge)

Logarithmic scaling:

- Consider order of magnitude, rather than direct value
- Good for heavy-tailed features (e.g., from power laws)

Min-max scaling:

$$x_i' = (x_i - min_{x_i})/(Max_{x_i} - min_{x_i})$$

The new feature $x_{i'}$ lies in the interval [0,1]

Standardization:

$$x_i' = (x_i - \mu_i)/\sigma_i$$

Where μ_i is the mean value of feature x_i, and σ_i is the standard deviation. The new feature x_i' has mean 0 and standard deviation 1.

Dangers of standardization and scaling:

- Standardization: we assume that the data has been generated by a Gaussian. So, it uses mean and std → not meaningful for heavy-tailed data (may be mitigated by log-scaling)
- Min-max scaling: If the data has outliers, they scale the typical values to a very small interval.

MODEL SELECTION:

→ k-Fold Crossvalidation!

- High Level: Need to choose the type of the model (e.g Logistic regression, decision trees, random forest, etc.)
- Low-level: Usually a classifier has some "hyperparameters" to be tuned. Set of features to include:
  - Threshold (e.g., logistic regression)
  - Distance function (e.g., k-NN)
  - Number of neighbors (e.g., k-NN)
  - Number of trees (e.g., random forest)
  - Regularization parameter

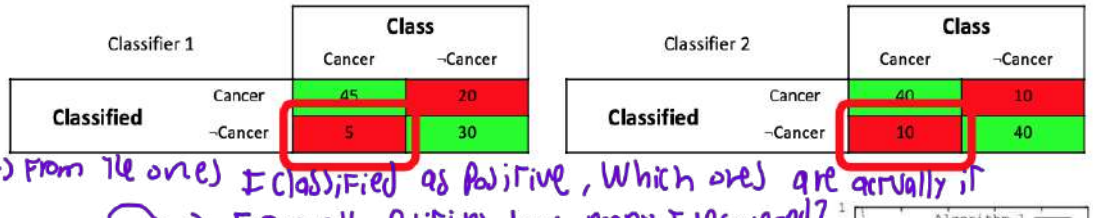
▶ Select the loss function { Categorical {0-1} or Continuous SE, absolute error.

Performance metrics for binary classification:

For categorical binary classification, the usual metrics are based on the **confusion matrix,** which has 4 values:

- True Positives (positive examples classified as positive)
- True Negatives (negative examples classified as negative)
- False Positives (negative examples classified as positive)
- False Negatives (positive examples classified as negative)

Accuracy: $A = (TP + TN)/N$ → Appropriate metric when classes are not skewed and errors (FP, FN) have the same importance. Example of skewed classifier:

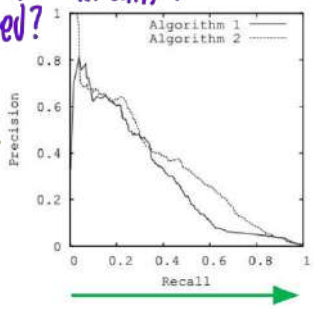


→ From the ones I classified as positive, which ones are actually it

Precision and Recall: → From all positives, how many I recovered?

Can be weighted if of what we want.

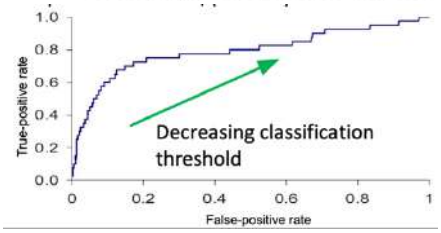
$P = TP / (TP + FP)$ and $R = TP / (TP + FN)$ → Fscore $F1 = 2\,P\,R\,/(P + R)$ → Harmonic mean! => useful to have a single metric



Receiver-operating characteristics (ROC) plots:

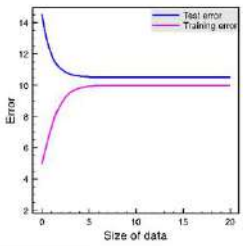
Y-axis is the recall and X-axis the true positive rate. ROC AUC is the "area under the curve", a single number that captures the overall quality of the classifier. It should be between 0.5 (random classifier) and 1.0 (perfect).

Decreasing classification threshold

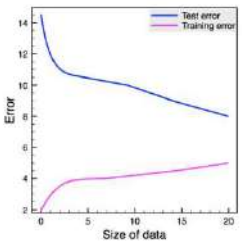


Bias and Variance:

Bias and variance can be assessed by comparing the error metric on the training set and the test set → always **plot learning curves** (training set size vs. error). In the ideal case, we want low bias (small training error) and low variance (small test error).



High bias



High variance

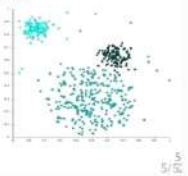
- TPR = recall = $\frac{TP}{TP+FN}$
- FNR = miss rate = $\frac{FN}{TP+FN}$
- TNR = specificity = $\frac{TN}{TN+FP}$
- FPR = fallout = $\frac{FP}{TN+FP}$

Lecture 9: The clustering problem:

- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, such that
  - members of a cluster are close (i.e., similar) to each other
  - members of different clusters are far apart from each other
- **Usually:**
  - Points live in a high-dimensional space
  - Similarity is defined via a distance measure
    - Euclidean, cosine, Jaccard, edit distance, …

(cf. lecture 7)

More similar = less distance: equivalent.

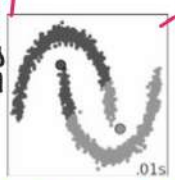


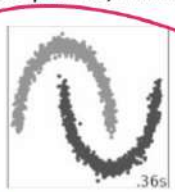
Characteristics:

Quantitative: scalability (many samples), dimensionality (many features)

Qualitative: types of features (numerical, categorical, etc.), type of shapes (polyhedra, hyperplanes, manifolds, etc.)

▶ Not convex => k-means is only useful for convex shapes.





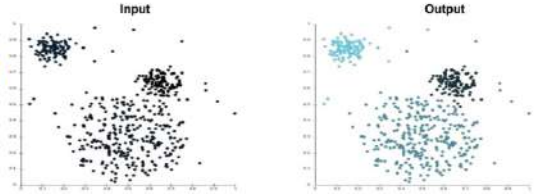
▶ Not appropriate w/ k-means => usually works w euclidian distances and shapes.

Characteristics:

Robustness: sensitivity to noise and outliers, sensitivity to the processing order

User interaction: incorporation of user constraints (e.g., number of clusters, max size of clusters), interpretability and usability

▶ Example

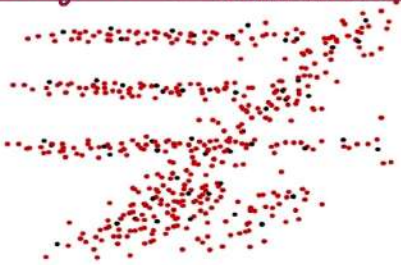


Note: Above is 2D; real scenarios often much more high-dimensional, e.g., 10,000-dimensional for 100x100 images.

Some use cases for clustering:

- Data exploration (especially for high-dimensional data, where visualization fails)
- Partitioning of data for more fine-grained subsequent analysis
- Marketing: building personas
- Supporting data labeling for supervised learning
- Supporting feature discretization for supervised learning (cf. lecture 8)
- Data compression (next slide)
- …

Clustering For Condensation/Compression:



▶ Basically replace our data by centroids and only work w/ this!

Here we don't require that clusters extract meaningful structure, but that they give a coarse-grained version of the data.

11

Cluster bias:

▶ We tend to see the world through categories ... => but sometimes we cluster when we shouldn't.

- **Clustering is used more than it should be**, because people assume an underlying domain has discrete classes in it
  - Especially true for characteristics of people, e.g., Myers-Briggs personality types like "ENTP".
- In reality the underlying data is often **continuous**.
- In such cases, continuous models (e.g., matrix factorization, "soft" clustering) tend to do better (cf. next slide)

▶ For continous data it is better to use other methods (like dimensionality reduction) than clustering ! => Ex: netflix matrix factorisation.

▶ With "continous" they mean data varies as a gradient : ~~~~~ instead of C1 C2 C3 ↓ ↓ ↓ ~ ~ ~

=> In these cases, PCA or TSNE are better for dimensionality reduction!

Terminology:

▶ Hierarchical : Clusters form a tree-shaped hierarchy. Can be computed bottom-up or top-down.
▶ Flat clustering: no inter-cluster structure
▶ Hard clustering: items assigned to a unique cluster
▶ Soft clustering: cluster membership is a proba.

Hard problem:

▶ In 2-D : easy but not in high dim...
▶ High-dimensional spaces are different ! => many dim => volume grows exponentially => space is sparse => clusters are less right ! => difficult!

Example of problems:

▶ Galaxies: 2 billion objects (→ sky.) clustered by 7-dimensional.
▶ Movies: Represent movies by a set of clusters who watched it => similar movies have similar sets of customers.

* 1 Dim = 1 customer * A movie: $(x_1, \dots, x_n)$: $X_i = 1$ iff i-th customer watched movie =>
* Find clusters of similar movies.

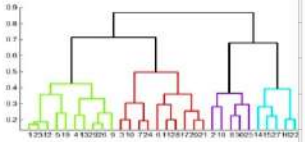
▶ Documents: a document is a vector $(x_1, x_2, \dots, x_n)$ where $x_i = 1$ iff i-th word appears in doc.
* Documents with similar sets are about same topic.

▶ Cells: Represent cells as expression vectors of genes => cluster by similar expression => (possibility to do further clustering inside).

Distances: ▶ Sets as vectors: 1) Euclidian: better for magnitude (real numbers) 2) Cosine: better for angle (documents / words).
▶ Sets: Jaccard similarity index. (1 - Inter/union)

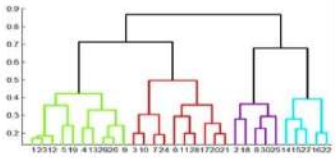
Clustering Methods:

- **Hierarchical methods:**
  - **Agglomerative** (bottom-up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one
  - **Divisive** (top-down):
    - Start with one cluster and recursively split it
- **Flat methods (a.k.a. point-assignment methods):**
  - Maintain a set of clusters
  - Assign points to "nearest" cluster; recompute clusters; repeat



Agglomerative hierarchical clustering

- **Key operation:**
  Repeatedly combine two nearest clusters



- **Three important questions:**
  (1) How to represent a cluster of more than one point?
  (2) How to determine the "nearness" of clusters?
  (3) When to stop combining clusters?

Agglomerative hierarchical clustering

Key operation: Repeatedly combine two nearest clusters

(1) How to represent a cluster of many points?

- Euclidean case: represent a cluster via its **centroid** = average of points in cluster
- What about non-Euclidean case?

(2) How to determine "nearness" of clusters?

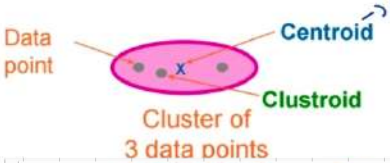
- Euclidean case: distance between clusters = distance between centroids
- What about non-Euclidean case?

Non-Euclidean case: clustroids

(1) How to represent a cluster of many points?

clustroid = actual data point that is **"closest"** to the other points

- **Possible meanings of "closest":**
  - Smallest average distance to other points (a.k.a. **medoid**)
  - Smallest sum of squares of distances to other points
  - Smallest maximum distance to other points



Centroid is the avg. of all data points in the cluster. This means centroid is an "artificial" point.

Clustroid is an **existing** data point that is "closest" to all other points in the cluster.

Non-Euclidean case: cluster "nearness"

(2) How do you determine the "nearness" of clusters?

- Approach 1:
  Intercluster distance = minimum of the distances between any two points, one from each cluster; or average of distances; or distance between clustroids; etc.
- Approach 2:
  Pick a notion of "**cohesion**" ("tightness") of a cluster
  Then: nearness of clusters = cohesion of their *union*

Cohesion

- **Approach 2.1:** Use the **diameter** of the merged cluster = maximum distance between points in the merged cluster
- **Approach 2.2:** Use the **average distance** between points in the merged cluster

▶ Branching points in dendogram with N data points: N-1 ⇒ N points ⇒ merge and merge ! and then you have final outcome !

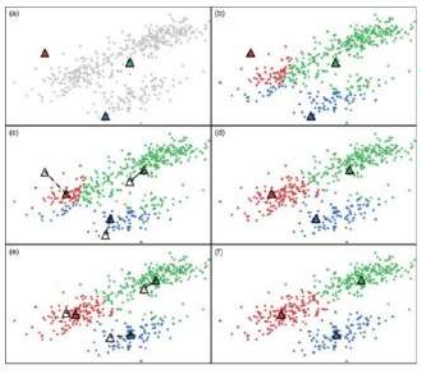
Implementation of hierarchical clustering:

1) Naive implementation of hierarchical clustering:
   - Compute pairwise distances between all pairs of clusters, then merge.
   - $O(N^3)$ where N = data point.

2) Careful implementation using priority queue: Can reduce time to $O(N^2 \log N)$... ⇒ still to expensive for big datasets.

K-means:

- **Goal:** assign each data point to one of k clusters such that the total distance of points to their centroids is minimized
- Solved by a simple greedy algorithm (Lloyd's algorithm): Locally minimize the "distance" (usually squared Euclidean distance) from data points to their respective centroids:
  - Find the closest cluster centroid for each data point, and assign the point to that cluster.
  - Recompute the cluster centroid (the mean of data points in the cluster) for each cluster.



How long to iterate?

- For fixed number of iterations
- or until no change in assignments (guaranteed to happen)
- or until only small change in cluster "tightness" (sum of [squared] distances from points to centroids)

▶ Decide K in advance ⇒ Guaranteed to converge !

K-means Initialization:

We need to pick some points for the first round of the algorithm:

- **Random sample:** Pick a random subset of k points from the dataset. → Luck in play
- **K-Means++:** Iteratively construct a random sample with good spacing across the dataset. ▶ Assignment is proportional to the distance to last chosen one.

Note: Finding an optimal k-means clustering is NP-hard. The above help avoid bad configurations. → like decision tree

K-means ++:

Start: Choose first cluster center at random from the data points

Iterate:

- For every remaining data point x, compute the distance $D(x)$ from x to the closest previously selected cluster center.
- Choose a remaining point x randomly with probability proportional to $D(x)^2$, and make it a new cluster center.

→ Have a centroid far from it!

Intuitively, this finds a sample of widely-spaced points from dense regions of the data space, avoiding "collapsing" of the clustering into a few internal centers.

K-means Properties:

- Greedy algorithm with random initialization – **solution may be suboptimal** & vary significantly with different initial points
- Very simple convergence proofs
- **Performance is O(*nk*) per iteration** — not bad, and can be heuristically improved
  n = number of points in the dataset, k = number clusters
- Many variants, e.g.
  - Fixed-size clusters
  - Soft clustering
- Works well for data condensation/compression

42

K-means drawbacks:

Often terminates at a **local but non-global optimum** (mitigated by smart initialization such k-means++, or by re-running multiple times with different initializations) ▶ We compare different runs to see.

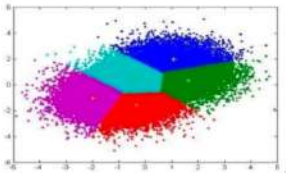
Requires the notion of a **mean**

Requires specifying k (number of clusters) in advance (also exist k-medoid)

Doesn't handle **noisy data and outliers** well

Clusters **only** have **convex shapes**

→ Fundamental part of the algo!
→ not good for non-convex!



Select good # clusters: k:

For k = 1, 2, 3, …

Run k-means with k clusters

For each data point i, compute "silhouette width"

S = average of $s(i)$ over all i

Plot S against k

Pick k for which S is greatest

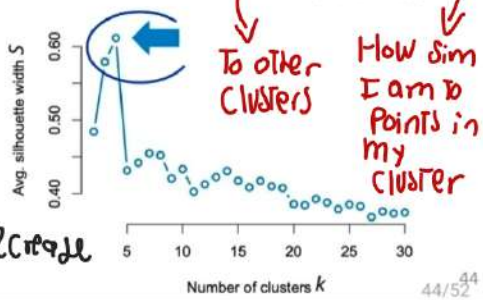
$b(i)$: avg. distance to points in closest other cluster

$a(i)$: avg. distance to points in own cluster

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

To other clusters

How sim I am to points in my cluster

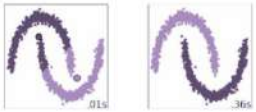


+ Also use elbow method: where validation error doesn't decrease as much.

44/52

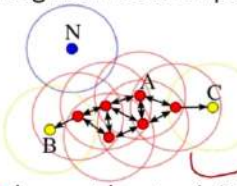
DBSCAN: ▶ No convex shapes:

- "Density-based spatial clustering of applications with noise"
- **Motivation:** centroid-based clustering methods like k-means favor clusters that are spherical, and have great difficulty with anything else



- But with real data we often have:

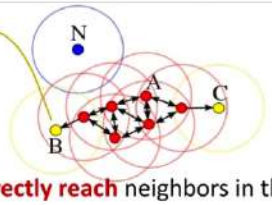
- DBSCAN performs density-based clustering, and follows the shape of dense neighborhoods of points.



At least three points in each sphere

- **Def.: core points** have at least *minPts* neighbors in a sphere of diameter ε around them.
- The red points here are core points with at least *minPts* = 3 neighbors in an ε-sphere around them.

46



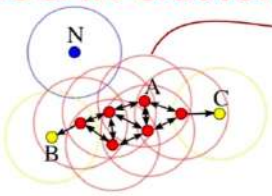
Like B or C From the red points

More **definitions (!):**

- Core points can **directly reach** neighbors in their ε-sphere
- From non-core points, no other points can be reached
- Point q is **density-reachable** from p if there is a series of points $p = p_1, \ldots, p_n = q$ such that p_{i+1} is directly reachable from p_i
- All points not density-reachable from any other points are **outliers/noise**

47/52

DBSCAN clusters



Even more **definitions:**

- Points p, q are **density-connected** if there is a point o such that both p and q are density-reachable from o.
- A **cluster** is a set of points that are **mutually density-connected**.
- That is, if a point is density-reachable from a cluster point, it is part of the cluster as well.
- In the above figure, red points are mutually density-reachable; B and C are density-connected; N is an outlier.

48/52

Algorithm (Intuition):

DBSCAN works by **looking for dense regions of points** in the data. It starts from an arbitrary point and checks whether this point lies in a sufficiently crowded neighborhood.

If the point is **not in a dense area**, it is temporarily considered noise and the algorithm moves on.

If the point **is in a dense area**, DBSCAN declares that point as the **seed of a new cluster**. From there, the cluster is grown outward by repeatedly adding all nearby points that are directly or indirectly reachable through dense neighborhoods.

Whenever DBSCAN encounters another dense point during this growth process, it keeps expanding the cluster further from that point as well. Points that are not dense themselves but lie close to dense points are absorbed into the cluster, even though they do not trigger further expansion.

This expansion continues until no more points can be reached without leaving dense regions. At that moment, the cluster is complete. DBSCAN then resumes scanning the dataset to find another dense region and repeats the process.

Any point that never becomes part of a cluster through this density-based expansion is ultimately labeled as **noise**.

Performance:

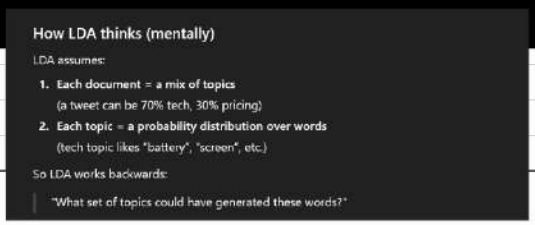
- DBSCAN uses all-pairs point distances, but using an efficient indexing structure ⇒ finding neighbors in ε-sphere takes only $O(\log n)$ time!
- Total time $\approx O(n \log n)$ time! For huge things becomes worse though :/.

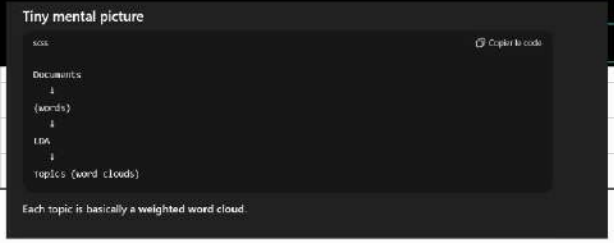
Question 1: Which of the following real-world ML applications is not unsupervised learning?

1. Netflix matrix factorization pipeline to discover users with similar interests
2. Speaker recognition (recognition of the identity of who is talking) in phones and smart assistant devices
3. LDA topic modeling on Twitter content to discover customers' opinions about a product
4. K-means clustering of Web domains

3 IS NOT. It is the Second one (voice recognition cuz we need the samples of which voice belongs to who). They didn't mean LDA linear discriminant anylysis but Latent Dirichlet Allocation (LDA) (which they didn't mention in the curse):

→ Basically :)





Lectures 10 and 11: Handling Text

▶ We need to derive cleaned data! ⇒ much cleaned data from uncleaned

From what we're looking for!

Typical Tasks:

Document Retrieval:

- Given a document collection and query document (a corpus; short string: that ranks important document ⇒)
- Task: rank all docs in collection by similarity to query
- Tool: KNN → define a distance function between documents and find the docs with smallest distance to q

▶ Important for libraries and web search
▶ Query: "What is ML?" ⇒ Task: look for document that answer ⇒ talk about it! ⇒ Sort documents by distance (similarity) to q: Closest first!

Hard part: ▶ Craft/learn a distance function (and scale it to WWW) ⇐⇒ need to measure similarity.

Document Classification:

- Given a document d and a set of classes (news, sports, Tech, music, romance)
- Task: decide to which one of the classes d belongs → example find articles about sports events
- Tool: Supervised learning classifier based on the labelled docs (kNN, logistic regression, decision tree, etc)

NN...

▶ Obtain docs, label to class, represent as feature vector, train classifier

Sentiment Analysis:

- Given: document d ▶ Infer what people think about a product from text, eg.
- Task: sentiment score capturing how positive/negative d is
- Tool: supervised learning (regression/classification) → same setup as for document classification. Learn from already labelled data. ▶ Label w/ "Ground Truth - sentiment scores" ⇒ train: kNN, Logistic...

Topic Detection:

- Given: unlabeled document collection
- Task: determine a set of prevalent topics in the docs and determine for each document to which topics it belongs
- Example: detection of trending topics in social media
- Tool: clustering → represent documents as feature vectors. Run hierarchical, point-assignment clustering.

▶ Don't scale well for big data :(

(agglo/divisive) (k means/ DBSCAN) (or matrix Factorization)

Feature Vectors:

▶ How do we present texts as feature vectors?

Nearly all ML methods work with feature vectors. Text is not immediately a feature vector:

- Variable length
- Even for fixed length (e.g., tweet...): Positions don't correspond to meaningful features

▶ And sometimes the meanings of words can vary in context.

Need to transform arbitrarily long string to fixed-length vector:

- Traditional and vetted: bag of words
- More recent: learn mapping from string to dense vector as "word embeddings"

{ More similar, closer embedding vectors.

Bag of Words:

Bag == multiset: (counts how many times words occur)

- Keep multiplicity of words
- Don't keep order of words
- E.g., document "what you see is what you get" → bag of words {get:1, is:1, see:1, what:2, you:2}

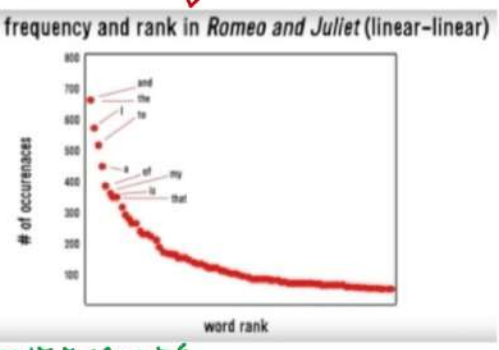
▶ We only get then frequency of each word!

To have fixed-length representation for all documents:

- One entry for each unique word in vocabulary ▶ we define a priori a vocabulary of fixed sized. ▶ Count how many words are there from it for each document
- Bag-of-word vectors are very high-dimensional (typically 1e5 or 1e6) and very **sparse** ! Lots of memory ∴
- E.g., above: [0...0 1 0...0 1 0...0 1 0...0 2 0...0 2 0...0]

Zipf's law: ▶ A power law. $P(w_i) \propto 1/i$

Take all the words in a document collection and plot them based on their frequency. By looking at the probability that words occur scales inversely with its rank. So, a lot of words are going to be extremely rare to find and most of the columns in the document matrix are going to be 0 → not an efficient representation. "Second most frequent word is half as likely to appear as the first most frequent"



Matrix representation:

Combine document vectors as rows in a matrix → one row per doc and one column per word in vocabulary. This matrix is huge! E.g., Wikipedia: 5M docs, 2M words → 10 trillion entries. Use a sparse matrix format:

- Triples: (doc_idx, word_idx, count)
- With matrix representation, you're ready to use any ML model . . . (Almost! => we need more tricks to optimize performance). Else: Garbage in => Garbage out!

Example of format:

Step 2: Dense Bag-of-Words matrix

Rows = documents
Columns = words
Values = word counts

doc \ word	I	like	cats	dogs	milk
Doc 0	1	1	1	0	0
Doc 1	1	1	0	1	0
Doc 2	0	1	1	0	1

Same example in sparse (triplet) format

We store triples:

```
scss
(doc_index, word_index, count)
```

From the dense matrix above:

Non-zero entries only:

```
arduino
(0, 0, 1)   # Doc 0 has "I" once
(0, 1, 1)   # Doc 0 has "like" once
(0, 2, 1)   # Doc 0 has "cats" once

(1, 0, 1)   # Doc 1 has "I"
(1, 1, 1)   # Doc 1 has "like"
(1, 3, 1)   # Doc 1 has "dogs"

(2, 1, 1)   # Doc 2 has "like"
(2, 2, 1)   # Doc 2 has "cats"
(2, 4, 1)   # Doc 2 has "milk"
```

BAG OF TRICKS FOR BAGS OF WORDS, PRE-PROCESSING:

Character encoding:

Need to map binary information to text characters because algorithms deal with binary format.

- Mapping from (abstract) characters to bytes
- Old School: ASCII, Latin-1 → 2^8 characters
- New school: Unicode (e.g., UTF-8, UTF-16, UTF-32) ▶ We need to know which one were used!
- Reading text from file: need to read with encoding that was used to write file. Especially important for non-English text: à, ê, ü, ß, ...
- Writing to file: Always use UTF-16; hard-code the output format! Get in the habit of specifically being explicit about the encoding.

```
file = codecs.open("temp", "w", "utf-8")
file.write(codecs.BOM_UTF8)
file.close()
```

Language identification:

- Typically, you're interested in text from a single language but increasingly, content is multilingual (e.g., Twitter)
- Ideally, language code is specified (e.g., headers in HTML; JSON field in Twitter API results). But not always...
- There are great libraries: most commonly based on letter trigrams → go through the entire corpus with corpus size of 3 and count all the sequences of 3 characters. Based on the distribution of frequencies of these 3 characters we can get a pretty good estimation. (e.g., "eau", "ghi", "ijs", "sch", "eiß", "ção"). This is much harder if you messed up character encoding...

Tokenization:

Chop the text into pieces:

- Maps character string into sequence of tokens (≈ words). E.g., "Hello! How are you?" → Hello_!_How_are_you_?
- Tempting to do this yourself by splitting at whitespaces and punctuation. But many corner cases: here we split on the wrong places because of the punctation. The dot of Mr. is not the end of a sentence.
  "Hello, Mr. President! How are you?! :-)"
  → Hello_,_Mr._President_!_How_are_you_?!_:-)
- Don't do it yourself, use libraries instead. Python: nltk; Java: Stanford CoreNLP
- Optimal tokenizer is different for different languages → so punctuation in between words for example (e.g., Swedish "Saint Peter" → "S:t Peter"), but English tokenizer often good enough → but need to use specific rules that make sense with the context. → Example: Advanced models can split complex words in german !

Stopword removal:

- Very frequent, "small" words carry little information for most tasks and can "drown out" information contained in real content words. E.g., "a", "the", "is", "you", "I", punctuation marks.
- Many stopword lists online but be careful! → ▶ Try to adapt it to dataset.
  - Different tasks require removing different stopwords
  - Good heuristic: remove words appearing in at least p% of all documents (but what should p be...?) → CVs To see Which one is better!
  - Sometimes stopword removal hurts! Author identification, psychological modelling; punctuation can be useful as well: e.g., "!!!", ":-)" → look at how can the writing style and use of different words correspond to documents with unknown authors. So, content can be very different but it's hard to hide writing style.

Casefolding:

E.g., "I love yams. Yams are yummy."

- Should "yams" and "Yams" really be different features?
- Simple solution: casefolding → make everything lower-case ("casefolding")
- But then: "I'd rather have an apple than an Apple."
- In practice (especially when dataset is large), typically best to **not** do casefolding → Because of These!
- But when dataset is small, might help because less sparsity

Stemming:

Stemming:

E.g., "walking", "walks", "walked" → "walk"
"business", "busy" → "busi"

- Map different forms of same word to same, normalized form, by stripping affixes
- Typically done in hacky, heuristic way
- Pro: decreases sparsity in bag-of-words matrix
- Con: discards information. E.g., "business" vs. "busy"; "operating" (as in "op. system")
- In English (esp. with big data) typically not done anymore
- Still very useful in morphologically richer languages (e.g., German, Finnish, Bantu languages)

Lemmatization: Basically putting things in context!

E.g., "U.S.A.", "US" → "United States"
"Grüße", "Gruesse" → "Grüße"
"You **lie** in the grass" vs. "You **lie** to me"

Lemmatization == stemming++. Nicer way of doing stemming → performs better in almost all cases.

- Map tokens to lexicon entries
- Frequently omitted, as it requires complete lexicon and complex mapping rules
- Especially hard for non-English
- Usually we should look at the semantics of the context of the sentences to understand different uses. So, lemmatization is easier for documents than small unique sentences.

Tokens vs n-grams:

- So far: bag-of-words matrix where rows = documents and columns = tokens (a.k.a. unigrams, or 1-grams)
- Frequently, longer sequences belong together and should not be separated into two words: E.g., "United States", "operating system" → Words that have ≠ meaning when together!
- Brute-force approach: use n > 1
  - E.g., all bigrams (n=2), all trigrams (n=3)
  - Using all 5-grams can beat neural networks→ scaling it up gives a good performance but the problem is that we can't do it on large scaled data. Problem: combinatorial explosion
- Smarter way:
  - Feature selection ("multi-word expressions", "phrase extraction")
  - Simple approach for bigrams: keep bigram if *mutual information* between constituent tokens is large

For bigger see how much the n-gram gets repeated!

BAG OF TRICKS, POSTPROCESSING THE BOW MATRIX:

Inverse document frequency:

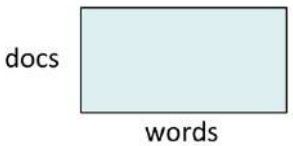
- Not all words equally informative. This is the reason for removing stopwords ("a", "the", "is", …)
- Beyond discarding stopwords, want to give less weight to more common words: E.g., "permanent" vs. "perceptron" or in law documents, "law" is going to appear a lot but does not differentiate the documents
- Standard way: **IDF = inverse document frequency**

$$idf(w) = -log(\frac{docfreq(w)}{N}) = log(N) - log(docfreq(w))$$

  - docfreq(w): number of documents that contain word w
  - N: overall number of documents
- Interpretation: information content (in terms of #bits) of event "randomly drawing a document that contains w". Beyond this theoretical justification, IDF weighting has been shown to work well in practice.

TF-IDF matrix:

- $tf(w,d)$: term frequency of word w in doc d
  - This is what the bag of words captures



- E.g., document "what you see is what you get" → bag of words {get:1, is:1, see:1, what:2, you:2}

- $idf(w)$: inverse doc freq of w (computed on entire corpus)
- TF-IDF matrix:
  - Entry in row d and column w has value $tf(w,d) * idf(w)$
  - Amounts to multiplying column w with constant $idf(w)$

Allows us to see which words are specific and apply to that document!

Row normalization:

Might also make sense to just normalize the vectors in the matrix → the more words we have the more non zero entries we have. There is a bias towards long documents being more similar to each-other just because they are longer.

- Longer docs have more non-zero entries. Interpreted as vectors, longer docs have longer vectors.
- This may throw off ML algorithms. Long vectors are far away from short vectors and regarding the dot product, random vector has a higher dot product with longer vector.
- Fix: normalize doc vectors, i.e., rows of TF-IDF matrix
  - L2-normalization: all rows have Euclidean distance 1 from origin (all data points lie on a unit sphere)
  - L1-normalization: all rows sum to 1, i.e., can be interpreted as distribution

Column normalization: It may help to apply any of the normalization techniques like Min-max scale or standardisation. What type we do? => Cross validation!

REVISITING TYPICAL TASKS:

Task 1: Document Retrieval

- Nearest-neighbour method in spirit of kNN
- Compare query doc q to all documents in the collection (i.e., rows of the TF-IDF matrix)
- Rank docs from collection in increasing order of distance
- Distance metrics:
  - Typically, cosine distance (= 1 - cosine similarity)
  - Recall: cosine similarity of q and v = <q/|q|, v/|v|>
  - If rows are L2-normalized, may simply take dot products <q,v>
- For efficiency
  - Start by filtering documents by presence of query terms (use efficient full-text index)
  - Hugely narrows down set of documents to be ranked

Task 2: Document Classification

- Use TF-IDF matrix as feature matrix for supervised methods
- Often more features (words) than documents. What's the danger with this? The columns correspond to words in the vocabulary → one column for each word we've seen → very large number. Words follow a certain distribution. The matrix is fat and short (more columns than rows). Usually for Machine Learning we want a lot of data points and few features → so tall and thin. Otherwise we have a lot of features that only appear in a few documents. So, the model could easily overfit to the few times it sees those features.
- High model capacity can lead to overfitting (high variance). Potential solutions:
  - Use more data (i.e., more labelled training docs)
  - Decrease model capacity → decrease the number of parameters
    - Feature selection (decrease number of columns, e.g. class of applied ML)
    - Regularization (two slides from now)
    - Dimensionality reduction (a few slides from now)
  - Use ensemble methods such as random forests → Idea is to decrease the variance of the models and let us avoid overfitting.

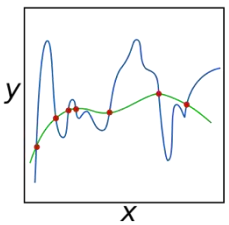
Task 3: Sentiment analysis

Regularization:

- E.g., linear regression: Find weight vector β that minimizes (z_i: feature vector of i-th data point; y_i: label of i-th data point, i.e., here: sentiment [1-5 stars] in document i)

$$\sum_{i=1}^{n}(y_i - \mathbf{z}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

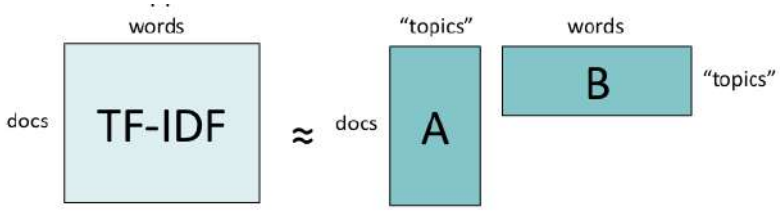
- If one word j appears only in docs with sentiment 5, we can obtain training error 0 on these docs by making β_j large enough. But doesn't generalize to unseen test data!
- Remedy: penalize very large positive and very large negative weights so that we keep the model from learning big parameters and thus overfitting.



Task 4: Topic detection

- Cluster rows of TF-IDF matrix (each row a data point)
- Manually inspect clusters and label them with descriptive names (e.g., "news", "sports", "romance", "tech", "politics")
- Unsupervised → typically, use k-means or k-medoids algorithms
- Can be difficult if dimensionality is large (#words >> #docs):
  - "Curse of dimensionality" → as we increase the number of dimensions, the volume of that space grows exponentially
  - Many outliers

Alternative approach: matrix factorization and latent semantic analysis (LSA)

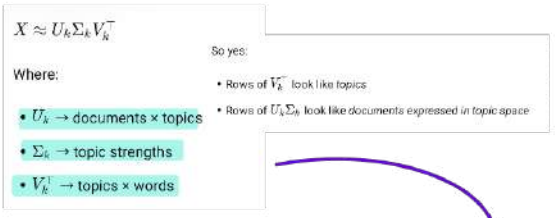


- #topics << #words (→ "dimensionality reduction")
- Assume docs and words have representation in "topic space" → we assume they have a representation we don't know yet where every document is characterized by a set of topics. So, there is a mix of topics in each document and each word is associated with a different degree to each topic.
- Word frequency (IDF-weighted) modelled as dot product of doc's vectors and word's vectors in topic space
- Topics interpretable in doc space (A's cols) and word space (B's rows)
- Optimization problem:
  - Find A, B such that A*B is as close to TF-IDF matrix as possible. Note: because right now B is also short and fat, we transpose it.
  - By close we mean, minimize :

$$\sum_{d=1}^{N}\sum_{w=1}^{M}(T_{dw} - A_d{}^T B_w)^2$$

Latent Semantic Analysis!

where T is TF-IDF matrix, A_d is d-th row of A, B_w is w-th column of B



You already know how to efficiently compute this, from your linear algebra class: singular-value decomposition (SVD):

$$T = USV^T$$

- Freebie: U and V are orthonormal bases (yay!) → the columns of U and V are mutually orthogonal to each-other. It means that the vectors are uncorrelated and don't carry much information about each-other. So, taking the columns of A as our representation, one topic is kind of independent from the other topics. Every new topic adds new irredundant information.
- S is diagonal and captures "importance" of topic (amount of variation in corpus w.r.t. topic)

- If you want k topics, keep only the first k columns of U and V, and the first k rows and columns of S → U', S', V'. E.g., $A = U', B = S'V'^T$

Recall potential problem with clustering and classification and regression: "curse of dimensionality". Matrix factorization via LSA solves these problems for you:

- Use A instead of original TF-IDF matrix
- That is, cluster (or learn to classify or regress) in topic space, rather than word space

Like if you set a value and if it is negative then what?

Topic representation from LSA is simply a vector, not a probability distribution over topics → columns ranging between – infinity to infinity. Problem is that we cannot form easy intuition about them anymore. The numbers don't really resonate intuitively with us. It would be nicer to have topics that are probability distributions over words. Probabilistic: LDA = Latent Dirichlet Allocation.

How do you interpret that?!

LDA: probabilistic topic modelling

- **L**atent **D**irichlet **A**llocation (*not* Latent Discriminant Analysis!)
- Topic := probability distribution over words and document := bag of words
- Each document has a distribution over topics. Dirichlet distribution = distribution over multinomial distributions. =) "IF documents were produced by mixing a few hidden topics, what must those topics
- Generative story not how documents are really generated → just a model! We assume that there is a probabilistic process that is generating models.

and mixtures?"

- "Generative story" for generating a doc of length n:

  d := sample a topic distribution for the doc (← "Dirichlet") → draw from a distribution (the Dirichlet distribution) over distributions.

  for i = 1, …, n

  t := sample a topic from topic distribution d

  w := sample a word from topic t

  Add w to the bag of words of the doc to be generated

▶ A topic: • P(w|t) =) probability distribution over word) =) non-negative, sums to 1.

▶ A document: • Represented as a bag words (order ignored). =) distro over topics: P(t|d).

More clearly:

1) Choose a topic distribution: $\theta_d \sim Dirichlet(\alpha)$ =) "How much of each topic will this document contain?"

2) For each word position $i = 1 \dots n$ =) 2a) Choose a $t_i \sim$ Categorical(θ_d): For this word, which is the topic. b) Choose a word from that topic =) $w_i \sim$ Categorical(β_{t_i}) =) "Pick a word of that topic". c) =) Add to bag of words.

Now, given these words and k topics =) do inverse =) determine which topics / document mixture generated this: Find P(w|t) and P(t|d) =)

max the likelihood: $\arg\max_{\theta,\beta} P(\text{words}|\theta,\beta)$

doc) topic

Topic inference:

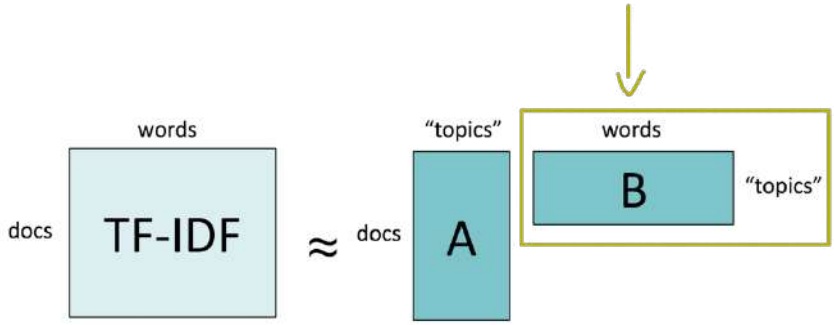
- LDA is unsupervised (topics come out "magically")
- Input:
  - Docs represented as bags of words
  - Number K of topics
- Output:
  - K topics (distributions over words)
  - For each doc: distribution over K topics
- How is this done? find distributions (i.e., topics, docs) that maximize the likelihood of the observed documents (maximum likelihood).

QUANTIFYING CLOSENESS:

- "Which of these word pairs is more closely related?" car, bus or car, astronaut
- How to quantify this?
  - Detour: How to quantify closeness of two docs? E.g., cosine of **rows** of TF-IDF matrix
  - Retour: How to quantify closeness of two words? E.g., cosine of **cols** of TF-IDF matrix

Sparsity in TF-IDF matrix:

- Two docs (i.e., rows of TF-IDF matrix) "Do you love ladies?" and "Adorest thou women?". Cosine of row vectors == 0 → Semantically the same but 0 overlap in terms of words. So, how can we still manage to capture the fact that the two sentences have the same meaning?
- Solution: Move from sparse to dense vectors so that we have non-zero numbers most of the time. But how → Latent semantic analysis (LSA). Now the topics these sentences talk about are the same.



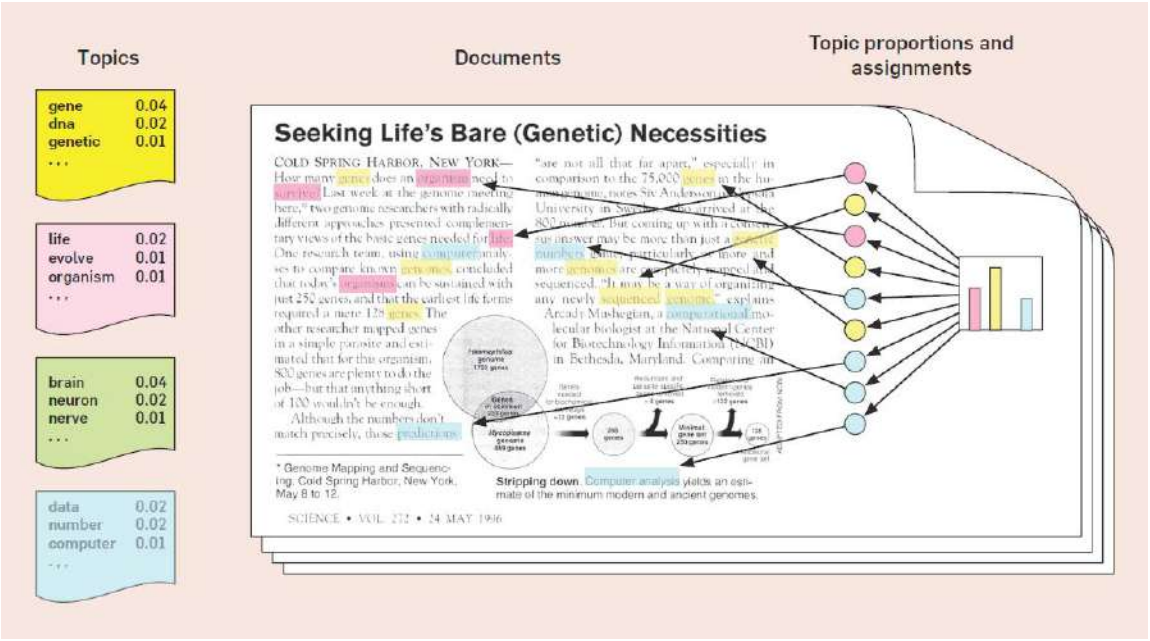
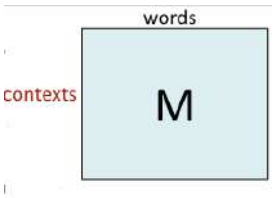
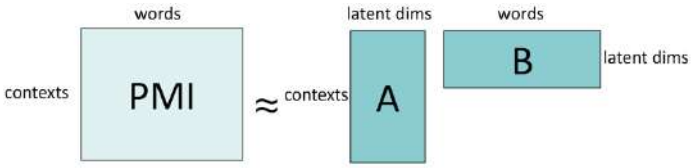


Figure 5: Generative modelling for LDA

Word vectors:

- Columns of TF-IDF matrix (sparse) or of word-by-topic matrix B (dense)
- Problem: entire doc treated as one bag of words and we use all information about word proximity, syntax, etc. Because bags of words are multisets and don't consider the word order.
- Solution: Instead of full docs, consider local contexts: windows of L (e.g., 3) consecutive words to left and right of the target word, Rows of matrix: not docs, but contexts
- What to use as entries of word/context matrix?
  - Straightforward: same as TF-IDF, but with contexts as "pseudo-docs": $M[c,w] = TF - IDF(c,w)$
  - May use any other measures of statistical association. M would store how many times we see a word in a given context. E.g., pointwise mutual information (PMI): $\mathrm{M}[\mathrm{c},\mathrm{w}] = \mathrm{PMI}(\mathrm{c},\mathrm{w}) = \log\frac{\Pr(c,w)}{\Pr(c)\Pr(w)}$ "How much more likely are c and w to occur together than if they were independent?"
  - word2vec: factor PMI matrix and use columns of B as word vectors





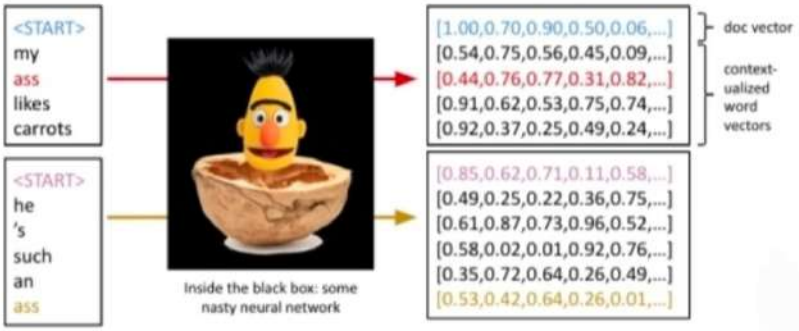
From words to texts:

How to represent larger units, such as sentences, paragraphs, docs?

- Typical approach: take sum/average of word vectors. Note: this is roughly also what bags of words are (when using "one-hot" encoding for words, i.e., vector with exactly one 1, rest 0). Current research: **learn** vectors for longer units (e.g. Cr5, sent2vec, convolutional neural networks, etc)

Contextualized word vectors:

Motivating example: "My ass likes carrots" vs. "He's such an ass". Classic word vectors (e.g., word2vec) cannot distinguish these two cases; same vector used for both instances of "ass". Solution: contextualized word vectors: e.g., BERT (see below)



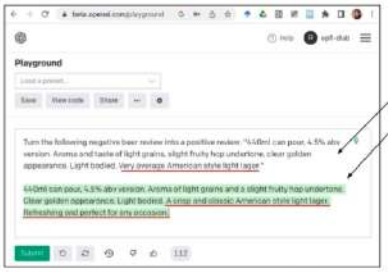
NLP pipeline

- Tokenization
- Sentence splitting
- Part-of-speech (POS) tagging
- Named-entity recognition (NER)
- Coreference resolution
- Parsing
  - Shallow parsing (a.k.a. chunking)
  - Constituency parsing
  - Dependency parsing

NLP pipeline

- Implemented by Stanford CoreNLP, nltk, spaCy, etc.
- Sequential model
  - Fixed order of steps
  - Early errors will propagate downstream
  - Fixed order not optimal for all cases (e.g., syntax usually done before semantics, but semantics might be useful for inferring syntax)
- Hence, current research: learn all tasks jointly (early example)
- To learn how all this magic is implemented
  - Take CS-431 (Intro to NLP), CS-552 (Modern NLP)

Today's trend: generative language models



- E.g., OpenAI GPT-*, ChatGPT
- Input: text
- Output: text
- Many NLP tasks can be formulated in this framework, by "prompting" the language model with the right input

Lecture 12: Scaling up:

▶ **Problem**: We have so so so much data we need to distribute computation! otherwise not possible in current age!

Usually use cheap hardware:
- ▶ Easy to add capacity.
- ▶ Cheaper per CPU and per disk.

↳ **Problems:**
1) Failures
2) Commodity network (vs RAM): much slower
3) Uneven performance: Inconsistent hardware and network latency!

Challenges of cluster computing:

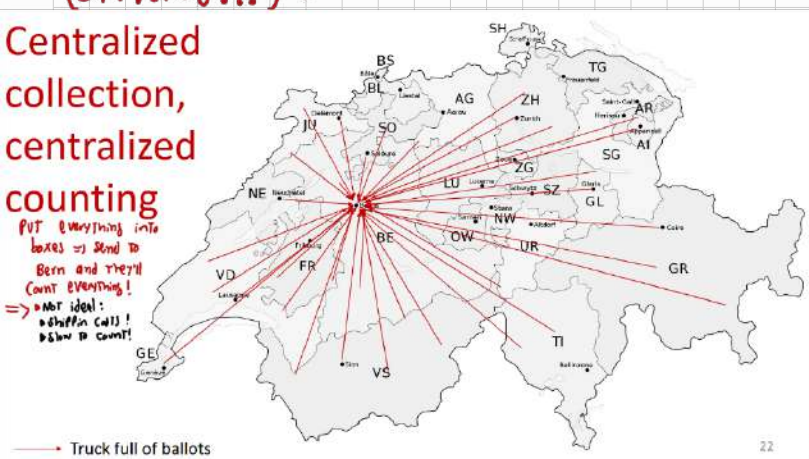
1) How to split work across machines? 2) How to deal w/ failures?

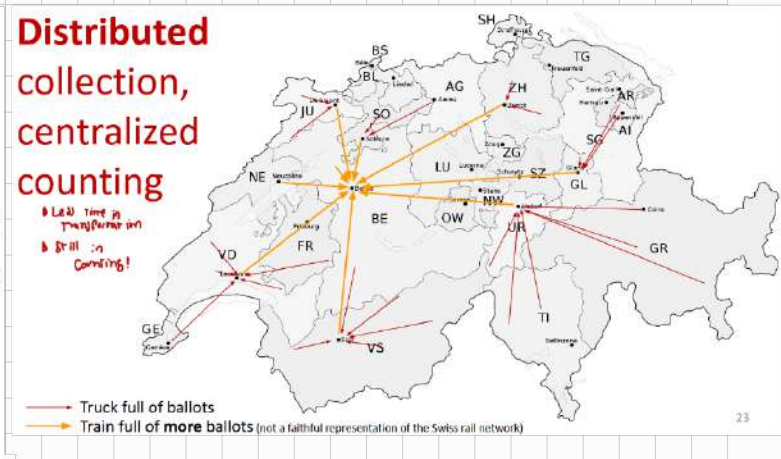
Models of distributed computation:

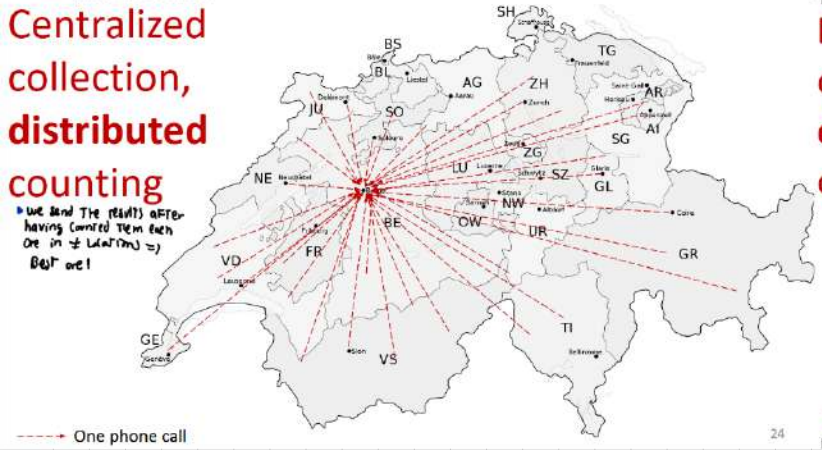
1) Distributed aggregation. 2) MapReduce 3) Spark RDDs 4) Spark high-level APIs.

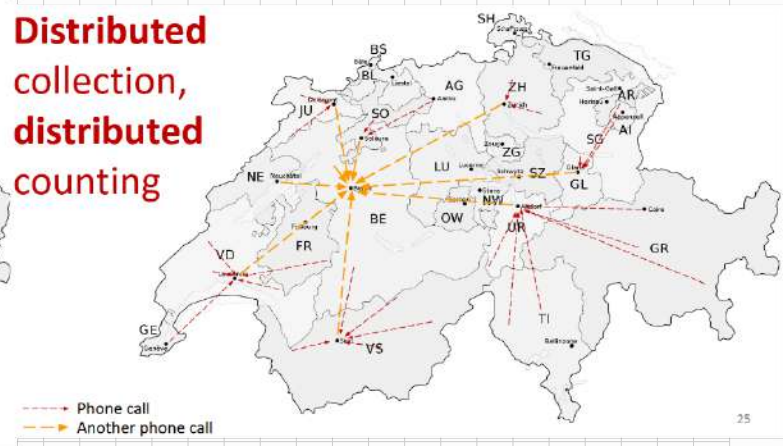
Distributed Aggregation:

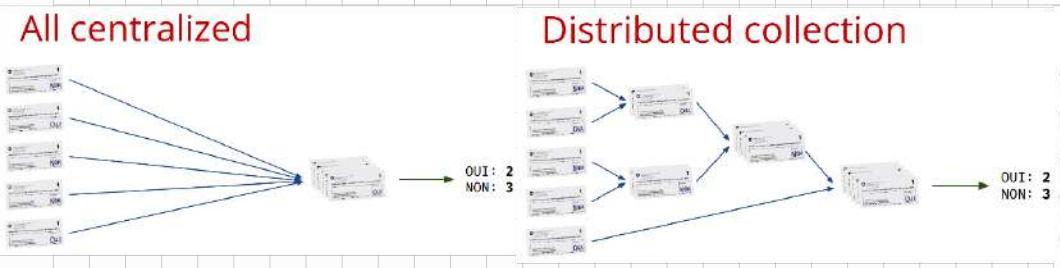
(Inverse...):

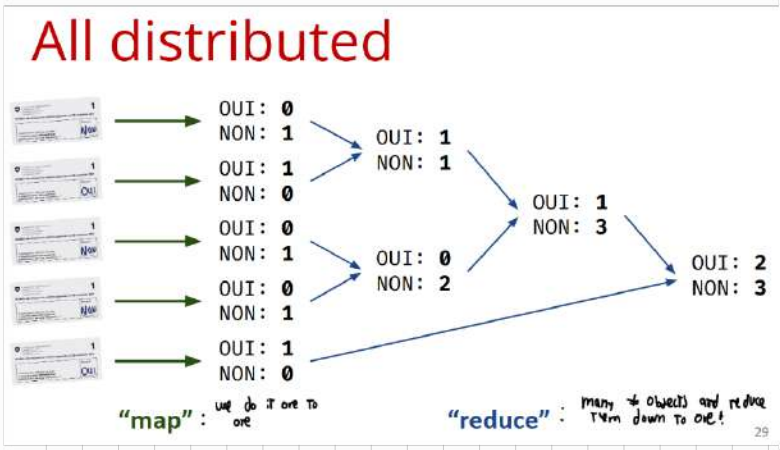


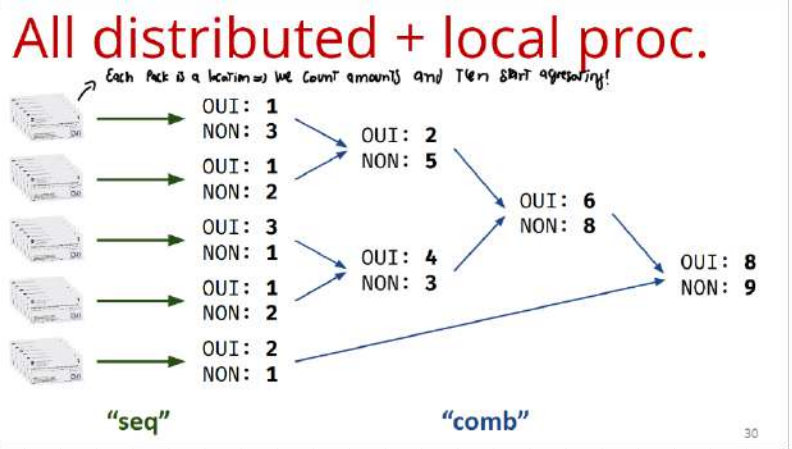






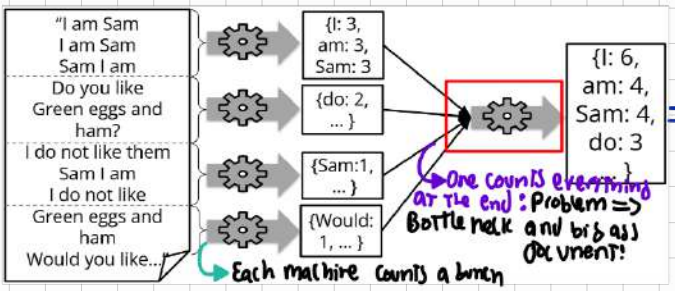






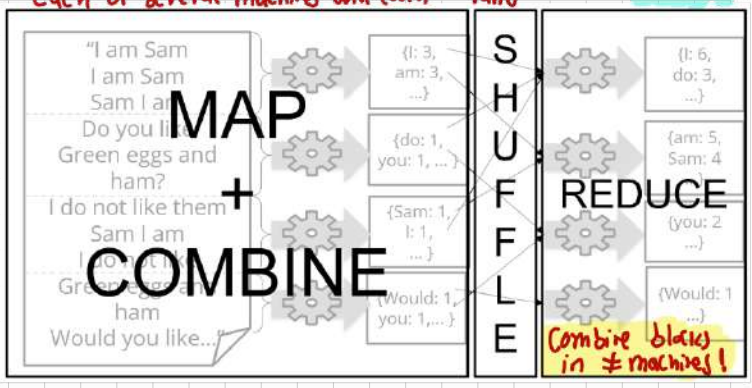
2) MapReduce / Hadoop => not only map + reduce cuz we will not necessarily reduce to sm small!

- Count number of words in a paragraph? => use a hashable => dictionary!
- BUT => happens if the document is really big? => we may run into bottle neck: too long to process!



Sol: =>

Each of several machines will count rand



What's hard about clustering computing?

1) Division of work { - movin data := $$$
- must consider data locality.

2) Deal w/ Failure => 1) one server fails every 3 years.
2) Worse: stragglers => node not failed but is very slow!

Solution : MapReduce

MAPREDUCE AND SPARK:

Does automatically: (smart system engineers have already done all work)

- Task scheduling
- Virtualization of file system
- Fault tolerance (incl. data replication)
- Job monitoring

"All" you do is implement Mapper and Reducer classes → Attention: still need to break more complex jobs into sequences of MapReduce jobs.

- How to deal w/ Failure and slow task? => start another one!!!

Example task:

Suppose you have user info in one file, website logs in another, and you need to find the top 5 pages most visited by users aged 18–25

- We need to be able to speak in the Cloud language!

Enter Spark: => A high level functional programming API:

```
print("I am a regular Python program, using pysp

users = (sc.textFile('users.tsv') # user <TAB> age
         .map(lambda line: line.split('\t'))
         .filter(lambda user_age: 18 <= int(user_age[1]) <= 25))

pages = (sc.textFile('pageviews/*.tsv')  # user <TAB> url
         .map(lambda line: line.split('\t')))

counts = (users.join(pages)
               .map(lambda user_age_url: (user_age_url[1][1], 1))
               .reduceByKey(lambda x, y: x + y)
               .takeOrdered(5))
```

Order by visits

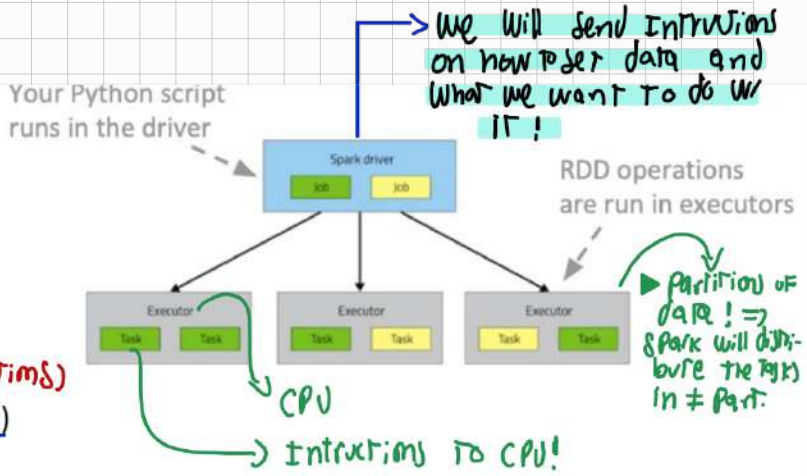
Take top 5

→ High level Programming for optimizing later!

RDD: resilient distributed dataset.

To programmer: looks like one single list (each element represents a "row" of a dataset).

- RDDs "live in the cloud": split over several machines, replicated, etc.
- Can be processed in parallel (Transformations)
- Can be transformed to single, real list (if small...) (actions)
- Typically read from the distributed file system (HDFS)
- Can be written to the distributed file system



1 What is an RDD?

An **RDD (Resilient Distributed Dataset)** is:

- A **distributed** collection of data
- **Immutable** (you create new RDDs from old ones)
- **Fault-tolerant** (can be recomputed if a node fails)
- Processed **in parallel** across a cluster

Formally:

> An RDD is Spark's low-level abstraction for distributed memory.

2 How Spark *uses* RDDs internally

Spark programs work like this:

1. You define **transformations**

```python
rdd2 = rdd1.map(...)
rdd3 = rdd2.filter(...)
```

→ *lazy*, nothing runs yet

2. You call an **action**

```python
rdd3.count()
```

→ Spark builds a **DAG (lineage graph)** and executes it

RDDs are the **nodes of that DAG**.

4 RDDs vs DataFrames / Datasets (important)

Today, when you write:

```python
df = spark.read.csv("data.csv")
```

You *don't see* RDDs — **but they're still there underneath.**

Level	You write	Spark executes
High-level	DataFrame / SQL	Optimized query plan
Low-level	RDD	Actual distributed execution

So:

- RDDs = execution foundation
- DataFrames = optimized, user-friendly layer

5 When would you use RDDs directly?

Rarely, but useful when:

- You need **very custom low-level logic**
- You work with **unstructured data**
- You want **full control** over partitions and execution

Most of the time:

> **Use DataFrames — Spark will convert them to RDD operations internally anyway.**

One-sentence exam answer

> **RDDs are Spark's core low-level abstraction for distributed data; Spark programs ultimately execute as transformations and actions on RDDs, even when using higher-level APIs like DataFrames or SQL.**

Commands on Spark to treat data!

RDD operations:

"Transformations"

- Input: RDD; output: another RDD
- Everything remains "in the cloud"
- Example: for every entry in the input RDD, count chars → RDD: ['I', 'am', 'you'] → RDD: [1, 2, 3]

"Actions" : ▶only these ones require computation!

- Input: RDD; output: a value that is returned to the driver → or to storage sources.
- Result is transferred "from cloud to ground"
- Example: take a sample of entries from RDD ; or agregations.

Lazy execution:

Transformations (i.e., RDD→RDD operations) are not executed until it's really necessary (a.k.a. "lazy execution"). Execution of transformations triggered by actions. Why?

- If you never look at the data, there's no point in manipulating it...
- Smarter query processing possible: E.g., rdd2 = rdd1.map(f1), rdd3 = rdd2.filter(f2) Can be done in one go → no need to materialize rdd2.

RDD transformations:

- **map**(*func*): Return a new distributed dataset formed by passing each element of the source through a function *func*
  - {1,2,3}.map(lambda x: x*2) → {2,4,6}
- **filter**(*func*): Return a new dataset formed by selecting those elements of the source on which *func* returns true
  - {1,2,3}.filter(lambda x: x <= 2) → {1,2}
- **flatMap**(*func*): Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a list rather than a single item)
  - {1,2,3}.flatMap(lambda x: [x,x*10]) → {1,10,2,20,3,30}
- **sample**(*withReplacement?, fraction, seed*): Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed* → To be consistent between the different systems => 20% is always 20% but 1000 samples may vary from data to data
- **union**(*otherDataset*): Return a new dataset that contains the union of the elements in the source dataset and the argument.
- **intersection**(*otherDataset*): ...
- **distinct**(): Return a new dataset that contains the distinct elements of the source dataset.
- **groupByKey**(): When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.
  - {(1,a), (2,b), (1,c)}.groupByKey() → {(1,[a,c]), (2,[b])}
- **reduceByKey**(*func*): When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V, V) => V.
  - {(1, 3.1), (2, 2.1), (1, 1.3)}.reduceByKey(lambda (x,y): x+y) → {(1, 4.4), (2, 2.1)}
- **sortByKey**(): When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs sorted by keys. Note: before doing this we can't assume that anything in the whole RDD is sorted.
- **join**(*otherDataset*): When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
  - {(1,a), (2,b)}.join({(1,A), (1,X)}) → {(1, (a,A)), (1, (a,X))}
- Analogous: **leftOuterJoin**, **rightOuterJoin**, **fullOuterJoin**

RDD actions:

- **collect**(): Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
- **count**(): Return the number of elements in the dataset.
- **take**(*n*): Return an array with the "first" *n* elements of the dataset.
- Reduce(Func.): Compute a single value.

- **saveAsTextFile**(*path*): Write the elements of the dataset as a text file in a given directory in the local filesystem or HDFS.

Broadcast variables:

- Example:
  my_set = set(range(1e80))
  rdd2 = rdd1.filter(lambda x: x in my_set)
  This is a bad idea: my_set needs to be shipped with every task (one task per data partition, so if rdd1 is stored in *N* partitions, the above will require copying the same object *N* times)
- Better:
  my_set = sc.broadcast(set(range(1e80)))
  rdd2 = rdd1.filter(lambda x: x in my_set.value)
  This way, my_set is copied to each executor only once and persists across all tasks (one per partition) on the same executor

Broadcast variables are **read-only.** The broadcast constructor allows the datastructure to be persisted on the executor till it's done.

Accumulators:

- def f(x): return x*2
  rdd2 = rdd1.map(f)
  How can we easily know how many rows there are in rdd1 (without running a costly reduce operation)?
- Side effects via accumulators!
  counter = sc.accumulator(0)
  def f(x): counter.add(1);
  return x*2 rdd2 = rdd1.map(f)
- Accumulators are **write-only** ("add-only") for executors since many executors do this in parallel, we might be reading a completely inconsistent state.
- Only the driver can read the value: counter.value

RDD persistence:

Because of lazy execution. We need to add persistence if we want to keep the intermediate results. Persistence is a trade-off because time and space. The upper case does not need extra space to persist rdd2 while in the lower case we save extra execution time but need to store an extra RDD.

```
rdd2 = rdd1.map(f1)
list1 = rdd2.filter(f2).collect()
list2 = rdd2.filter(f3).collect()
```
} map(f1) transformation is executed twice

```
rdd2 = rdd1.map(f1)
rdd2.persist()
list1 = rdd2.filter(f2).collect()
list2 = rdd2.filter(f3).collect()
```
} Result of map(f1) transformation is cached and reused (can choose between memory and disk)

🧠 One-page comparison

Concept	Purpose	Read / Write	Lives where
Broadcast	Share large constants	Read-only	Executors
Accumulator	Aggregate side info	Write-only (exec) / read (driver)	Driver
Persistence	Reuse computation	Read/write internally	Executors

DataFrames and SQL:

Spark DataFrames

- Bridging the gap between your experience with Pandas and the need for distributed computing
  - RDD = collection of records
  - DataFrame = table with rows and typed columns
- Important to understand what RDDs are and what they offer, but today most of the tasks can be accomplished with DataFrames **(higher level of abstraction ⇒ less code)**
- Getting Started with DataFrames | Databricks

Example task

Suppose you have user info in one file, website logs in another, and you need to find the top 5 pages most visited by users aged 18–25

Spark DataFrames

```
users = spark.read.csv("users.csv", header=True)
views = spark.read.csv("pageviews/*.csv", header=True)
counts = (
    users[users.age.between(18, 25)]
      .drop("age")
      .join(views, on="name")
      .groupBy("url").count()
      .sort_values("name", ascending=True)
      .head(5)
).toPandas()
```

Spark SQL

```
counts = spark.sql("""
  SELECT url, COUNT(*)
  FROM users
  JOIN views ON users.name == views.name
  WHERE 18 <= age AND age <= 25
  GROUP BY url
""")

counts.toPandas()
```

Lecture 13: Handling Network data

▶ We've been dealing w/ flat data up until now. So let's see now when data is connected among them.

Networks ... and Graph ?

▶ Network: a real-world system of dependent variables ! => Something that actually exists !
- WWW
- Society
- Metabolic pathways.

- Graph: Mathematical abstraction for describing networks { Mathematical tool simply.

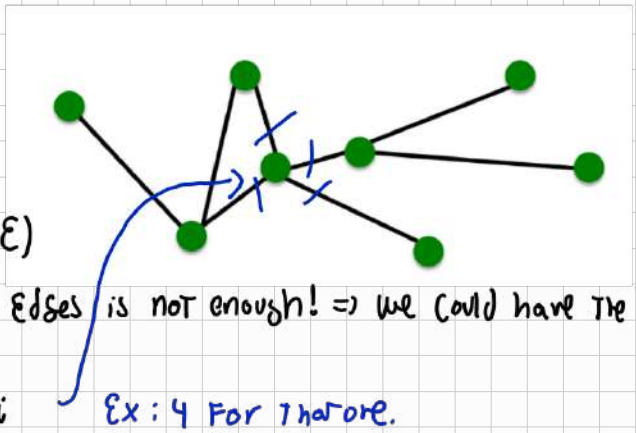
▶ Usually interchangeable but remember difference!

Types of Graphs:

Undirected Graphs:

▶ Vertices/nodes: ● V.

▶ Relationships: edges/links E \. ▶ System: G = (V, E)



▶ We need to take both into account ! => Mentioning edges is not enough ! => We could have the case of a node w/ no edge for example !

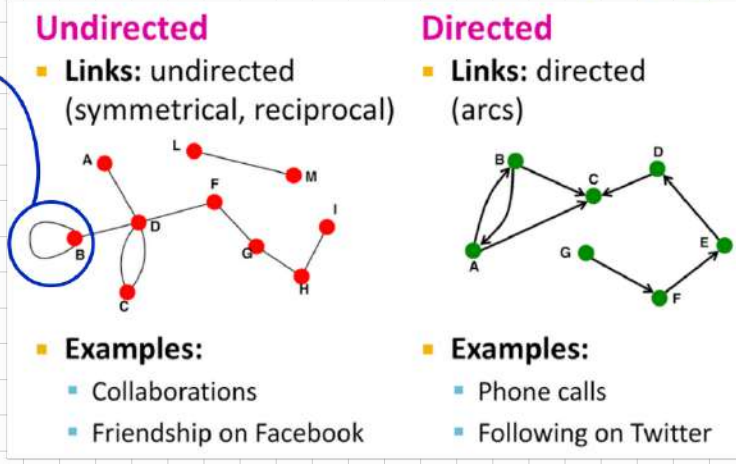
Node degree:

▶ Number of edges adjacent to node i

Ex: 4 for Thatore.

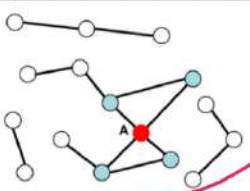
Two types:

▶ Liking my own post for example



Node degrees for directed and undirected graphs:

Undirected

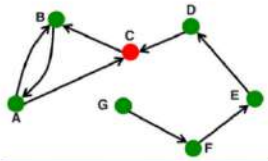


Node degree, k_i: the number of edges adjacent to node i

$k_A = 4$

Avg. degree: $\bar{k} = \langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$

Directed



Source: Node with $k^{in} = 0$
Sink: Node with $k^{out} = 0$

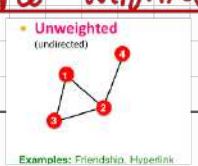
In directed networks we define an **in-degree** and **out-degree.**
The (total) degree of a node is the sum of in- and out-degrees.

$k_C^{in} = 2 \quad k_C^{out} = 1 \quad k_C = 3$

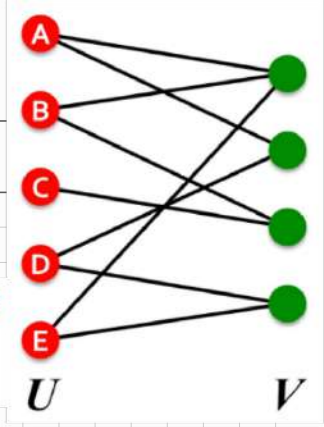
$\bar{k} = \frac{E}{N} = \bar{k}_{out} = \bar{k}_{in}$

▶ Every edge increases nodes by two usually!

Types: Weighted: ▶ Represent strength of relationship by thickness of edge => depends on your network.



▶ The sum of degrees of nodes in one partition is always equal to the sum of degrees of nodes in the other one!



Bipartite: ▶ Graph whose nodes can be divided into two disjoint sets U and V such that every link connects a node U to one in V.

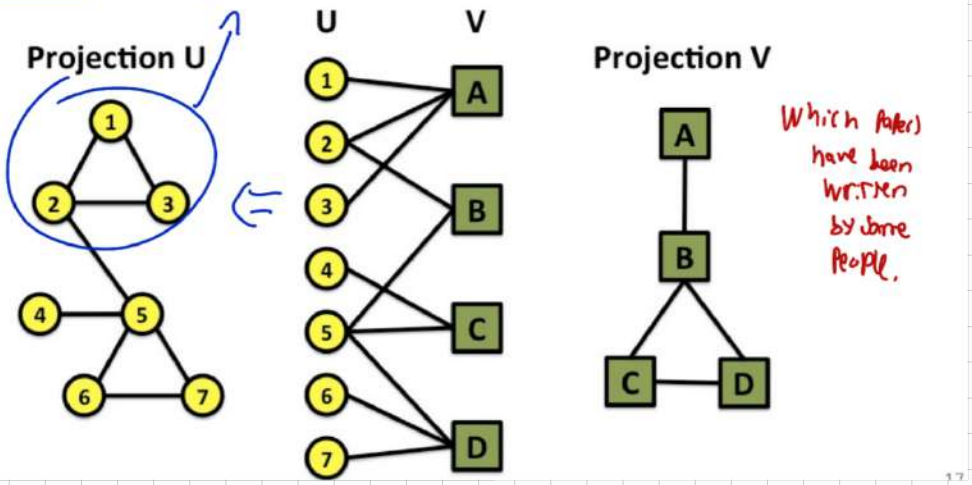
- **Examples:**
  - Authors-to-Papers (they authored)
  - Actors-to-Movies (they appeared in)
  - Users-to-Movies (they rated)
  - Recipes-to-Ingredients (they contain)
- **"Folded" networks (a.k.a. "projections"):**
  - Author collaboration networks
  - Movie co-rating networks

▶ We can then explain relations among groups of U or V!

Projections:

All these people wrote the article A for example

Projection:
=) ▶ IFF two form connection between them!



Which papers have been written by some people.

17

Representing graphs in a computer:

1) Adjacency matrix:

$A_{ij} = 1$ if there is a link from node i to node j
$A_{ij} = 0$ otherwise

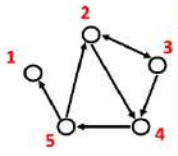
$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

▶ For a directed graph =) A is **not** symmetrical!
▶ Usually very sparse =) takes lots of memory for huge N $O(n^2)$ =) :(

2) More efficient: Edge list:

▶ Represent graph as list of edges: O(#edges) << $O(n^2)$

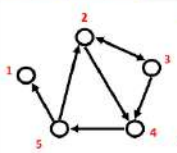
- (2, 3)
- (2, 4)
- (3, 2)
- (3, 4)
- (4, 5)
- (5, 2)
- (5, 1)



3) Adjacency list:

▶ Easier to work if large, sparse. ▶ All neighbors of a node.

- 1:
- 2: 3, 4
- 3: 2, 4
- 4: 5
- 5: 1, 2



IF directed =) we can specify a list for in and a list for out!

Properties of real-world networks:

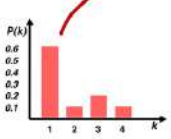
▶ In spite of being made of ≠ stuff =) they actually share some properties!

1 Sparsity:

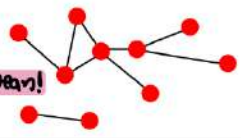
▶ Nodes are connected to only small fraction of other nodes =) $k_i << N$
▶ Maximum edges: $n(n-1)/2$ (undi) ; $n(n-1)$ (directed).
▶ Often bounded by a constant (e.g.: Dunbar number for people).

2.) Degree distribution:

- **Degree distribution $P(k)$:** Probability that a randomly chosen node has degree k
  - N_k = # nodes with degree k
- Normalized histogram:
  - $P(k) = N_k / N$ → plot

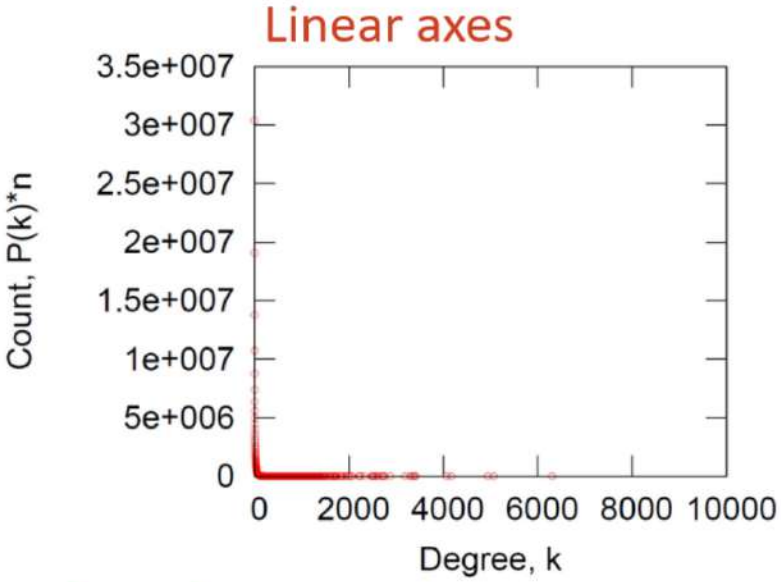


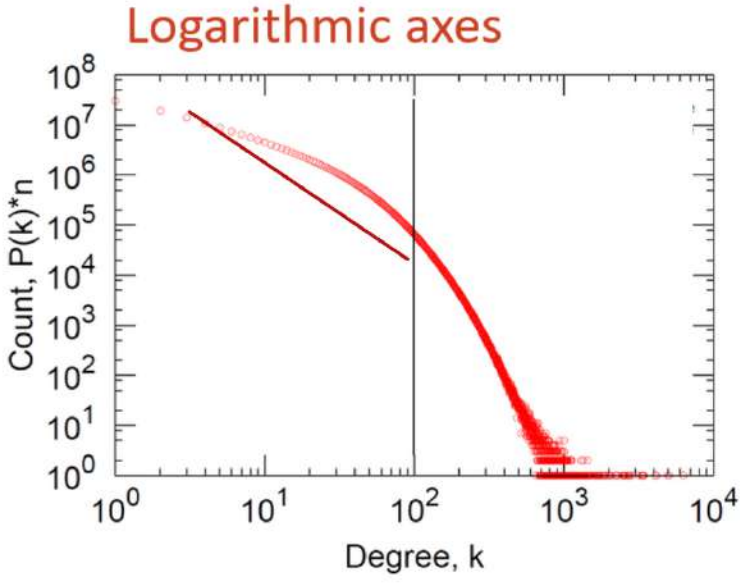
What percentage of nodes had 1 connection...



▶ More or less like a power law: So mean not really useful!

⟹
▶ Median
▶ Avg. of log of degrees =) Geo mean!





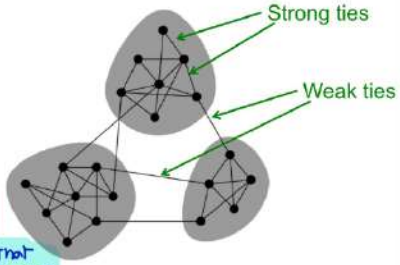
3) Triadic structure and Clustering coeff: "A Friend of a Friend is my Friend."

- Clustering coefficient: $C_i = \frac{\#\text{Edges among neighbors of } i}{\#\text{Potential Edges among " " " "}} = \frac{\#\text{Edges among neighbors of } i}{k_i(k_i-1)/2}$
- In real networks they're usually very large => Restart network keeping same coefs but randomly rewire and test that after 1000-10000 times p val is sig that clust coefs are high!

4) Community Structure:

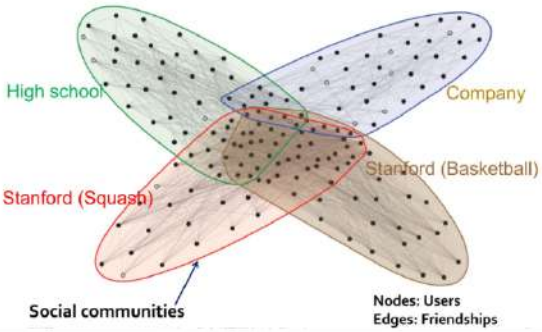
- Triadic closure makes real networks cluster into locally dense "communities"
- Communities connected via "weak ties"
- "The strength of weak ties" (Granovetter 1973)
- Weak ties fill "structural holes"

(cross communication people => connect ≠ communities => important/influential people => researchers that work w/ ≠ groups.



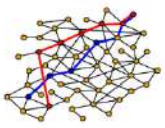
33

- In real life, communities are often not disjoint, but overlapping:



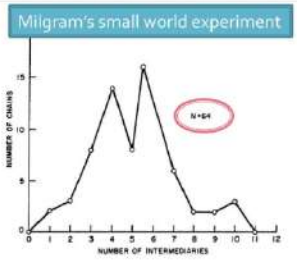
5) Average shortest-path length and navigability:

- **What is the typical shortest path length between any two people?**
  - **Experiment on the global friendship network**
    - Can't measure, need to probe explicitly
- **Small-world experiment** [Milgram '67]
  - Picked 300 people in Omaha, Nebraska and Wichita, Kansas
  - Ask them to get a letter to a stock-broker in Boston by passing it through friends
- **How many steps did it take?**

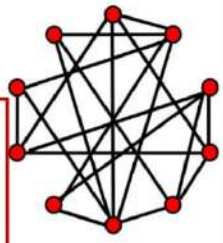


How many hops B sat to destination ≈ only 6.

- **64 chains completed:**
  (i.e., 64 letters reached the target)
  - It took 6.2 steps on the average, thus "6 degrees of separation"
- **Further observations:**
  - People who owned stock had shorter paths to the stockbroker than random people: 5.4 vs. 6.7
  - People from the Boston area have even closer paths: 4.4



- For decades, people focused on the fact that short paths exist in social networks
- But this is true even in random graphs (e.g., Erdős-Rényi model)
- The truly amazing fact is not that short paths exist, but that they are discoverable via greedy decentralized routing (as in Milgram's experiment)
- Intrigued? Read Jon Kleinberg: "The Small-World Problem: An Algorithmic Perspective"



38

TL;DR: Shortest paths not only exist in organized networks but they also exist in random ones and can be discovered!

6) Homophily and Heterophily:

- "Birds of a Feather Flock Together".
- Especially in social networks.
- Big confound and cause debate: Influence vs homophily.
- E.g.: $F = G M_1 M_2 / r^2$: m_1: obese person 1, m_2: obese person 2.
  - 1) Influence: I copy eating behaviour of people around me
  - 2) Homophily: people w/ same eating habits are likely to be friends

⇒ } ∴: difficult to say which one is cause/effect.

Summary of properties:

Real-world networks (across many types)

- are sparsely connected,
- but some nodes are much more connected than most others (i.e., skewed degree distribution);
- form locally dense clusters via triadic closure,
- which leads to community structure;
- have short paths between random node pairs (partly due to "hubs" [skewed degree distribution!]),
- and the short paths are easily discoverable.

Measuring Node Importance: ▶ Formalized via centrality measures.

1) Degree centrality:

▶ Simplest. ▶ $C(i)$ = # neighbors of i. ▶ Very fragile and easy to fix (bots...).

2) Closeness centrality

- Farness(x) = total distance to x from other nodes
- $C(x)$ = 1 / Farness(x)
  - Reciprocal to turn farness into closeness
  - Only defined for connected graphs (otherwise d infinite)
- Under closeness centrality, nodes that are easy to reach from anywhere in the network are considered important
- Variant: harmonic centrality: switch sum and reciprocal
  - $C(x)$ = total reciprocal distance of x to other nodes
  - Defined even for disconnected graphs (define $1/d$ of disconnected nodes as 0)

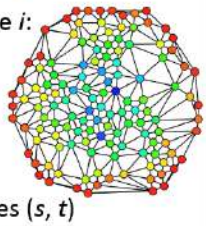
$$C(x) = \frac{1}{\sum_y d(y,x)}$$

(and for connected node). (d = shortest path)

$$C(x) = \sum_{y \neq x} \frac{1}{d(y,x)}$$

3) Betweenness centrality

- $C(i)$ = average fraction of all shortest paths in the network that pass through node i ⇒ Importance ⇒ if many short paths pass through my node.
- Computation of betweenness centrality of node i:
  - For each pair of vertices (s, t):
    - Find all shortest paths from s to t
    - Compute the fraction of these shortest paths that pass through i
  - Average this fraction over all pairs of vertices (s, t)
- Expensive to compute { Iterate through all pairs and compute the shortest path.



4) Katz centrality

- Generalization of degree centrality
- Degree centrality counts only number of direct neighbors (i.e., neighbors at distance 1)
- Katz centrality also counts neighbors at distances 2, 3, ... (we give them less weight.)
- More precisely, number of paths from other nodes to i, giving less weight to larger distances:

$$C(i) = \sum_{k=1}^{\infty} \sum_{j=1}^{N} \alpha^k (A^k)_{ji}$$

(α: penalizing for neighbors of higher rank.)

k-th power of adjacency matrix A contains number of length-k paths for each node pair

- More robust than degree centrality

5) PageRank centrality

(I depend not only on the importance of me but also the one's of my neighbors.)

- Recursive definition: my centrality $C(i) =: x_i$ is high if I receive inlinks from many other central nodes:

$$x_i = \sum_j a_{ji} \frac{x_j}{L(j)}$$

$$L(j) = \sum_i a_{ji}$$

a_{ji}: entry (j, i) of adjacency matrix A (1 if j links to i, else 0)
$L(j)$: out-degree of j

- Some extra tweaks to make it work with any graph (e.g., disconnected)

PageRank centrality

$$x_i = \sum_j a_{ji} \frac{x_j}{L(j)}$$

- Matrix notation: $x = M x$
  (where M is computed from adjacency matrix A)
- Do you recognize this?
  - x is eigenvector of M with eigenvalue 1 ⟶ we're in linear-algebra land (home sweet home)
  - x is the steady state the Markov chain induced by the network: x_i is fraction of time a random walker will have spent in node i, after a very long ($\rightarrow\infty$) random walk

PageRank centrality

- The technology that made Google huge
  - "Page" in PageRank for Larry Page
  - MapReduce was invented to compute PageRank on full Google Web crawl
  - "The $25,000,000,000 eigenvector" [link]
- Bottom line:
  Pay attention in your linear algebra class