Applied Data Analysis:

1. Handling Data:

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:

- 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

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).

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:

- **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.

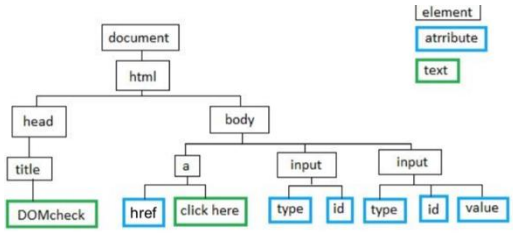


Figure 1: XML and DOM

JSON:

(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)

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).

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:

- 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.

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

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**
- 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

3. Data Visualization

VISUALIZATION FOR DATA EXPLORATION:

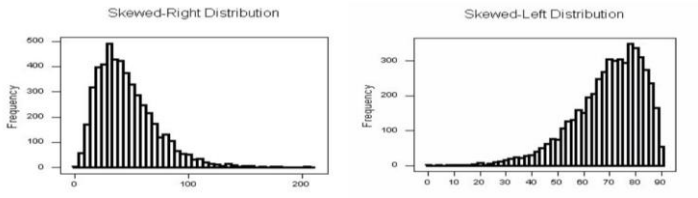


Figure 2: Histograms

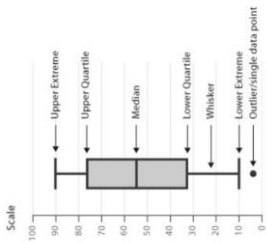
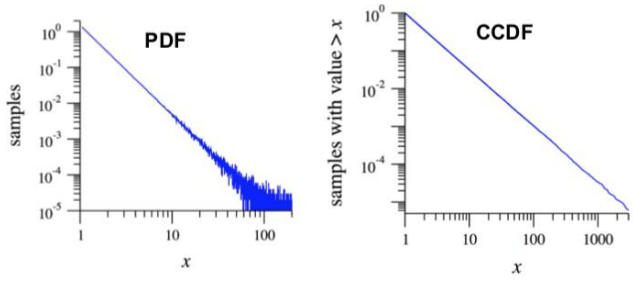


Figure 3: Boxplot

Heavy-tailed data: power laws

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 α – 1)



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)

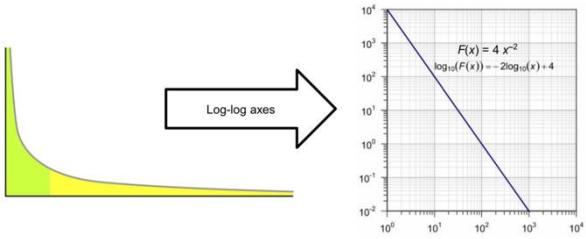
$$P(x) = C\int_x^\infty x'^{-\alpha}\,\mathrm{d}x' = \frac{C}{\alpha-1}x^{-(\alpha-1)}.$$

Heavy-tailed distributions:

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

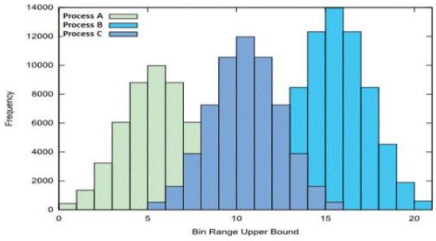
E.g. power laws: $f(x) = ax^{-k}$

- 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.)



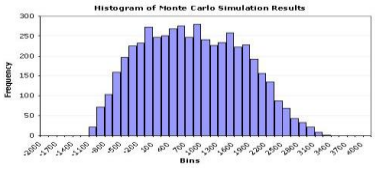
Multimodal data:

- 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.



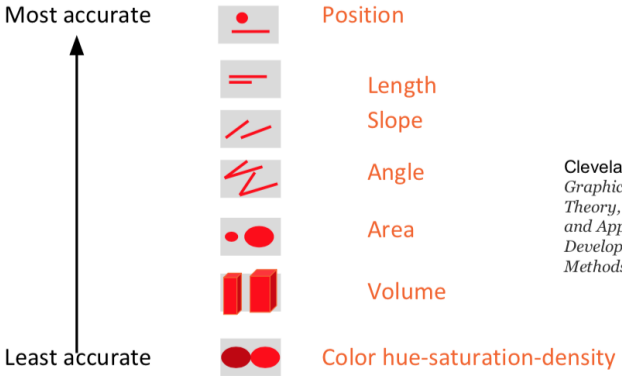
Weird data:

- 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.



PRINCIPLES OF DATA VISUALIZATION:

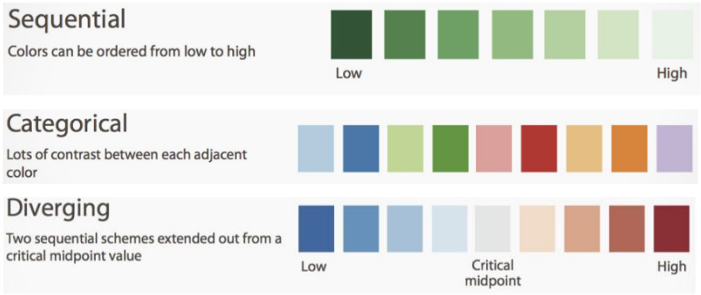
Perception of magnitudes:



Cleveland, McGill (1984)
Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods

Colours:

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



4. Read the stats:

DESCRIPTIVE STATISTICS:

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:

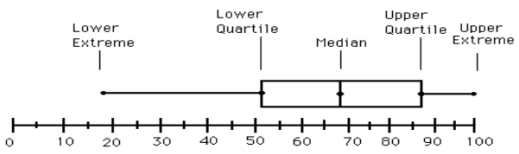
A statistic is said to be robust if it is not sensitive to outliers

- 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)

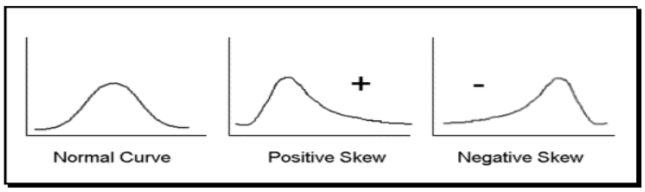
Distributions:

- **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.

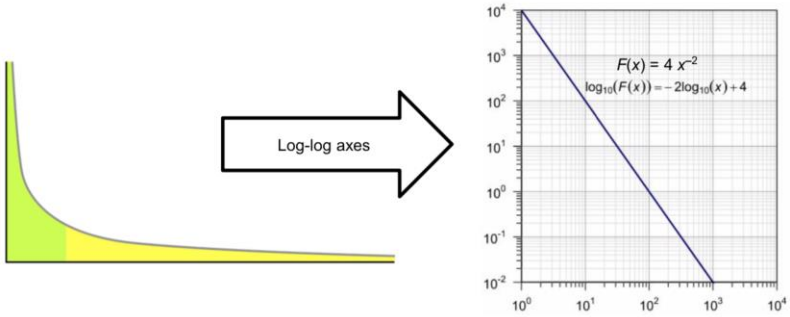


Box plot



(Smoothed) histogram

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



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.

Sampling is tricky:

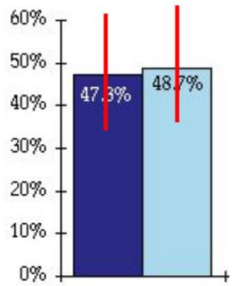
- Sometimes need to subsample for computational efficiency:
  - Uniformly at random
  - Stratified sampling (e.g., equal number per quantile)
  - Importance sampling
- Sometimes need to subsample during data collection:
  - E.g., Google Flu Trends: only Google users → Careful: bias!

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:

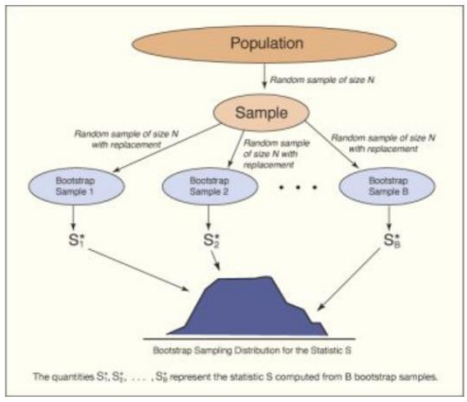


Can represent many things (→ always explain; always question):

- 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.

Bootstrap resampling:



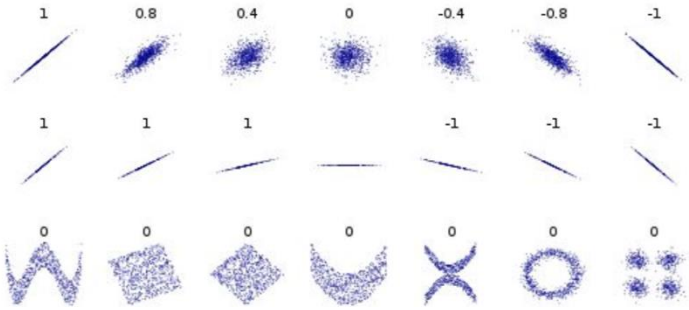
Idea : start with one dataset and simulate new populations from the dataset that we have. We pick uniformly at random samples (they can be resampled). For these resampled datasets we compute the parameters (mean, etc). We repeat that many times → this gives us a distribution of our parameters which is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

Relating two Variables:

Pearson's correlation coefficient:

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.

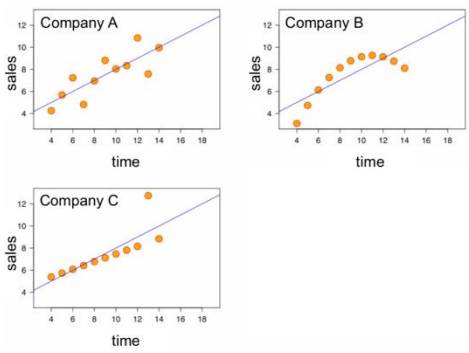


Figure 4: Anscombe's quartet

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

Hypothesis testing:

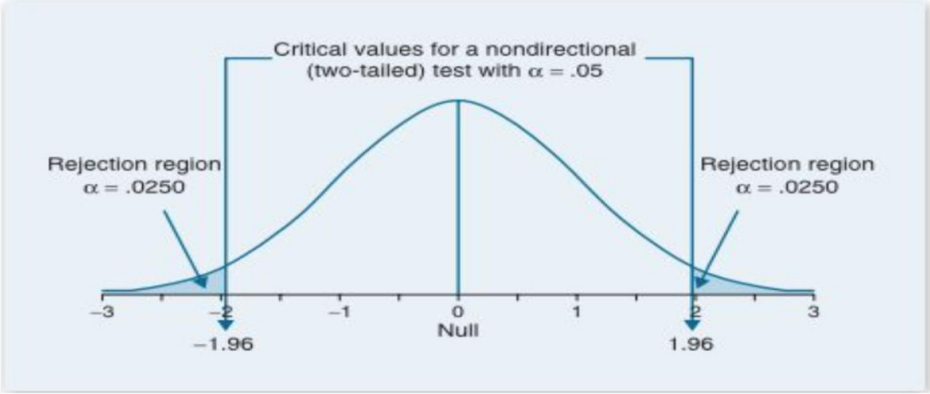
Hypothesis testing:

Reasoning:

- Flip a coin 100 times; 40 heads; "Is the coin fair?"
- Null hypothesis: "yes"; alternative hypothesis: "no" → "How likely would I be to see 40 or fewer heads if the null hypothesis were true?"
- If this probability is large, the null hypothesis suffices (and is thus not rejected)
- Otherwise, keep experimenting
- **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)**.
- 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.

Another example:

- Two samples *a* and *b*, normally distributed, from *A* and *B*.
- Null hypothesis H_0: mean(*A*) = mean(*B*)
- test statistic is: $s = \text{mean}(a) - \text{mean}(b)$.
- 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



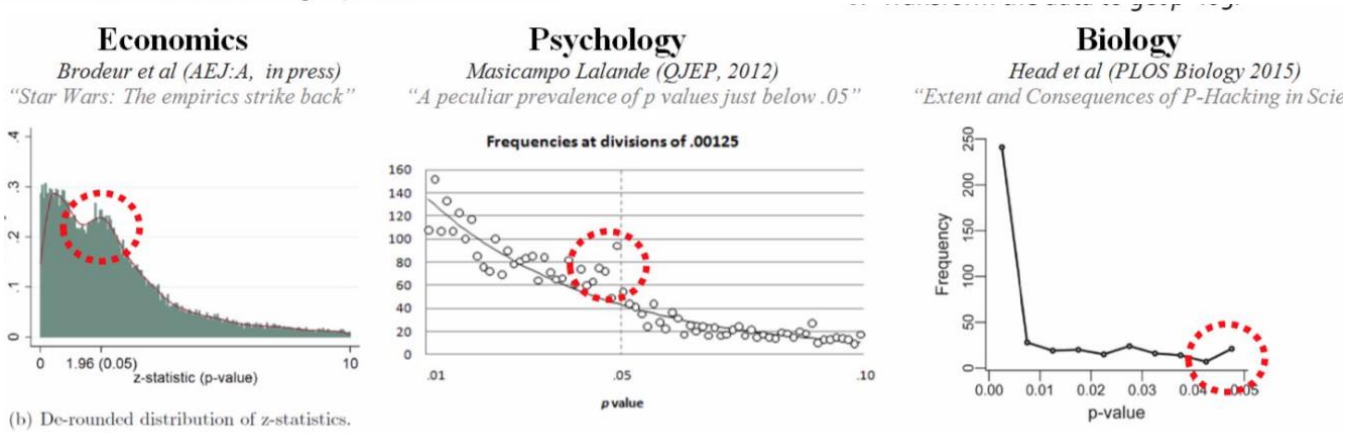
p-values:

Standard method to report significant results.

- 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!

p-value hacking:

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

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:

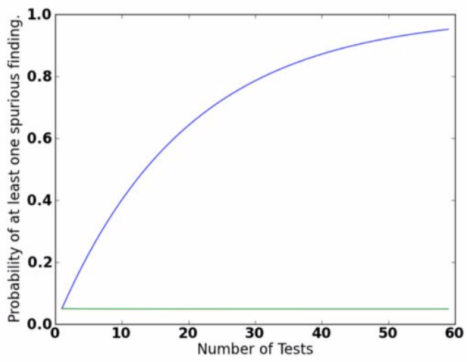
- Bonferroni correction: $\alpha_c = \alpha/k$
- Sidak correction: $\alpha_c = 1 - (1-\alpha)^{1/k}$

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"

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)

- Regression: y is continuous, e.g. linear/logistic regression

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
- y is continuous: Matrix factorization, topic modelling, unsupervised neural networks, word2vec, ...

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)
  - Similarity metric

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.

k-NN distance measures and metrics:

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:

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

Hamming Distance: For string data:

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

Manhattan Distance: Coordinate-wise distance

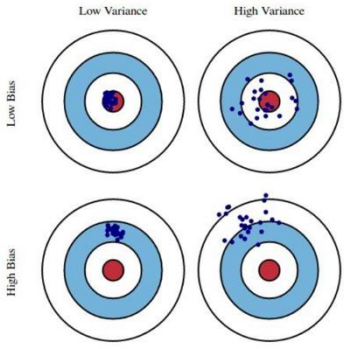
$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

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 $f_D(X)$



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:

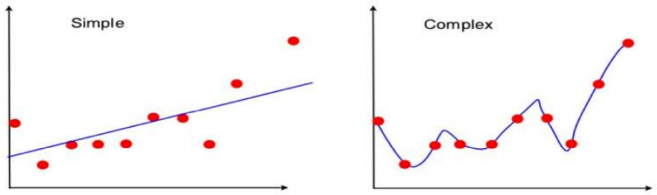
- Bias: if we train models $f_D(X)$ on many training sets D, bias is the expected difference between their predictions and the true y's → $Bias = E[f_D(X) - y]$
- Variance: if we train models $f_D(X)$ on many training sets D, variance is the variance of the estimates.

$$Variance = E[(f_D(X) - E[f_D(X)])^2]$$

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

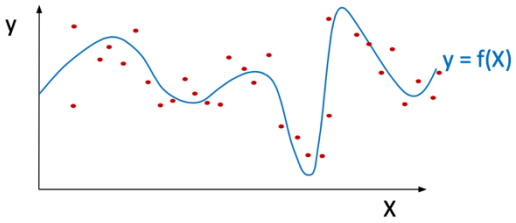
Trade-off: There is usually a bias-variance trade-off caused by model complexity.

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



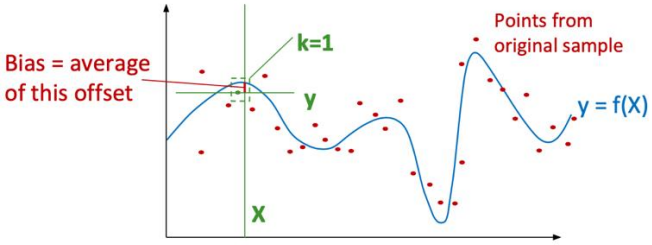
- The total expected error is: $Error = Bias^2 + Variance$
- Because of the trade-off, we want to balance these two contributions. If variance strongly dominates, there is too much variation between models and causes **over-fitting**. If Bias strongly dominates, the models are not fitting the data well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

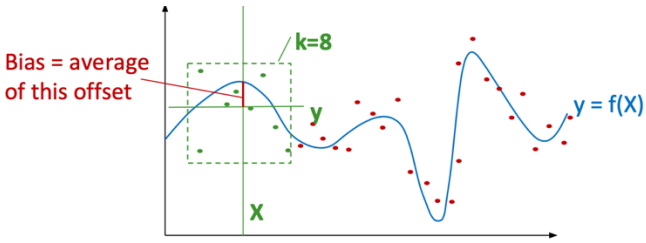
Choosing k for K-NN:



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

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.





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.

Curse of dimensionality:

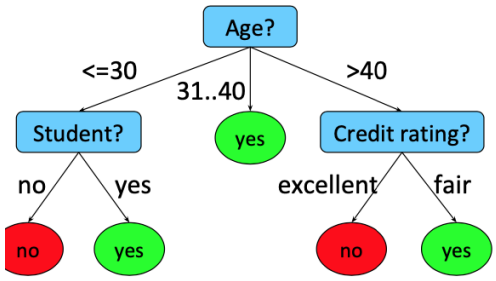
The curse of dimensionality refers to a phenomena that occurs in high dimensions (100s to millions) that does not occur in low-dimensional (e.g. 3-dimensional) space. In particular, data in high dimensions are much sparser (less

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:

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:

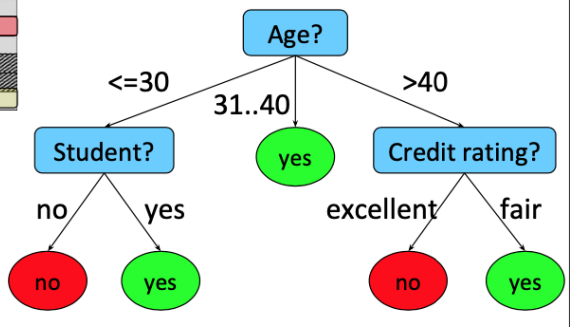
$$H(P,\ N) = -\frac{P}{P+N}\log_2\frac{P}{P+N} - \frac{N}{P+N}\log_2\frac{N}{P+N}$$

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).

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



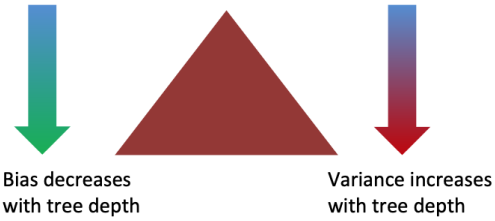
Pruning:

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)



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.

<u>Random Forests:</u>

- 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
- Needs many passes over the data – at least the max depth of the trees (can be helped by boosted trees).

Boosted decision trees:

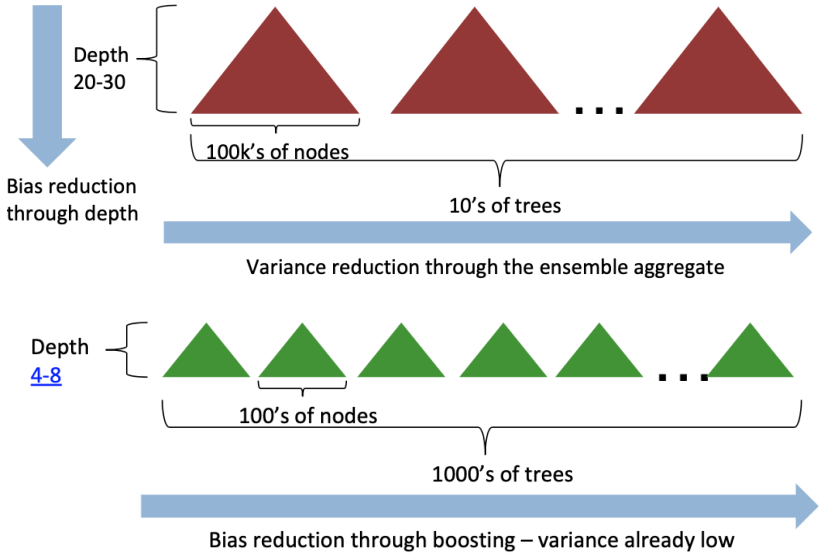
- A more recent alternative to Random Forests
- In contrast to RFs whose trees are trained independently, BDT trees are trained sequentially by boosting: Each tree is trained to predict error residuals of previous trees (→ bias reduction).
- Both RF and boosted trees can produce very high-quality models. 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:

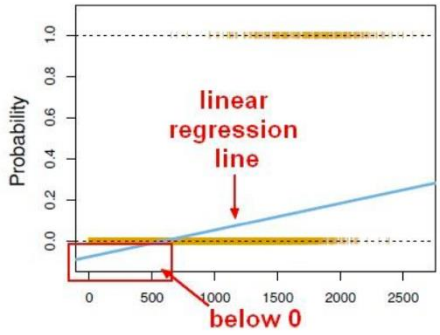
Geometry of the methods is very different. Random forest uses 10's of deep large trees while Boosted trees use 1000's of shallow small trees:

- 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

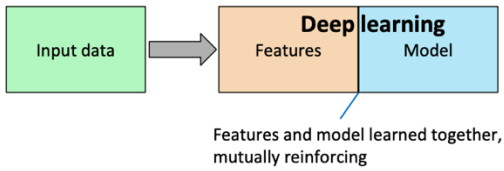
- E.g., keep features that correlate with the label y
- Forward/backward feature selection
- Regularization (e.g., penalize norm of weight vector)

6. Applied Machine Learning:

DATA COLLECTION:

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.



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.

Feature selection:

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

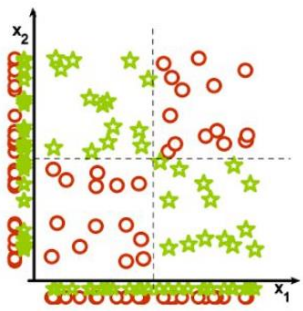
Filtering as preprocessing: offline feature selection

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.!)
- Categorical features and labels: mutual information. $\mathbf{X^2}$ 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)



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:

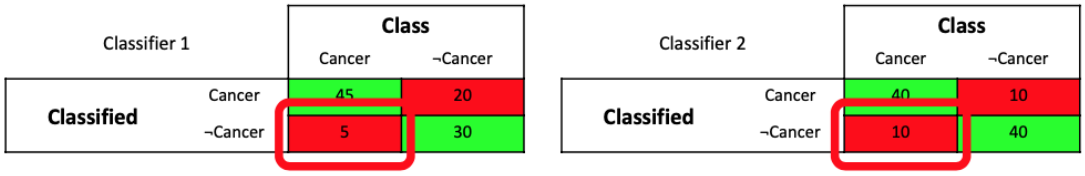
- High Level: Need to choose the type of the model (e.g Logistic regression, decision trees, random forest, etc.)
- Low-level: Usually a classifier has some "hyperparameters" to be tuned. Set of features to include:
  - Threshold (e.g., logistic regression)
  - Distance function (e.g., k-NN)
  - Number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

Performance metrics for binary classification:

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

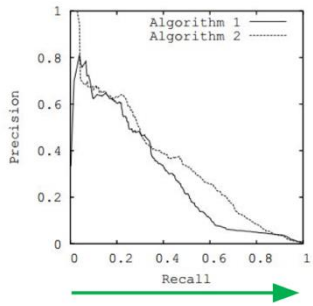
- True Positives (positive examples classified as positive)
- True Negatives (negative examples classified as negative)
- False Positives (negative examples classified as positive)
- False Negatives (positive examples classified as negative)

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



Precision and Recall:

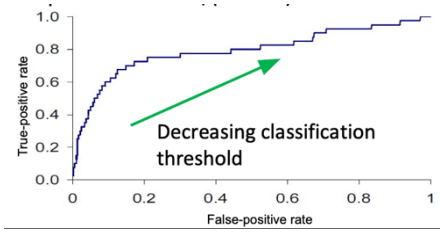
$P = TP / (TP + FP)$ and $R = TP / (TP + FN)$ → Fscore $F1 = 2\,P\,R\,/(P + R)$



Decreasing classification threshold

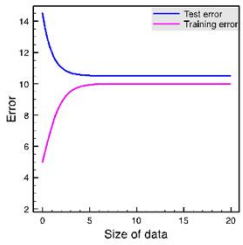
Receiver-operating characteristics (ROC) plots:

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).

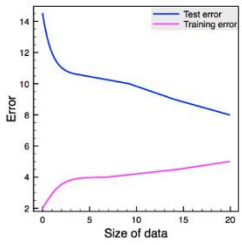


Bias and Variance:

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

7. Scaling Up:

MapReduce and Spark:

Does automatically:

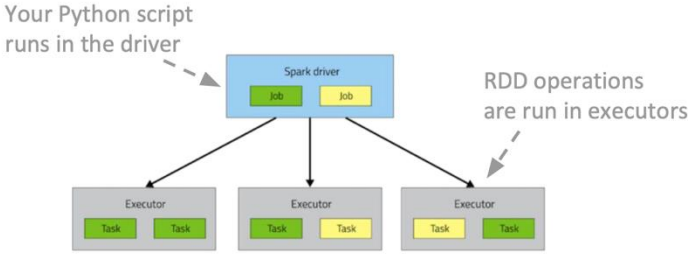
- Task scheduling
- Virtualization of file system
- Fault tolerance (incl. data replication)
- Job monitoring

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

Spark:

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
- Can be transformed to single, real list (if small...)
- Typically read from the distributed file system (HDFS)
- Can be written to the distributed file system



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

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*
- **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.

- **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)

8. Handling Text:

Typical Tasks:

Document Retrieval:

- Given a document collection and query 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

Document Classification:

- Given a document d and a set of classes
- 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)

Sentiment Analysis:

- Given: document d
- 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.

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.

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

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"

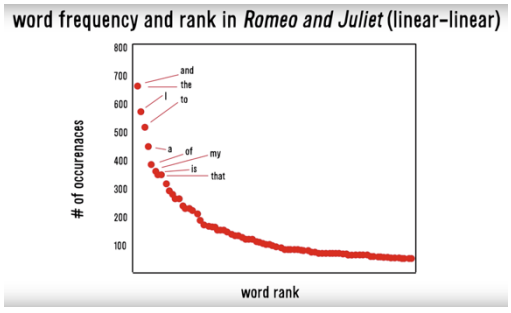
Bag of Words:

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}

To have fixed-length representation for all documents:
- One entry for each unique word in vocabulary
- Bag-of-word vectors are very high-dimensional (typically 1e5 or 1e6) and very **sparse**
- E.g., above: [0…0 1 0…0 1 0…0 1 0…0 2 0…0 2 0…0]

Zipf's law:

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.



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

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
- New school: Unicode (e.g., UTF-8, UTF-16, UTF-32)
- 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.

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!
  - Different tasks require removing different stopwords
  - Good heuristic: remove words appearing in at least p% of all documents (but what should p be…?)
  - 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
- But when dataset is small, might help because less sparsity

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:

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”
- 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

BAG OF TRICKS, POSTPROCESSING THE BOW MATRIX:

Inverse document frequency:

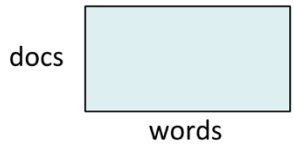
- Not all words equally informative. This is the reason for removing stopwords (“a”, “the”, “is”, …)
- Beyond discarding stopwords, want to give less weight to more common words: E.g., “permanent” vs. “perceptron” or in law documents, “law” is going to appear a lot but does not differentiate the documents
- 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 $idf(w)$

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

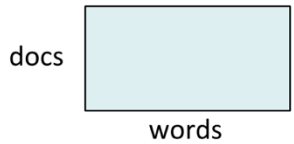
Column normalization: It may help to apply any of the normalization techniques like Min-max scale or standardisation.

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.

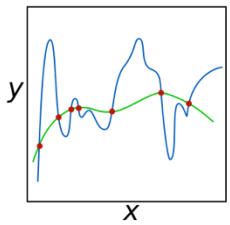
Task 3: Sentiment analysis

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*)

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

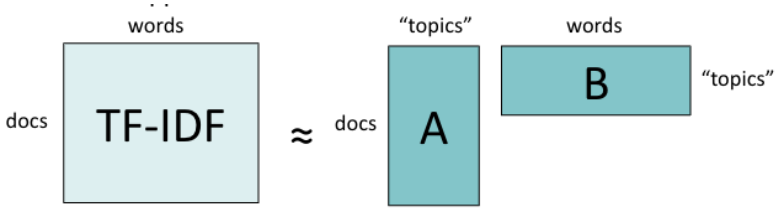
- If one word *j* appears only in docs with sentiment 5, we can obtain training error 0 on these docs by making β_j large enough. But doesn't generalize to unseen test data!
- Remedy: penalize very large positive and very large negative weights so that we keep the model from learning big parameters and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

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

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

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.

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.
- Generative story not how documents are really generated → just a model! We assume that there is a probabilistic process that is generating models.
- "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

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.

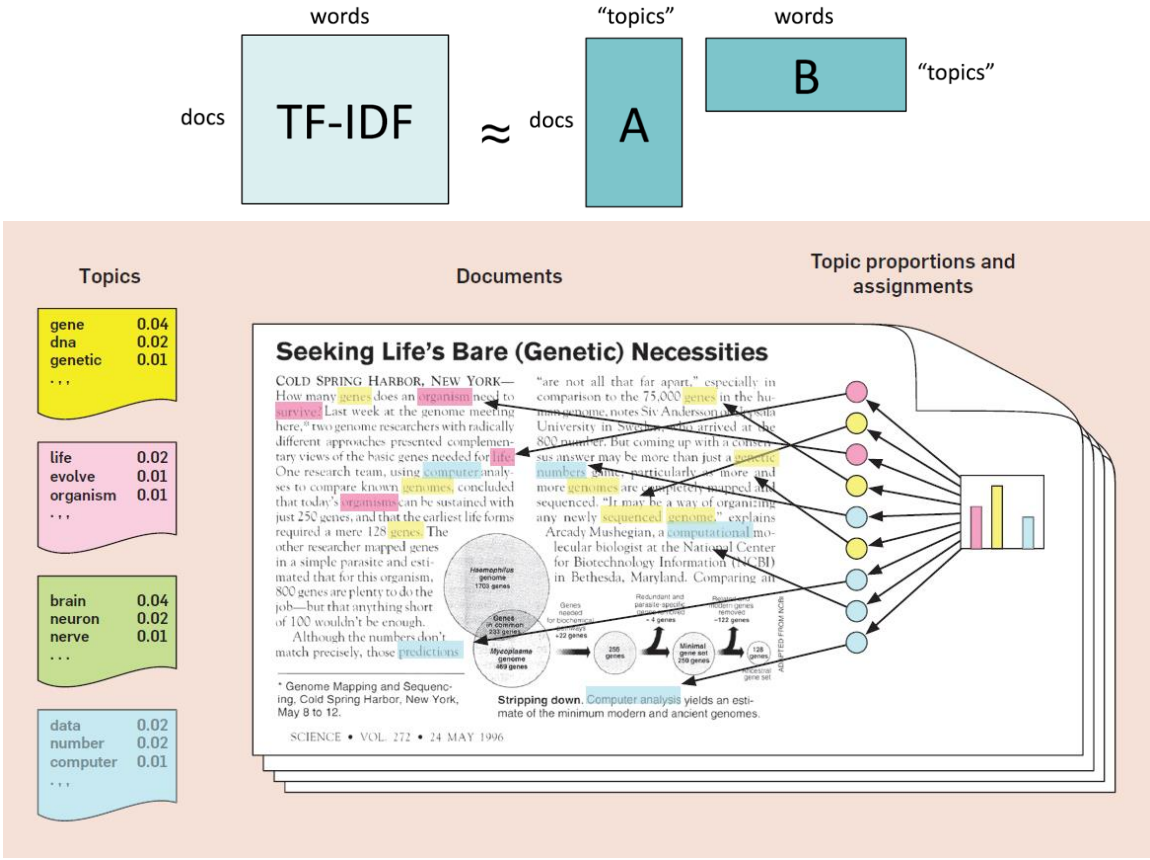
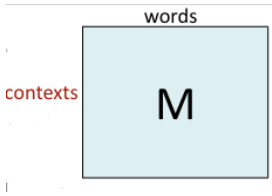
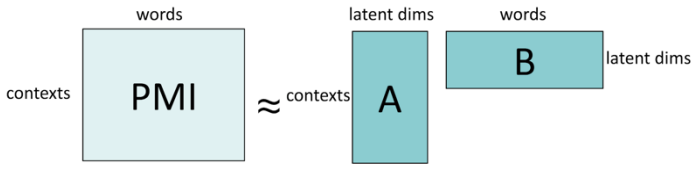


Figure 1: Generative modelling for LDA

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): $\mathrm{M[c,w]} = \mathrm{PMI(c,w)} = \log\frac{\Pr(c,w)}{\Pr(c)\Pr(w)}$ "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





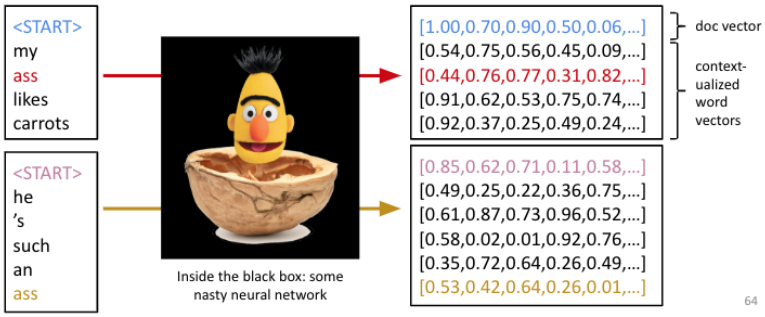
From words to texts:

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)



9. Observational studies:

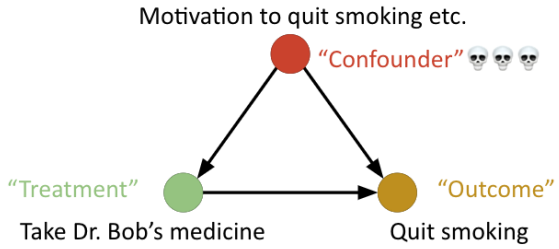
Introduction:

Goal: Clarify difference between experimental and observational studies and highlight pitfalls of observational studies**.** Give you tools for drawing valid conclusions from “found data”.

Dr. Bob’s smoking cure:

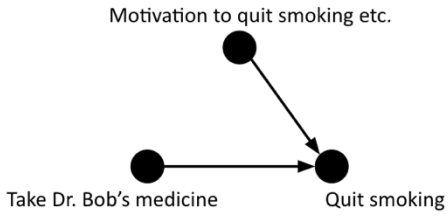
- I claim to have developed a medicine that helps you quit smoking
- I ask all smokers: “Do you want to try my medicine?”
- Smokers = treated smokers ∪ untreated (“control”) smokers
- Fraction of successful quitters is higher in the treated group
- I conclude: “My medicine helps you quit smoking! Buy it!”
- Do you believe me?

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

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)
- 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

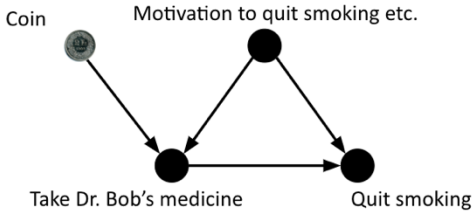
We need to make sure that the people who take and ho don't take the medicine are completely indistinguishably from each-other → Idea of *randomized*: flip a coin for each person on whether they take the treatment or not

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



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

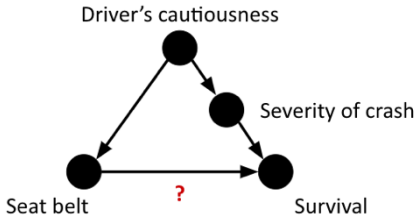
ALTERNATIVE: OBSERVATIONAL STUDIES

- 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

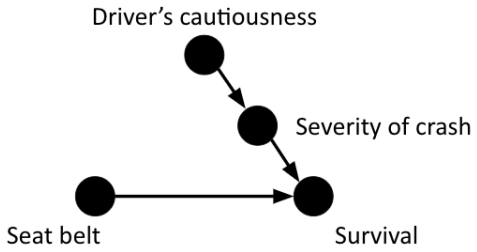
Example: seat belts revisited

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

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?

- 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.

PROBABILISTIC MODEL:

Some notation:

$$\pi_\ell = \Pr(Z_\ell = 1 \mid r_{T\ell}, r_{C\ell}, \mathbf{x}_\ell, u_\ell)$$

- ℓ : a subject participating in the study

- π_ℓ : probability of being treated, *given full knowledge of the world*
- Z_ℓ : treatment assignment (1 := treated; 0 := control)
- $r_{T\ell}$: response if subject is treated (observed iff $Z = 1$)
- $r_{C\ell}$: response if subject is control (observed iff $Z = 0$)
- $\mathbf{x}_\ell$: observed covariates (a.k.a. features we know about subject ℓ)
- u_ℓ : unobserved covariates

Ideal matching:

- Recall: we match 1 treated with 1 control subject
- Ideal matching: equal probability to be treated:
  $\pi_k = \pi_\ell$ for all matched pairs (k, ℓ)
- Why is this ideal? Because it entails that each individual is equally likely to be the treated one in the pair

$$\Pr(Z_k = 1, Z_\ell = 0 \mid r_{Tk}, r_{Ck}, \mathbf{x}_k, u_k, r_{T\ell}, r_{C\ell}, \mathbf{x}_\ell, u_\ell, Z_k + Z_\ell = 1)$$

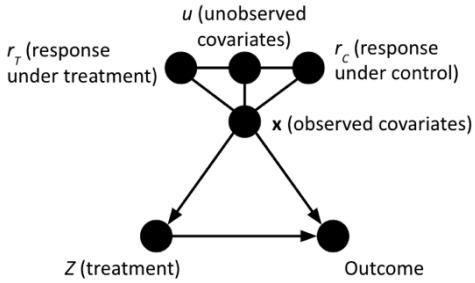
$$= \frac{\Pr(Z_k = 1, Z_\ell = 0 \mid r_{Tk}, r_{Ck}, \mathbf{x}_k, u_k, r_{T\ell}, r_{C\ell}, \mathbf{x}_\ell, u_\ell)}{\Pr(Z_k + Z_\ell = 1 \mid r_{Tk}, r_{Ck}, \mathbf{x}_k, u_k, r_{T\ell}, r_{C\ell}, \mathbf{x}_\ell, u_\ell)}$$

$$= \frac{\pi_\ell^{1+0}(1-\pi_\ell)^{(1-1)+(1-0)}}{\pi_\ell^{1+0}(1-\pi_\ell)^{(1-1)+(1-0)} + \pi_\ell^{0+1}(1-\pi_\ell)^{(1-0)+(1-1)}}$$

$$= \frac{\pi_\ell(1-\pi_\ell)}{\pi_\ell(1-\pi_\ell) + \pi_\ell(1-\pi_\ell)} = \frac{1}{2}$$

Problem: You generally don't know the probabilities to treat:

$$\pi_\ell = \Pr(Z_\ell = 1 \mid r_{T\ell}, r_{C\ell}, \mathbf{x}_\ell, u_\ell)$$



A naive model: "People who look comparable are comparable", or "Only observed variables determine treatment assignment"

$$\pi_\ell = \Pr(Z_\ell = 1 \mid r_{T\ell}, r_{C\ell}, \mathbf{x}_\ell, u_\ell) = \Pr(Z_\ell = 1 \mid \mathbf{x}_\ell)$$

i.e., $Z \perp\!\!\!\perp (r_T, r_C, u) \mid \mathbf{x}$.

Assume in this model that u, r_T and r_c are kind of shielded by x. x blocks information from flowing from u to z and z cannot be influenced by the other ones.

If the naïve model were true:

You could "simulate" a randomized experiment:

- Simply match subjects with identical observed variables x
- Subjects in a pair have the same probability to treat
- So, who gets treated is up to chance, as in experiment
- Analysis: compare outcome for treated to outcome of control (e.g., mean difference treated-minus-control)

Two problems:

- Even if naive model were true: matching on x exactly may not be possible. E.g., if x contains 20 binary features: 1 million possible instantiations of x, so likely no match. Solution: propensity scores
- The naive model is naive and rarely true → Solution: sensitivity analysis

<u>Case 1: naïve model true, but matching on x exactly may not be possible</u>

Propensity score:

- Matching on observed covariates x is rarely feasible
- Idea: reduce the information to a single number
- Definition: propensity score: $e(x) = \Pr(Z = 1 \mid x)$
- Defined even when naive model not true. But if naive model is true, it equals the probability to treat

$$\pi_\ell = \Pr(Z_\ell = 1 \mid r_{T\ell}, r_{C\ell}, \mathbf{x}_\ell, u_\ell) = \Pr(Z_\ell = 1 \mid \mathbf{x}_\ell) = e(\mathbf{x}_\ell)$$

- Typically computed via logistic regression → features: x; label: Z

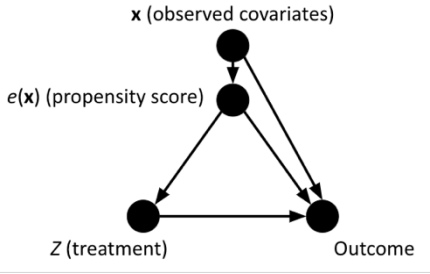
Balancing property of propensity scores:

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

$$\Pr\{\mathbf{x} \mid Z = 1, e(\mathbf{x})\} = \Pr\{\mathbf{x} \mid Z = 0, e(\mathbf{x})\}$$

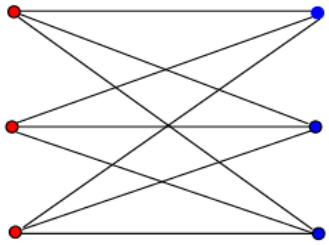
or equivalently

$$Z \perp\!\!\!\perp \mathbf{x} \mid e(\mathbf{x}),$$



- Subjects in a matched pair might not have equal x, but treated and control groups will have similar distributions of x → by fixing the score, the treated and control will look the same. That is, matching on e(x) is just as good as matching on x.

Matching:



Once we have the scores we want to do matching (with the same score now). Unlikely that 2 subjects have identical propensity scores e(x) → Matching (as you know it from your algorithms class)

- Complete bipartite graph: each subject connected to all other subjects
- Edge weights: absolute (or squared) difference of propensity scores
- Find minimum matching, e.g., via Hungarian algorithm

<u>Case 2: naïve model is naïve and rarely true</u>

Sensitivity analysis:

- Idea: Quantify the degree to which the naive model is wrong
- More concretely, model assumes that treatment odds of two identically looking subjects (i.e., identical observed covariates x) differ by a bounded factor Γ
- Then reasoning: "To change the conclusions of my study, two identically 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."

Model:

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

Bounded odds ratio. Odds isomorphic to probabilities. E.g., prob 2/3 = odds 2/1; prob 1/2 = odds 1/1

- Sensitivity Γ = 1 → naive model is true
- Sensitivity Γ = 2 → subject with same observed covariates x up to twice as likely to receive treatment
- Γ = ∞ → void statement

Example: smoking and lung cancer

- Under **naive model**: matching on observed covariates gives a very small p-value for the null hypothesis, which states that smoking does not increase lung cancer risk (using some hypothesis test)
- 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!"
- Under **sensitivity analysis model**, increasing sensitivity Γ increases the p-value for null hypothesis
- Anti-tobacco lobby: "But making p > 0.05 would require Γ > 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!"

<u>Mechanical vs scientific</u>

In observational studies, there are two parts, a mechanical part = Create pairs (1 treated + 1 control) with similar observed covariates (using exact or propensity-score matching) and a 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)