DATA MINING

Course notes

KUL

YSALINE DE WOUTERS

2016-2017

Table of content

CHAPTER 1: INTRODUCTION TO DATA MINING

Data mining refers to the discovery of models for data. However, a model can be one of several things. Originally, data minor was a derogatory term referring to attempts to extract information that was not supported by the data. Today, it is seen as the construction of a statistical model, that is, an underlying distribution from which the visible data is drawn.

Some people regard data mining as synonymous with machine learning. There is no question that some data mining appropriately uses algorithms from machine learning. Machine learning practitioners use the data as a training set, to train an algorithm of one of the many types used by machine-learning practitioners, such as Bayes net, support vector machines, decision trees, …

More recently, computer scientists have looked at data mining as an algorithmic problem. There are many different approaches to modelling data. We could for instance construct a statistical process whereby the data could have been generated.

- **Summarizing** the data succinctly and approximately;
  Ex: page rank idea. The entire complex structure of the Web is summarized by a single number for each page. This number is the probability that a random walker on the graph would be at that page at any given time. It reflects the importance of the pages.
- **Extracting** the most prominent features of the data and ignoring the rest. It represents the data by these examples.
  - **Frequent item sets:** Makes sense for data that consists of "baskets" of small sets of items. Look for small sets of items that appear together in may baskets, these frequent itemsets are the characterization of the data that we seek.
  - **Similar items:** find pairs of sets that have a relatively large fraction of their elements in common.
    Ex: treating customers at an online store like amazon as the set of items they have bought. Amazon can look for similar customers and recommend something many of these customers have bought. This is called **collaborative filtering**.

1. Background on data mining

This course provides a broad survey of several important and well-known subfields. The main goal is to develop an overall sense of how to extract information from data in a systematic way.

- The **How**: Gain insight into the working of specific algorithms. How to extract information from data. We want to understand typical questions underlying the how. How do algorithms work, what do they try to do, how do they extract information?
- The **Why**: Understand the "big picture" of data mining

Goals: understand the challenges in data mining.

Many **definitions** are possible:

- Phrase to put on CV to get hired
- Non-trivial extraction of implicit, previously unknown and useful information from data
- Buzzword used to get money from funding agencies and venture capital firms
- (Semi-)automated exploration and analysis of large dataset to discover meaningful patterns

Data mining can be interpreted as the process of automatically identifying models and patterns from massive observational databases that are

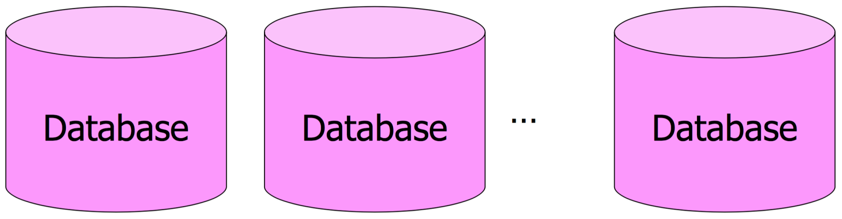
- **Valid**: hold on new data with some certainty
- **Novel**: non-obvious to the system.
- **Useful**: should be possible to act on the item. Should allow us to have a look inside the problem for instance in order to allow us decision making.
- **Understandable**: humans should be able to interpret the pattern. Often we are working with expert and we don't even understand what he is talking about.

Data mining can either be **data driven** or **learning driven**. You will have data and use it to make inferences, strategies about fact, draw models and patterns.

The process of automatically identifying models and patterns from massive observational databases.

- Models and patterns can be decision trees for instance. These are a kind of representation of the knowledge.
- Massive, we are talking about database systems and scalability.
- Observational

⇨ Observe data



There are **three goals** to data mining

- **Understand** the data, representation, what is in the data, …
- **Extract** the knowledge of the data
- Make **predictions** about the future, what can happen tomorrow, next week of year, et.

1.1 Technological advances

25-30 years ago there was essentially no data mining, and in the last years, it is becoming extremely popular and successful. It is usually popular, many companies apply data mining, the same goes for universities, sports teams, etc. It is also taught in academia. There are two reasons for this popularity.

- Technology has greatly improved.
- Databases and the web means everyone has data, everyone collects data.

Storage is **larger** and **cheaper**

⇨ Moore's law for magnetic disk density: *"capacity doubles every 18 months".*

Storage cost per byte is falling rapidly. Things like storage has becoming larger and cheaper over the years. There is more offer regarding storage capacity.
Next, we also observe improvements in computing power. The super computer of 15 years ago is equivalently powerful as modern desktops. Many large datasets are available today.

Online text sources
- MEDLINE has 19 million published articles.
- Wikipedia has huge number of articles. People try to analyse Wikipedia.
- Web search engines which can index billions of webpages every day. 100's of millions of site visitors per day
- Retail transaction data
- Ebay, Amazon, Walmart: >100 million transactions per day
- Visa, Mastercard: similar or larger numbers

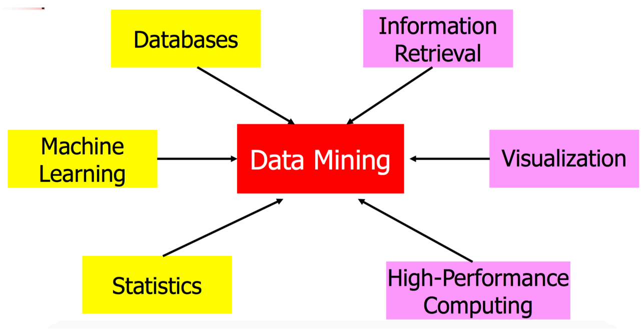
Scientific uses
- Data collected and stored at GB/hour
- Remote sensors on a satellite
- Microarrays generating gene expression data
- Scientific simulations
- Traditional techniques infeasible for raw data
- Data mining helps scientists to
- Classifying and segmenting data
- Form hypotheses
- Find hidden patterns and correlations

Furthermore, data mining appears as commercially useful. Indeed, data analysis helps companies, stores, to develop and to know how they should display the shelves, what and how much they should sell. Many companies collect and store data Search-engines: click data, advertising, automating the search.
- Stores: purchases records
- Banks: credit card transactions, want to be able to catch fraud.

Competition is strong and data mining can help
- Marketing and advertising
- Search
- Inventory. Inventory remains expensive, you don't want to have to much stock so you should be sure to optimize the inventory level.

Data mining crosses many different disciplines.
- Databases: large data sets are stored
- Machine learning
- Statistics

- Visualization of the data
- Information retrieval, good retrieval to allow an efficient search of the data
- High-performance computing to ensure the analysis goes fast.



The two most similar things are Machine Learning and statistics. What is statistics? Statisticians think in terms of models. There are lot of people, called population and we take some sample of that population. The first thing you can do is to write a distribution, describing the sample. Another thing is to make statistical inference, I have a sample and I try the hypothesis, I apply it to the mode and then see if it fits the data.
There is always a relationship between some target variable and interpretable variable.

There is usually no hypothesis. Statistics is often more used when we have this prospective data. Data mining focuses on analysis of existing data. It is more algorithmic based; data mining defines algorithms that encompass the data.

Machine learning is interested in designing algorithms that are able to improve the performances or experience of a task. Performances could relate to some metrics we want to optimize (accuracy, ROC curve, …) whereas a task refers to a certain problem, and experience is often data.

1.2 Data mining vs. Machine Learning

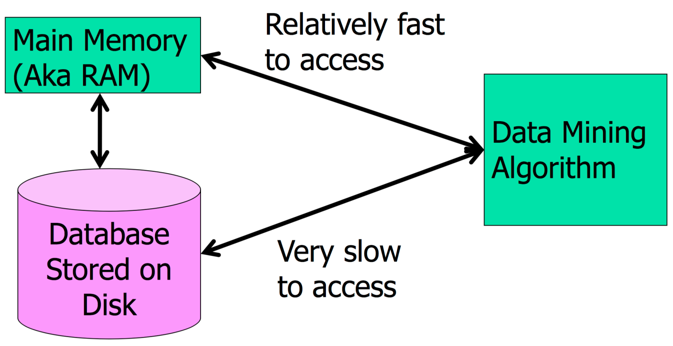
Data mining

- Data mining focuses more on scalability.
- Data mining is much more application focused.
- Data mining is more often used in industrial setting.

Machine Learning

- More theoretical emphasis
- Term more used in research/academia then data mining.

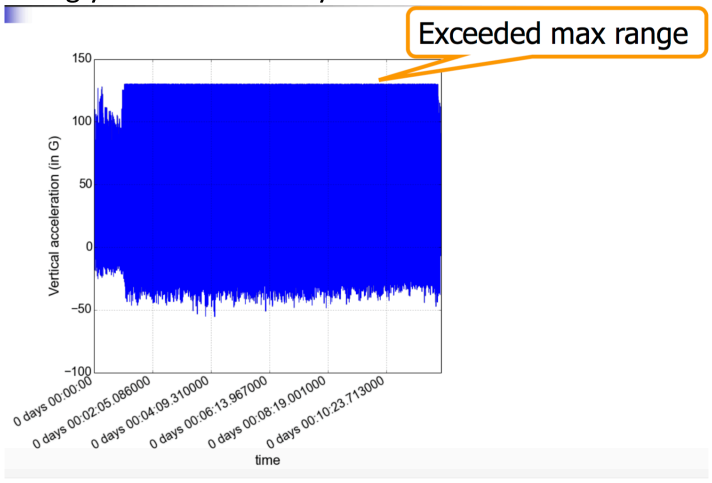
In terms of scaling up. As we see on the picture, we have both a database stored on disk and a main memory. It is very slow to access data that is stored on a disk whereas the main memory is relatively fast to access. Only a small subset of the data can be stored at the main memory. Often data mining tries to limits the number of types that it has to retain on the disk.



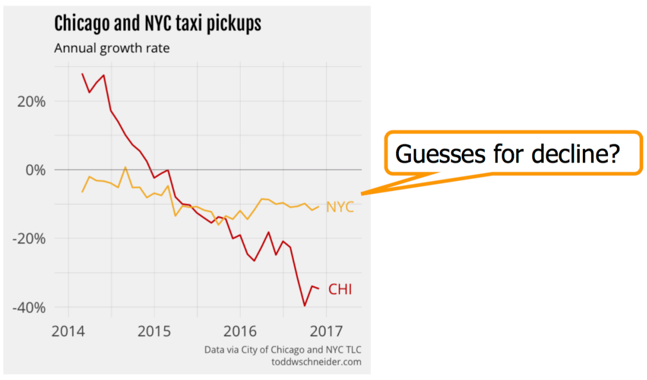
2. Data mining tasks

⇨ There are **four tasks** included.

- **EXPLORE**: try to understand and get to know the data. Therefore, we do several tasks related to statistics that is to say, define relevant and meaningful summary statistics of the data: number of variables, averages, means, modes, etc. You also try to identify flaws or problems in the data: skew, missing values, visually inspect the data, classification problems, …
Exploration is the first thing you will do when you receive data.



⇨ What is wrong with this picture? The data exceeds the maximum range. Whatever the range is, it is exceeded, this tends to be a problem. Another scale would be more appropriate.



Another examples relates to the taxi data, from 2014. Guesses for the decline? UBER could be a good reason, people using Uber instead of taxis. Again we cannot assume a causal relationship here, but probably it is one reason for this decline.

- **DESSCRIPTITVE MODELING**
  You try to build models that describe and summarize the data. You can simulate how the data was generated, modelling the process that generated the data.
  Techniques
  - Clustering
  - Density estimation/probabilistic models

- **PREDICTIVE MODELING**
  We have some variables X and we want to predict the future vale of another variable, Y, also called the target variable.
  - Classification: Y is discrete
  - Regression: Y is continuous
  - Probability estimation: Prob(Y=y)

  The goal is to explore the relationship between the set of variables X and the single variable Y. You try to approximate a function, mapping configurations of X → Y. This is what many machine learning and statistic algorithms do. Often the focus is on accuracy, not comprehensibility. We mainly focus on getting the prediction for Y.
  One big application is in politics, people making campaigns make lot of use of data. Which candidate do you prefer? Who will vote? Who will give money? Who can persuade?

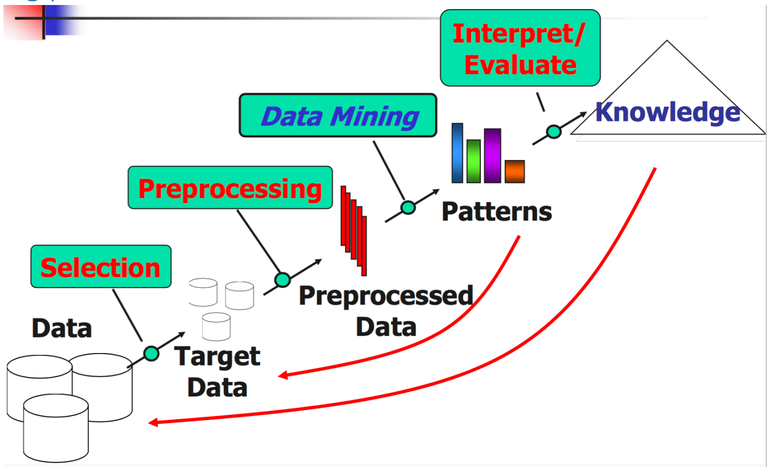
  Another good application of this is web search. You have some query, and you want to know which documents are related to the query. How do people solve this? The first attempt is to manually curate directories. The problem is that it works for small data sets but it does not scale.
  Next, another natural thing to do is try to match words in query to words in document. This is rather challenging since there are many words in the query.
  Google's idea was to exploit the web structure. View links as votes, so the more links means the more important web pages are. The intuition behind this is that you can trust what other people say.

- **DISCOVERING PATTERNS**
  The goal is to discover interesting "local" patterns in the data, not to characterize the data globally. The focus is to find human-interpretable patterns that describe the data.
  Techniques
  - Item-set mining
  - Pattern mining
  - Sequence mining

  One example would be to find repeated DNA sequences. Another thing would be product recommendation, finding commonly co-purchased items.
  Again, this step could be illustrated by sports. NBA logs all play by play information: which players are in the game? Shots attempts, etc. The questions are: which line-ups work well? Offensive efficiency? Defensive efficiency?

Machine Learning class	Data mining class
• ML has lots and lots of techniques for predictive models: naïve Bayes, decision trees, rule learning, ILP, … • Experimental methodology like cross validation • Reinforcement learning	• Predictive modelling motivated by applications • Pattern mining (unsupervised learning) • Scalability

3. The data mining process



First select data to get the target data. Look for outliers, missing values, etc. Then you do some sort of pre-processing, a key step where you apply transformation to your data, feature transformation. Then you have data mining where you find models and patterns.
Ideally you have knowledge at the end.

In a data mining system, you want something that is computationally sound. You want it to be scalable, without it taking too much time and space complexity. You want things to be parallelizable. Next you want things to be statistically sound, finding meaningful patterns. Do our results generalize to new data?
You also want the data to be easy to used and comprehensible.

3.1 Components of a data mining system

- **Representation** is about how you actually represent the data. There are two aspects of representation. The way data is represented and the way models are represented.
  - **Data** is represented using feature vectors, relational database, free text, images, graphs, etc.
  - **Model**: decision trees, graphical models, rule set, association rules, graph patterns, sequential patterns, etc.
- **Evaluation** can either subjective or objective
  - Objective: accuracy, precision and recall, cost, utility, fast, etc.

- Subjective: interesting, novel, actionable
- **Search**
  - Combinatorial optimization: Greedy search
  - Convex optimization: Gradient descent, with functions you want to minimize, maximize, etc.
  - Constrained search: Constrained search
- **Data management**
- **User interface**

To sum up, we live an age where large amounts of data are common place. Data mining is hugely popular and hugely successful because it extracts useful information from this data. Data mining is able to help people solving problems. The information comes in many forms like models, patterns, etc. Finally, data mining is challenging!

4. Statistical limits on Data Mining

⇨ A common data-mining problem involves discovering unusual events hidden within massive amounts of data.

- **Total Information Awareness**: the results you obtain all depends on how narrowly you define the activities that you look for.
- **Bonferroni's Principle**: suppose you have a certain amount of data, and you look for events of a certain type within that data. You can expect events of this type to occur, even if the data is completely random, and the number of occurrences of these events will grow as the size of the data grows.
  Calculate the expected number of occurrences of the events you are looking for, on the assumption that data is random. If this number is significantly larger than the number of real instance you hope to find, then you must expect almost anything you find to be bogus.

5. Things useful to know

5.1 Importance of Words in Documents

⇨ In many applications we shall be faced with the problem of categorizing documents (sequences of words), by their topic. Topics are typically identified by finding the special words that characterize documents about that topic.

Classification often sorts by looking at documents, and finding the significant words in those documents. Our intuition might be that the words appearing most frequently in a document are the most significant. But this is not true. I fact, the several hundred most common words in English ("the" or "and") are often removed from documents before any attempt to classify them. Actually, the indicators of the topic are relatively rare words. But not all rare words are equally useful as indicators.

TD.IDF = Terms Frequency Times Inverse Document Frequency. This is the formal measure of how concentrated into relatively few documents are the occurrences of a given word.

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

5.2 Hash Functions

A **hash function** h takes a hash-key value as an argument and produces a bucket number as a result. The bucket number is an integer, normally in range 0 to B-1, where B is the number of buckets.
Hash-keys can be of any type. There is an intuitive property of hash functions that they “randomize” hash-keys. If hash-keys are drawn randomly from a reasonable population of possible hash-keys, then h will send approximately equal numbers of hash-keys to each of the B buckets.

5.3 Indexes

An index is a data structure that makes it efficient to retrieve objects given the value of one or more elements of those objects. The most common situation is one where the objects are records, and the index is on one of the fields of that record. Given a value v for that field, the index lets us retrieve all the records with value v in that field.
Ex: file of (name, address, phone), and an index on the phone field. Given a phone number, the index allows us to find quickly the record or records with that phone number.

CHAPTER 2: GET TO KNOW YOUR DATA

Analysing data requires
- Collecting data
- Understanding data
- Putting data into format such that analysis can be performed.

Collecting the data is the most complicated, time consuming, and important part of the analysis. 50-75% of the time is spent on collecting the data. Most of the mistakes arise here, especially in the transformation phase.

1. Retrospective vs. Prospective data

Correlational vs. causation

- Retrospective for correlation A and B are related
- Prospective for causality A causes B

There exist two types of data
- **Prospective data**: collected as part of controlled scientific experiment.
- **Retrospective** or **observational data**: collected passively or some particular reason.

  ⇨ The type of data you have influences the conclusions you can draw from it. With prospective data, you can find things like causality. With retrospective data it is difficult to find causality.

PROSPECTIVE DATA

Prospective data is like original, traditional, scientific experiment design. We have a hypothesis H we want to test and design experiment, with controls, to test H. we collect data to try to determine the cause, before analysing results and see if they confirm H.

Ex: Clinical trials, gene knockout experiments, etc.

⇨ Very expensive and time consuming since you need a lot of samples, divide data into groups, trials are needed, etc. Today, this seems completely outdated.

Data mining is largely about **correlation**.
- Data passively collected
- Data collected for some other reason
- Data is relatively cheap to collect.
  ⇨ We can draw stronger conclusions from prospective data, but it is expensive and time consuming to collect.

OBSERVATIONAL DATA

Now, we have huge observational data sets. Ex: Web logs, customer transactions at retail stores, human genome, etc.

It makes sense to leverage available data since it may contain useful information and it appears very cheap to collect.

The assumptions of experimental design are violated.

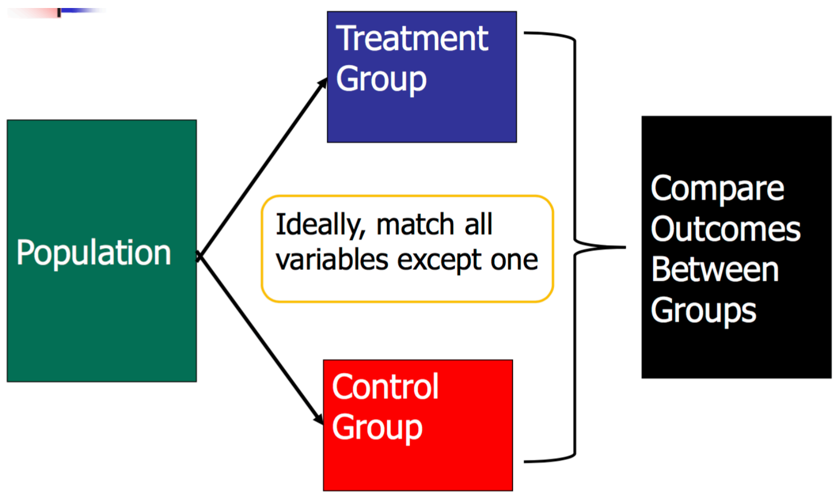
- How can we use such data to do science?
- Can we do model exploration, hypothesis testing?

One exception: the web. It tends to be very easy to experiment on the Web as it counts a huge number of users. Amazon for instance has a huge number of users, which makes it easy to quickly and easily collect data.

Amazon applies a recommendation system. Greg Linden's shopping cart recommendations add items based on what is in shopping cart. A marketing vice president states that it might distract people away from checking out. This result in a prohibition to continue. He disobeyed and ran a test to see and found not having recommendations was costing Amazon a lot of money.

This illustrates that our intuitions are often wrong. Why should we debate if we can collect data? Cultural reasons:

- Upton Sinclear: it is difficult to get a man to understand something when his salary depends upon his not understanding it
- Corollary: no not trust the HIPPO: Highest Paid Person's Opinion. By definition, these people are probably biased in their opinion.

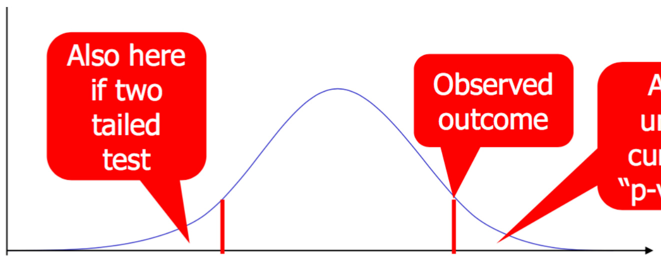

Treatment Group
Population
Ideally, match all variables except one
Control Group
Compare Outcomes Between Groups

We have two groups: the treatment group and the control group. We are gonna divide our population, usually 50-50 among these groups. The treatment group will see one variant of the website; the control group will see another variant. We will then compare the outcomes between the two groups. We could for instance see how much money people spend in one group.
This test for causation not correlation. If it is done correctly, there are only two things that can explain the change in outcome.

- A vs. B
- Random Chance

Everything else affects both variants. Statisticians try to answer if A or B is better with a formal hypothesis test. Note that this does not solve everything.

Hypothesis testing outline: we define two hypotheses: the null hypothesis and the alternative.
The H_0 states that the original model is better whereas the alternative or H_a states that the variant is better.
We run an experiment to get data about the alternative hypothesis. We then compare how probable the observed outcome is compared to what is expected if the null hypothesis were true.
The p-value is needed to perform the testing. The v-value is the probability of the data or something more extreme under the null hypothesis. Usually we use p<= 0,05 to be confident that a difference is statistically significant. The probability of getting the outcome corresponds to the area under the curve.



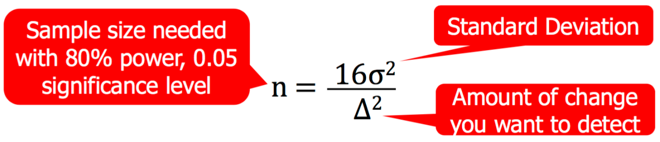
Distribution of NULL Hypothesis

Looking at hypothesis testing, what kind of errors can we make ?

- Type I Error : reject a true null hypothesis aka a false positive.
- Type II Error : Do not reject a false null hypothesis aka a false negative. The Null hypothesis is actuall false.

- Type-I Error or α
  - Reject the null when the effect isn't real
- Type-II Error or β :
  - Fail to reject the null when the effect is real
- POWER (the flip side of type-II error: 1- β):
  - Probability of seeing a true effect if it exists

Some key things to think about is the overall Evaluation criteria (OEC), a metric we are measuring. It is important to think carefully about this. Decide it upfront and do not change it.
Effect size : difference of OEC between treatment and control.



$$n = \frac{16\sigma^2}{\Delta^2}$$

Sample size needed with 80% power, 0.05 significance level
Standard Deviation
Amount of change you want to detect

- Want to detect 5% change in conversion rate
- Probability of purchase is 10%

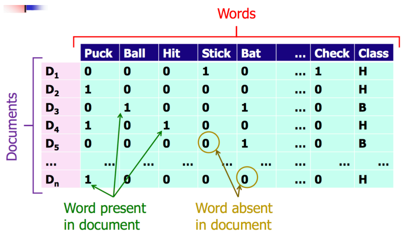
$$n = \frac{16\sigma^2}{\Delta^2} = \frac{16 * (0.1 * (1\text{-}0.1))}{(0.1 * 0.05)^2} = 57{,}600$$

⇨ We have to stay aware of day of week and seasonal effects. Beware to of newness effects.
Assigning users to groups can be difficult as need to ensure that this is
- Random
- Consistent across platforms.
- No interactions between different experiments, no correlations.
- Fast, people are intolerant to slowness of operations.

2. Types of data

⇨ What can data look like?

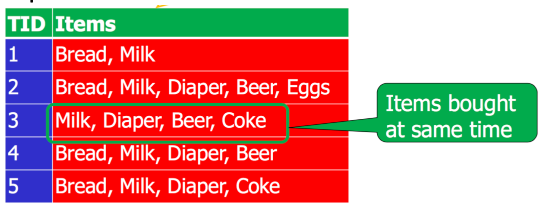
- **Feature vector**



	Puck	Ball	Hit	Stick	Bat	...	Check	Class
D_1	0	0	0	1	0	...	1	H
D_2	1	0	0	0	0	...	0	H
D_3	0	1	0	0	1	...	0	B
D_4	1	0	1	0	0	...	0	H
D_5	0	0	0	0	1	...	0	B
...	...	...	...	...	...	...	...	...
D_n	1	0	0	0	0	...	0	H

Words
Documents
Word present in document
Word absent in document

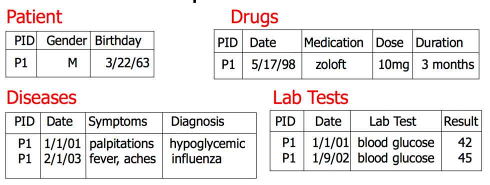
- **Transaction data** occurs for instance when visiting and analysing data on Amazon's website. Implicitly you can use it as sparse representation.



TID	Items
1	Bread, Milk
2	Bread, Milk, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Items bought at same time

- **Relational data:** extremely common sort of data representation, including tables made out of rows and columns. There might be dependencies between tables, and dependencies between rows in the table. Data may be sorted in one or several places.



Patient

PID	Gender	Birthday
P1	M	3/22/63

Drugs

PID	Date	Medication	Dose	Duration
P1	5/17/98	zoloft	10mg	3 months

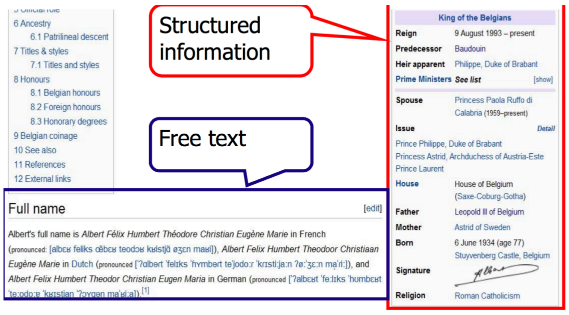
Diseases

PID	Date	Symptoms	Diagnosis
P1	1/1/01	palpitations	hypoglycemic
P1	2/1/03	fever, aches	influenza

Lab Tests

PID	Date	Lab Test	Result
P1	1/1/01	blood glucose	42
P1	1/9/02	blood glucose	45

- **Semi-structured data:** Little boxes where we get structured information about someone.



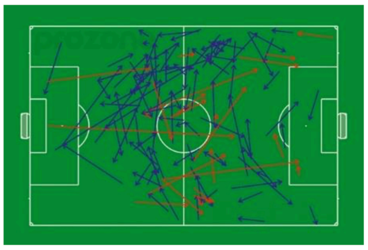
6 Ancestry
6.1 Patrilineal descent
7 Titles & styles
7.1 Titles and styles
8 Honours
8.1 Belgian honours
8.2 Foreign honours
8.3 Honorary degrees
9 Belgian coinage
10 See also
11 References
12 External links

Full name [edit]

Albert's full name is *Albert Félix Humbert Théodore Christian Eugène Marie* in French (pronounced: [albɛʁ feliks œbɛʁ teodɔʁ kʁistjɑ̃ øʒɛn maʁi]), *Albert Felix Humbert Theodoor Christiaan Eugène Marie* in Dutch (pronounced [ˈʔɑlbərt ˈfelɪks ˈɦʏmbərt teˈjodoːr ˈkrɪstiːjaːn ʔøːˈʒɛːn maˈriː]), and *Albert Felix Humbert Theodor Christian Eugen Maria* in German (pronounced [ˈʔalbɛʁt ˈfeːlɪks ˈhʊmbɛʁt ˈteːodoːɐ ˈkʁɪstian ˈʔɔʏɡən maˈʁiːa]).[1]

King of the Belgians	
Reign	9 August 1993 – present
Predecessor	Baudouin
Heir apparent	Philippe, Duke of Brabant
Prime Ministers	*See list* [show]
Spouse	Princess Paola Ruffo di Calabria (1959–present)
Issue	*Detail*
Prince Philippe, Duke of Brabant Princess Astrid, Archduchess of Austria-Este Prince Laurent	
House	House of Belgium (Saxe-Coburg-Gotha)
Father	Leopold III of Belgium
Mother	Astrid of Sweden
Born	6 June 1934 (age 77) Stuyvenberg Castle, Belgium
Signature	[illegible]
Religion	Roman Catholicism

- **Graph:** Molecules
- **Sequencial data:** DNA. This is one type of ordered data
- **Time Series Data**
- **Spatial Data:** could be locational data for instance.
- **Spatial-Temporal**



3. Simple descriptive statistics

First get an overall sense of the data, analysing data type of variables for instance: numerical data, text data. In other words, first look at obvious things like: number of variables, number of data points, missing values, class skew for prediction problems, etc. Missing values are tricky! Class skew looks at number of positive, negative instances in the sample for instance.
Next, dive into the individual variables.

- Discrete variables: For a certain variable you could for instance look at the number of values this variable can take. How often does each value occur, etc. Which values are ordered, mode (most common value)?
- Continuous variables: look at the mean, sample variance, median, quartile. This gives an idea about how the data is distributed.
  - Q1: value at position 0,25n
  - Q3: value at position 0,74n
  - Interquartile range: Q3-Q1

When looking at averages, we are often interested in weighted averages. The mean assumes that each data point is of equal importance in average. But actually, some data may be more important than other data. So, the estimate will be more reliable, more valuable, more representative, more recent, by taking the weight of each individual variable into account.

$$\mu_X = \frac{\sum_i w_i x_i}{\sum_i w_i}$$

Simpson's paradox, or the Yule–Simpson effect, is a paradox in probability and statistics, in which a trend appears in different groups of data but disappears or reverses when these groups are combined. It is sometimes given the descriptive title reversal paradox or amalgamation paradox. This result is often encountered in social-science and medical-science statistics, and is particularly confounding when frequency data is unduly given causal interpretations. The paradoxical elements disappear when causal relations are brought into consideration.

The most famous example is the UC-Berkely admissions. UC-Berkley got sued for gender bias in grad school admissions in the 70s.

- Men's acceptance rate: 44%
- Women's acceptance rate: 35%

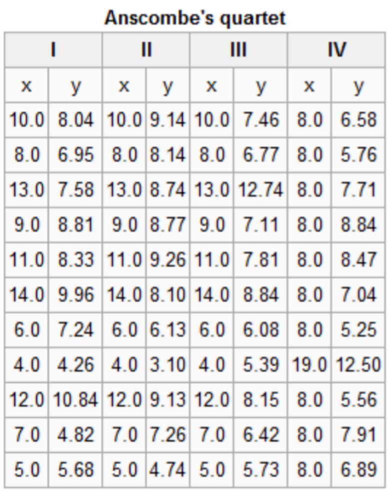
It was observed that no department was biased against women. Actually, most slightly favoured women. Women tended to apply to competitive programs with lots of applicants and low admit rates.

Describing networks

- Geodesic: shortest_path(n,m)
- Diameter: max(geodesic(n,m)) n,m actors in graph
- Density: number of existing edges / all possible edges.
- Degree distribution, counting each node, attributes they have.

4. Visualization

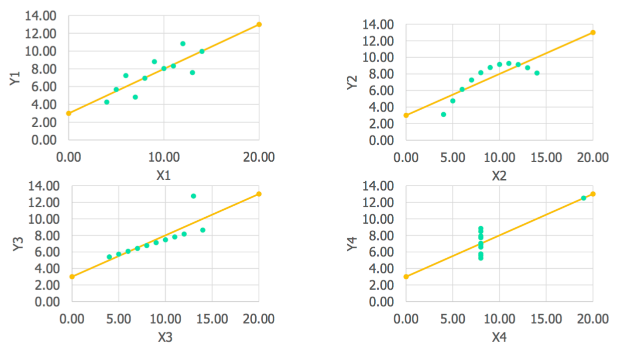
This figure shows four data sets, made of simple data. In each data set, the x variable has a mean of 9, and Y has a mean of 7,5. Each has same correlation(X,Y): 0,82. They also all have the same linear regression: y = 3 + 0,5x.



Anscombe's quartet

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

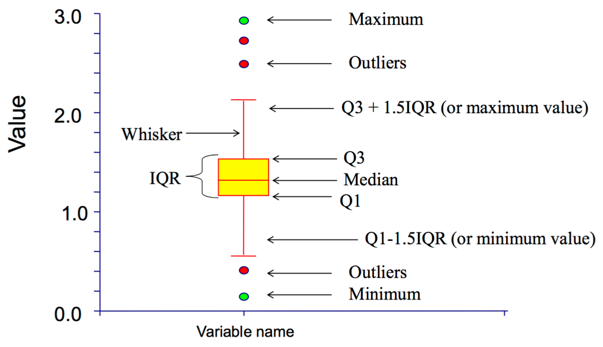
Looking at the data, there are many different ways to plot it and represent this visually.



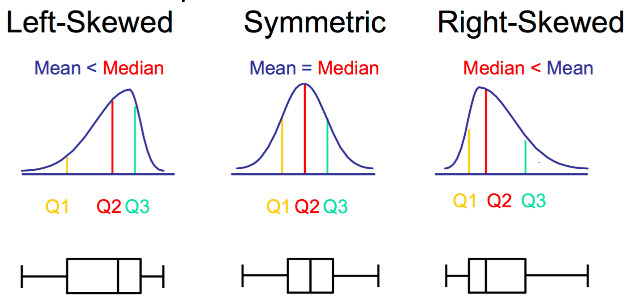
Linear regression lines have been drawn. We observe that each graph is different. The two on the left are more or less the same. The bottom less is more or less a straight line, except the outlier. On the right, the shape of the distribution is different.

The graph on the top has a curved shape. Whereas on the bottom right, it looks like a straight line.

Next, we can look at the box plot of the data. With continuous data, people use this kind of plots.



Another common visualization is the bar chart. There are different positions, and each position includes a count. We can get a good insight into what's going on.
Histograms are another way to plot continuous variables. We divide data into bins, usually each of equal width, we then count the number of points in each bin. Regarding histograms, the bin size is important! We can also look at the distribution' shape.



To conclude, typically, you start the analysis by trying to understand the data. Is it observational or prospective? What is being measured? What does the data look like?

CHAPTER 3: INTRODUCTION TO PREDICTIVE MODELING IN DATA MINING

This chapter is close to machine learning, we will cover some algorithms and see how they come in handy when analysing data.

1. Inductive learning

1.1 Scoring problems

- Given: Data S = {(x1, y1),...,(xn, yn)}
- Learn: Function F: x -> y

⇨ If Y is discrete, we will use classification, if it is continuous, we use regression.

Very classic examples of scoring problems are credit scoring, where you have to predict who is likely to default on loan. The score is the probability people will repay. Another examples relates to marketing. Marketing costs a lot of money, you have to mail people, call, … Target marketing allows us to identify people who are more likely to respond. Doing this, a score could be a probability of someone responding to the marketing campaign. Finally, churn prediction is another common thing, where you rank customers who are likely to switch to another service, drop service. You will provide incentives to those most likely to leave.

1.1.1 A regression approach

The goal of regression is to learn a function to approximate E[Y|X][1] for each X.

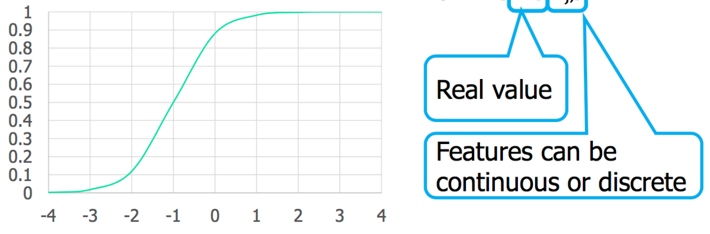
$$\begin{aligned} E[Y \mid X] &= \Sigma_y \; P(Y \mid X) \; * y \\ &= P(Y=1 \mid X) * 1 + P(Y=0 \mid X) * 0 \\ &= P(Y=1 \mid X) \end{aligned}$$

⇨ We try to estimate she posterior class probabilities = binary classifier.
Ex: What's the probability that a client responds to my advertising campaign.

The most common thing we can do for this is the logistic regression. This is a discriminative model for learning the probability of Y given X. We don't care about modelling the single values of X, modelling the probability of X.
This is a sigmoid applied to linear function of data.

[1] Expected value of Y given X

$$P(Y_j{=}0 \mid X_j, w) = \frac{1}{1 + \exp(w_0 + \Sigma_i \boxed{w_i}\, \boxed{x_{j,i}})}$$



2. Features

- There are binary variables, which means that for each word, we create a binary variable that is either O or 1.
- There are discrete variables, that we take on k values. We will use k-1 values.
- Real: just as is
- Complex variables
  - Democrat or independent
  - Word A =Present AND Word B = Present
  - Real sin(x) or X_1X_2

In logistic regression, we have two possibilities: Y = 0 and Y = 1.

$$P(Y{=}0 \mid X) = \frac{1}{1 + \exp(w_0 + \Sigma\, w_i x_i)}$$

$$P(Y{=}1 \mid X) = \frac{\exp(w_0 + \Sigma\, w_i x_i)}{1 + \exp(w_0 + \Sigma\, w_i x_i)}$$

1-Dimensional case (w > 0)

- $X \to +\infty$, then $P(Y{=}1 \mid X) = 0.5 \to 1$
- $X \to -\infty$, then $P(Y{=}1 \mid X) \to 0$
- $P(Y{=}1 \mid X) = 0.5$ when $-w_0 - w\,x = 0$ or $x = -w_0 / w$
- Location of logistic curve controlled by $-w_0 / w$
- Steepness of curve controlled by w

Both are linear classifiers!

- Naïve Bayes

$$\log\frac{P(Y_j{=}1 \mid x_j)}{P(Y_j{=}0 \mid x_j)} = \log\frac{P(Y{=}1)}{P(Y{=}0)} + \Sigma_i \log\frac{P(X_i = x_{j,i} \mid Y_j{=}1)}{P(X_i = x_{j,i} \mid Y_j{=}0)}$$

- Logistic regression

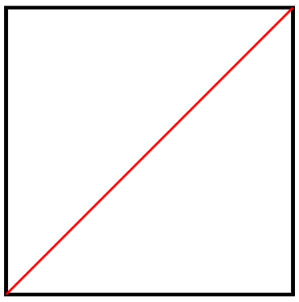
$$\ln\frac{P(Y{=}1 \mid X)}{P(Y{=}0 \mid X)} = w_0 + \Sigma\, w_i x_i$$

Why pick up logistic regression? In many tasks, calibration of the probability estimates is important. By estimating a probability, it should be a probability. Ex: if say 75% of being in positive class, then in test set you would expect ¾ of the instances with estimate to be truly positive.

However, not all classifiers are well-calibrated. For instance, Naïve Bayes is not whereas logistic regression is well calibrated. If you look at probabilities provided by Naïve Bayes, you don't get good probabilities.

3. Calibration check

Perform the check on validation data, divide predicted probability estimates into multiple bins. Plot Y-Axis, for each bin, the number of positive examples in the bin or the total number of examples in the bin. The X axis represents the predicted probability. A good calibration remains close to the diagonal.



4. Bayes Rule & Naïve Bayes Classifiers

Let's consider a supervised learning problem in which we wish to approximate an unknown target function f: X → Y or equivalently P(Y|X). We assume Y is a Boolean-valued random variable, and X is a vector containing n Boolean attributes. Applying Bayes rule, we see that:

$$P(Y=y_i|X=x_k)=\frac{P(X=x_k|Y=y_i)P(Y=y_i)}{\sum_j P(X=x_k|Y=y_j)P(Y=y_j)}$$

⇨ Intractable sample **complexity** for leaning Bayesian classifiers, hence, we must look for ways to reduce this complexity. The Naïve Bayes classifier does this by making a conditional independence assumption that dramatically reduces the number of parameters to be estimated when modelling P(X|Y).

The Naïve Bayes classifier assumes all attributes describing X are conditionally independent given Y. this dramatically reduces the number of parameters that must be estimated to lean the classifier.

Given three sets of random variables X, Y and Z, we say that X is **conditionally independent** of Y given Z, if and only if the probability distribution governing X is independent of the value of Y given Z.

$$(\forall i,j,k)P(X=x_i|Y=y_j,Z=z_k)=P(X=x_i|Z=z_k)$$

The Naive Bayes algorithm is a classification algorithm based on Bayes rule and a set of conditional independence assumptions. Given the goal of learning P(Y|X) where X = ⟨X_1 ...,X_n⟩, the Naive Bayes algorithm makes the assumption that each X_i is conditionally independent of each of the other X_ks given Y, and also independent of each subset of the other X_k's given Y. This assumption dramatically simplifies the representation of (X|Y) and the problem of estimating it from the training data.

$$\begin{aligned} P(X|Y) &= P(X_1, X_2|Y) \\ &= P(X_1|X_2, Y)P(X_2|Y) \\ &= P(X_1|Y)P(X_2|Y) \end{aligned}$$

5. Logistic regression, training task

Logistic Regression is an approach to learning functions of the form f: X→ Y, or P(Y|X) in the case where Y is discrete-valued, and X = (X_1, …, X_n) is any vector containing discrete or continuous variables.

Logistic Regression assumes a parametric form for the distribution P(Y|X), then directly estimates its parameters from the training data. The parametric model assumed by Logistic Regression in the case where Y is Boolean is:

$$P(Y=1|X) = \frac{1}{1+\exp(w_0+\sum_{i=1}^n w_i X_i)}$$

and

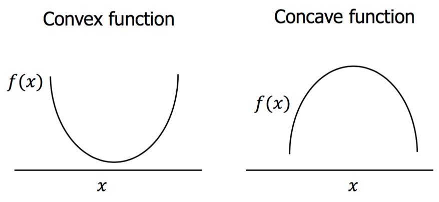
$$P(Y=0|X) = \frac{\exp(w_0+\sum_{i=1}^n w_i X_i)}{1+\exp(w_0+\sum_{i=1}^n w_i X_i)}$$

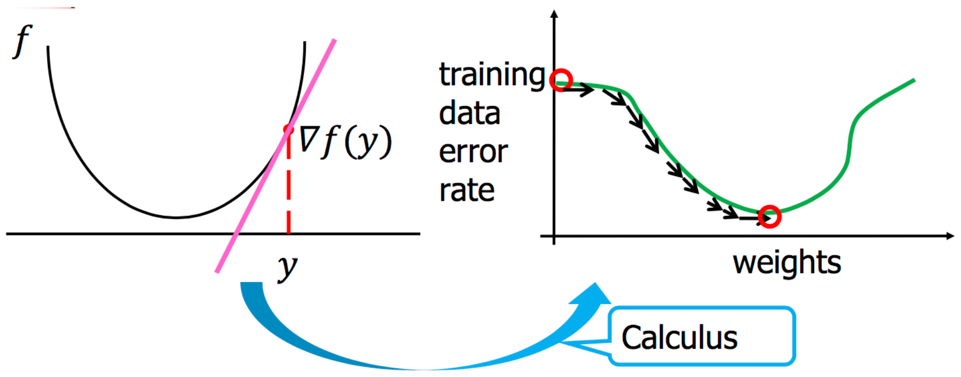
Both equations must sum to 1.

Given: Data D = ((x_1, y_1),…,(x_n,y_n))

You want to learn weights that **maximize the initial probability**: P(Y|X). In other words, the maximum weigh that push all predicted probability to zero for negative examples and to 1 for positive examples. The good news is that this function is concave, which means this function is easy to optimize. However, there is no closed-form solution, which is bad news.

$$\text{Argmax}_w \Sigma_j P(Y_j \mid X_j, w)$$





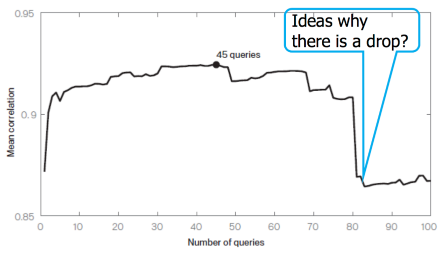
⇨ Logistic regression is often referred to as a discriminative classifier because we can view the distribution P(Y|X) as directly discriminating he value of the target value Y for any given instance X.

6. Challenge Problem

Ex: Let's say you try to predict flu outbreaks. Your model will help to model where, when flu occurs. What data could we use? What features would we look at? What else would be important in the prediction?

We could for example get data from a pharmacy, as people often go buying some medicine when they feel sick. When we get sick, we also tweet about it on Facebook, go on the internet to get some more information, hence, social media data could be interesting too. However, social media may be biased. Nevertheless, looking at social media provides us with large amounts of data.
Logistic regression could be a useful tool to predict the tool. Ex: Influenza results in 250 000 – 500 000 deaths in the world each year. Ruckly detecting outbreaks can reduce risk. Currently, the centre for disease control tracks this by aggregating counts of people with flu like illnesses: you go to the doctor, makes a guess, then he sends it to the CDC, which aggregates it. However, this tends to be slow and not timely enough, because data needs to be sent and then aggregated.

The idea now is to exploit query logs. The goal is to predict the proportion of doctor visits that are flu-related as calculated by the centre for disease (CD). Data includes historical Google Search data. A model would be the logistic regression. Features: queries most correlated to target variable, to try to select the right features.



There are several guesses about the drop. Flus are correlated with the flu season. But other queries are correlated too with seasons.

Often times, there is a variable not be used in the analysis that is correlated with the target variable, Y, and the predictor variables, X. in this example, it is season. In causal studies, these are called confounding variables. This occurs often, watch out for it!

Another example is **credit scoring**, where a person applies for a loan. This application is either rejected or accepted. However, some people accepted for a loan may default. The goal is to predict for those granted a loan, who will default on it.
Traditionally loans were granted by experts. However, there were scepticism that automation is better than experts. Hence, it was first adopted in credit-card approvals. Later it was then broadly adopted in home-loans, etc. Now, it is widely accepted and used by almost all banks, credit-granting agencies, etc.

Customer provided data: Age, address, income, job, number of credit cards, savings, etc. We also get internal data about customer: how long with the bank, previous loans, etc. We also buy external customer-level data like credit reports and legal proceedings. Also macro-level data comes in handy: demographics (post code for instance).

- ⇨ This is challenging as data might be incorrectly entered. People also deliberate false information, entering a higher income or lower income than the true one.
  Also legal issues may occur: illegal to use race, colour, religion, national origin, sex marital status, or age in the decision to grant credit.

The approach we take relates to logistic regression and decision trees. We want to make sure that data is relevant and fits customers. We need to access to relevant variables for these customers.
Defining labels remains a tricky problem.

Practical issues

- Cost/benefit analysis
  - Saving of employing system, small gains in accuracy may be very valuable.
  - Costs of developing, testing, maintaining, checking legal requirements, etc.
- Other points:
  - Set threshold: when should loans be granted
  - Override: bank employees can ignore system
  - When is a new model needed?
  - Continuous evaluation.

Another place where logistic regression is used is in suggesting Facebook Friends. How does Facebook come up with people we may know? The first insight is that most come from friends of friends. The problem is that there are lots of candidates. The average user has 130 friends: 17000 candidates. Hence, the trick is to define lots of features, exploiting the right data.
You can think of things like the number of friends in common, a good friend of yours just friended this person, demographics (age, gender, etc.).
A model is then built, to predict whether a user will click on a given suggestion. What is the probability that the user clicks on it? Therefore, a logistic regression (+ decision tree) is used to produce a ranking.

One final application in logistic regression is the prediction of CTR for advertising. When you Google, you enter a keyword.

On the right you have ad slots. The links are algorithmic searches. Every time someone clicks on the add, you get a certain amount.
Advertisers

- Bid x$ per keyword w
- Pay x$ every time someone clicks on add
- Gives a maximum budget
  - ⇨ Someone searches on a keyword w, which ad should the search engine display?

Naïve solution: just return the ad with the highest bid on w.
Google's insight: profit = x$ * click through rate (CTR). Display based on this CTR.

To sum up, we often want to assign a score to examples in prediction problems. The logistic regression is a workhorse in practice for this types of problems. There are many relevant industry applications.

CHAPTER 4: RECOMMENDING PRODUCTS

⇨ How would you recommend products? What kind of data do we need? How do you collect this?
We could for instance look at products that are usually sold together. We could also look at everything the customer has bought, and cluster among customers. Another option could be to look at similar customer profiles.

There is an extensive class of Web applications that involve predicting user responses to options. Such a facility is called a **recommendation system**.
Ex: offering news articles to on-line newspaper readers, based on a prediction of reader interest; offering customers of an on-line retailer suggestions about what they might like to buy, based on their past history of purchases or product searches.

1. Problem overview

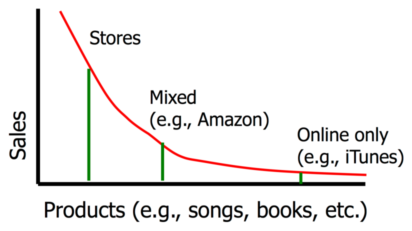
When we think about **economics of traditional retailer market**, space is scarce and expensive commodity.

- Retailers: physical. You need some space to rent. Physical delivery systems are characterized by a scarcity of resources.
- TV networks: time
- Movie theatres: space and time.
- Brick-and-mortar stores have limited shelf space, can show the customer only a small fraction of all the choices that exist.
  >< on-line stores can make anything that exists available to the customer.

⇨ People are not willing to travel far for products (buying, movie, etc.), so you need to have a customer based.
Implication: focus on popular products.

On the other hand, if you have an **online economy**, for instance on the web, storage space is cheap, it is much cheaper to store things on the web. Next, sites cater to everyone, you are not restricted to people near you, you can reach people all around the world.
Implication: low cost and easy access, which means it is possible to offer more choice.

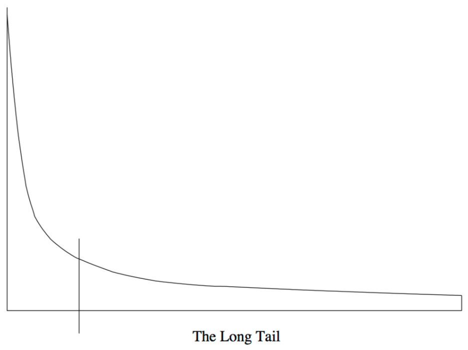
⇨ Problem: how can we find products given huge number of choices?
Solution: systems that can recommend products, particularly unusual or unpopular ones.



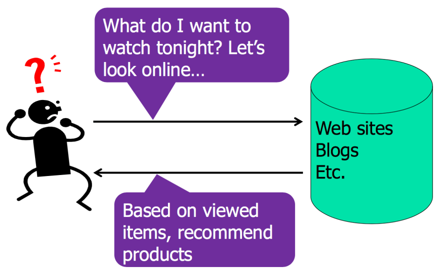
Ranking of products and sales.

- **Physical stores**, they can stock physical products, limited offering.
- **Mixed**: Amazon, on the internet but they also have physical retailers
- **Online only**: purely online. Ex: iTunes

The distinction between the physical and on-line worlds has been called the **long tail phenomenon**. Items are ordered on the horizontal axis according to their popularity. Physical institutions provide only the most popular items to the left of the vertical line, while the corresponding on-line institutions provide the entire range of items: the tail as well as the popular items.


The Long Tail

Making recommendations. Ex: what do I want to watch tonight? Let's look online. What website could I use? Based on the viewed items, the system will recommend me a product.


What do I want to watch tonight? Let's look online…
Web sites
Blogs
Etc.
Based on viewed items, recommend products

An important concept in recommendation-system application is the **utility matrix**. There are two classes of entities, which we shall refer to as users and items. Users have preferences for certain items, and these preferences must be teased out of the data. The data itself is represented as a utility matrix, giving for each user-item pair, a value that represents what is known about the degree of preference of that user for that item.

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

without a utility matrix, it is almost impossible to recommend items. However, acquiring data from which to build a utility matrix is often difficult. There are two general approaches to discovering the value users place on items:

- Ask users to rate items
- Make inferences from users' behaviour

⇨ Goal of a recommendation system is to predict the blanks in the utility matrix. It is not necessary to predict every blank but only to discover some entries in each row that are likely to be high.

⇨ Find a large subset of those with the highest ratings.

1.1 Recommendation types

There exist loads of recommendation types.

- **Editorial**: these are list of favourites. Ex: newspapers, travel magazines. News services have attempted to identify articles of

interest of readers, based on the articles that they have read in the past. Might be based on the similarity of important words in the documents, or on the articles that are read by people with similar reading tastes.
You can also have lists of essential items.

- **Aggregates**
  - Top 10 lists
  - Most emailed articles
  - Most recent posts
- **Personalized user recommendations:** They make recommendations based on your preferences. Perhaps the most important use of recommendation systems is at on-line retailers.
  - Amazon
  - Movie sites (Netflix): Netflix offers its customers recommendations of movies they might like. These are based on ratings provided by users.

1.2 Challenges

There are **3 key challenges**

- How do we get user fee**d**back?
- How do we **predict** an unknown rating? You want to be able to nail the things.
- How do we evaluate predictions?

How could we obtain ratings? There exist many ways to do. I want to predict what a specific user will like, based on his preferences.

- Explicit rating of products
- Bought items
- Items on "wish lists"
- Recently clicked product pages/link
- Length of time spent on product page
- Printed links
- Etc.

Problem: predict ratings

- **Given** information about a user's preferences, interests, likes/dislikes
- **Predict**: will a specific user like a given product, service, etc.?

Challenges …

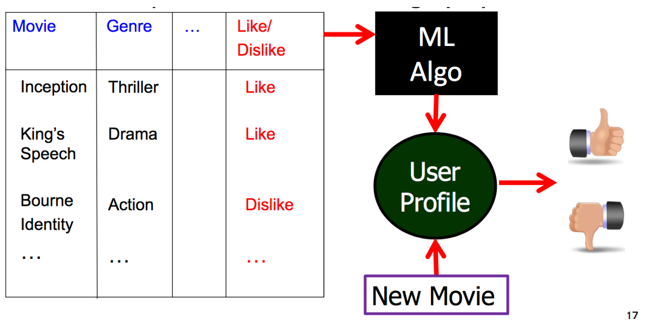
- Sparse data: most users rate very few items
- Cold start: new items have no ratings
- Main paradigms

- Content based filtering
- Collaborative filtering
- Hybrid: use both prior approaches

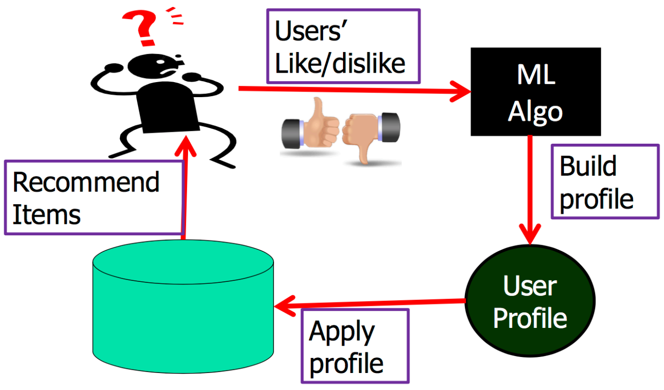
2. Content based filtering

Idea: focus on properties of items. Similarity of items is determined by measuring the similarity in their properties.

Machine learning problem, so we can just apply basic standards seen. We can easily recommend items to a customer based on profiles of the past. We try to find things that are similar to what the customer has visited before, similar to previous items rated highly by the customer.



Movie	Genre	…	Like/ Dislike
Inception	Thriller		Like
King's Speech	Drama		Like
Bourne Identity	Action		Dislike
…	…		…



So we need to define a number of features, and obtain data from the users. We will then apply any machine learning algorithm to this data.
Ex: for every movie, I construct features (actors, genre, director …), I then try to find out whether the user will like the movie.

A key **challenge** is building item profile. An item profile is a way to describe each item. Usually these are hand-crafted.
You have to spend a lot of time building this profile. Usually, for such tasks, they are not done in an automated way.
Ex: for a news article: Words, title, author, etc.

Pro's	Con's
• Only need data about one user. No need to have data about millions of users. • Results in a more personalized approach (Ex: good if user has unique taste). • More easily recommend new/unpopular items • Can provide context for recommendation (Ex: path down decision tree) • Lots of research on classification algorithms. Every time there is a new one, I use it instead of the old one.	• Need to construct meaningful features that are predictive of user's preferences Ex: lots of hand-coding, guess & check) • Never recommends items outside user's content profile • Hard to build a profile for a new user. I need to have sufficient number of examples in order to recommend something. • Ignores information about other users

⇨ Supervised classification problem

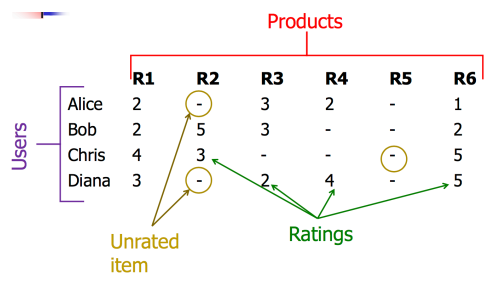
We not only need to create vectors describing items; we need to create vectors with the same components that describe the user's preferences; the utility matrix represents the connection between users and items. The best estimate we can make regarding which items the user lies is some aggregation of the profiles of those items.

A completely different approach to a recommendation system using item profiles and utility matrices is to treat the problem as one of machine learning. Regard the given data as a training set, and for each user, build a classifier that predicts the rating of all items. The construction of a decision tree might be handy. Nevertheless, it requires the selection of a predicate for each interior node. There are many ways of picking the best predicate. Once the predicate has been choosing, we divide the items into the two groups: those that satisfy the predicate and those that do not.

3. Collaborative filtering

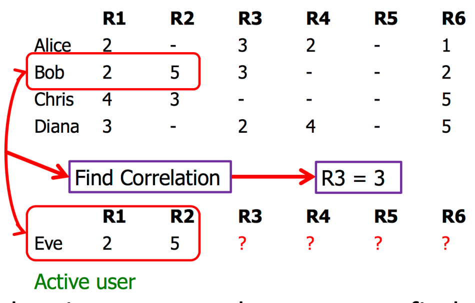
Idea: Find users with similar tastes and recommend products they liked. The big insight is that we can do this just by looking at ratings, and just use these ratings to make the predictions, no need to look at features.

So, the big idea is to find other users whose ratings are similar to the current user. And then propagate the (dis)likes to the current user.



	R1	R2	R3	R4	R5	R6
Alice	2	-	3	2	-	1
Bob	2	5	3	-	-	2
Chris	4	3	-	-	-	5
Diana	3	-	2	4	-	5

Standard matrix where each row corresponds to a user, so we have loads of users. Each column is a product. The entries are the ratings. There are many users, many products, and most of the entries will be empty. Very sparse! Most of the entries are missing here.



	R1	R2	R3	R4	R5	R6
Alice	2	-	3	2	-	1
Bob	2	5	3	-	-	2
Chris	4	3	-	-	-	5
Diana	3	-	2	4	-	5

	R1	R2	R3	R4	R5	R6
Eve	2	5	?	?	?	?

So, we have our database, and active users, and we want to find similarities between the current user and all the other users in the database. For instance, Bob and Eve have the same ratings, this rating is different from Alice, Chris, etc. We then try to find out how strong the similarity is.
The algorithm works in **four steps**.

- **STEP 1:** Measure similarity between user of interest and all other users. The question is, how can we compute similarity between two users? There are loads of similarity metrics.
  I have two users, I know what the first is interesting in and I should make recommendations for the second one.

  Idea 1: Jaccard similarity: $\mathrm{sim}(S_i, S_j) = \dfrac{S_i \cap S_j}{S_i \cup S_j}$
  Problem: Ignores actual rating

  Idea 2: Cosine similarity: $\mathrm{sim}(S_i, S_j) = \dfrac{S_i \cdot S_j}{||S_i|| \cdot ||S_j||}$

  Problem: treats missing ratings as zero since it treats every entry as a vector. This treating as zero is a wrong way of doing since it is clearly not the case.

  The Pearson correlation is a possible solution.

$$W_{ij} = \frac{\Sigma_k (R_{ik} - \bar{R}_i)(R_{jk} - \bar{R}_j)}{[\Sigma_k (R_{ik} - \bar{R}_i)^2 \, \Sigma_k (R_{jk} - \bar{R}_j)^2]^{0.5}}$$

  $\bar{R}_i = \dfrac{\Sigma_e R_{ie}}{m}$ (Average user I's rated items)

  R_{ik} = User I's rating on item k

  The key thing to remember with the Pearson correlation is that it only considers items k rated by both users. This is just a standard correlation we could use. If this correlation equals 1, this is a perfect linear relationship with both increasing in the same direction. Whereas as a correlation equals to -1 means that when one rating decreases, the other one will increase.

$$W_{ij} = \frac{\Sigma_k (R_{ik} - \overline{R}_i)(R_{jk} - \overline{R}_j)}{[\Sigma_k (R_{ik} - \overline{R}_i)^2 \, \Sigma_k (R_{jk} - \overline{R}_j)^2]^{0.5}}$$

Positive if both predictions on same side of average

- **STEP 2:** (Optional): select a smaller subset consisting of most similar users. You could use the whole database. But there would be loads of people with no correlation, that's why we will ignore "far away" users, and pick only the "k" nearest users. We could also decide just including all users above a predetermined weight threshold.
  Note: could use any k-NN strategy here.
- **STEP 3:** Predict ratings as a weighted combination of "nearest neighbours".
  Now we want to predict ratings. One big problem here is that people have different rating scales. So, one user might use 5 stars, while others may use only 3. We will first compute an average rating, and then try to implement an algorithm to find out about the nearest users.

$$P_{ik} = \overline{R}_i + a \, \Sigma_j W_{ij} (R_{jk} - \overline{R}_j)$$

$$a = \frac{1}{\Sigma |W_{ij}|}$$

$$W_{ij} = \text{Pearson correlation}$$

One potential problem here is that correlations which are based on very few co-rated items may be inaccurate.
Simple solution adjust the Pearson correlation based on number of co-rated items. I take some threshold. I then adapt the correlation.

$$W'_{ij} = \begin{cases} W'_{ij} \ * k / 25 & \text{if } k <= 25 \\ W'_{ij} & \text{if } k > 25 \end{cases}$$

- **STEP 4:** return the highest rated items. We don't want to recommend items with a low correlation rate. Next, we don't want to overload the users, for instance, returning thousands of possible products.
  We usually return the top 2, 5 or 10 items. It is then possible to give the user the option to view more items.

⇨ Could you employ the basic collaborative filtering algorithm between pairs of items? Would this work?

⇨ An example is provided in the slides. Given a user movie pair, I want to predict the rating.

The big practical issue is that we have large data sets: we could have millions of users and 100 000s of items. The key issue is how to efficiently find similar items? I want to compute pairwise similarity between users.

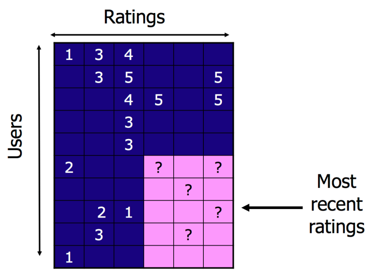
- Pairwise similarity for 1M users: +- 5 days
- Pairwise similarity for 10M users: +- 1 year!

The trick is to use the **Locality Sensitive hashing**.

Pro's	Con's
• Simple and intuitive approach • Works for any kind of item ○ No feature design ○ No feature selection • Works between pairs of Users • Exploits information about other users/items	• **Data sparsity**: even lots of data, hard to find users that rated the same items. • **Cold start**: need for enough users in database • **First rate**: can't recommend unrated items ○ Now product ○ Unique items • **Popularity bias**: favours items that lots of people like (i.e., bad if you have unique taste).

4. Evaluation

⇨ How do we evaluate these algorithms? Here, we have a huge rating matrix. We think about the matrix, taking out the most recent ratings. We then make predictions on these most recent ratings.



The two most typical metrics are **Root-Mean Square error** (RMSE) and the **Rank Correlation Spearman's.**

Root-mean square error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n} \Sigma_k (P_k - R_k)^2}$$

Precision at top 10: % of those in top 10

Rank Correlation: Spearman's, r_s, between system's and user's complete rankings

$$R_s = 1 - \frac{6\Sigma_k (U_k - S_k)^2}{n(n^2 - 1)}$$

Assume 0/1 data

- Coverage: number of items/users for which the system can make a prediction. You want to be able to make as many predictions as possible.
- Precision: accuracy of predicted items
- Receiver operator characteristic (ROC) curve
    - False positive rate
    - True positive rate

⇨ **Weakness** with Metrics

- Focusing on accuracy misses important points
    - Prediction **diversity**
    - Prediction **context**
    - **Order** of prediction
- Only high ratings matter: RMSE might penalize a method that does well for high ratings and badly for others.

5. Case study: Netflix challenge

Netflix is a monthly subscription service, providing movies and TV. The original model was a Mail order DVD service. People could buy digital rights to many movies. It was also possible to rank movies of interest and receive in mail based on movies you want to see.

Today, things have changed, we now face a new model: streaming online content. Capitalize on "binge watching" create their own content

In **2009**, Netflix had

- 100,000 movies
- 10 millions customers
- More US-based

Currently it tends to be very international. It was first based in the US, now it is very international.

- The content based on region.

- In US: ~5,000 movies and ~2,000 TV shows.
- 75 million subscribers

They wanted to have a **recommender system**. The idea was to create a competition for 1 million $. That's why, in 2006, they announced the Netflix Prize, a machine learning and data mining competition for movie rating prediction. $1 million was offered to whoever improved the accuracy of the existing system called Cinematch. Proposed algorithms were tested on their ability to predict the ratings in a secret remainder of the larger dataset.
A team came up with the final combination of 107 algorithms. To put these algorithms to use, they had to work to overcome some limitations, for instance that they were built to handle 100 million ratings, instead of the more than 5 billion that they have. They were not built to adapt as members added more ratings.

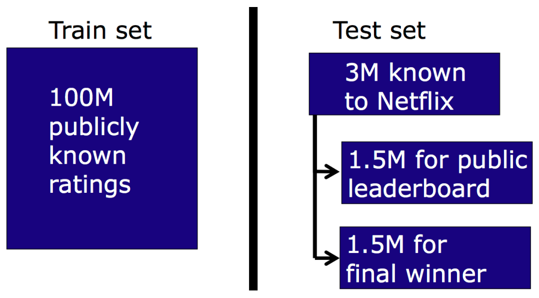
Given: a training set of 100 million rating
Do: build a recommendation system that improves the root-mean squared error by 10% over Netflix's system.

- Provided 100 million ratings
  - Matrix is 99% sparse
  - 480,000 users
  - 17,700 movies
- Rating include
  - User
  - Movie
  - Rating
  - Time-stamp

⇨ **Why sponsor a challenge?** Netflix only makes money if they keep customers. The Cinematch, their internal system was expensive to develop and made slow progress. Next, publicity is good and the awarding prize will pay for itself.

⇨ **What is to lose?**

- Negative publicity
  - Concerns about user privacy (i.e., user backlash to data release)
  - Prize won too quickly
  - No one wins the prize
- Time and effort associated with running the competition (set up server)
- Results not useful in practice (e.g., too slow).



Score based on RMSE

Evaluation

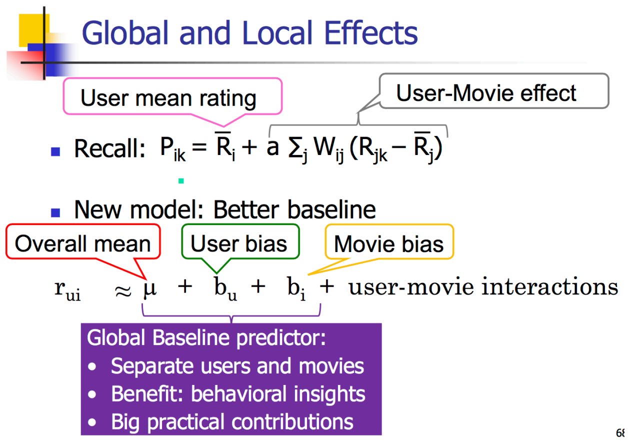
⇨ Why was the test set split? To avoid overfitting.

The competition began in October of 2006. There were many more teams than anticipated, eventually around 40 000. There were lots on lots of initial progress and within weeks a 1% improvement over baseline system was achieved.

There are **four important big ideas**

- Modelling global and local biases
- Latent factor models
- Modelling temporal dynamics
- Try lots and lots of different models and combine the output

5.1 Modelling global and local biases



Global and Local Effects

User mean rating

User-Movie effect

- Recall: $P_{ik} = \bar{R}_i + a \sum_j W_{ij} (R_{jk} - \bar{R}_j)$
- New model: Better baseline

Overall mean

User bias

Movie bias

$$r_{ui} \approx \underbrace{\mu + b_u + b_i}_{} + \text{user-movie interactions}$$

Global Baseline predictor:
- Separate users and movies
- Benefit: behavioral insights
- Big practical contributions

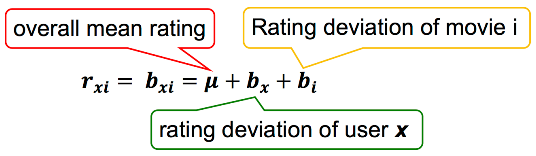
We have two components
- User mean rating
- User-Movie effect

The Better baseline rating wants to take the mean rating and divide it into three components. I want to look at the overall mean, and then add the user bias and movie bias.
- **User bias**: how much better is it than other movies. For the user bias, each user has a different rating scale. What the number means depends on the individual user. Other things affect the user bias like the values of other ratings the user gave recently (day-specific mood, anchoring, multi-user accounts).

Ex: if you are given a number, ask people to write their cell phone number, and you then ask "how many countries are there in Africa?". People will respond based on the previously given numbers.

- **Movie bias**: How much higher or low is this movie rating compared to other. You may be interested in the (recent) popularity of movie I, selection bias; related to the number of ratings user gave on the same day ("frequency"). By looking at these we have indications and expectations.

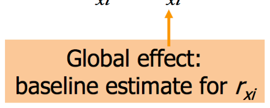
⇨ Capturing the global effects.



$$r_{xi} = b_{xi} = \mu + b_x + b_i$$

- μ: Average rating of all movies in data
- b_x: (Average rating of user x) $-\ \mu$
- b_i: (Average rating of movie i) $-\ \mu$

$$\hat{r}_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$



Problems/Issues:

1) Similarity measures are "arbitrary"

2) Pairwise similarities ignore interactions among users/items

3) Similarities are normalized to 1 and hence can overfit

Solution: Estimate weight w_{ij} directly from data

⇨ Instead of having similarities, we will replace it by the weight.

- Weighted **sum** rather than **weighted avg.**:

$$\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj})$$

- **A few notes:**
  - $N(i;x)$: movies rated by user x that are similar to movie i
  - w_{ij} is the interpolation weight (some real number)
  - w_{ij} models interaction between pairs of movies (it does not depend on user x)

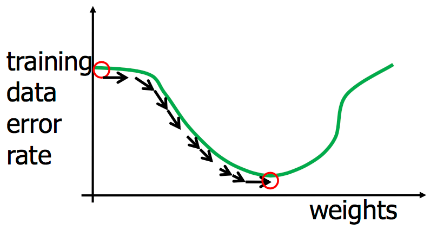
- Competition objective: Sum of squared errors

$$\sum_{(i,x)\in R} (\hat{r}_{xi} - r_{xi})^2$$

- Idea: Pick weights to minimize this objective!

$$J(w) = \sum_{x,i} \Bigg(\underbrace{\Big[b_{xi} + \sum_{j\in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \Big]}_{\text{Predicted rating}} - \underbrace{r_{xi}}_{\text{True rating}} \Bigg)^2$$

We can actually train the weights to optimize the objective function, that is to say, minimize this objective!

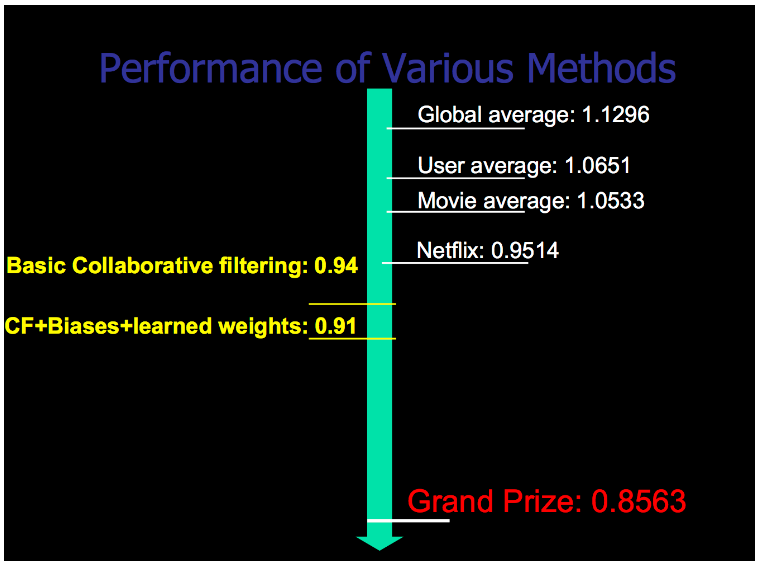


- Take initial guess for weights
- Take function derivative
- Update values of weight in direction
- Repeat until convergence

- **Global:** Baseline estimation
  - Average rating: 3.7
  - *The Sixth Sense* is **0.5** stars above avg.
  - Joe rates **0.2** stars below avg.

    ⇒ ***Joe* will rate *The Sixth Sense* 4 stars**
- **Local:**
  - *Joe* didn't like related movie *Signs*
  - **Final estimate:** Combine the two

    ⇒ ***Joe* will rate *The Sixth Sense* 3.8 stars**



Performance of Various Methods

Global average: 1.1296

User average: 1.0651

Movie average: 1.0533

Netflix: 0.9514

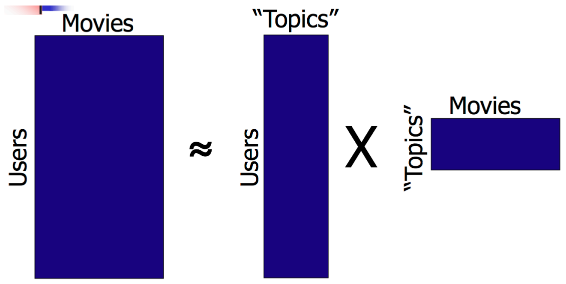
Basic Collaborative filtering: 0.94

CF+Biases+learned weights: 0.91

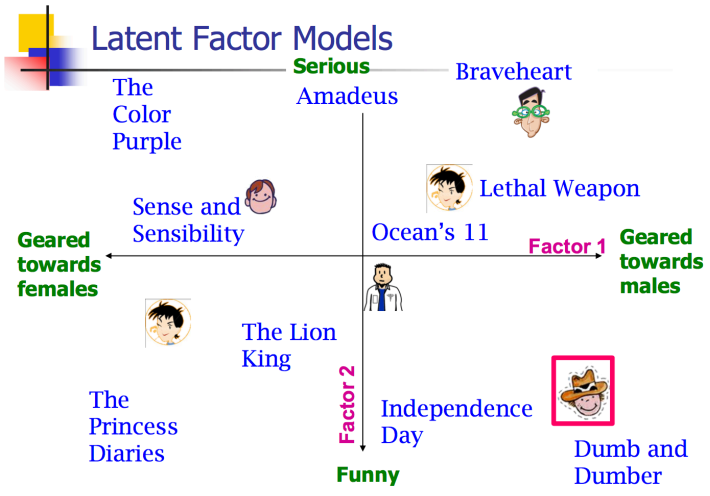
Grand Prize: 0.8563

In 2007, data appears as the challenge task at KDD and there was an associated workshop. The winners of the improvement prize were from BellKor, 8.4% improvement (Yehuda Koren, Bob Bell, Chris Volinksy, AT&T Research).

5.2 Latent factor models



Idea: make some matrix factors and decompose it into interesting, smaller parts. Topics capture shared hidden structures. You want the number of topics to be as small as possible.



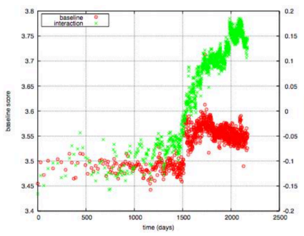
⇨ Examples are provided in the slides

Solve via an optimization problem

- Challenges:
  - Sparse matrix with many missing entries
  - Slow to compute gradient
  - Need to avoid overfitting
- Solutions
  - Solve least squares over just observed data
  - Stochastic gradient descent
  - Shrink towards mean if little data

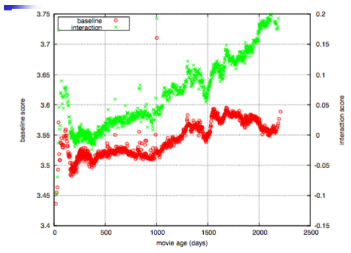
In 2008, the progress slowed down and many teams dropped out due to the time needed to complete. Many papers were published on the task.
Leaders had 9,4% improvement.

5.3 Modelling temporal dynamics



- Pre 2004: 3.4 stars
- Post 2004: >3.6 stars

In 2004, there was a considerable jump in the rating people assign to movies. Could be due to improvements of cinematch. Netflix improved matching, leading to higher rankings?
Another explanation could be that people are biased towards higher ratings: meaning of rating changes?



- High initial ratings
- Ratings increase with movie age

- **Original model:** $r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$
- **Add time dependence to biases:**
  $r_{xi} = \mu + b_x(t) + b_i(t) + q_i \cdot p_x$
  - Make parameters b_x and b_i to depend on time
  - **(1)** Parameterize time-dependence by linear trends
    (2) Each bin corresponds to 10 consecutive weeks
    $$b_i(t) = b_i + b_{i,\text{Bin}(t)}$$
- **Add temporal dependence to factors**
  - $p_x(t)$… user preference vector on day t

5.4 Try lots and lots of different models and combine the output

People are getting desperate. They tried lots of things, but still have not reached the 10% threshold.
Idea: “Kitchen Sink Approach”. Build lots and lots of predictors

- Classifiers
- Collaborative filters
- Ensembles

Come up with clever ways to blend the results

2009, at the end of June, the leading team submits results that exceed the 10% threshold. The competition enters the 30-day final period.

A new "ensemble" team is formed based on collaboration for others near at top of leader board and quickly beat the 10% threshold too.
The race is on …

Take away messages

- Knowing your data is crucial
- Must account for unpredictable users' behaviour
- Discovering hidden structure can help
- When in doubt, combine lots of models

Thanks to the competition, it served as publicity (and money) for the winning team and its members. Netflix received much attention and valuable new algorithms.
Data mining/machine learning

- Excitement and publicity for the field
- Interesting new publications
- Valuable data set

To sum up, recommender systems are an important and active area or research and use.
Three paradigms

- **Content**: based on designing features and applying classification/regression algos
- **Collaborative**: based on comparing users/items (aka nearest neighbour)
- **Hybrid**: blends both approaches

⇨ Netflix challenge was interesting and successful

One of the reasons their focus in the recommendation algorithms has changed is because Netflix as a whole has changed dramatically in the last few years. Netflix launched an instant streaming service in 2007. Streaming members are looking for something great to watch right now; they can sample a few videos before settling on one, they can consume several in one session, statistics are provided such as whether a video as watched fully or partially.

We have adapted our personalization algorithms to this new scenario in such a way that now 75% of what people watch is from some sort of recommendation. We reached this point by continuously optimizing the member experience and have measured significant gains in member satisfaction whenever we improved the personalization for our members.

Over the years, Netflix discovered that there is a tremendous value to their subscribers in incorporating recommendations to personalize as much of Netflix as possible. Personalization starts on their homepage, consisting of groups of videos arranged in horizontal rows.
Ex: Top 10 row: best guess at the ten titles people are most likely to enjoy.

An important element is **awareness**. Netflix want members to be aware of how they are adapting to their tastes. It then encourages members to give feedback that will result in batter recommendations.
Next, recommendations are partly based on our friends. Knowing about your friends not only gives Netflix another signal to use in the personalization algorithms, but it also allows for different rows that rely mostly on the social circle to generate recommendations.

Similarity is another important source of personalization in their service. Similarity can be thought of in a very broad sense; it can be between movies or between members and can be in multiple dimensions such as metadata, ratings, or viewing data.

In most of the previous contexts – be it in the Top10 row, the genres, or the similars – ranking, the choice of what order to place the items in a row, is critical in providing an effective personalized experience. The goal of Netflix' ranking system is to find **the best possible ordering** of a set of items for a member, within a specific context, in real-time. Their business objective is to **maximize member satisfaction** and month-to-month subscription retention, which correlates well with maximizing consumption of video content.

CHAPTER 5: MODEL ENSEMBLES

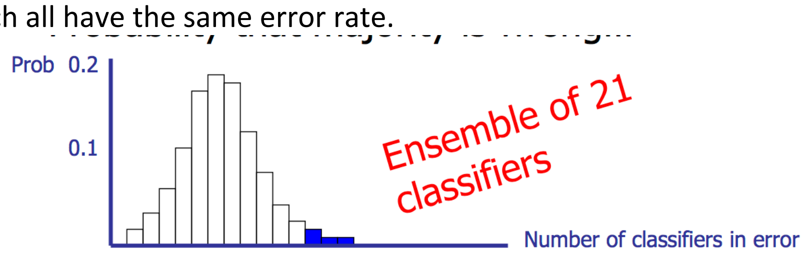
1. Motivation and overview

An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples.
One good learner produces one effective classifier: could learning many classifiers help?
The main discovery is that ensembles are often much more accurate than the individual classifiers that make them up.
An ensemble can be more accurate than its component classifiers only if the individual classifiers disagree with one another.

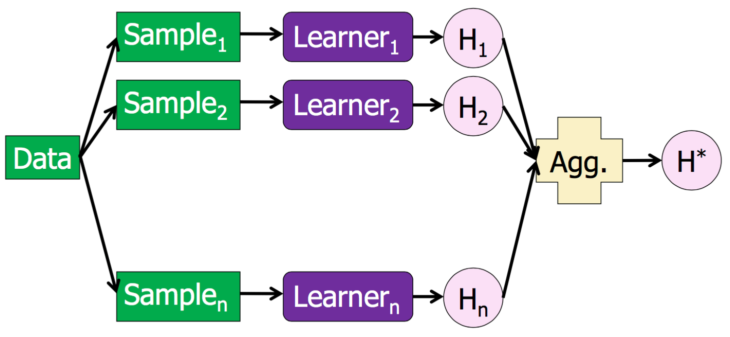
Human ensembles are demonstrably better.
Ex: how many jelly beans in the jar? Individual estimates vs. group average.
Ex: Who wants to be a Millionaire: Expert friend vs. audience vote.

⇨ If classifiers make **INDEPENDENT** mistakes, then h* (combined hypothesis) is more accurate, because the probability that they all make a mistake is low.

Let's say we assume independent errors (30%) and a majority vote. We have 21 classifiers which all have the same error rate.



Area under curve for >= 11 wrong is 0.026.
⇨ Order of magnitude improvement!



How to generate the base classifiers?

- Different learners?
- Bootstrap samples?
- Etc.

How to integrate/combine them?

- Average
- Weighted Average
- Instance-specific decisions
- Etc.

Ensemble approaches

Sample data set: create replicates in some way, or reweight the examples in some way.

- Bagging
- Boosting

Manipulate features

- **Input feature.** for instance, I can only show a subset of them, train one classifier first, then another, etc.
- **Target features**. Create a new classifier. Grouping together some labels.

Add randomness

- Data
- Algorithm

Stacking

2. Sampling-based approaches

This first method manipulates the training examples to generate multiple hypotheses. The learning algorithm is run several times, each time with a different subset of the training examples. This works especially well for unstable learning algorithms. The key idea here has to do with the stability of an algorithm.

Unstable learner: minor variations in training data result in major changes in classifier output, lead to a different model.

- **Unstable**: Decision tree, neural network, rule learning algorithms
- **Stable**: linear regression, nearest neighbour, linear threshold algorithms, etc.

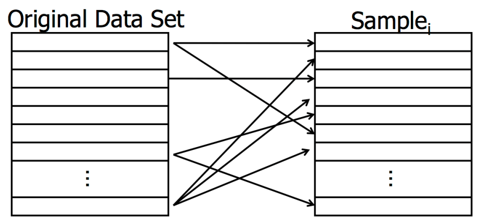
⇨ Subsampling is best for unstable learners

- Bagging
- Boosting
- Cross-validated Committees

2.1 Bagging: Bootstram Aggregating

Given: Data set S, integer T
For i = 1, ..., T
S_i = Bootstrap replicate of S
(i.e., sample with replacement)
h_i = Apply learning algorithm to Si

Classify test instance using unweighted vote



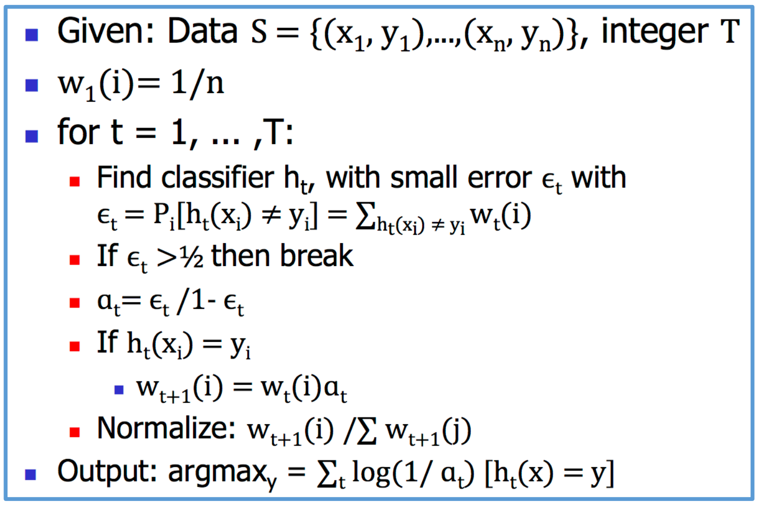
Draw |Data Set| examples with replacement. Each sample' contains 63.2% of original examples (+ duplicates)

Bagging is the most straightforward way of manipulating the training set. On each run, bagging presents the learning algorithm with a training set that consists of a sample of m training examples drawn randomly with replacement from the original training set of m items.

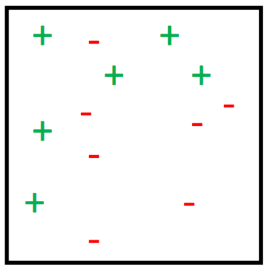
2.2 AdaBoost (Addapted Boosting)

- Idea 1: Assign weights to examples. It will iteratively change the weights.
  Decrease weight of correctly labelled examples
  Increase weight of incorrectly labelled examples =>Focus attention on misclassified examples
  The algorithm wants to decrease the focus of examples that are well-classified. I want to make sure I get inexact instances.

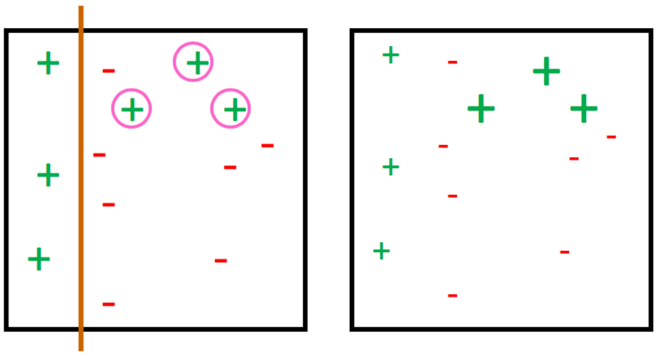
- Idea 2: Assign weights to each learned hypothesis based on how accurate it is. Weighted vote to label new examples.



```
Given: Data $S = \{(x_1, y_1),\ldots,(x_n, y_n)\}$, integer $T$
$w_1(i) = 1/n$
for t = 1, ... ,T:
    Find classifier $h_t$, with small error $\epsilon_t$ with
    $\epsilon_t = P_i[h_t(x_i) \neq y_i] = \sum_{h_t(x_i) \neq y_i} w_t(i)$
    If $\epsilon_t > ½$ then break
    $a_t = \epsilon_t / 1 - \epsilon_t$
    If $h_t(x_i) = y_i$
        $w_{t+1}(i) = w_t(i) a_t$
    Normalize: $w_{t+1}(i) / \sum w_{t+1}(j)$
Output: $\text{argmax}_y = \sum_t \log(1/a_t)\,[h_t(x) = y]$
```

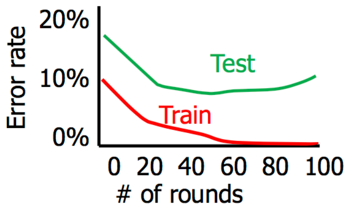


Assume that we are going to make one axis parallel cut through the feature space.



We have 3 misclassified instances. What we want to do is to assign a higher weight to these 3 misclassified examples. In brief, we up weight the mistakes and we downweight everything else.

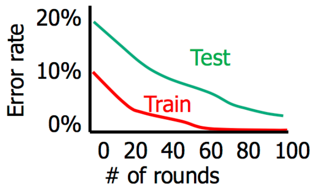
⇨ How will the number of rounds effect generalization?



Expect

- Training error to drop or reach 0
- Test error to increase when h* becomes too complex: "Occam's razor" (i.e. overfitting)
- Hard to know when to stop training

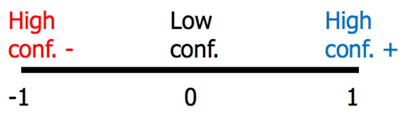
>< But of the, the test error does not increase, even after 100 rounds. The test error continues to drop, even after training error is 0!.
Occam's razor: "simpler is better" appears to not apply!



The key idea here is called **margins**. The training error only measures whether classifications are right or wrong. But it should also consider confidence of classifications.

Margins try to exploit the **confidence**.

H* is a weighted majority vote of weak classifiers. Each of the individual classifiers is better than 50-50. We can measure the confidence by margin. We take the weighted vote and calculate how much weight we have on the positive cases and how much we have on negative cases.
→ (weighted vote +) – (weighted vote -)



It turns out that boosting really tries to build up a classifier with a high margin. Boosting increases the margin very aggressively since it concentrates on the hardest examples. If the margin is large, more weak learners agree and hence more rounds does not necessarily imply that the final classifier is getting more complex.
Theorem: large margins yield better bound on generalization error. If all training examples have large margins, then we can approximate final classifier by a much small classifier. It is similar to how polls can predict outcome of a not-too-close election.

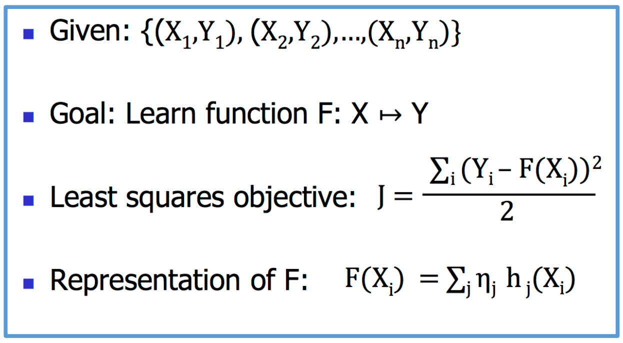
2.3 Cross-validation

Another training set sampling method is to construct the training sets by leaving out disjoint subsets of the training data.
Ex: the training set can be randomly divided into 10 disjoint subsets. Then 10 overlapping training sets can be constructed by dropping out a different one of these 10 subsets.

2.4 Gradient Tree Boosting

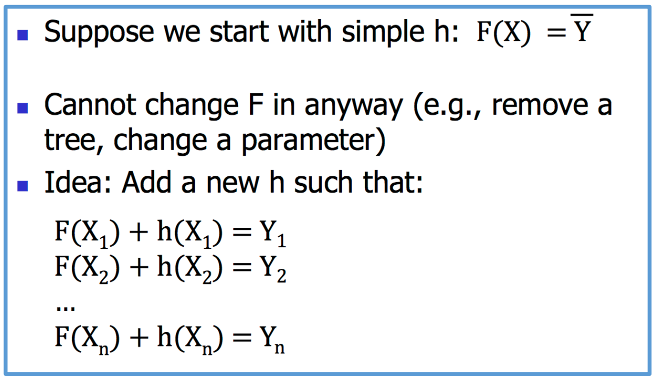
The base algorithm is old but very hyped now. We will focus on least squares regression case. We will ignore some of mathematical details.

Gradient Boosting = Gradient Descent + Boosting. It fits an additive model (ensemble) in a greedy forward stage-wise manner. Each stage introduces a weak learner to address the shortcomings of the current model. Shortcomings are identified by **gradients**.



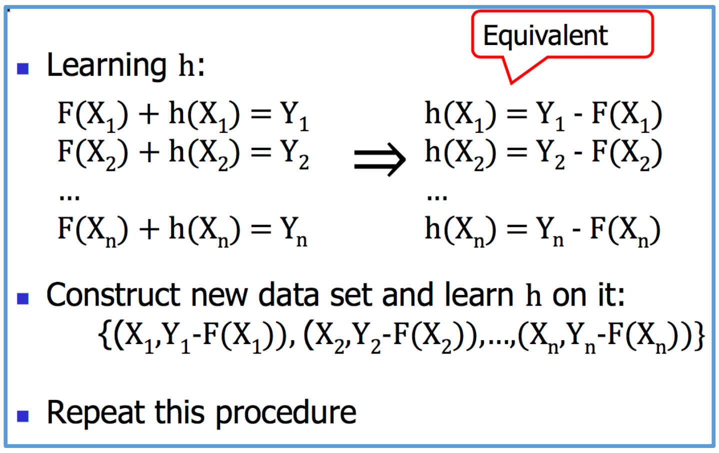
- Given: $\{(X_1,Y_1), (X_2,Y_2),\ldots,(X_n,Y_n)\}$
- Goal: Learn function F: X $\mapsto$ Y
- Least squares objective: $J = \dfrac{\sum_i (Y_i - F(X_i))^2}{2}$
- Representation of F: $F(X_i) = \sum_j \eta_j \, h_j(X_i)$

Intuitively, we start with a very **simple hypothesis**.



- Suppose we start with simple h: $F(X) = \overline{Y}$
- Cannot change F in anyway (e.g., remove a tree, change a parameter)
- Idea: Add a new h such that:

$$F(X_1) + h(X_1) = Y_1$$
$$F(X_2) + h(X_2) = Y_2$$
$$\dots$$
$$F(X_n) + h(X_n) = Y_n$$



- Learning h:

$$\begin{matrix} F(X_1) + h(X_1) = Y_1 & & h(X_1) = Y_1 - F(X_1) \\ F(X_2) + h(X_2) = Y_2 & \Rightarrow & h(X_2) = Y_2 - F(X_2) \\ \dots & & \dots \\ F(X_n) + h(X_n) = Y_n & & h(X_n) = Y_n - F(X_n) \end{matrix}$$

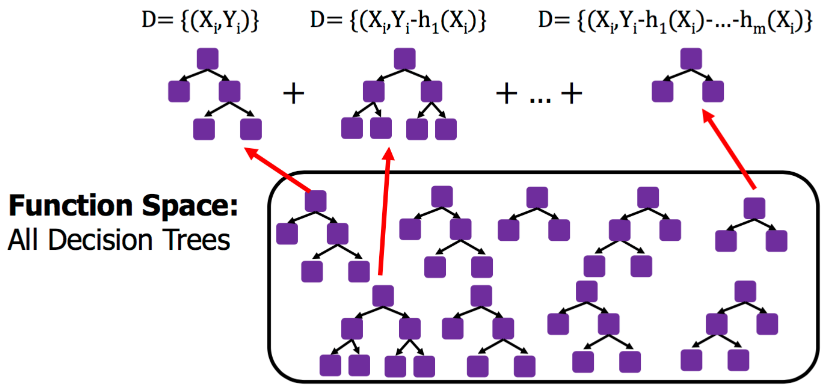
Equivalent

- Construct new data set and learn h on it:

$$\{(X_1,Y_1-F(X_1)), (X_2,Y_2-F(X_2)),\dots,(X_n,Y_n-F(X_n))\}$$

- Repeat this procedure

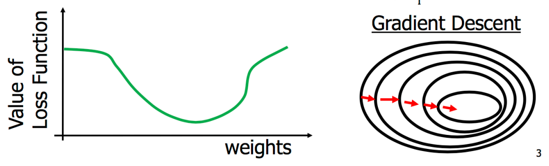
Additive Model: $F(X) = h_0(X) + h_1(X) + \dots + h_m(X)$



The function is just an additive function and the regression model is just a decision tree.
I start with the initial data set and I want to learn a model. hence, I evaluate all possible decision trees.
Idea: incrementally get closer to tree level.

Recall, gradient descent

- Given: Function l(**w**), with **w** = $[w_1,\ldots,w_n]$
- Goal: Set parameters **w** to minimize l(**w**)
- Approach:
  - Guess random initial value of **w**
  - Iteratively update $w_{i+1} \leftarrow w_i - \eta \frac{\partial l(w)}{\partial w_i}$



⇨ Connection to Gradient Descent

- So far, reweight with residual: $= Y_i - F(X_i)$
- By changing F, we minimize our loss function

$$J = \tfrac{1}{2} \textstyle\sum_i (Y_i - F(X_i))^2$$

- View $F(X_i)$s as parameters and take derivative

$$\frac{\partial J}{\partial F(X_i)} = \frac{\partial \tfrac{1}{2} \sum_i (Y_i - F(X_i))^2}{\partial F(X_i)} = F(X_i) - Y_i$$

- Residual is negative gradient: $Y_i - F(X_i) = -\frac{\partial J}{\partial F(X_i)}$

Chain rule: $F(f \circ g\,(x))$ then $F' = f'(g(x))\,g'(x)$

$$\frac{\partial J}{\partial F(X_i)} = \frac{\partial \tfrac{1}{2} \sum_i (Y_i - F(X_i))^2}{\partial F(X_i)}$$

$$= \frac{\partial \tfrac{1}{2}[(Y_1 - F(X_1))^2 + \ldots + (Y_i - F(X_i))^2 \ldots + (Y_n - F(X_n))^2]}{\partial F(X_i)}$$

$$= \frac{\partial \tfrac{1}{2}[(Y_i - F(X_i))^2]}{\partial F(X_i)}$$

No other term involves $F(X_i)$
These terms' derivatives are 0

$$= \frac{\partial \tfrac{1}{2}[(Y_i - F(X_i))^2]}{\partial F(X_i)} = F(X_i) - Y_i$$

By chain rule

Can view as:

$$= \frac{\partial \tfrac{1}{2}[(Y_i - F(X_i))^2]}{\partial F(X_i)}$$

View above as $F(f \circ g\,(x))$ with $g\,(x) = Y_i - F(X_i)$

By chain rule $F' = f'\,(g(x))g'(x)$

$f'(g(x)) = Y_i - F(X_i)$ and $g'(x) = -1$, thus

$$= \frac{\partial \tfrac{1}{2}[(Y_i - F(X_i))^2]}{\partial F(X_i)} = F(X_i) - Y_i$$

Why is Gradient Boosting so powerful? It is a generalization of AdaBoost. AdaBoost designs a particular function and tries to optimize classification.

Gradient Boosting tends to abstract away the algorithm from the loss function and hence the task.
Thus, it can plug in any differentiable loss function and use the same algorithm

- Other regression loss functions
- Classification
- Ranking

Several details were skipped

- Regularization to avoid overfitting
    - Restrict depth of trees by not considering full possible trees. I enumerate trees up to a certain depth.
    - Add penalty term to objective function J
- Setting the learning rate η
- Derivations and discuss of all loss functions

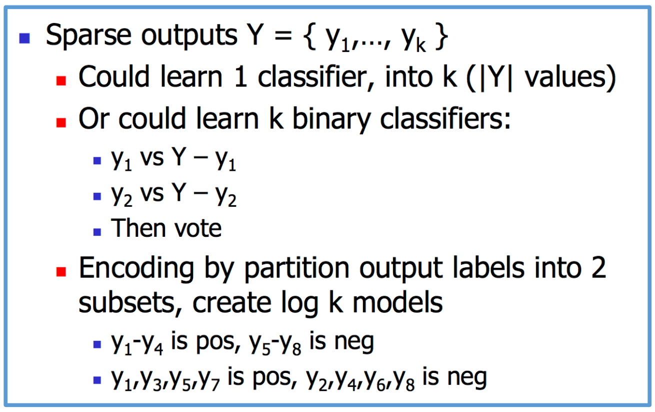
AdaBoost vs. Gradient Boosting

⇨ How are they similar and how are they different

- **Similarities**
    - Stage wise (1st model then second model, etc.) greedy learning of additive model.
    - Focus on mispredicted examples. The model makes some mistakes and both techniques want to focus on these mistakes.
- **Differences**
    - **Focus on mispredictions**
        - AdaBoost : High-weight data points
        - Gradient Boosting : Gradient of loss function. Focus on a little part of each example.
    - **Generality**
        - AdaBoost : Just classification.
        - Gradient Boosting : Any differentiable loss

3. Manipulate features

<u>Idea</u>: different learners see different subsets of features (of each training instances). Empirically it provided mixed results. The technique works best when input features are highly redundant.

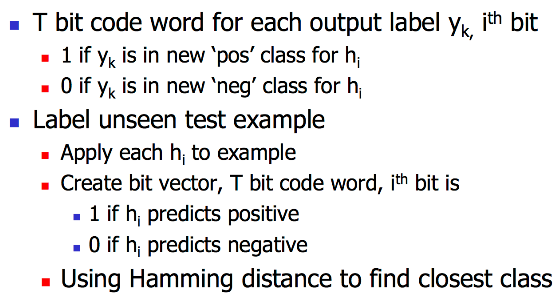


- Sparse outputs Y = { y_1,…, y_k }
  - Could learn 1 classifier, into k (|Y| values)
  - Or could learn k binary classifiers:
    - y_1 vs Y – y_1
    - y_2 vs Y – y_2
    - Then vote
  - Encoding by partition output labels into 2 subsets, create log k models
    - y_1-y_4 is pos, y_5-y_8 is neg
    - y_1,y_3,y_5,y_7 is pos, y_2,y_4,y_6,y_8 is neg

⇨ How to use this in a sampling context? The idea is to create more than log k models, to get some redundancy.
"Error-Correcting Codes" (some redundancy)

Given: Integer T
For I = 1 to T

- Partition labels into two disjoint sets
- Build classifier to distinguish between these sets of examples



- T bit code word for each output label $y_{k,}$ ith bit
  - 1 if y_k is in new 'pos' class for h_i
  - 0 if y_k is in new 'neg' class for h_i
- Label unseen test example
  - Apply each h_i to example
  - Create bit vector, T bit code word, ith bit is
    - 1 if h_i predicts positive
    - 0 if h_i predicts negative
  - Using Hamming distance to find closest class

4. Add randomness

Neural networks:

- Different initial values
- Not really independent

Decision trees:

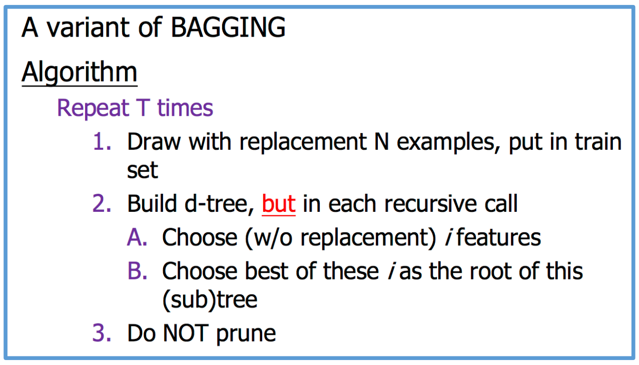
- Consider top 20 attributes choose one at random?
- Produce 200 classifiers
- To classify new instance: Vote

FOIL:

- Choose any test w/foil gain within 80% of top
- Good empirical performance

Random Forests

Random forests or random decision forests[1][2] are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.


A variant of BAGGING

Algorithm

Repeat T times

1. Draw with replacement N examples, put in train set
2. Build d-tree, but in each recursive call
   - A. Choose (w/o replacement) *i* features
   - B. Choose best of these *i* as the root of this (sub)tree
3. Do NOT prune

Increasing i

- Increases correlation among individual trees (**BAD**)
- Also increases accuracy of individual trees (**GOOD**)

Use tuning set to choose good setting for i

Overall, random forests

- Are very fast
- Deal with large # of features
- Reduce overfitting substantially
- Work very well in practice

Random forests differ in only one way from the general scheme of bagging: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.

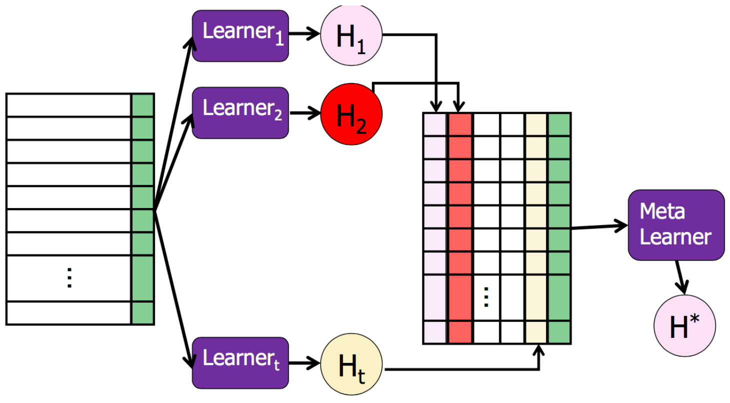
5. Stacking

Stacking is a similar to boosting: you also apply several models to your original data. The difference here is, however, that you don't have just an empirical formula for your weight function, rather you introduce a **meta-level** and use another model/approach to estimate the input together with outputs of every model to estimate the weights or, in other words, to determine what models perform well and what badly given these input data.

Idea: you want to learn whether each learner is good

- Given: Learners $L_1,\dots,L_t$
- Idea: Learn when each learner is good

- Let $h_j(-i) = L_j(S - x_i)$ be classifier learned using L_j, on all but instance x_i
- Let $y'_i(j) = h_j(x_i)$
- New train set: $\{ [[y'_i(1), y'_i(2),\dots, y'_i(t)], y_i] \}i$



6. Why do ensemble methods work?

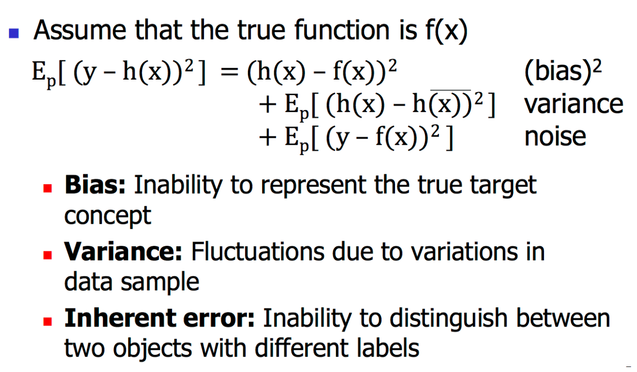
⇨ There are **four possible explanations** to why ensembles are better than basic classifiers.

6.1 Expected Error

- Given a new data point **x**, what is the **expected prediction error?**
- Assume data points are drawn i.i.d. from a unique underlying probability distribution P
- Goal: compute for an arbitrary new point **x**

$$E_p[(y - h(x))^2]$$

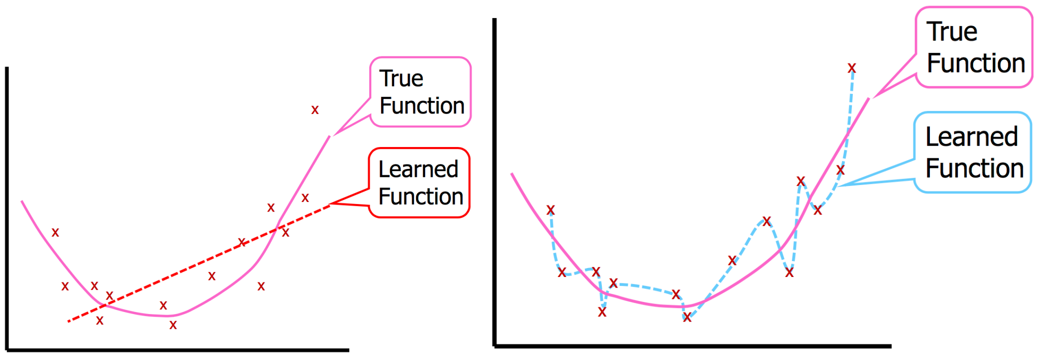
  - E_p is the expectation across all training sets
  - y is true label
  - h(x) is predicted label
  - Note: looking at squared error



- Assume that the true function is f(x)

$$
\begin{aligned}
E_p[\,(y - h(x))^2\,] &= (h(x) - f(x))^2 && \text{(bias)}^2 \\
&+ E_p[\,(h(x) - \overline{h(x)})^2\,] && \text{variance} \\
&+ E_p[\,(y - f(x))^2\,] && \text{noise}
\end{aligned}
$$

  - **Bias:** Inability to represent the true target concept
  - **Variance:** Fluctuations due to variations in data sample
  - **Inherent error:** Inability to distinguish between two objects with different labels



- Bias: left picture.
- Variance: right picture. Complicated hypothesis space.

Sources of Bias

- Bias comes from the inability to represent certain decision boundaries. E.g., linear threshold units, decision trees.
- You make incorrect assumptions
  - E.g., failure of independence assumption in naïve Bayes
- Classifiers that are "too global"
  - E.g., single linear separator, small decision tree
- If bias is high, the model is underfitting the data

Sources of variance

- Statistical sources. Classifiers are "too local" and can easily fit the data. (e.g., nearest neighbor, large decision trees)
- Computational sources
  - Decision made on small subsets of the data (e.g., decision tree splits near the leaves)
  - Randomization in the learning algorithm (e.g., neural nets with random initial weights)
  - Unstable learning algorithms (e.g., decision boundary can change if one training example changes)
  - High variance $\Rightarrow$ model is overfitting the data

6.2 Bias/variance explanation

Bias and variance are two **components of error**.

- **Bias**: inability to represent the true concept. I have some function I want to predict, based on hypothesis and model. I might have some error which is called “bias”. We can think about bias when we don’t have a good hypothesis space.
- **Variance**: fluctuations due to random variations in the data. Appears when the hypothesis class is too big.

⇨ Hence, we are faced with a **trade-off!**
- More expressive class of hypotheses, which generates higher variance
- Less expressive class, which will generate higher bias.

I have to make a decision beforehand based on whether I prefer high bias or high variance.

⇨ Ensembles can address both bias and variance!

- **Bagging**: if bootstrap approximation is correct, then it would reduce variance without changing bias. In practice, bagging can reduce both.
  - For high-bias classifiers, can reduce bias
  - For high-variance classifiers, can reduce variance
- **Boosting:** Attends to allow you to work with more expressive hypothesis.
  - Early iterations: primarily reduces bias
  - Later iterations: primarily reduces variances (apparently).

6.3 Statistical explanation

To motivate this explanation, we could look at an example of flipping coins.

Flip same coin twice and observe: Heads then Tails

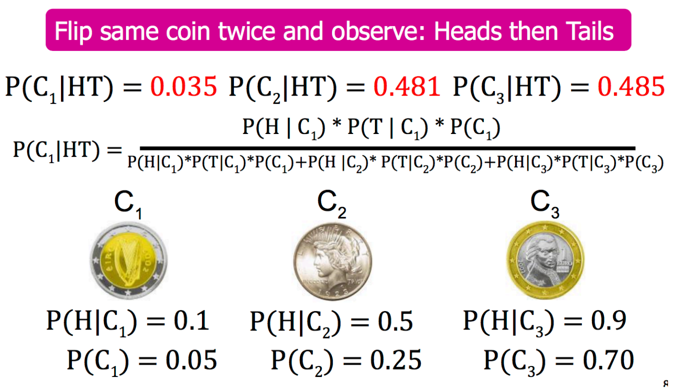
$P(C_1|HT) = ?$ $P(C_2|HT) = ?$ $P(C_3|HT) = ?$

$$P(C_1|HT) = \frac{P(H \mid C_1) * P(T \mid C_1) * P(C_1)}{P(H|C_1)*P(T|C_1)*P(C_1)+P(H \mid C_2)* P(T|C_2)*P(C_2)+P(H|C_3)*P(T|C_3)*P(C_3)}$$

C_1 C_2 C_3

$P(H|C_1) = 0.1$ $P(H|C_2) = 0.5$ $P(H|C_3) = 0.9$

$P(C_1) = 0.05$ $P(C_2) = 0.25$ $P(C_3) = 0.70$



Flip same coin twice and observe: Heads then Tails

$P(C_1|HT) = 0.035$ $P(C_2|HT) = 0.481$ $P(C_3|HT) = 0.485$

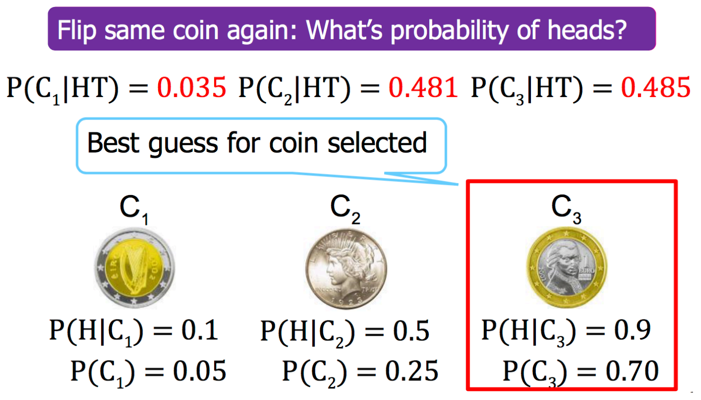
$$P(C_1|HT) = \frac{P(H \mid C_1) * P(T \mid C_1) * P(C_1)}{P(H|C_1)*P(T|C_1)*P(C_1)+P(H|C_2)*P(T|C_2)*P(C_2)+P(H|C_3)*P(T|C_3)*P(C_3)}$$

C_1 C_2 C_3

$P(H|C_1) = 0.1$ $P(H|C_2) = 0.5$ $P(H|C_3) = 0.9$

$P(C_1) = 0.05$ $P(C_2) = 0.25$ $P(C_3) = 0.70$

⇨ I can then compute the probability that I use one of the coins, based on this figures. I see that there is a small change that I flipped C1, but there is a bigger chance that I swap C2 and C3.



Flip same coin again: What's probability of heads?

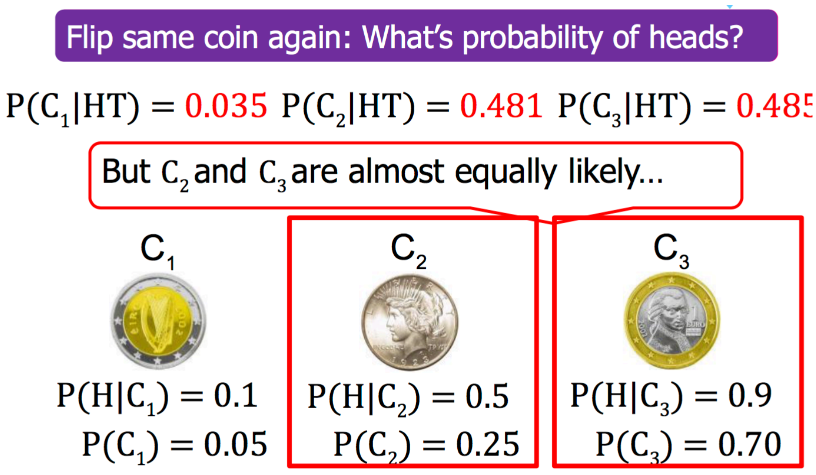
$P(C_1|HT) = 0.035$ $P(C_2|HT) = 0.481$ $P(C_3|HT) = 0.485$

Best guess for coin selected

C_1 C_2 $\boxed{C_3}$

$P(H|C_1) = 0.1$ $P(H|C_2) = 0.5$ $P(H|C_3) = 0.9$

$P(C_1) = 0.05$ $P(C_2) = 0.25$ $P(C_3) = 0.70$



Flip same coin again: What's probability of heads?

$P(C_1|HT) = 0.035$ $P(C_2|HT) = 0.481$ $P(C_3|HT) = 0.48$

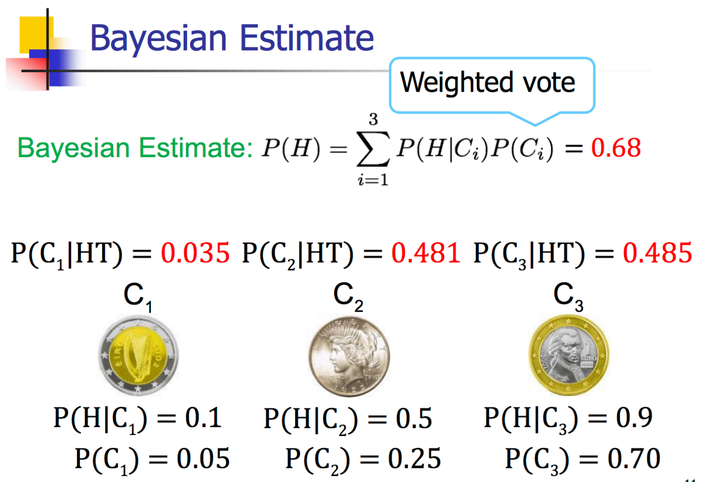
But C_2 and C_3 are almost equally likely…

C_1 C_2 C_3

$P(H|C_1) = 0.1$ $P(H|C_2) = 0.5$ $P(H|C_3) = 0.9$

$P(C_1) = 0.05$ $P(C_2) = 0.25$ $P(C_3) = 0.70$

Bayesian estimate could be a better idea. We make the probability of seeing a head a weighted vote among all possibilities. We then get a weighted vote about the predictions.



Bayesian Estimate

Weighted vote

Bayesian Estimate: $P(H) = \sum_{i=1}^{3} P(H|C_i)P(C_i) = 0.68$

$P(C_1|HT) = 0.035$ $P(C_2|HT) = 0.481$ $P(C_3|HT) = 0.485$

C_1	C_2	C_3
$P(H\|C_1) = 0.1$	$P(H\|C_2) = 0.5$	$P(H\|C_3) = 0.9$
$P(C_1) = 0.05$	$P(C_2) = 0.25$	$P(C_3) = 0.70$

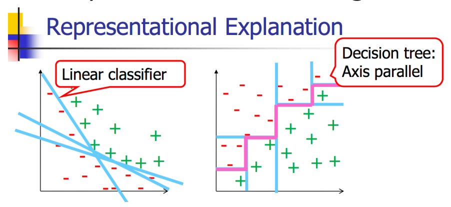
⇨ How can the learning algorithm select among the set of (almost) equally good hypotheses? How can I pick the best hypothesis? I can for example thing about the error rate.

From theory we know it is called the Bayes optimal classifier, making a weighted majority vote among all possible hypothesis. That is the best way we can do: weighted by their posterior probability, probably the best possible classifier, that is, it minimizes the error rate. But it is actually infeasible.

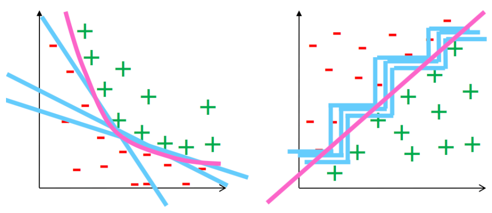
⇨ Ensemble learning approximates Bayes optimal

6.4 Representational explanation

Each leaner works with a given hypotheses class (i.e., set of possible models). It is possible that the target function lies outside of the considered hypothesis class.



- Linear classifier: I will have a y function that separates positive and negatives examples. I won't have a single separation line.
- No straight line, a decision tree can represent this diagonal line.



An ensemble averaging may approximate the true target function with arbitrarily good accuracy.

- Curved boundary by averaging lines
- Diagonal boundary by averaging "staircases".

6.5 Computational explanation

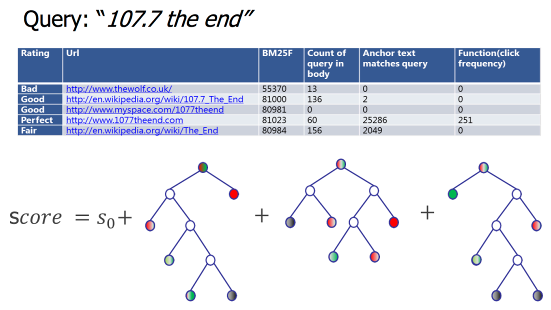
Most learning algorithms search through hypotheses space to find one "good" model. Hypothesis space could be a space of decision trees, regression lines, etc.

The most interesting hypothesis space are huger or infinite, so we have to do a heuristic search.

The learning might get stuck in a **local minimum**. One strategy for avoiding local minima is to **repeat** the search many times with random restarts. This is essentially what bagging does. This method is really powerful.

7. Some applications

The best one or at least the most successful one is in the **web search.**



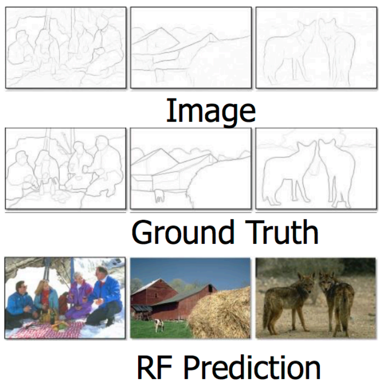
Query: "*107.7 the end*"

Rating	Url	BM25F	Count of query in body	Anchor text matches query	Function(click frequency)
Bad	http://www.thewolf.co.uk/	55370	13	0	0
Good	http://en.wikipedia.org/wiki/107.7_The_End	81000	136	2	0
Good	http://www.myspace.com/1077theend	80981	0	0	0
Perfect	http://www.1077theend.com	81023	60	25286	251
Fair	http://en.wikipedia.org/wiki/The_End	80984	156	2049	0

$$Score = s_0 + \ldots + \ldots + \ldots$$

Uses ensembles of decision trees. They use the gradient boosting. We can think about the data, we have a query, here, a radio station. And we then have different Url's (= training data). We also have a ranking among the lines.
They will then induce an ensemble of trees.
A score is used to rank the Url's. I want to make sure I pick the perfect one first, then a good one, etc. They want to predict a score and learn about decision trees to perform the ranking.
One thing that is important is to evaluate the trees. Another reason is that this method tends to be very accurate.

Another area where ensembles are widely used is in **edge detection**. Given an image, you want to detect edges, finding the outline of the different figures. What you could do then is to use an ensemble, for instance random forest.
We will have some probabilistic models, train decision trees, etc.



A third area refers to **motion capture**. I want to model a person. The goal standard is to have a good camera tracking system. You then have a random map and you try to get location of the people. This is widely used in sports domain to look at people's behaviour on the field.
This is method is really accurate. However, the lab setup is incredibly expensive and very time consuming.

Video games that tries to recognize your gesture. You gen try to infer what the poses of people are. You get a depth image and you then go to a body part before moving on to a 3D joint proposal. Decision trees are used to move on through the procedure.

To sum up, a committee of experts is typically more effective than a single supergenius.

Key issues

- Generating base models
- Integrating responses from base models: I have to combine or aggregate the predictions.

Popular ensemble techniques

- Manipulate training data: bagging and boosting
- Manipulate output values: error-correcting output coding

⇨ Bias/variance is really important in data mining.

INTERLUDE: ACTIONABLE DATA MINING

- How many of you go to the same grocery store every week?
- How many of you go to the same bar?
- How many of drink one type of beer?
- What would make you change?
- Why do we work this way?

What we want to take away is that people typically have similar habits. Similar habits make some things easier to analyse, entering a certain routine.
Once you have certain habits, it becomes hard to change.

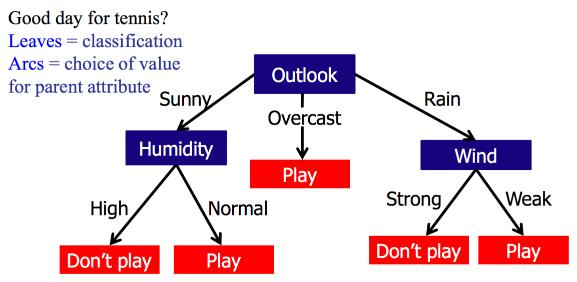
Task: based on customer information
Do: You want to predict whether the customer is pregnant or not.

Many **questions** arise then

- **What data is needed? Where would you get it?** We need data about gender, age. Purchase data. We also need data about search engines, looking at which products people are looking for. I should try to figure out what pregnant people are looking for.
- **What techniques are applicable to this problem?** This is a binary classification problem, where you may want to assign a score to pregnant people.
- **What would you look for?**
- **How could you exploit your findings?** The main goal of data mining is to act upon the data. So if I know if someone is pregnant, a want to use this information to act in a certain manner. I may be tended to recommend some products for instance, proposing some adds, etc.
- **Are there risks involved? If so, what are they?** There are some risks related to privacy. Then questions arise: is this rational or not?
  There is also a risk of taking false positive. Take for instance a person that makes searches for his or her friends but who might not be pregnant at all.

INTERLUDE: LARGE SCALE DECISION TREE LEARNING

1. Decision tree overview



- **Internal nodes** are choice nodes.
- **Leave nodes** are prediction nodes. They represent the classification.

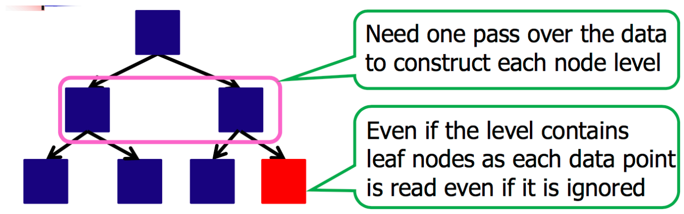
```
BuildTree(TraingData)
    Split(TrainingData)

Split(D)
    If (all points in D are of the same class)
        Then Return
    For each attribute A
        Evaluate splits on attribute A
        Use best split to partition D into D1, D2
        Split(D1)
        Split(D2)
```

I have a bunch of training data. I then split, picking the best internal node. You first want to check if all points are of the same class. Otherwise you have to run through all attributes and evaluate the splits on each of them. You then find the best one and partition the data you have.

⇨ When will this basic decision tree algorithm be inefficient? Think about very large data sets.
Answer: if my decision tree does not fit in the memory. Every time I want to do a split, I have to do a full scan of the data. So, I might keep a list of instances I want to look at. There are many examples I can ignore.

2. Rainforest


Need one pass over the data to construct each node level
Even if the level contains leaf nodes as each data point is read even if it is ignored

⇨ Can we improve? Yes! We just need aggregate information at each node in order to compute the split point. We can just store the aggregate information.

The data structure is called **Attribute Value Class** Label set or AVC set. Counts for each class are aggregated and it stores the class label distribution for each attribute value.

The **AVC group** is an AVC set for all possible attributes.

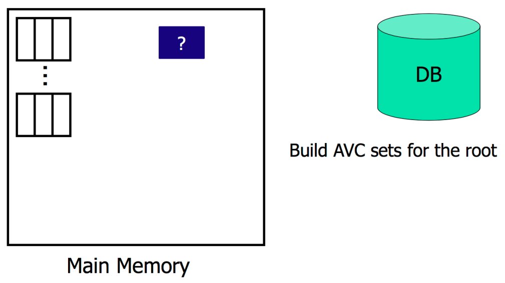
RainForest Algorithms

- **RF-Write:** Always write out partitions to disk

Pro: Do not have to read whole DB each time
Con: Writing to disk is expensive

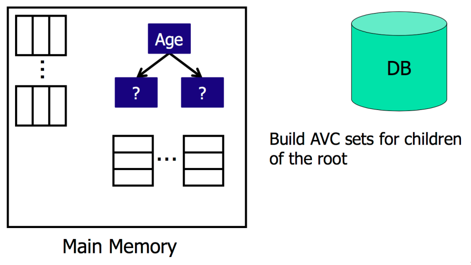
- **RF-Read**: Scan the whole database at each node level
  Pro: Avoid expensive write operations
  Con: Read lots of irrelevant information
- **RF-Hybrid:** Mixed strategy
  Use RF-Read until AVC-Groups of child nodes don't fit in memory. For each level where AVC-Groups don't fit in memory, partition child nodes into sets M & N
  - AVC-Groups for n ∈ M all fit in memory
  - AVC-Groups for n ∈ N are build on disk.

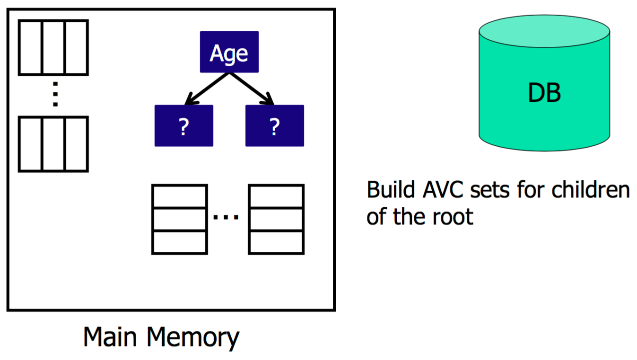
  Process nodes in memory then fill memory from disk



I have a database and I scan the data. I build the AVC set for each of my possible nodes. I pick my root node.

I now get two nodes. For each node I have to consider the possible splits. I consider each attribute. So, I doubled the number of attributes I have to consider.





3. BOAT

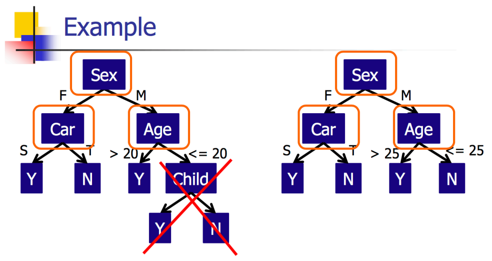
Another way to think about scaling up is the BOAT algorithm. This stands for Bootsrapped Optimist Algorithm for Tree Construction. This is very scalable to learning decision trees. Indeed, it needs very few scans (perhaps 2) over the data. And it constructs multiple levels at once.

The algorithm runs in two phases

- **Sampling phases**: it is gonna look at as much as the data set we can. Take a sample D′⊂ D from the training database such that D’ fits in memory. Construct b bootstrap trees T1,...,Tb by creating training samples D1,...,Db obtained by sampling with replacement from D’. **Perform bagging!** I then end up with a large number of trees. It will then try to combine them into a unified, single tree. So, it will try to create a summary tree.
- **Create Coarse Splitting Criteria**:
  Process the trees in a top-down fashion. At each node N, check if the splitting criteria is identical for all trees. If not, delete N and its subtrees in all trees. If they agree construct coarse splitting criteria

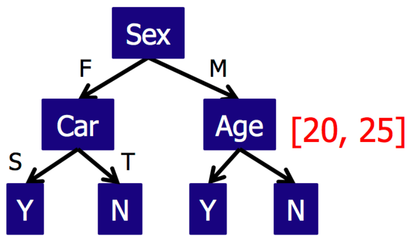
⇨ **Coarse splitting criteria**

- **Discrete**: Just the attribute-values
- **Continuous**: Confidence interval [L, U] such that the true split is likely in the interval. We have a range of possible values.



Both trees agree on the root node. On the left subtree they all agree. On the right subtree they agree. But then they disagree.

On the left tree we have >20, on the other side >25. So for age we don’t know where the actual threshold will be.
For the final tree, we compute an exact split.



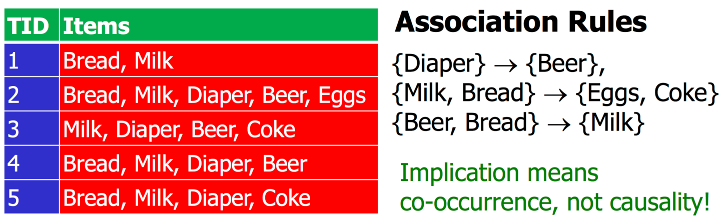
- **Cleaning phase**
  For discrete attribute the coarse splitting criteria is the exact splitting attribute, whereas, for continuous data, only try splits within confidence interval.
  Collect all examples that follow this path AND fall within the confidence interval. Then compute exact split based on these examples

CHAPTER 6: ASSOCIATION RULE MINING

⇨ How to find patterns?

1. Introduction and definitions

Given: Set of transactions
Find: If-Then rules that predict the occurrence of an item based on other items in the transaction.



TID	Items
1	Bread, Milk
2	Bread, Milk, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Eggs, Coke}
{Beer, Bread} → {Milk}

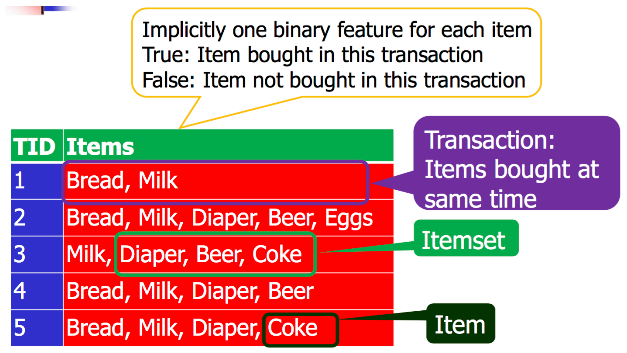
Implication means co-occurrence, not causality!

The main motivation is **finding regularities** in data. What products were often purchased together? What kinds of DNA are sensitive to new drug? Try to find co-occurrences. We try to find patterns in the data, patterns that predict a certain variable.

Foundation for many data mining tasks:

- Association
- Correlation

⇨ Algorithms do not require labelled data or for a user to specify a predefined target concept.


Implicitly one binary feature for each item
True: Item bought in this transaction
False: Item not bought in this transaction
Transaction: Items bought at same time
Itemset
Item

TID	Items
1	Bread, Milk
2	Bread, Milk, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

- An **itemset** is a collection of one or more items. Ex: {Bread, Milk}
- A **k-itemset** is an itemset that contains k items. Ex: 3-itemset {Bread, Milk, Diaper}

Association rules are if-then rules about the contents of baskets
Given: Set of items: I = {i1, i2, ..., im}
Set of transactions: D = {d1, d2, ..., dn}
An association rule: A → B

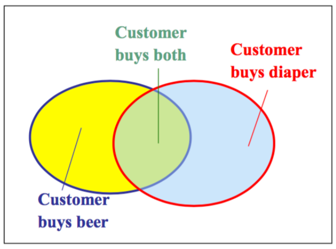
- $A \subset I$
- $B \subset I$
- $A \cap B = \varnothing$

{i1, i2,...,ik} → j means: "if a basket contains all of i1,...,ik then it is likely to contain j."

When dealing with association rules, a simple question is: find sets of items that appear "frequently" in the baskets.

The **support count** for itemset I is the number of baskets containing all items in I. the support is the fraction of transactions that contain an itemset. Basically we are just counting how many times we see an itemset in the data.
Given a support threshold s, frequent itemsets appear in at least s baskets.

$$A \Rightarrow C = \text{support}(\{A\} \cup \{C\}) / |T|$$



The **confidence** of an association rule is the conditional probability of j given i1, ..., i_k. This gives a measure of how accurate the rule is. Given I have observed A, what is the probability I also observe B?
Confidence ($A \Rightarrow B$) = $P(B|A)$ = sup({A,B}) /sup(A)
The confidence of the rule is the fraction of the baskets with all of I that also contain j.
Confidence alone can be useful, provided the support for the left side of the rule is fairly large. We usually want the confidence of the rule to be reasonable high, perhaps 50%, or else the rule has little practical effect.

Given an association rule i→ j, **Interest** = Confidence (j | I) - Support(j).

- Interest = 0 : I has no influence on J
- Interest > 0 : I may cause the presence of j
- Interest < 0 : I discourages the presence of j

If I has no influence on j, then we would expect that the fraction of baskets including I that contain j would be exactly the same as the fraction of all baskets that contain j. such a rule has interest 0.

In data mining and association rule learning, **lift** is a measure of the performance of a targeting model at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model. A targeting model is doing a good job if the response within the target is much better than the average for the population as a whole. Lift is simply the ratio of these values: target response divided by average response.
If some rule had a lift of 1, it would imply that the probability of occurrence of the antecedent and that of the consequent are independent of each other. When two events are independent of each other, no rule can be drawn involving those two events.

If the lift is > 1, it lets us know the degree to which those two occurrences are dependent on one another, and makes those rules potentially useful for predicting the consequent in future data sets.

Why could the lift measure be useful? By dividing confidence by support, we can "normalize" data and compare the results.

The **association rule mining task** is simple.
Given: Transaction data, support s, and confidence c.
Find: All association rules with support ≥ s and confidence ≥ c

Task focus:

- General many-many mapping (association) between items and baskets
- Connection among "items" not "baskets"
- Focus on common event not on rare events

There exist different **types of associations**

- **Boolean** associations

  $\boxed{\text{Bread} \wedge \text{Milk} \rightarrow \text{Diapers}}$

  Items are either purchased or not

- **Quantitative** associations

  $\boxed{\text{age in 30..39}} \wedge \text{income in 42..48K} \rightarrow \text{buys PC}$

  Look at a range of value

Number of predicates captured

- Single attribute

  $\boxed{\text{Bread} \wedge \text{Milk} \rightarrow \text{Diapers}}$ Just purchases

- Multiple attributes

  $\boxed{\text{age}}\text{ in 30..39} \wedge \boxed{\text{income}}\text{ in 42..48K} \rightarrow \text{buys PC}$

  $\neq$

- Multi-relational

  $\text{buys(x, PC)} \wedge \boxed{\text{friends(x,y)}} \rightarrow \text{buys(y, PC)}$

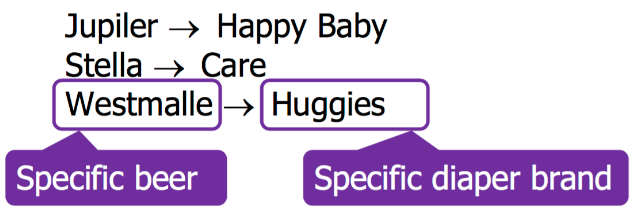
  Look at relationship between individuals

Single or Multiple Level

- Single Level

  $\boxed{\text{Beer} \rightarrow \text{Diapers}}$ Generic item types

- Multiple Level



Association rules are often used in the **retail sector**.

- **Baskets**: sets of products someone bought in one trip to the store
- **Items**: products
  Ex: given that many people buy beer and diapers together, run a sale on diapers; raise price of beer. Only useful if many buy diapers and beer. What items should store stock up on.

By finding frequent itemsets, a retailer can learn what is commonly bought together. Especially important are pairs or larger sets of items that occur much more frequently than would be expected were the items bought independently.
Ex: diapers and beer. One would hardly expect these two items to be related. Through data analysis one chain store discovered that people who buy diapers are unusually likely to buy beer.

Another application is **plagiarism**

- Baskets = sentences
- Items = documents contain those sentences
  Items that appear together too often could represent plagiarism. Notice that items do not have to be "in" baskets. Documents having a high count contain plagiarism.

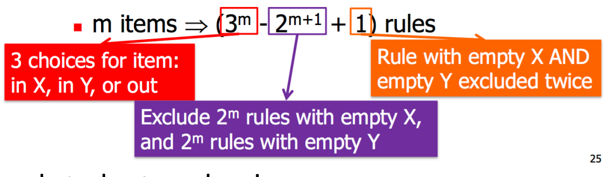
Association rules are also often used in **web pages** analytics.

- Baskets = web pages
- Items = words
  Unusual words appearing together in a large number of documents, e.g., "Brad" and "Angelina", may indicate an interesting relationship.

2. Naïve Algorithm

Walmart sells 100 000 items and can store billions of baskets. The Web has billions of words and many billions of pages. We have access to lots and lots of data …

Naïve Generate and Test
Goal: Find all association rules with support ≥ s and confidence ≥ c.
For each association rule X => Y, keep it if it meets support & confidence threshold.
>< Complexity: exponential in number of items.


m items ⇒ (3^m - 2^{m+1} + 1) rules
3 choices for item: in X, in Y, or out
Exclude 2^m rules with empty X, and 2^m rules with empty Y
Rule with empty X AND empty Y excluded twice
25

This Naïve algorithm tends to be too slow!

Association Rule Mining Goal

Insight: Given frequent itemsets,

- Trivial to derive all association rules
- These will contain all rules with support s and confidence c

Hard part: finding the frequent itemsets.
Note: if {i1, i2,...,ik} → j has high support
and confidence, then both {i1, i2,...,ik} and {i1, i2,...,ik ,j } will be "frequent"

Creating association rules
Given: Support s, confidence c

- Step 1 : Find all itemsets with support s
- Step 2 : For each frequent itemset L, for each non-empty subset s of L. Output the rule s → {l-s} if its confidence ≥ c
  Once I have frequent itemsets, finding association rules is quite easy.

Typically, data is kept in flat files rather than in a database system. Data is stored on disk basket-by-basket. Use k nested to expand baskets into pairs, triples, etc. as you read basket.

The true cost of mining disk-resident data is usually the number of disk I/O's.

- Read data in passes: all baskets read in turn
- Cost: Number of passes an algorithm takes

The bottleneck is main memory. For many frequent-itemset algorithms, **main memory** is the critical resource. As we read baskets, we need to count something, e.g., occurrences of pairs. The number of different things we can count is limited by main memory.
Swapping counts in/out is a disaster (why?).

The hardest problem often turns out to be finding the frequent pairs, often frequent pairs are common, frequent triples are rare. The probability of being frequent drops exponentially with size. The number of sets grows more slowly with size.
First, focus on pairs, then extend to largest sets.

Naïve Algorithm

What is the naïve way to think about this? For each basket that I process, I generate all possible pairs and I process them.
Read file once, counting in main memory the occurrences of each pair. From each basket of n items, generate its n(n-1)/2 pairs by two nested loops.
Fails if $(\#items)^2$ exceeds main memory
Remember: #items can be 100K (Wal-Mart) or 10B (Web pages)

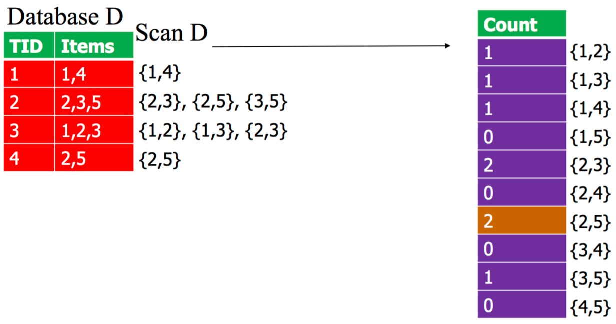
2 approaches

1) **Count all pairs, using a triangular matrix.** Observe all possible pairs in there. I can figure out the identifiers of the items.
    - Assign each item a number
    - Count {i,j} only if i < j
    - Keep pairs in the order: {1,2} … {1,n } : {2,3} ... {n -1,n }
    - Pair{i,j} at the position: (i–1)(n–i/2)+j–i

⇨ 4 bytes/pair store all pairs

Assign each transaction a number and keep a count for each itemset.

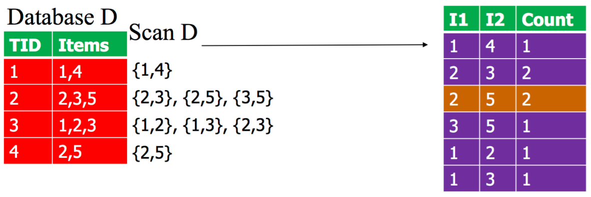
We have a simple transactional database containing four transactions. Then I have my array where I generated all possible itemsets. I have to keep one entry for each possible pair. I want to scan my data. For each transaction I generate all possible pairs.



TID	Items	
1	1,4	{1,4}
2	2,3,5	{2,3}, {2,5}, {3,5}
3	1,2,3	{1,2}, {1,3}, {2,3}
4	2,5	{2,5}

Count	
1	{1,2}
1	{1,3}
1	{1,4}
0	{1,5}
2	{2,3}
0	{2,4}
2	{2,5}
0	{3,4}
1	{3,5}
0	{4,5}

2) **Table of triples [i, j, c] = pair {i, j} count is c.** This method is more memory efficient than previous approach.
The total number of bytes used is about 12p, where p is the number of pairs that actually occur. This method requires us to store three integers, rather than one, or every pair that does appear in some basket. This approach beats the triangular matrix if at most 1/3 of the possible pairs actually occur. It requires extra space for retrieval of structure. The triples method does not require us to store anything if the count for a pair is zero.



TID	Items	
1	1,4	{1,4}
2	2,3,5	{2,3}, {2,5}, {3,5}
3	1,2,3	{1,2}, {1,3}, {2,3}
4	2,5	{2,5}

I1	I2	Count
1	4	1
2	3	2
2	5	2
3	5	1
1	2	1
1	3	1

So far, we focused on identifying frequent pairs. But, there exist many more possible frequent triples than frequent pairs. How can we avoid generating all of them?
Number of frequent itemsets of size two will be bigger than frequent itemset of size three. But on the other hand, there are more possible frequent triples than possible doubles.

3. Apriori

In practice, the most main memory is required for determining the frequent pairs. The number of items, while possible very large, is rarely so large we cannot count all the singleton sets in main memory at the same time.
In practice the support threshold is set high enough that it is only a rare set that is frequent. Monotonicity tells us that if there is a frequent triple, then there are three frequent pairs contained within it. Thus, we expect to find more frequent pairs than frequent triples, more frequent triples than frequent quadruples, and so on.

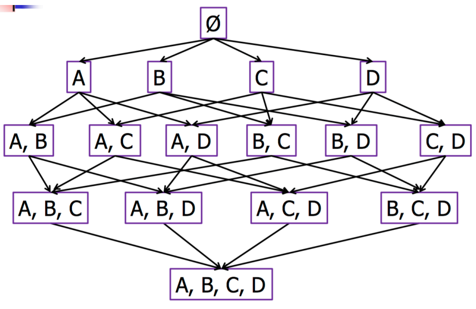
The **A-Priori algorithm** is designed to reduce the number of pairs that must be counted, at the expense of performing two passes over data, rather than one. This is the job of the A-Priori Algorithm and related algorithms to avoid counting many triples or larger sets.

The main purpose here is to generate and test approach for discovering frequent itemsets. The apriori algorithm is an iterative approach.

- Find all frequent itemsets of size k before finding frequent itemsets of size k+1
- One pass through the data for each frequent itemset size.

If an itemset appears at least s times, so do all its subsets. The **contrapositive** for pairs states that if item I does not appear in s baskets, then no pair including I can appear in s baskets. It will never be frequent. As soon as I know that something is infrequent, there is no point of counting an itemset that is bigger.

The Apriori principle holds due to the following property of the support measure: $\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$



- **Pass 1**: Create two tables. The first table translates item names into integers from 1 to n. The other table is an array of counts. Read baskets and count in main memory the occurrences of each item. It requires memory proportional to the number of items. Frequent items are those that appear s times.

  After the first pass we examine the counts in order to determine the frequent itemsets. We then create a new number from 1 to m for just the frequent items. This table is an array indexed 1 to n, and the entry for I is either 0, if item I is not frequent, or a unique integer in the range 1 to m if item I is frequent.

- **Pass 2**: Read baskets again and count in main memory only those pairs both of which were found in Pass 1 to be frequent. In a double loop, generate all pairs of frequent items in that basket. For each such pair, add one to its count in the data structure used to store counts.
  It requires memory proportional to the square of frequent items, plus a list of the frequent items. Frequent itemsets are those that appear s times.

```
Apriori(Data D, Support S, Candidates C)
while (|C| > 0) do
    count = [0,….0]; $L_k = \emptyset$
    for each transaction t ∈ D do
        for each Candidate c ∈ C do
            if ($c \subset t$) then count[c]++
    for each Candidate c ∈ C do
        if count[c] ≥ s then $L_k \cup \{c\}$
    C = Join($L_k$, $L_k$)
    C = Prune(C)
```

Verify if each candidate satisfies the threshold.

⇨ If no frequent itemsets of a certain size are found, then monotonicity tells us there can be no larger frequent itemsets, so we can stop.

1) **Join Step**: Candidate set C_{k+1} is generated by joining L_k with itself Suppose the items in L_k are listed in an order. Join each element in L_k with itself. If l1, l2 ∈ Lk, they are joinable if:
   - The first k-1 items in l1 and l2 are the same
   - l1[1] = l2[1] AND l1[2] = l2[2] AND … AND l1[k-1] = l2[k-1]

   L_3={abc, abd, acd, ace, bcd, bce}

   Self-joining: *L_3*L_3*
   - *abcd* from *abc* and *abd*
   - *acde* from *acd* and *ace*
   - *bcde* from *bcd* and *bce*
   - Note: other joins are illegal:
     abc and acd: Differ on second item
     abc and ace: Differ on second item
     etc.
2) **Prune Step**: Any k-itemset that is not frequent cannot be a subset of a frequent (k+1)-itemset
   For each candidate itemsets C_{k+1}, look at each subset of size k, i.e., drop one item from the candidate.
   If any of these subsets isn't frequent, discard this candidate.
   → Application of the Apriori Principle

L_3={*abc, abd, acd, ace, bcd, bce*}

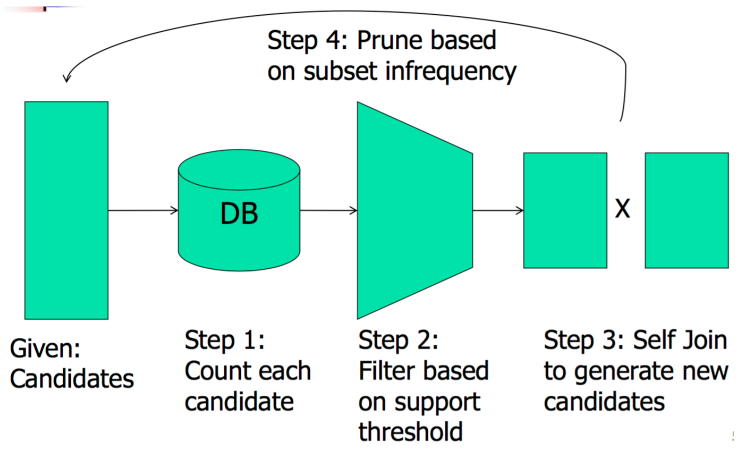
$C_{initial}$={*abcd*, ~~*acde, bcde*~~}

$\downarrow$ *acde* is removed because *ade* is not in L_3

bcde is removed because *bde* is not in L_3

C_4={*abcd*}

The real trick is the pruning step. This is the key step.


Step 4: Prune based on subset infrequency
DB
X
Given: Candidates
Step 1: Count each candidate
Step 2: Filter based on support threshold
Step 3: Self Join to generate new candidates

The size of the file of baskets is sufficiently large, hence it does not fit in main memory. Thus, a major cost of any algorithm is the time it takes to read the baskets from disk.

Besides cost, another data-related issue may be related to the use of the main memory for itemset counting. All frequent-itemset algorithms require us to maintain many different counts as we make a pass through the data. We might for instance need to count the number of times that each pair of items occurs in baskets. We must be careful as we cannot count anything that does not fit in main memory. Thus, each algorithm has a limit on how many items it can deal with.

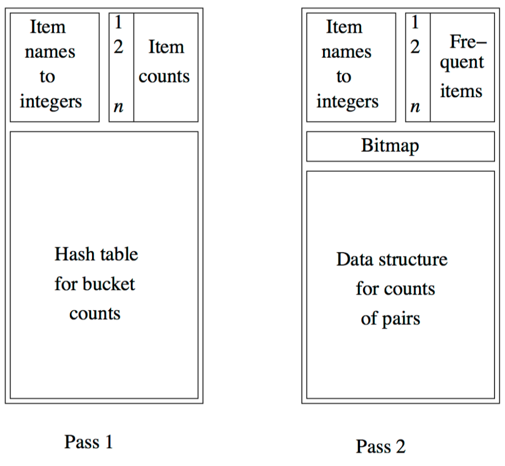
4. PCY

In this section we consider the PCY algorithm, which takes advantage of the fact that in the first pass of A-Priori there is typically lots of main memory not needed for the counting of single items.

Simple problem: check if an object has been previously encountered.

Filter strings: Scan a larger file F of strings and output those that are in S (e.g., emails) Has someone visited the set yet (a bit more complicated …).

⇨ Often, files will not fit in memory. This algorithm exploits the observation that there may be much unused space in main memory on the first pass. The PCY Algorithm uses that space for an array of integers that generalizes the idea of a Bloom filter.

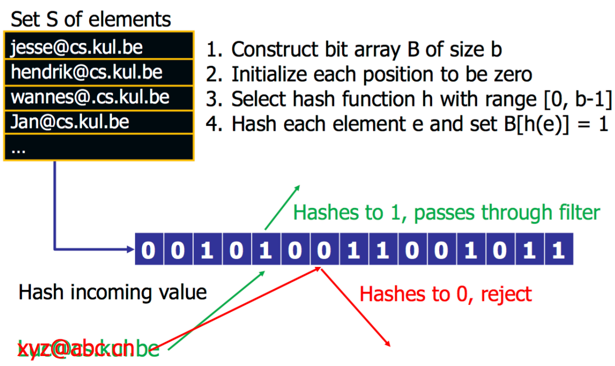


Think of an array as a hash table, whose buckets hold integers rather than sets of keys or bits. Pairs of items are hashed to buckets of this hash table. As we examine a basket during the first pass, we not only add 1 to the count for each item in the basket, but we generate all the pairs, using a double loop.
Hash each pair and add 1 to the bucket into which that pair hashes.

At the end of the first pass, each bucket has a count, which is the sum of the counts of all the pairs that hash to that bucket. If the count of a bucket is at least as great as the support threshold s, it is called a frequent bucket.

- We can say nothing about the pairs that hash to a frequent bucket; they could all be frequent pairs from the information available to us.
- If the count is less than s, we know no pair that hashes to this bucket can be frequent.

Next define the set of candidate pairs for the next pass.

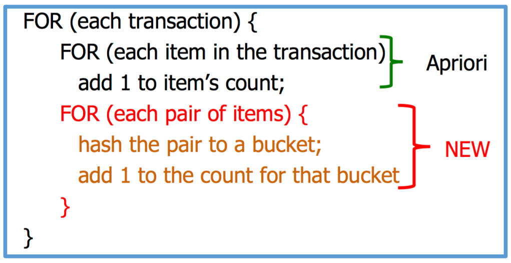

Set S of elements
jesse@cs.kul.be
hendrik@cs.kul.be
wannes@.cs.kul.be
Jan@cs.kul.be
...
1. Construct bit array B of size b
2. Initialize each position to be zero
3. Select hash function h with range [0, b-1]
4. Hash each element e and set B[h(e)] = 1
Hashes to 1, passes through filter
0 0 1 0 1 0 0 1 1 0 0 1 0 1 1
Hash incoming value
Hashes to 0, reject

Analysis

- No false negatives: catches all elements in F
- False positives are possible
- If b=n|S| at most 1/n of the bit array is 1, only 1/nth of elements not in S get through filter
  - Note will be less than 1/N given collisions
  - Assuming hashing uniformly at random

During the first pass of the Apriori algorithm, most memory is idle.
<u>Idea</u>: use free memory for a hash table

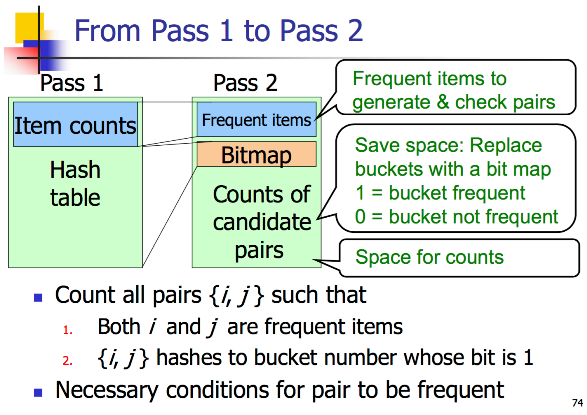
- Hash pairs of items that appear in a transaction: we need to generate these
- Just the count, not the pairs themselves
- Interested in the presence of a pair AND whether it is present at least s (support) times.



```
FOR (each transaction) {
    FOR (each item in the transaction)    } Apriori
      add 1 to item's count;
    FOR (each pair of items) {            } NEW
      hash the pair to a bucket;
      add 1 to the count for that bucket
    }
}
```

Observation about buckets

- A bucket that a frequent pair hashes to meet minimum support threshold. Cannot eliminate any member of this bucket
- Even without any frequent pair, a bucket can be frequent. Cannot eliminate any member of this bucket
- Best case: count for a bucket is less than minimum support. Eliminate all pairs hashed to this bucket even if the pair consists of two frequent items.



From Pass 1 to Pass 2

- Count all pairs {*i*, *j*} such that
  1. Both *i* and *j* are frequent items
  2. {*i*, *j*} hashes to bucket number whose bit is 1
- Necessary conditions for pair to be frequent

⇨ An example is provided in the slides.

5. Limiting disk I/O

A-Priori, PCY, etc., take k passes to find frequent itemsets of size k. in other words, previously discussed algorithms use one pass for each size of itemset we investigate. If the main memory is too small to hold the data and the space needed to count frequent itemsets of one size, there does not seem to be any way to avoid k passes to compute the exact collection of frequent itemsets.

- Good: clever pruning strategy
- Bad: still requires lots of pass over data ☹

Many applications where it is not essential to discover every frequent itemset. Hence, it is quite sufficient to find most but not all of them.
Making one pass over the data is what the eliminating factor is.
Other techniques use 2 or fewer passes for all sizes:

- Simple algorithm
- SON (Savasere, Omiecinksi, and Navathe)
- Toivonen

⇨ The question is, can we find all frequent itemsets in 2 passes or less?

5.1 Simple Algorithm: Sample

Idea: instead of using the entire file of baskets, we could pick a random subset of the baskets and pretend it is the entire dataset.

Procedure: Set of the data, I have a database, then memory. We need to have two parts of the memory. We will randomly sample baskets and make sure these end up in the memory. We then have some data in memory, we then run Apriori on the sample. So we need to think about the support threshold, that scales to the size of the database. That is to say, scale back support count, e.g., use s/100 if sample is 1/100 of original DB size.

The safest way to pick the sample is to read the entire dataset, and for each basket, select that basket for the sample with some fixed probability p. In case our baskets appear in random order in the file already, then we do not even have to read the entire file.

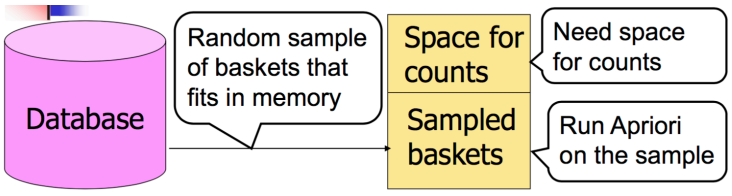
Once we have selected our sample of the baskets, we use part of main memory to store these baskets. The balance of the main memory is used to execute one of the algorithms we have discussed, such as A-Priori, PCY, … the algorithm must run passes over the main-memory sample for each itemset size, until we find a size with no frequent items.
Misses sets frequent in whole but not in sample

- Can use even smaller support
- E.g., s/125 instead of s/100

Optional: verify guesses on entire data set.

This algorithm requires at most two passes: one pass to construct the sample and then one sample if you want to verify guesses.

This can be very fast; It is also very simple. Of course the algorithm will fail if whichever method seen cannot be run in the amount of main memory left after storing the sample.


Database
Random sample of baskets that fits in memory
Space for counts
Sampled baskets
Need space for counts
Run Apriori on the sample

⇨ Be careful, it cannot be relied upon either to produce all the itemsets that are frequent in the whole dataset, nor will it produce only itemsets that are frequent in the hole; an itemset that is frequent in the whole but not in the sample is a false negative, while an itemset that is frequent tin the sample but not the whole is a false positive.

⇨ Retain as frequent only those itemsets that were frequent in the sample and also frequent in the whole. This improvement will eliminate all false positives, but a false negative is not counted and therefore remains undiscovered.

5.2 The SON Algorithm

⇨ Savasere, Omiecinski, and Navathe (SON)

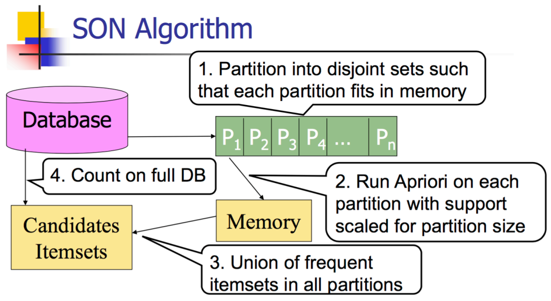
This improvement avoids both false negatives and false positives, at the cost of making two full passes.

Idea: divide the input file into chunks, treat each chunk as a sample, and run the simple, randomized algorithm ion that chunk. Use *ps* as threshold, if each chunk is fraction p of the whole file, and s is the support threshold. Store on disk all the frequent itemsets found for each chunk.

Once all chunks have been processed, take the union of all the itemsets that have been found frequent for one or more chunks. These are the candidate itemsets.

Monotonicity: every itemset that is frequent in the whole is frequent in at least one chunk, and we can be sure that all the truly frequent itemsets are among the candidates; i.e., there are no false negatives.

- 1st pass: read each chunk and process it
- 2nd pass: count all the candidate itemsets and select those that have support at least s as the frequent itemsets


SON Algorithm
1. Partition into disjoint sets such that each partition fits in memory
Database
P_1 P_2 P_3 P_4 ... P_n
4. Count on full DB
2. Run Apriori on each partition with support scaled for partition size
Candidates Itemsets
Memory
3. Union of frequent itemsets in all partitions

5.3 Toivonen's Algorithm

Again, it uses the simple algorithm, but lower the threshold s for the sample. Ex if the sample is 1% of the baskets, use s/125 vs. s/100.

Goal: avoid missing truly frequent itemsets.

Toivonen's algorithm begins by selecting a small sample of the input dataset, and finding from it the candidate frequent itemsets. It is essential that the threshold is set to something less than its proportional value. The smaller we make the threshold, the more main memory we need for computing all itemsets that are frequent in the sample, but the more likely we are to avoid the situation where the algorithm m fails to provide an answer.

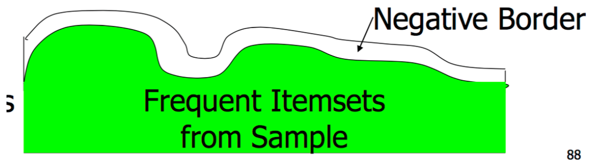
We then construct the negative border. This is the collection of itemsets that are not frequent in the sample, but all of their immediate subsets are frequent in the sample.

Example: ABCD is in the negative border if and only if:

- It is not frequent ni the sample, and
- ABC, ABD, ACD, BCD are all frequent in sample.

A is negative border if and only of

- It is not frequent in the sample
- Empty set is frequent unless the number of baskets < support



To complete the algorithm, we make a pass through the entire dataset, counting all the itemsets that are frequent in the sample or are in the negative border.

There are two possible outcomes

- No member of the negative border is frequent in the whole dataset. In this case, the correct set of frequent itemsets is exactly those itemsets from the sample that were found to be frequent in the whole.
- Some member of the negative border is frequent in the whole. The we cannot be sure that there are not some even larger sets, in neither the negative border nor the collection of frequent itemsets for the sample, that are also frequent in the whole. So, we can give no answer at this time and must repeat the algorithm with a new random sample.

Theorem: if there is an itemset that is frequent in the whole, but not frequent in the sample, then there is a member of the negative border for the sample that is frequent in the whole.

Proof: prove it by contradiction

Suppose not; i.e.;

1) There is an itemset S frequent in the whole but not frequent in the sample, and
2) Nothing in negative border is frequent in the whole

Let T be a smallest subset of S that is not frequent in the sample. T is frequent in the whole (S is frequent + monotonicity). T is in the negative border because otherwise it would not be the smallest.
Any removal from T results in something that is frequent.

⇨ Why are these variants interesting? These employ standard things to improve efficiency: hashing and sampling. Hashing is a good technique, used in many algorithms to make things go fast. These are standard techniques that we will often encounter.
They can be used in many other contexts, so good to know there strengths and weaknesses.

Always think about how you can improve an algorithm!

6. FP Growth

Quick reminder, where are we so far?

- A-Priori
  - Guaranteed to be complete
  - Slow due to repeated DB scans to generate candidate itemsets

  Although the Apriori heuristic achieves good performance gained by reducing the size of candidates sets, it may suffer from some costs in situations with a large number of frequent patterns, long patterns, or quite low minimum support thresholds.

- Toivonen's Algorithm
  - Only requires two DB scans
  - Not guaranteed to be complete

⇨ Can we get fast and complete?

There is an approach called **Frequent-pattern tree** (FP tree) that tries to do this. It mainly consists out of two big ideas. This is an extended prefix-tree structure storing crucial, quantitative information about frequent patterns.

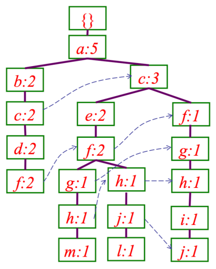
Everything we have seen so far is based on the A-Priori algorithm, count everything of size 1 before moving to size 2, and so on. Studies have shown that FP-growth is about an order of magnitude faster than Apriori, especially when the data set is dense and/or when the frequent patterns are long.

- **Data compression**: Exploit redundancy to compactly but completely represent frequent items in DB. Compress the database to something smaller that fits in the main memory.
- Avoids repeated DB scans.

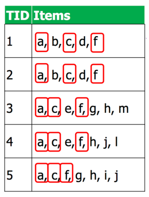
A compact data structure can be designed based on the following observations

- Since only the frequent items will play a role in the frequent-pattern mining, it is necessary to perform one scan of transaction database DB to identify the set of frequent items
- If the set of frequent items of each transaction can be stored in some compact structure, it may be possible to avoid repeatedly scanning the original transaction database
- If multiple transactions share a set of frequent items, it may be possible to merge the shared sets with the number of occurrences registered as count. It is easy to check whether two sets are identical if the frequent items in all of the transactions are listed according to a fixed order.
- If two transactions share a common prefix, according to some sorted order of frequent items, the shared parts can be merged using one prefix structure as long as the count is registered properly. If the frequent items are stored in their frequency descending order, there are better chances that more prefix stings can be shared.

The FP-tree construction takes exactly **two scans** of the transaction database: the first scan collects the set of frequent items, and the second constructs the FP-tree.



To discover item sets in the FP-tree, partition the data and count within each partition: aka Divide-and-conquer. It avoids candidate generation.

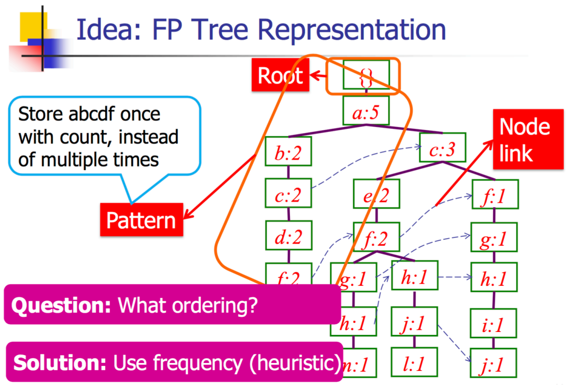


TID	Items
1	a, b, c, d, f
2	a, b, c, d, f
3	a, c, e, f, g, h, m
4	a, c, e, f, h, j, l
5	a, c, f, g, h, i, j

Looking at the five transactions, we observe that the first one and the second one are exactly the same transactions. The transaction is repeated so we could just store it once with count. However, it is very unlikely to find man exact repeats. The question is, how to do better than this?
When we look at the data, there are still repeated structures in this data. We can see things like f appears in all transactions.
<u>Question</u>: how could we exploit the structure to compress? The goal is to very quickly and cheaply compress the data. Ideally we should compress the data in one or two passes.

TID	Items
1	a, b, c, d, f
2	a, b, c, d, f
3	a, c, e, f, g, h, m
4	a, c, e, f, h, j, l
5	a, c, f, g, h, i, j

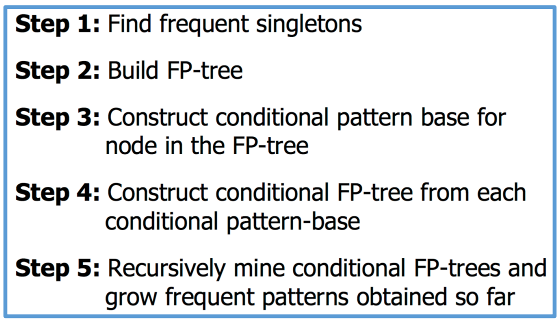
Instead of storing f once. We store it once and we keep track of the counts. Each node in the three is just an item with the count. We can reconstruct the pattern by going down the tree.
<u>Question</u>: is there a better way to order the data?


Idea: FP Tree Representation
Root
Store abcdf once with count, instead of multiple times
Pattern
Node link
Question: What ordering?
Solution: Use frequency (heuristic)
a:5
b:2
c:3
c:2
e:2
f:1
d:2
f:2
g:1
f:2
g:1
h:1
h:1
h:1
j:1
i:1
n:1
l:1
j:1

General idea: Divide-and-conquer to recursively grow frequent pattern path using the FP-tree.

Method:

- For each item, construct is conditional pattern-base and then its conditional FP-tree
- Repeat process on each newly created conditional FP-tree until
  - The resulting FP-tree is empty, or
  - It contains only one path



Step 1: Find frequent singletons

Step 2: Build FP-tree

Step 3: Construct conditional pattern base for node in the FP-tree

Step 4: Construct conditional FP-tree from each conditional pattern-base

Step 5: Recursively mine conditional FP-trees and grow frequent patterns obtained so far

⇨ A detailed example is given in the slides.

⇨ Why is this a beneficial technique? It is a very compact representation of the database. We just increase the counts instead of just keeping the itemsets several times into the database.

Why is FP-Growth Fast?

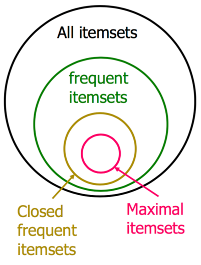
- No candidate generation, no candidate test
- Use compact data structure
- Eliminate repeated database scans
- Basic operation is counting and FP-tree building

7. Incorporating constraints into mining

It is not realistic to find all patterns in a database autonomously.

- Too many patterns
- Patterns too unfocused

Data mining is interactive: users should provide advice and direction to algorithms. Usually we have a hypothesis we want to evaluate by looking at the data.



Constraint-based mining

- User provides constraints on what to mine
- System exploits constraints for efficiency

Type of constraints

- Interest constraint: <u>Ex</u>: rule meets minimum support, confidence, interest thresholds
- Data constraint. Ex: products sold together in Leuven, in March 2017-04-29
- Rule or pattern constraint.
    - Content: small male (sum <10) triggers large sale (sum >200)
    - From: meta-rule. P(x, y) ∧ Q(x, w) → takes (x, database systems)

Given constraints, a mining algorithm should be

- **Sound**: only finds itemsets that satisfy constraints
- **Complete**: finds all itemsets that satisfy constraints

A Naïve solution finds all frequent itemsets, then test them for constraint satisfaction. This is an efficient technique.

The efficient approach is to analyse the itemset and push the constraint into the mining process. Then, things will be much more efficient.

Anti-Monotonicity: when an itemset S violates the constraint, so does any of its superset.

- Support(S) $\geq v$ is anti-monotone
- Support(S) $\leq v$ is not anti-monotone
- *sum(S.Price)* $\leq v$ is anti-monotone
- *sum(S.Price)* $\geq v$ is not anti-monotone

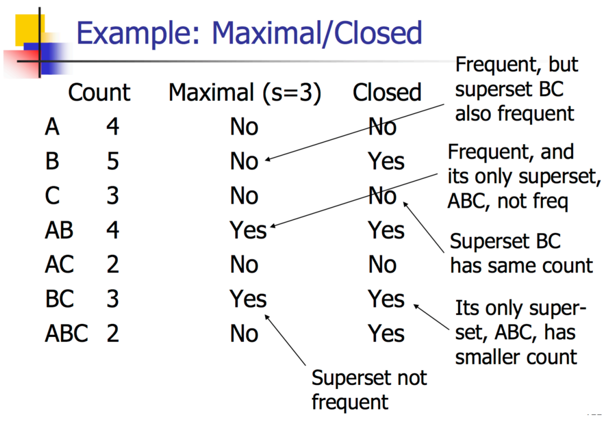
Succinctness: if A is succinct you can enumerate all, and only those, that satisfy it.

<u>Idea</u>: without looking at the data, can determine itemsets that satisfies the constraint.
<u>Ex</u>: min(S.Price) ≤ v is succinct

8. Presenting results, other metrics

- Maximal itemsets: no immediate superset is frequent

- Closed itemsets: no immediate superset has the same count (>0)
    - Stores what is frequent and exact counts
    - Lossless encoding of frequent itemsets



Example: Maximal/Closed

	Count	Maximal (s=3)	Closed
A	4	No	No
B	5	No	Yes
C	3	No	No
AB	4	Yes	Yes
AC	2	No	No
BC	3	Yes	Yes
ABC	2	No	Yes

Frequent, but superset BC also frequent

Frequent, and its only superset, ABC, not freq

Superset BC has same count

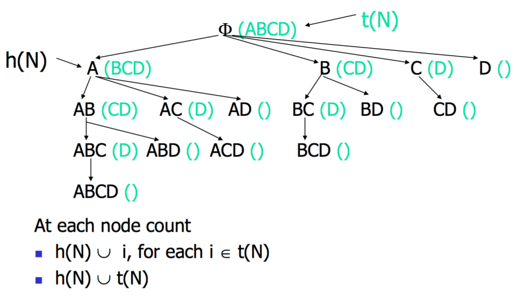
Its only super-set, ABC, has smaller count

Superset not frequent

⇨ Why do we need these notions of maximal and closed itemsets?

- Way to compress the search space. What is the intuition if we only keep the closed itemsets? Or if we only keep the maximal ones, in which case do we lose information?
    - If we only keep the closest, we can generate all the frequent itemsets.

Mining Maximal Itemsets: MaxMiner



At each node count
- h(N) ∪ i, for each i ∈ t(N)
- h(N) ∪ t(N)

We want to try something that is as big as possible and frequent. Always add sub-branches.

Pruning techniques

- Local pruning (e.g., at node A)
    - If h(N)∪t(N) is **frequent**, prune whole sub- tree (e.g., if ABCD is frequent don't explore)
    - If h(N) ∪ i, for some i ∈ t(N) is **NOT frequent** remove i from "h" before expanding (e.g., if AC not frequent, remove C)
- Global pruning: When max pattern found (e.g., ABCD). Prune all nodes (e.g., B, C, D) where h(N)∪t(N) is a subset (e.g., subset of ABCD)

⇨ An example is given in the slides

We have looked at two popular objective measurements: support and confidence. There are other things we can look at, which are more subjective measures: **interestigness**. A rule (pattern) is interesting if it is

- Unexpected (surprising to the user)
- Actionable (the user can do something with it)

- $P(S \wedge B) = P(S) \times P(B) \Rightarrow$ Statistical independence
- $P(S \wedge B) > P(S) \times P(B) \Rightarrow$ Positively correlated
- $P(S \wedge B) < P(S) \times P(B) \Rightarrow$ Negatively correlated
- Also written: $\text{Lift}(A \rightarrow B) = \dfrac{P(B \mid A)}{P(B)}$

To sum up …

Characterization of algorithms

- Search techniques:
  - Breadth-first: Apriori and its variants
  - Depth-first: FP-Growth
  - Look ahead
- Pruning techniques
  - Subset infrequent
  - Hashing
  - Superset frequency
- Limiting disk accesses is the primary goal

Presenting rules

- Which itemsets
  - All itemsets
  - Closed itemsets
  - Maximal itemsets
- What metrics
  - Support, confidence, interest
  - Lift
  - Correlation

⇨ What is the difference between standard rule induction and association rules?
  - Rule induction: the rule is just a target. In rule mining we perform a classifier.
  - Association rules

Association rules are an efficient way to mine interesting information in very large databases: get probabilities. It does not require (but can exploit) user guidance for interesting patterns.

A-priori algorithm and its extensions allow the user to gather a good deal of information without too many passes through data.

Food for thought: Pattern explosion is a well-known problem in frequent itemset mining. High support thresholds typically result only in few well-known patterns; but for low support thresholds, the number of frequent itemsets can easily be orders of magnitude larger than the number of transactions. Knowledge discovery in such humongous itemset collections is virtually impossible.
What are the causes of pattern explosion? Can you think of a way to solve or at least alleviate this issue?

- Problem with the choice of support because there is a trade-off.
  - Too low: lots and lots of itemsets
  - Too high: few itemsets which are well-known
- If there are statistical dependencies between some items then you see them appearing everywhere together, which might not add additional information.

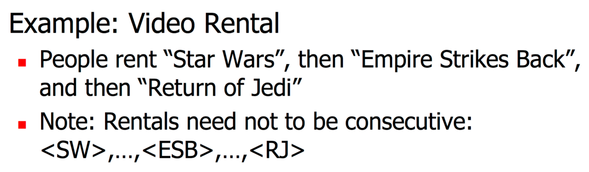
  How can we deal with it? We could for instance use constraints. If we have some background information, we can exclude some itemsets.

CHAPTER 7: MINING SEQUENCES

- Understand how sequential pattern mining relates to association rule mining
- Introduce the basic terminology and concepts of sequential pattern mining
- A brief introduction to the different algorithms for sequential pattern discovery

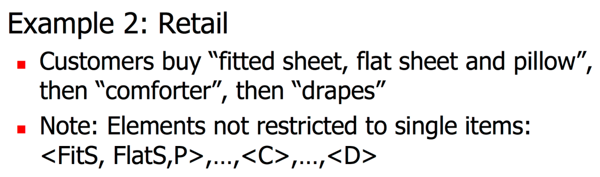
1. Motivation

Idea: Sequential pattern mining discovers frequent subsequences as patterns in a sequence database. It is an important data mining problem with broad applications.



Example: Video Rental

- People rent "Star Wars", then "Empire Strikes Back", and then "Return of Jedi"
- Note: Rentals need not to be consecutive: <SW>,...,<ESB>,...,<RJ>



Example 2: Retail

- Customers buy "fitted sheet, flat sheet and pillow", then "comforter", then "drapes"
- Note: Elements not restricted to single items: <FitS, FlatS,P>,...,<C>,...,<D>

- **Association rule mining**: find relationships between items that co-occur simultaneously. Looking at associations rule mining we were looking at associations and connections between items in a market basket for instance.
- **Sequential pattern** analysis considers time (order transactions occur in). In sequential pattern mining we will take the order into account. The order matters here. Often the order can be important in decision making.

  Many different applications domains are characterized by sequential data. Often the time or order of actions/events can be relevant in decision making. It is important to be able to capture those types of dependencies.

  The Sequential pattern mining task is quite similar to that of association rule mining.

  Given: A sequential database and minimum support threshold.
  Find: All (maximal) frequent sub sequences

2. Background and definitions

We previously worked on transaction database, assuming a transaction ID. Here, we will time stamp a customer identifier. So we need to make a **sequence database** based on the transactions of each customer.

Transaction Database

Time	CID	Items
03/01/11	2	CDs, DVDs
05/01/11	4	PC
08/01/11	2	PC
09/01/11	4	Printer, Ink
09/01/11	1	PC
15/01/11	4	Paper
17/01/11	3	PC, Chair,Ink
18/01/11	1	Paper

Sequence Database

CID	Items
1	<(PC) (Paper)>
2	<(CDs, DVDs) (PC) >
3	<(PC, Chair, Ink)>
4	<(PC) (Printer, Ink) (Paper)>

The customer-sequence database is a time-order list of a customer's transactions. For instance, customer 1 bought a PC before paper. Customer 4 bought a PC before a printer and ink.

We can group all transactions by users.

Time	CID	Items
10/01/11	1	PC
18/01/11	1	Paper
03/01/11	2	CDs, DVDs
08/01/11	2	PC
09/01/11	2	Printer, Table, Ink
10/01/11	3	PC, Chair, Ink
09/01/11	4	PC
22/01/11	4	Printer, Ink
23/01/11	4	Paper
05/01/11	5	Paper

A **sequence** is just an ordered list of itemsets. Items in the transactions remain unordered.

E.g.:< (7) (3 8) (9) (4 5 6) (8) >

Transaction: Items are unordered

The length of a sequence is the number of itemsets in the sequence, the number of transactions in it.

E.g.: <(7) (3 8) (9) (4 5 6) (8)> has length 5

E.g.: <(1 3 5 6) (2 3)> has length 2

A sequence S1 = (a1,a2,...,an) is contained in another sequence S2 = (b1,b2,...,bm) IF

- Exists integers i1 < i2<...< in such that
- a1 ⊆ bi1 AND a2 ⊆ bi2 AND ... AND an ⊆ bin

I can find some transactions in S2 such that S1 occurs in it. So we have to find a way to map those transactions of S1 belonging to S2.

Examples

- <(3)(4 5) (8)> is <(7) (3 8) (9) (4 5 6) (8)>
- <(3)(5)> is NOT in <(3 5)>
- <(3 5)> is not in <(3)(5)>

An item can occur at most once in an element of a sequence, but can occur multiple times in different elements of a sequence.

- A **customer supports** a sequence s if s is contained in the customer sequence.
  The **support** is the fraction of customers who support the sequence.
- A **sequence is frequent** if it meets a minimum support threshold.
- A sequence s is **maximal** if it is not contained in any other sequence.

Ex:

- Customer 2 supports <(PC) (Printer)>
- Support of <(PC) (Printer)> = 2
- Support of <(Printer) (Ink)> = 0
- <(PC) (Printer)> and <(PC) (Paper)> are frequent with minimum support threshold of 2

CID	Items
1	<(PC) (Paper)>
2	<(CDs, DVDs) (PC) (Printer, Ink, Table)>
3	<(PC, Chair, Ink)>
4	<(PC) (Printer, Ink) (Paper)>

Rmq: <(Printer) (Ink)> means that the printer and the ink are bought in two separate transactions. Notice that customer 4 buys both the printer and ink but in the same transactions. Which is different from <(Printer) (Ink)>.

Remember the Apriori property: if a sequence is not frequent, then none of its super-sequences are frequent.

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

For example, with support threshold = 2 <hb> is infrequent

Thus <hab> and <(ah)b> are also infrequent

Sequential pattern mining faces some challenges. Indeed, databases contain a huge number of possible sequential patterns. Because we are now considering the order. Hence, generating candidates tends to be quite challenging.
Algorithms should find all patterns (if possible) that meet a minimum support threshold. They should also be efficient and scalable and incorporate the user provided constraint.

⇨ The big challenges reside in generating candidates.

There are **2 possible** ways to extend a possible candidate.

- Extend the sequence to include another itemset (i.e., a separate transaction). I have a sequence of a certain length, for instance length 2 and I add a new transaction, leading to a sequence of length 3.
  - Extend <(2)> to
  - <(2) (3)> or <(2) (4)>, …
- Extend the number of items included in one itemset (i.e., model **co-occurring** items).
  - Extend <(2)> to
  - <(2 3)> or <(2 4)>,…

Assume that we have 6 frequent length 1 sequences.
First: Extend the length of the sequence. This yields 6x6 = 36 candidates.

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

Second: we will extend the size of an item set. This yields 6x5/2 = 15 candidates. Instead of having 15 itemsets we will have 15+36 itemsets.

There are a number of **applications** that people look at.

- Think about retail shopping. First, buy a computer, second, by printer within the next 3 months.
- Other applications are in the medical domains. First distinguish symptoms, then diseases and third treatments.
- Financial markets: commonly occurring transaction orders.
- Web data. Ex: click streams. What order do people visit pages in?
- Telephone calling patterns: what order do people call other sin?
- DNA sequences: what structure is commonly repeated?

Quick check …

CID	Items
1	<(Ink, Paper)>
2	<(Printer, Table) (Ink, Paper)>
3	<(PC, Ink) (Paper) (Ink)>
4	<(PC) (Ink, Paper) (PC)>

What is the support of:

- (PC). Support is 2. We only care about the number of customers who bought a pc. Here: customers 3 and 4. We don't count the number of times a pc was bought.
- (Ink, Paper). Support is 3.
- (PC) (Ink). Support is 2

Given: 10 things of size 1 are found to be frequent.
How many item sets must be evaluated on the next search iteration? 45 (= 10x9/2)
How many sequences must be evaluated on the next search iteration? 45 (items) + 100 = 145 (= 10x9/2) + 10^2)

3. FreeSpan

⇨ Can we develop a method which may absorb the spirit of Apriori but avoid or substantially reduce the expensive candidate generation and test?

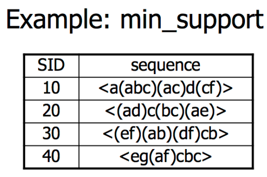
FreeSpan is basically the first thing we think of in sequential mining. The idea is to adapt the FP-Growth to the sequential case.

Idea: Use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database.

Overview: Divide-and-conquer approach
- Step 1: Count sequences to find f-list
- Step 2: Recurse
  - Project sequence database into a set of smaller databases
  - Mine frequent patterns in each projected database.

The big challenge here is that **order** matters!

Step 1: Find length-1 sequential patterns and list them in support descending order


Example: min_support = 2

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

⇨ f_list = a:4,b:4,c:4,d:3,e:3,f:3; g:1

Rmq: we don't count how often a pattern appears in the transactions. We count the number of customers' transactions in which the 1-length sequence appears.

Here, we can prune g, as it only has a support of 1. Hence, we don't care about g. we will use the other frequent sequences to partition the search space.

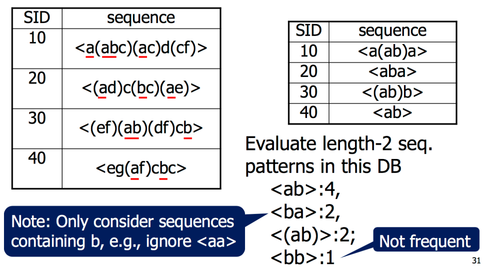
Step 2: Divide the search space. The complete set of sequence patterns can be partitioned into 6 disjoint subsets (move down the f_list).

- Contain item a
- Contain item b but no items after b in f_list
- Contain item c but no items after c in f_list
- Contain item d but no items after d in f_list
- Contain item e but no items after e in f_list
- Contain item f

The subsets of sequential patterns can be mined by constructing projected databases.

- Finding sequential patterns containing only item a. By scanning sequence database once, the only two sequential patterns containing only item a are: (a) and (aa).
- Finding sequential patterns containing item b but no item after b in f_list. This can be achieved by constructing the (b)-projected database.
- Finding other subsets of sequential patterns. Others subsets can be found similarly, by constructing corresponding projected database and mining them recursively. These databases are constructed simultaneously during one scan of the original sequence database.

Let's look at the **<b> projected database**.



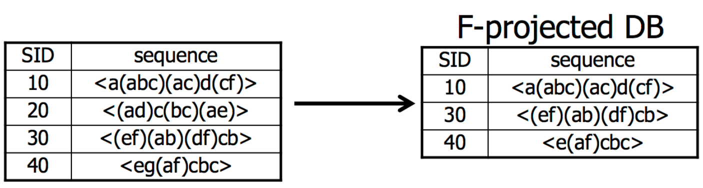
SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

SID	sequence
10	<a(ab)a>
20	<aba>
30	<(ab)b>
40	<ab>

Evaluate length-2 seq. patterns in this DB
<ab>:4,
<ba>:2,
<(ab)>:2;
<bb>:1

Note: Only consider sequences containing b, e.g., ignore <aa>
Not frequent
31

Pro's	Con's
• Projections replace candidate generate	• Con project at any point in sequence • Projected sequences may not be shorter



SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

F-projected DB

SID	sequence
10	<a(abc)(ac)d(cf)>
30	<(ef)(ab)(df)cb>
40	<e(af)cbc>

In this case I didn't save that much work. My sequences are not that shorter than the previous sequences. I just managed to remove one customer ID.

4. PrefixSpan

Idea: instead of projecting sequence databases by considering all the possible occurrences of frequent subsequences, the projection is based only on frequent prefixes because any frequent subsequence can always be found by growing a frequent prefix.

PrefixSpan mines the complete set of patterns but it greatly reduces the efforts of candidate subsequence generation. Moreover, it reduces the size of projected databases and leads to efficient processing.
PrefixSpan outperforms both the A-Priori algorithm and previously seen method, FreeSpan.

Pro's

- ○ Maintain projection idea, avoid candidate generation. We want to guarantee we do less work.
- ○ Projection based on sequence prefix: ensure that sequences get progressively shorter.

Definition: Prefix

Sequence β =<b1b2...bm> is a prefix of sequence α =<a1a2...an>, where m ≤ n, IFF

- $b_i = a_i$ for $i \leq m-1$
- $b_m \subseteq a_m$
- All items in $(a_m - b_m)$ are alphabetically after those in b_m

Example: α =<a(abc)(ac)d(cf)>

β =<a>
β =<a(ab)>
β =<a(abc)a>
Legal

β =<a(abc)c>
β =<a(ab)a>
Illegal

Definition : Projection

A subsequence α' of sequence α is a projection of α w.r.t. prefix β IFF

- β is a subsequence of α
- α' has prefix β
- No proper super-sequence α'' of α' such that α'' is a subsequence of α and has prefix β

Example: α =<a(abc)(ac)d(cf)>

- β =<(abc)a>
- α' =<(abc)(ac)d(cf)>

Similarly, what we can do is look at the **Suffix**. Split the projection into two part, the prefix (data) and the suffix.

- Let α' =<a1a2...an> be the projection of α w.r.t. prefix β =<a1a2...am-1a'm> (m ≤n)
- Sequence γ=<a''mam+1...an> is called the suffix of α w.r.t. prefix β, denoted as γ= α / β, where α''m=(am-a'm)

- We also denote α' =β · γ

  Example: α' =<a(abc)(ac)d(cf)>
  - β =<a(abc)a>
  - γ=<(_c)d(cf)>

Projected DB

- Given, prefix β and suffix γ
- We call γ the β projected DB

  When my prefix gets longer and longer, my suffix gets smaller and smaller.

  Example: α =<a(abc)(ac)d(cf)>
  - β =<(ac)d>
  - β projected DB = <(cf)>

  Example: α =<(ef)(ab)(df)cb>
  - β =<e>
  - β projected DB = <(_f)(ab)(df)cb>

  The first occurrence of e occurs in a transaction with two items, so we don't want to lose the fact that there is some structure and that e co-occurs. So, we add an "_", to indicate that f should occur with another item.

The PrefixSpan Algorithm performs a divide-and-conquer search to find all sequential patterns.

- Step 1: Find all frequent length 1 sequences
- Step 2: Divide search space based on length 1 prefixes and recurse: PrefixSpan(α, l, S| α)

  - α: sequential pattern,
  - l: the length of α
  - S| α : the α-projected database, if α ≠<>; otherwise; the sequence database S

  1. Scan S| α to find frequent items b such that:
     a. b can be assembled to the last element of α to form a sequential pattern; or
     b. <b> can be appended to α to form a sequential pattern

  2. For each frequent item b, append it to α to form a sequential pattern α', and output α';

  3. For each α', construct α'-projected database S| α', and call PrefixSpan(α', l+1, S| α').

  A detailed example is provided in the slides

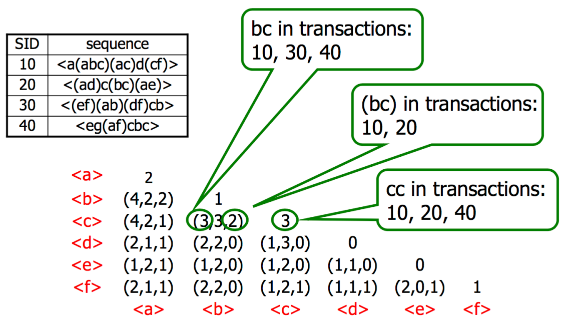
Projected DBs allow us to avoid generating candidate sequences.

- No candidate sequence needs to be generated by PrefixSpan. It only grows longer sequential patterns from the shorter frequent ones. It does not generate nor test any candidate sequence non-existent in a projected database.
- The projected databases keep shrinking: a projected database is smaller than the original one because only the postfix subsequences of a frequent prefix are projected into a projected database.
- Major cost of PrefixSpan is the construction of projected databases.

The costliest operation is building projected DBs. Every time I perform a new recursive step, i have to build a new database. If the number and/or size of projected databases can be reduced, the performance of sequential pattern mining can be improved substantially. There are two ways to improve computational efficiency.

- **Bi-level projections** to reduce the number and sizes of projected DBs. The **first scan** identifies frequent length 1 sequences. The second scan constructs a triangular matrix instead of projected databases and count.
    - Support(a,a) for each item
    - For each pair (a,b) of frequent items count
        - Support(<ab>)
        - Support(<ba>)
        - Support(<(ab)>
    - On the next level, construct projected DBs

  We will look at very small parts of the data. Look at things we think we need.



SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

<a>	2					
<b>	(4,2,2)	1				
<c>	(4,2,1)	(3,3,2)	3			
<d>	(2,1,1)	(2,2,0)	(1,3,0)	0		
<e>	(1,2,1)	(1,2,0)	(1,2,0)	(1,1,0)	0	
<f>	(2,1,1)	(2,2,0)	(1,2,1)	(1,1,1)	(2,0,1)	1
	<a>	<b>	<c>	<d>	<e>	<f>

bc in transactions: 10, 30, 40
(bc) in transactions: 10, 20
cc in transactions: 10, 20, 40

For each frequent length-2 sequential pattern construct a projected DB. Then on next scan build the matrix.

<ab>-projected database
<(_c)(ac)(cf)>
<(_c)(a)>
<c>

Seq	Support
<a>	2
<c>	2
<f>	1
<(_c)>	2

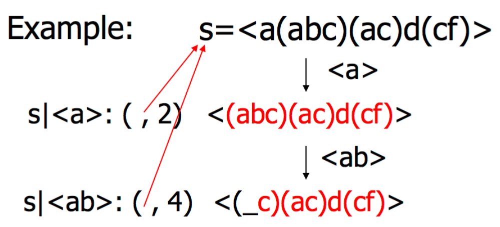
<a>	0		
<c>	(1,0,1)	1	
<(_c)>	(φ,2, φ)	(φ,1, φ)	φ
	<a>	<c>	<(_c)>

⇨ For more information, cfr. Hand-written notes in the slides.

- **Pseudo-projection** reduces cost if projected database fits in memory. We want to avoid copying.
  Insight: Postfixes of sequences often appear repeatedly in recursive projected databases.
  Pseudo-projection: avoid repeated physical copying postfix
  - Efficient if projected DB fits in memory
  - Costly if projected DB is disk resident

  Idea: use points to avoid copying

  With each projection store
  - Pointer to base sequence. Points to where I want to be in the sequence.
  - Offset to where post-fix starts



Instead of building the data, I just maintain a pointer.

To conclude, many important domains are characterized by the presence of sequential data.
Sequential pattern mining helps capture time dependences between events. It is similar to frequent itemsets but considers orders of elements.

⇨ Common algorithms are inspired by itemset mining.

CHAPTER 8: CLUSTERING

1. Unsupervised learning, clustering intro

Clustering is the process of examining a collection of “points”, and grouping the points into “clusters” according to some distance measure.

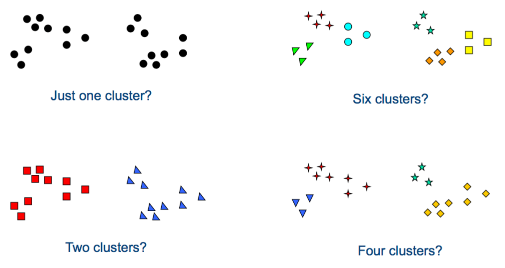
Goal: find groups of items

Produce clusters such that things that occur in the same group are highly similar, or are close together in distance.
A good clustering method will produce clusters with

- High intra-class similarity
- Low inter-class similarity

When you are faced with the clustering task, many things can make sense. Ex: divide set into groups. You can do grouping based on gender for instance.
Clustering is a subjective notion hence, it is difficult to give a precise definition of clustering quality. It is application dependent and ultimately subjective.

Thinking about how many clusters can also be a tricky question. How many clusters do I want?



Remember …

Supervised learning:

- Data are set of pairs <x,y>, where y=f(x)
- Goal: Approximate f

Unsupervised learning: the data is just x!
Goal: Find structure in the data
Challenge: Ground truth is often missing (no clear error function, like in supervised learning)

One way to think about clustering is that it is an unsupervised learning technique. In unsupervised learning, you don’t have a label. You just have attributes. The goal is to find a structure in the data, based on the attributes.

>< In supervised learning we know the label and we want to optimize a certain metric: ROC, AUC, …

Unsupervised learning tends to be more abstract.

Why is clustering useful? It can be interesting to look at. You can use it to compress the data.

- Groups of data are often useful
- Compress or reduce the data (e.g., establish prototypes)
- Identify outliers (or novel examples)
- Pre-processing for supervised learning (e.g., feature construction)
- Visualization of the data

Given: D = { x1, x2,..., xN }, where each xi is a d-dimensional feature vector (xi,1, xi,2,...,xi,d)
Do: Divide the N vectors into K groups such that the grouping is "optimal"

There are **3 key issues**

- Measure of distances: distance between examples, between clusters, between example and clusters.
- How will we represent clusters?
- Assigning items to clusters

Clustering methods use a **distance (similarity) measure** to assess the distance between

- A pair of instances
- A cluster and an instance
- A pair of clusters

Remember that the requirements for a function on pairs of points to be a distance measure are that

- Distances are always nonnegative, and only the distance between a point and itself is 0.
- Distance is symmetric; it doesn't matter in which order you consider the points when computing their distance.
- Distance measures obey the triangle inequality. The distance from x to y to z is never less than the distance going from x to z directly.

Given a distance value, we can convert it into a similarity value: sim(i,j) = 1/[1+dist(i,j)]
Not always straightforward to go the other way. We'll describe our algorithms in terms of distances.

Manhattan:

$$d(i,j) = (|x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{ip} - x_{jp}|)$$

Euclidean:

$$d(i,j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + \ldots + |x_{i_p} - x_{j_p}|^2)}$$

Edit distance: The number of required edits to transform a string into another string
edit_distance(mining,smiling) = 2

We can divide clustering algorithms into two groups. There exist several types of cluster structures: the hierarchical structures and the flat structures.

- **Hierarchical arrangement**: create a hierarchy between objects.
- **Flat arrangement**: straight partition the data into different groups.

Regarding the **cluster assignment**, we distinguish between different techniques

- Hard clustering: Each item occurs in exactly one cluster
- Soft clustering: Each item has a probability of belonging to a certain cluster
- Disjunctive vs. overlapping clusters. Items may occur in multiple clusters

Clustering approaches

- Hierarchical: Create a hierarchical decomposition of the set of objects using some criterion
- Partitioning: Construct various partitions and then evaluate them by some criterion
- Model-based: Hypothesize a model for each cluster and find best fit of models to data
- Density-based: Guided by connectivity and density functions

2. Hierarchical clustering

2.1 The algorithm

This algorithm starts with each point in its own cluster. Clusters are combined based on their "closeness". Combination stops when further combination leads to clusters that are undesirable for one of several reasons. Ex: we may stop when we have a predetermined number of clusters, or we may use a measure of compactness for clusters, and refuse to construct a cluster by combining two smaller clusters if the resulting cluster has points that are spread out over too large a region.

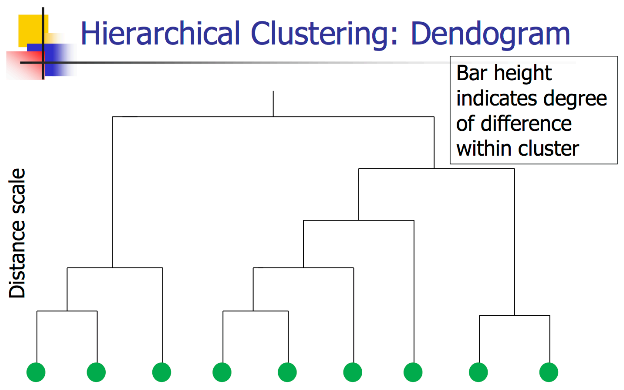
Hierarchical clustering can do top-down (divisive or bottom-up (agglomerative) clustering.

- Top-down: subdivide objects. Partition the items
- Bottom-up: every object is in an individual cluster, cluster them all together into one aggregated cluster.
  Given a set of instances, I create one cluster for each instance.
  I want to find the nearest two clusters.

In either case, we maintain a matrix of distance (or similarity) scores for all pairs of

- Instances;
- Clusters (formed so far)
- Instances and clusters

Dendograms are used to represent hierarchical clustering. We have distance on the x-axis. Leaves represent the instances. The bar height indicates the degree of difference within the cluster.


Hierarchical Clustering: Dendogram
Bar height indicates degree of difference within cluster
Distance scale

```
Given: instances $x_1,…,x_n$
For i = 1 to n, $c_i = \{x_i\}$
$C = \{c_1,…,c_n\}$
$j = n$
While $|C| > 1$
    $j = j+1$
    $(c_a,c_b) = \text{argmin dist}(c_u,c_v)$
    $c_j = c_a \cup c_b$
    add node to tree joining a and b
    $C = (C - \{c_a,c_b\}) \cup c_j$
Return tree with root node j
```

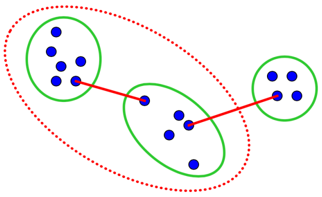
We have to decide in advance on:

- How will clusters be represented?
- How will we choose which two clusters to merge?
- When will we stop combining clusters? There are several approaches we might use to stopping the clustering process.
    - We could be told, or have a belief, about how man clusters there are in the data.
    - We could stop combining when at some point the best combination of existing cluster produces a cluster that is inadequate.
    - Continue clustering until there is only one cluster. However, it is meaningless to return a single cluster consisting of all the points.

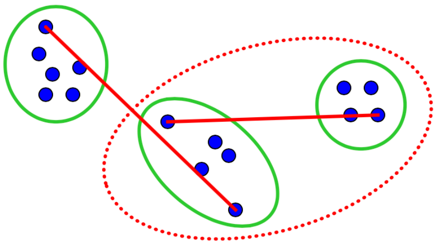
2.2 Distance measures

The distance between two clusters can be determined in several ways

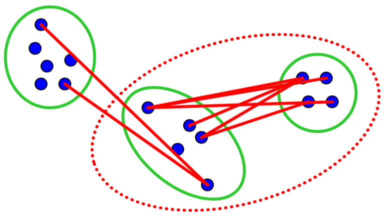
- **Single link:** distance of two most similar instances: dist(cu, cv) = min{dist(a, b) | a∈cu, b∈cv} Take the minimum distances.
  Issue: chain phenomenon.



- **Complete link**: distance of two least similar instances: dist(cu, cv) = max{dist(a, b) | a∈cu, b∈cv}. Take the maximum distance between two clusters. Usually when you do clustering you apply complete linkage.



- **Average link:** average distance between instances: dist(cu, cv) = avg{dist(a, b) | a∈cu, b∈cv}

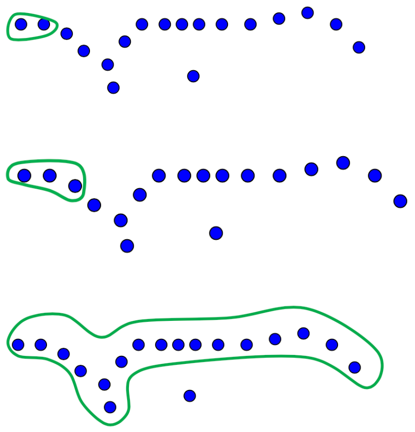


If we merged c_u and c_v into c_j, we can determine distance to each other cluster:

- Single link: dist(cj, ck) = min{dist(cu, ck) , dist(cv, ck)}
- Complete link: dist(cj, ck) = max{dist(cu, ck) , dist(cv, ck)}
- Average link:|c | * dist(c , c) + |c | * dist(c , c) dist(cj, ck) =

$$\frac{|c_u| * dist(c_u, c_k) + |c_v| * dist(c_v, c_k)}{|c_u| + |c_v|}$$

Single link - Chaining



Bottom line

- Simple, fast
- Often low quality

Complete Link

- Worst case O(n^3)
- Fast algorithm: Requires O(n^2) space
- No chaining
- Bottom line:
  Typically much faster than O(n^3)
- Often good quality

2.3 Evaluation of the algorithm

Weaknesses of agglomerative clustering methods:

- The basic algorithm is not very efficient. At each step, we must compute the distances between each pair of clusters, in order to find the best merger.
- Do not scale well: time complexity of at least O(n2), where n is the number of total objects
- Can never undo what was done previously

⇨ Integration of hierarchical with distance-based clustering

- **BIRCH**: uses CF-tree and incrementally adjusts the quality of sub-clusters
- **CURE**: selects well-scattered points from the cluster and then shrinks them towards the centre of the cluster by a specified fraction

BIRCH: Balanced Iterative Reducing and Clustering using Hierarchies (Zhang, Ramakrishnan & Livny, 1996). It incrementally construct a CF (Clustering Feature) tree.
Parameters: max diameter, max children

- Phase 1: scan DB to build an initial in-memory CF tree (each node: #points, sum, sum of squares)
- Phase 2: use an arbitrary clustering algorithm to cluster the leaf nodes of the CF-tree

Scales linearly: finds a good clustering with a single scan
Weaknesses: handles only numeric data, sensitive to order of data records

Centroid: $\vec{X}0 = \frac{\sum_{i=1}^{N} \vec{X}_i}{N}$

Radius: average distance from member points to cluster centroid $R = (\frac{\sum_{i=1}^{N} (\vec{X}_i - \vec{X}0)^2}{N})^{\frac{1}{2}}$

Diameter: average pairwise distance within a cluster

$$D = (\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (\vec{X}_i - \vec{X}_j)^2}{N(N-1)})^{\frac{1}{2}}$$

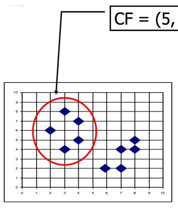
2.4 Cluster Feature Vector

Given: $X_1, \ldots, X_n$, data points in a cluster where each with d-dimensions

We define $\mathrm{CF} = (\mathrm{N}, LS, SS)$, where

- N: Number of data points
- LS: $\sum_{i=1}^{N} X_i$
- SS: $\sum_{i=1}^{N} X_i^2$

Note: CFs are additive!
E.g., CF1 + CF2 = (N1+N2, LS1+LS2, SS1+SS2)



CF = (5, (16,30), (54,190))
(3,4)
(2,6)
(4,5)
(4,7)
(3,8)
$LS_x = 3 + 2 + 4 + 4 + 3 = 16$
$LS_y = 4 + 6 + 5 + 7 + 8 = 30$
$SS_x = 3^2 + 2^2 + 4^2 + 4^2 + 3^2 = 54$
$SS_y = 4^2 + 6^2 + 5^2 + 7^2 + 8^2 = 190$

2d + 1 values represent any number of points
d = number of dimensions.

Averages in each dimension can be calculated as SUMi /N
Variance in dimension i can be computed by: (SUMSQi /N) – (SUMi /N)2
To get standard deviation take square root

Can also compute the **radius**.

$$R = (\frac{\sum_{i=1}^{N} (\overrightarrow{X_i} - \overrightarrow{X_0})^2}{N})^{1/2}$$
$$= (\frac{\sum_{i=1}^{N} (\overrightarrow{X_i}^2 - 2\overrightarrow{X_0}\overrightarrow{X_i} + \overrightarrow{X_0}^2)}{N})^{1/2}$$
$$= (\frac{\sum_{i=1}^{N} \overrightarrow{X_i}^2 - 2\sum_{i=1}^{N} \overrightarrow{X_i}\overrightarrow{X_0} + \sum_{i=1}^{N} \overrightarrow{X_0}^2}{N})^{1/2}$$
$$= (\frac{\sum_{i=1}^{N} \overrightarrow{X_i}^2 - 2\overrightarrow{X_0}\sum_{i=1}^{N} \overrightarrow{X_i} + N\overrightarrow{X_0}^2}{N})^{1/2}$$
$$= (\frac{SS - 2\frac{LS}{N} \cdot LS + N(\frac{LS}{N})^2}{N})^{1/2}$$

46

2.5 Cluster Feature Tree

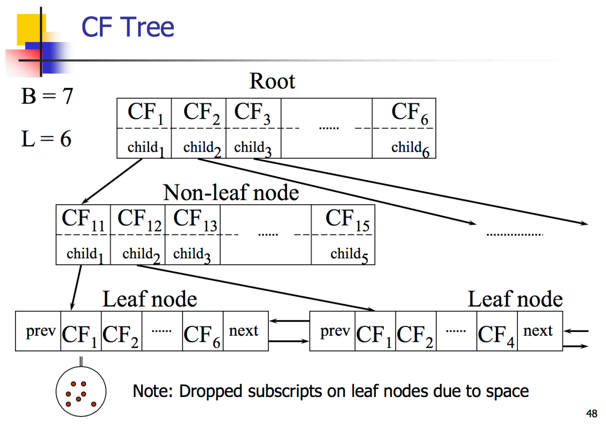
A CF-tree is a height-balanced tree with two parameters:

- **Branching** factor (non leaf nodes B, leaf nodes, L)
- **Threshold** T

Each non leaf node has the form [CF_i, $child_i$]
Each leaf node has CF

- Set of CFs
- Two pointers: prev and next

Radius of a sub cluster under a leaf node cannot exceed the threshold T


CF Tree
B = 7
L = 6
Root
CF1 CF2 CF3 ...... CF6
child1 child2 child3 child6
Non-leaf node
CF11 CF12 CF13 ...... CF15
child1 child2 child3 child5
Leaf node
prev CF1 CF2 ...... CF6 next
Leaf node
prev CF1 CF2 ...... CF4 next
Note: Dropped subscripts on leaf nodes due to space
48

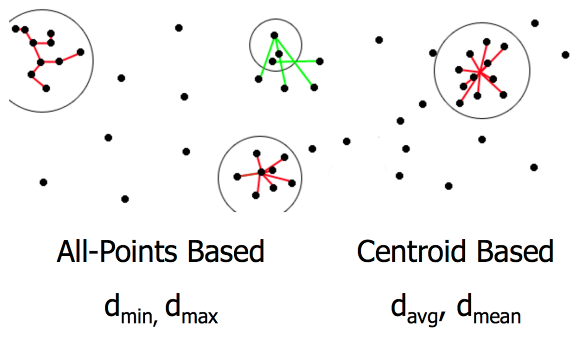
To construct the CF-Tree, scan the data set and insert the incoming data instances into the CF tree one by one. Each instance is inserted into the closest sub cluster under a leaf node. If insertion causes sub cluster diameter to exceed threshold, then create a new sub-cluster.
The new sub-cluster may cause its parents to exceed branching factor. If so, split the leaf node.

- Identifying the pair of sub-clusters with largest inter-cluster distance.
- Divide by proximity to these two sub-clusters

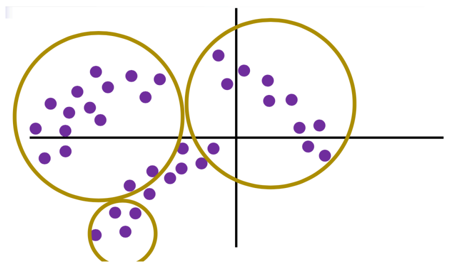
If this split causes the non-leaf node to exceed the branching fact, then recursively split. If the root node is split, then the height of the CF tree is increased by one.

⇨ An example is given in the slides

Traditional algorithms



What would BIRCH do?



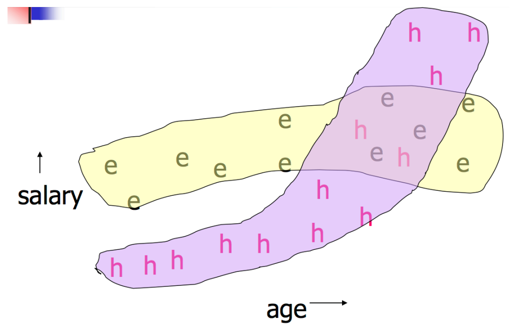
Birch assumes:

- Clusters are normally distributed in each dimension
- Axes are fixed: Ellipses at an angle are not ok

Clustering Using Representatives (CURE)

Cluster definition: Set of representative points. It enables clusters of differing shapes.
⇨ Requires an Euclidean space

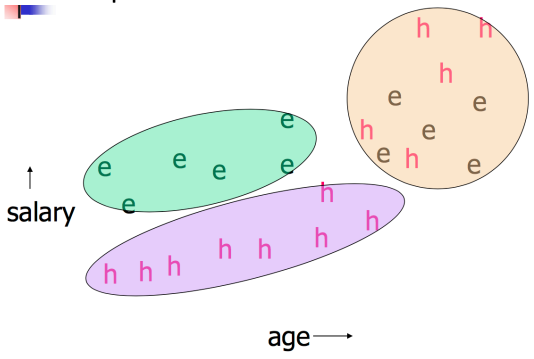
The CURE algorithm does not assume anything about the shape of clusters; they need not be normally distributed, and can even have strange bends, S-shapes, or even rings. Instead of representing clusters by their centroid, it uses a collection of representative points, as the name implies.



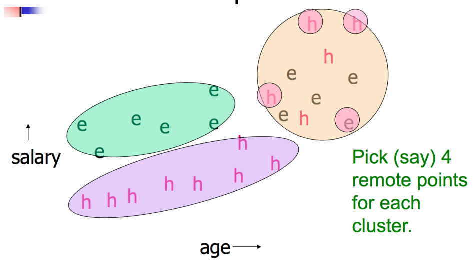
Two-pass (hierarchical) clustering approach

- **Pass 1**: Clustering of subset of data to pick "representative points".

- Take a small sample of the data and cluster it in main memory.

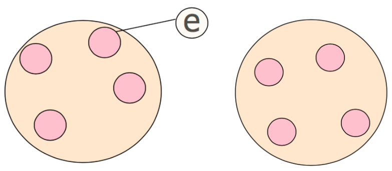


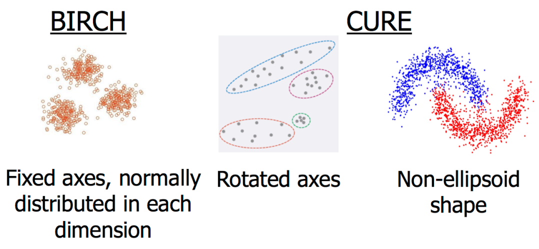
- Select a small set of points from each cluster to be representative points. These points should be chosen to be as far from one another as possible.



- Move each of the representative points a fixed fraction of the distance between its location and the centroid of its cluster. 20% is a good fraction to choose.

- **Pass 2**: Assign all points to clusters
  Scan the entire data set and assign each example e to the “closest” cluster.
  - Standard metric determines the closest
  - Done by finding representative with smallest distance to e.





3. Partitional clustering

3.1 Partitioning Algorithms

<u>Method</u>: Construct a partition of a database D of n objects into a set of k clusters

Given a k, find a partition of k clusters that optimizes the chosen partitioning criterion
Global optimal: exhaustively enumerate all partitions
Heuristic methods: k-means, k-medoids algorithms

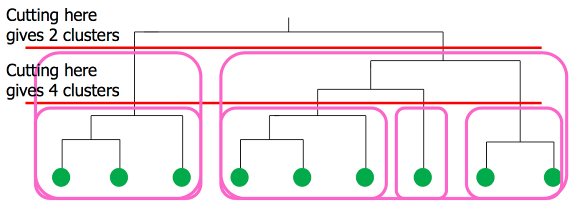
- k-means (MacQueen, 1967): Each cluster is represented by the center of the cluster
- k-medoids or PAM (Partition around medoids) (Kaufman & Rousseeuw, 1987): Each cluster is represented by one of the objects in the cluster

Idea: divide instances into disjoint clusters. Flat vs. tree structure.

Issues:

- How many clusters should there be?
- How should clusters be represented?

We can generate a Partitional clustering from a hierarchical clustering by "cutting" the tree at some level.



3.2 K-Means algorithm

The K-means algorithm is a commonly-used algorithm. It is both easy to implement and quick to run.

Assumptions

- Objects are n-dimensional vectors
- Distance/similarity measure between these instances

Goal: Partition the data in K disjoint subsets. Ideally the partition should reflect the structure of the data.

```
kMeans(Data D, number of clusters k, Distance
   function $\delta$)
Pick k centroids $C = [c_1, \ldots, c_k]$
while (Not converged) do
   for each example $x_i \in D$ do
       assign $x_i$ to $c_j$ such that $\min_j \delta(c_j, x_i)$
   for each $c_j \in C$ do
       $c_j = \mu(c_j)$
Return C
```

Initially choose k points that are likely to be in different clusters. Make these points the centroids of their clusters. For each remaining point p, find the centroid to which p is closest. Add p to the cluster of that centroid and then adjust the centroid of that cluster to account for p.

How to initialize the clusters? We want to pick points that have a good chance of lying in different clusters. There are 2 approaches

- Pick points that are as far away from one another as possible. Good choice: pick the first point at random.
- Cluster a sample of the data, perhaps hierarchically, so there are k clusters. Pick a point from each cluster, perhaps that point closest to the centroid of the cluster.

An optional step at the end is to fix the centroids of the clusters and to reassign each point, including the k initial point. To the k clusters.

The results depend on the seed selection. Some seeds can result in poor convergence or sub-optimal clusterings.

- Pick points randomly or from data instances
- Use results of another method
- Many runs of k-means: each with random seeds

$$\mu(c_j) = [1/ |c_j|] * [\Sigma_i x_i]$$

- $|c_j|$ is number of examples assigned to cluster cj
- $i \in c_j$, i.e., examples that are assigned to cluster cj
- This is a vector: Calculate μ along each dimension

⇨ An example is provided in the slides

A trick point of this algorithm is picking the right value of k. we may not know the correct value of k to use in a k-means clustering. However, if we can measure the quality of the clustering for various values of k, we can usually guess what the right value of k is. If we take a measure of appropriateness for clusters, such as average radius or diameter, the value will grow slowly, as long as he numbers of clusters we assume remains at or above the true number of clusters.
As soon as we try to form fewer clusters, than there really are, the measure will rise precipitously.

Time Complexity

- Distance between two instances: O(d), where d is the dimensionality of the vectors
- Reassigning clusters: O(kn) distance computations, or O(kdn)
- Computing centroids: Each instance vector gets added once to some centroid: O(dn)
- Assume these two steps are each done once for I iterations: O(Ikdn)

- Linear in all relevant factors, with fixed number of iterations, more efficient than O(n2) HAC

3.3 Bradly-Fayyad-Reina (BFR)

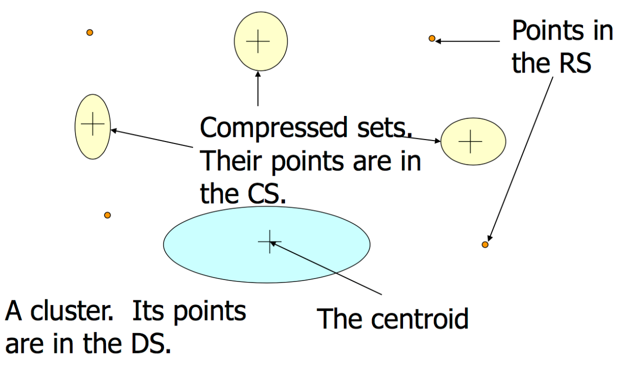
The BFR algorithm is a variant of the k-means that is designed to cluster dta in a high-dimensional Euclidean space.

Key assumption: clusters are normally distributed around a centroid in a Euclidean space. The standard deviations in different dimensions may vary but the dimensions must be independent.

Algorithm overview:

- Read points one main-memory-full at a time. The algorithm begins by selecting k points, using one of the methods we saw in previous section. Then, the points of the data file are read in chunks. These might be chunks from a distributed file system or a conventional file might be partitioned into chunks of the appropriate size. Each chunk must consist of few enough points that they can be processed in main memory. Also stored in main memory are summaries of the k clusters and some other data, so the entire memory is not available to store a chunk.

  Three classes of points
  - **The discard set:** points close enough to a centroid to be represented statistically. For each cluster, the discard set is represented by:
    - The number of points, N
    - The vector SUM, whose ith component is the sum of the coordinates of the points in the ith dimension.
    - The vector SUMSQ: ith component = sum of squares of coordinates in the ith dimension.
  - **The compression set:** groups of points that are close together but not close to any centroid. They are represented statistically, but not assigned to a cluster
  - **The retained set:** isolated points


Points in the RS
Compressed sets. Their points are in the CS.
A cluster. Its points are in the DS.
The centroid

- Summarize most points by simple statistics

- To begin, from the initial load we select the initial k centroids by some sensible approach

Initialization

- Take a small, random sample and cluster optimally
- Take a sample; pick a random point, and then k-1 more point, each as far from the previously selected points as possible

Processing points

- Find points close to a cluster centroid. Add them to cluster and the discard set.
- Perform in-memory clustering on remaining points and the old RS. Clusters become CS, others to RS.
- Adjust cluster statistics based on new points
- Consider merging pairs of CS
- On final round marge ell CSs and RS points into the nearest cluster.

Two key questions

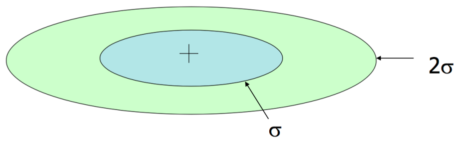
1) When is a point "close enough" to a cluster that we can it to that cluster? We need a way to decide whether to put a new point into a cluster.

   BFR suggests two ways

   - The **Mahalanobis distance** is less than threshold. This is a normalized Euclidean distance. For point (x1,...,xk) and centroid (c1,...,ck):

     - Normalize in each dimension: yi = |xi -ci|/ σi
     - Take sum of the squares of the y_i's
     - Take the square root
     - Accept a point for a cluster if its M.D. is < some threshold, e.g., 4 standard deviations

     Equal M.D. Regions

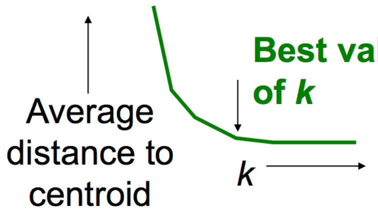
     

   - Low likelihood of the currently nearest centroid changing.
     Add p to a cluster if it not only has the centroid closest to p, but it is very unlikely that, after all the points have been processed, some other cluster centroid will be found to be nearer to p.

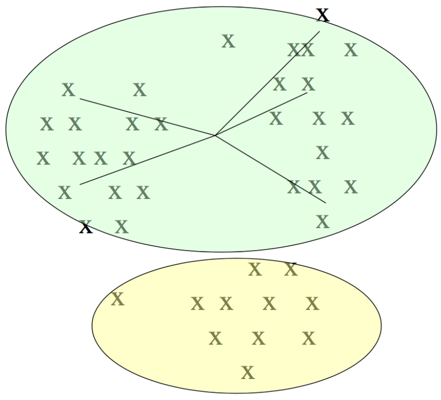
2) When should two compressed sets be combined into one?
   Compute the variance of the combined sub-cluster. N, SUM and SUMQ allow us to make that calculation. Combine if the variance is below some user provided threshold.

Getting the k right

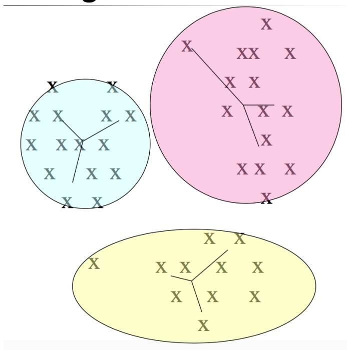
⇨ How to select k? Try different k, looking at the change in the average distance to centroid as k increases. Average falls rapidly until right k, then changes little.



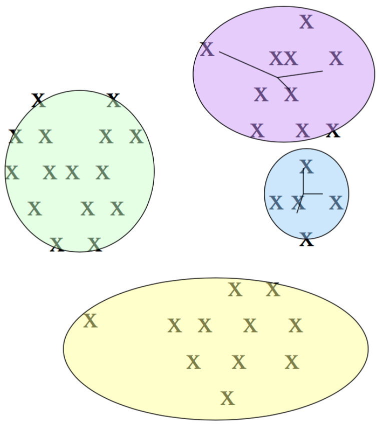
- **Too few**: many long distances to centroid



- **Just right**: distances rather short



- **Too many**: little improvement in average distance



To wrap up

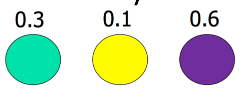
K-Means	
Strengths	*Weaknesses*
• Efficient: O(Ikdn) • Straightforward to implement • Fins local optimum, can use restarts more advanced search to find better solution	• Must be able to define mean • Need to provide k • Susceptible to noise and outliers • Cannot represent non-convex clusters

4. Model-Based clustering

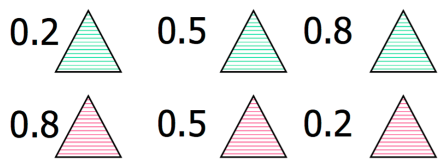
Basic idea: Clustering as probability estimation

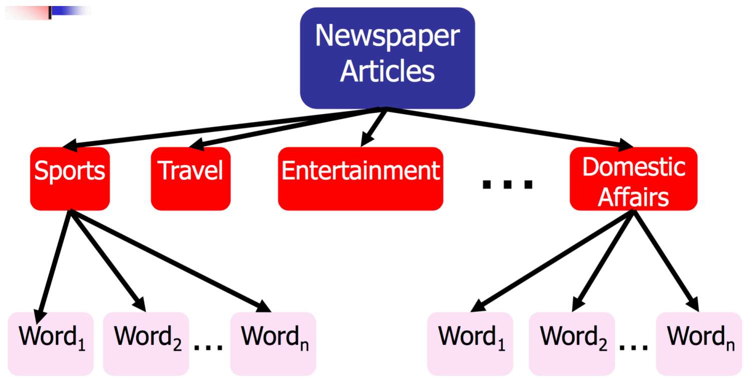
Generative model

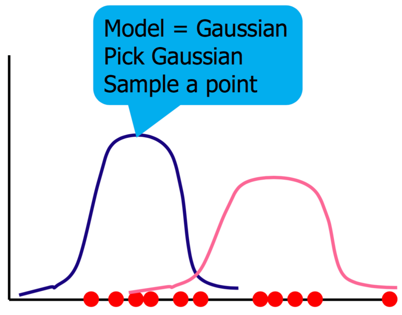
1) Probability of selecting a cluster



2) Probability of generating an object in cluster







Representation: Each cluster is represented by a probability distribution (density).
Given: Data points, number of clusters, model form of each cluster
Find: Maximum likelihood or MAP assignment of data points to clusters and learn parameters for each cluster.
Quality of clustering: Likelihood of test objects

$$N_j(x_i) = \frac{1}{(2\pi\sigma^2)^{0.5}} e^{\frac{-1}{2}\left[\frac{(x_i - \mu_j)}{\sigma}\right]^2}$$

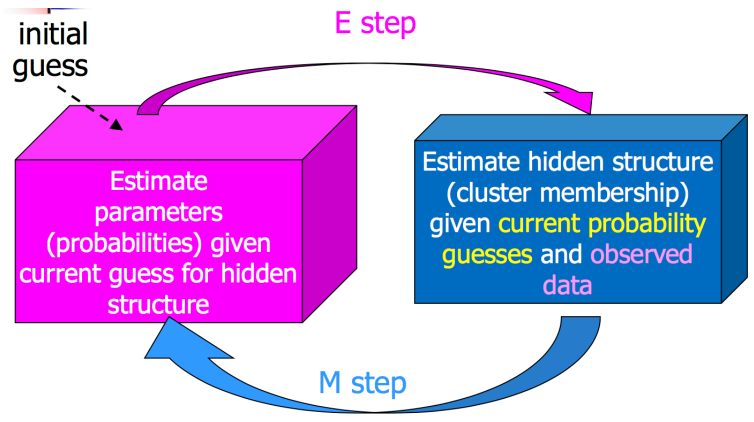
- Where
  - μ_j is the mean
  - σ^2 is the variance
  - $N_j(x_i)$ = probability(x_i | μ_j)

Aside: More or less as probability densities are tricky

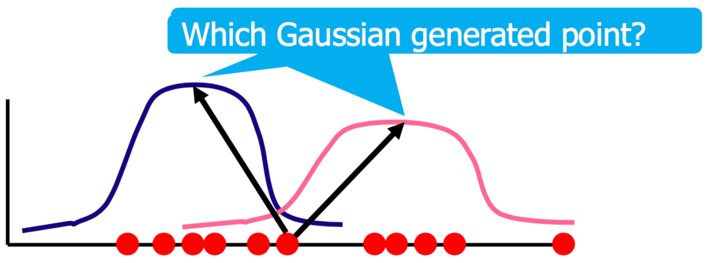
Key Challenge

- If we knew which examples belong to each cluster, then we can set the model parameters
- If we knew model parameters, then we can estimate probability that cluster generated an example

⇨ Chicken and egg problem ☹

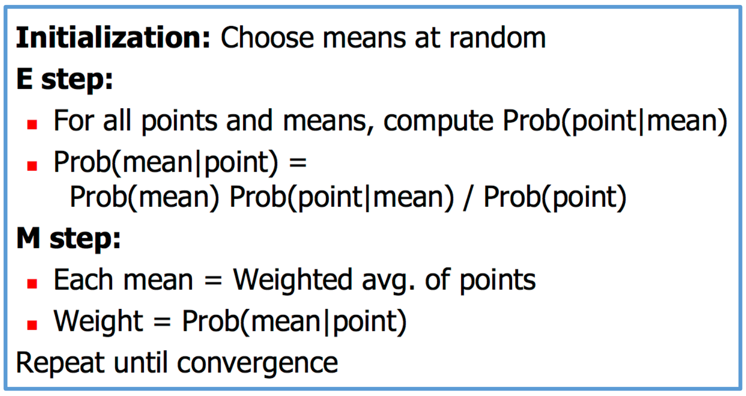

initial guess
E step
Estimate parameters (probabilities) given current guess for hidden structure
Estimate hidden structure (cluster membership) given current probability guesses and observed data
M step

Missing information: Cluster membership
Idea: hidden variables for cluster membership
Each instance x_i has a set of hidden variables zi1,...,zik
Intuition: z_{ij} = 1 if i belongs to cluster j and 0 otherwise



In k-means, instances are assigned to exactly one cluster. EM clustering is a "soft" k-means where ach example has the probability of belong to a cluster.
Guess the initial parameters for model in each cluster and then iterate until convergence.

- **E step:** Determine how likely it is that each cluster generated each instance
- **M step:** Adjust cluster parameters to maximize likelihood.



Initialization: Choose means at random
E step:
- For all points and means, compute Prob(point|mean)
- Prob(mean|point) =
  Prob(mean) Prob(point|mean) / Prob(point)

M step:
- Each mean = Weighted avg. of points
- Weight = Prob(mean|point)

Repeat until convergence

E-Step

Recall that z_{ij} is a hidden variable which is 1 if N_j generated x_i and 0 otherwise
In the E-step, we compute h_{ij}, the expected value of this hidden variable.

$$h_{ij} = \frac{P_j * N_j(x_i)}{\Sigma_l P_l * N_l(x_i)}$$

M-Step

Given the expected values h_{ij}, we re-estimate the means of the Gaussians and the cluster probabilities.

$$\mu_j = \frac{\Sigma_i x_i * h_{ij}}{\Sigma_i h_{ij}}$$

$$P_j = \frac{\Sigma_i h_{ij}}{N}$$

Note : "i" goes over examples

EM Clustering … To sum up
- Will converge to a local maximum

- Sensitive to initial means of clusters
- Have to choose the number of clusters in advance

Evaluating cluster results
Given random data without any "structure", clustering algorithms will still return clusters.
The gold standard: do clusters correspond to natural categories?
Do clusters correspond to categories we care about? (There are lots of ways to partition the world)

Approaches to cluster evaluation

- **External validation**. Ex: do genes clustered together have some common function?
- **Internal validation**: how well does clustering optimize intra-cluster similarity and inter-cluster dissimilarity?
- **Relative validation:** how does it compare to other clusterings? Ex: With a probabilistic method (such as EM) we can ask: how probable does held-aside data look as we vary the number of clusters?

5. Applications

5.1 Low Quality of Web Searches

- System perspective
  - Small coverage of Web (<16%)
  - Dead links and out of date pages
  - Limited resources
- IR perspective (relevancy of doc ~ similarity to query)
  - Very short queries
  - Huge database
  - Novice users

5.2 Document clustering

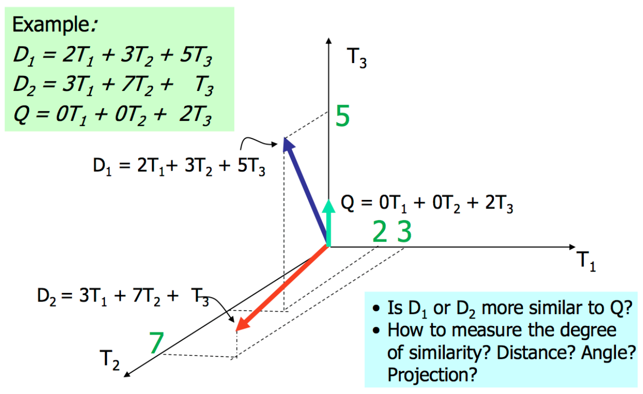
- User receives many (200-500) documents from web search engine
- Group documents in clusters by topic
- Present clusters as interface

⇨ We need a way to compare queries and documents

- Vector space model
  - How to determine important words in a document?
  - How to determine the degree of importance of a term within a document and within the entire collection?
  - How to determine the degree of similarity between a document and the query?

- In the case of the web, what is a collection and what are the effects of links, formatting information, etc.?

Assume t distinct terms remain after pre-processing: vocabulary
These "orthogonal" terms form a vector space Dimension = t = |vocabulary|
Each term, i, in a document or query, j, is given a real-valued weight, w_{ij}.
Both documents and queries are expressed as t-dimensional vectors:
$d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$


Example:
$D_1 = 2T_1 + 3T_2 + 5T_3$
$D_2 = 3T_1 + 7T_2 + T_3$
$Q = 0T_1 + 0T_2 + 2T_3$
T_3
5
$D_1 = 2T_1 + 3T_2 + 5T_3$
$Q = 0T_1 + 0T_2 + 2T_3$
2 3
T_1
$D_2 = 3T_1 + 7T_2 + T_3$
7
T_2
• Is D_1 or D_2 more similar to Q?
• How to measure the degree of similarity? Distance? Angle? Projection?

Vector space model represents a collection of n documents by a term-document matrix.
Each entry: "Weight" of a term in the document

$$\begin{array}{c|cccc} & T_1 & T_2 & \ldots & T_t \\ D_1 & w_{11} & w_{21} & \ldots & w_{t1} \\ D_2 & w_{12} & w_{22} & \ldots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \ldots & w_{tn} \end{array}$$

More frequent terms in a document are more important, i.e. mmore indicative of the topic
F_{ij} = frequency of term I in the document j
We may want to normalize the term frequency (tf) by dividing by the frequency of the most common term in the document: tf_{ij} = fij / maxi{f_{ij}}

Terms that appear in many different documents are less indicative of overall topic.
df $_i$ = document frequency of term I
= number of documents containing term i

idf_i = inverse document frequency of term i,
= log2 (N/ df i) (N: number of documents)
An indication of a term's discrimination power
⇨ Log used to dampen the effect relative to tf

A typical combined term importance indicator is tf-idf weighting:

wij = tfij idfi = tfij log2 (N/ dfi)

A term occurring frequently in the document but rarely in the rest of the collection is given high weight. Many other ways of determining term weights have been proposed. Experimentally, tf-idf works well

Example:

Given a document containing terms with given frequencies: A(3), B(2), C(1)
Assume collection contains 10,000 documents and document frequencies of these terms are: A(50), B(1300), C(250)
Then:
A: tf = 3/3; idf = log2(10000/50) = 7.6; tf-idf = 7.6
B: tf = 2/3; idf = log2 (10000/1300) = 2.9; tf-idf = 2.0
C: tf = 1/3; idf = log2 (10000/250) = 5.3; tf-idf = 1.8

A query vector is typically treated as a document and also tf-idf weighted. An alternative is for the user to supply weights for the given query terms.

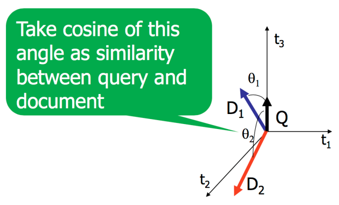
- **Inner product**

  $sim(d_j,q) = \Sigma\, w_{ij} * w_{iq}$

  - W_{ij} is weight of term I in doc j
  - W_{iq} is weight of term I in query
- Cosine similarity

  $$sim(d_j,q) = \frac{\Sigma_i\, w_{ij} * w_{iq}}{\Sigma_i\, (w_{ij})^2\, \Sigma_i\, (w_{iq})^2}$$

  - Measures the cosine of the angel between two vectors
  - Inner product normalized by the vector lengths


Take cosine of this angle as similarity between query and document
t_3
θ_1
D_1
Q
θ_2
t_1
t_2
D_2

Comparison

D1 = 2T1 + 3T2 + 5T3
D2 = 3T1 + 7T2 + 1T3
Q = 0T1 + 0T2 + 2T3

Weighted inner product
sim(D1 , Q) = 2*0 + 3*0 + 5*2 = 10

sim(D2 , Q) = 3*0 + 7*0 + 1*2 = 2

Cosine
sim(D1 , Q) = 10 / √ (4+9+25)(0+0+4) = 0.81
sim(D2 , Q) = 2 / √ (9+49+1)(0+0+4) = 0.13

⇨ D1 is 6 times better than D2 using cosine similarity but only 5 times better using inner product.

Comments on Vector Space Model

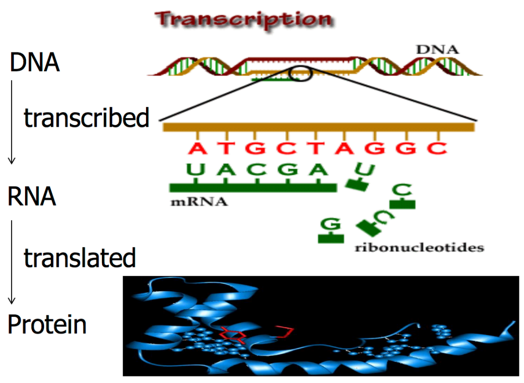
- Simple, mathematically based approach
- Considers both local (tf) and global (idf) word
- occurrence frequencies
- Provides partial matching and ranked results.
- Tends to work quite well in practice despite obvious weaknesses
- Allows efficient implementation for large document collections

Weakness with Vector Space Model
- Missing semantic information. Ex: word sense
- Missing syntactic information (Ex: phrase structure, word order, proximity information)
- Assumption of term independence (Ex: ignores synonymy)

5.3 Bioinformatics

Molecular biology has become a data-rich science. The Nucleic Acids Research Molecular Biology Database Collection indexes 1330 data sources. This represents lots and lots of data, but there is a lack of understanding of biological systems. Computational methods are crucial for understanding available data.

Genes



Genes are the basic units of heredity A gene is a sequence of DNA bases that carries the information required for constructing a particular protein. Such a gene is said to encode a protein. The human genome comprises ~ 25,000 protein coding genes

Types of data
- Full sequences of a genomes
- Single nucleotide polymorphisms

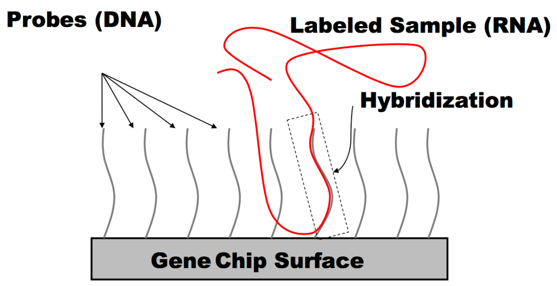
- Gene expression data
- Many more!

Microarrays allow us to measure gene expression. Microarray is a solid support, on which pieces of DNA are arranged in a grid-like array. It measures RNA abundances by exploiting complementary hybridization.

How active are various genes in different cell/ tissue types?
How does the activity level of various genes change under different conditions?

- Stages of a cell cycle
- Environmental conditions
- Disease states
- Knockout experiments

What genes seem to be regulated together?



Data points are genes

- Represented by expression levels across different samples (i.e., features=samples)
- Goal: categorize new genes

Data points are samples (e.g., patients)

- Represented by expression levels of different genes (i.e., features=genes)
- Goal: categorize new samples

Unsupervised Learning Task (1)

Given: a set of microarray experiments under different conditions
Do: cluster the genes, where a gen described by its expression levels in different experiments

Unsupervised Learning Task (2)

Given: a set of microarray experiments (samples) corresponding to different conditions or patients
Do: cluster the experiments

Ex :

- Cluster samples from mice subjected to a variety of toxic compounds (Thomas et al., 2001)
- Cluster samples from cancer patients, potentially to discover different subtypes of a cancer
- Cluster samples taken at different time points

CHAPTER 9: USING UNLABELED DATA

1. Introduction

When you do something like classification you have labelled data. The assumption is that someone is telling you what is labelled as positive or as negative.

Where does labelled data come from? It comes from some tasks; people are willing to label.

- Netflix, Amazon, etc.
- Spam
- Medical diagnoses

Often we have to get people to label data

- Web ranking
- Document classification

Problem: labelling data is expensive because you have to pay someone to do it and you need a lot of data. Next, it is also time-consuming, thinking about what the label should be, etc.

Learning methods need labelled data.

- Lots of <x, f(x)> pairs
- Hard to get … who wants to label data

But unlabelled data is usually plentiful … could we use this instead??

- Semi-supervised learning
- Active learning

Problem set-up

Motivation: Hard to get labelled data

Given: we assume we have some labled data and some unlabelled data.

- Labeled data: D= [<x, f(x)>]
- Unlabeled data: U = [<x, ??>]

⇨ Learn a hypotheses h that approximates f.

2. Semi-supervised learning

Self-training: use an ensemble technique such as bagging to build a set of classifiers on labelled data. Classify each unlabelled training example with each classifier. Make a prediction on each unlabelled examples. If all classifiers agree on an example's label, add it to the training set with its predicted label. Next, iterate. For the first iteration you have little training data.

Another way is called **co-training.**

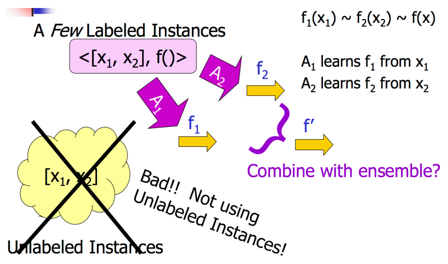
Idea: you have a little labelled data and lots of unlabelled data. And each instance has two parts:

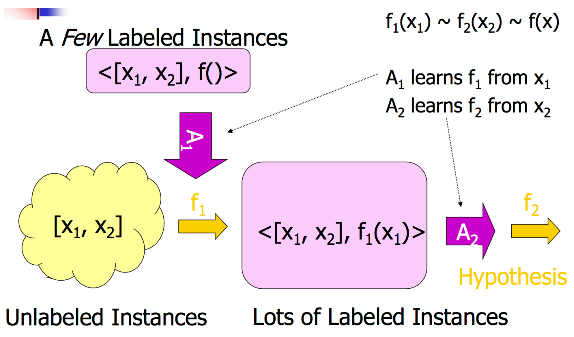
- x = [x1, x2]
- x1, x2 conditionally independent given f(x)

Each half can be used to classify instance
∃f1, f2 such that f1(x1) ~ f2(x2) ~ f(x)

Both f1, f2 are learnable
f1 ∈ H1, f2 ∈ H2, ∃ learning algorithms A1, A2

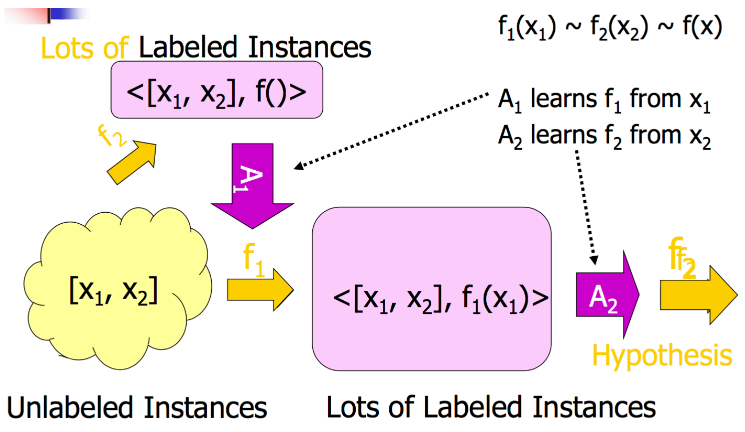
Ex: you can think of classifying document on web pages, having two features on the web page. One: words on the webpage. Second: words appearing on any internet page.





Can apply A_1 to generate as much training data as one wants
If x_1 is conditionally independent of x_2 / f(x), then the error in the labels produced by A1 will look like random noise to A2 !!!

Thus **no limit** to quality of the hypothesis A2 can make



Learning to classify web pages as course pages.
Two features:

- x1 = bag of words on a page
- x2 = bag of words from all anchors pointing to a page

Naïve Bayes classifiers

- 12 labeled pages
- 1039 unlabeled pages

	Page-based classifier	Hyperlink-based classifier	Combined classifier
Supervised training	12.9	12.4	11.1
Co-training	6.2	11.6	5.0

One thing that is surprising is that accuracy and error rate highly differ. With co-training, the error rate is halfed.

3. Active learning

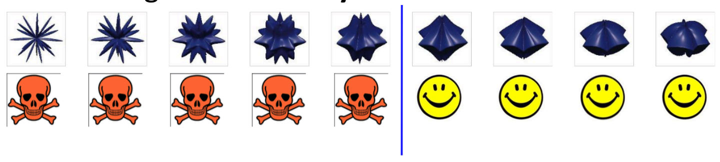
3.1 The concept

Active learning is a subfield of machine learning and, more generally, artificial intelligence. The key hypothesis is that if the learning algorithm is allowed to choose the data from which it learns—to be “curious,” if you will—it will perform better with less training. Why is this a desirable property for learning algorithms to have? Consider that, for any supervised learning system to perform well, it must often be trained on hundreds (even thousands) of labelled instances.

Suppose you are the leader of an Earth convoy sent to colonize planet Mars.

People who ate the round Martian fruits found them *tasty!*
People who ate the spiked Martian fruits ***died!***

Problem: there’s a range of spiky-to-round fruit shapes on Mars. You need to learn the “threshold” of roundness where the fruits go from poisonous to safe. And … you need to determine this risking as few colonists’ lives as possible.
A way to think about doing this is a **binary search**. You need to test all of the instances.



This is just a **binary search**, so…

Under the PAC model, assume we need $O(1/\varepsilon)$ i.i.d. instances to train a classifier with error ε.

Using the binary search approach, we only needed $O(\log_2 1/\varepsilon)$ instances!

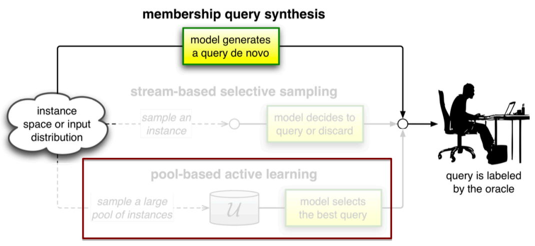
Key idea: the learner can choose training data. You throw of some labelled example and choose which labels you want to preserve.

- On Mars: whether a fruit was poisonous/safe
- In general: the true label of some instance

Goal: reduce the training costs
- On Mars: the number of "lives at risk"
- In general: the number of "queries"

Active learning systems attempt to overcome the labeling bottleneck by asking queries in the form of unlabeled instances to be labeled by an oracle (e.g., a human annotator). In this way, the active learner aims to achieve high accuracy using as few labeled instances as possible, thereby minimizing the cost of obtaining labeled data. Active learning is well-motivated in many modern machine learning problems where data may be abundant but labels are scarce or expensive to obtain.

How do we do this?

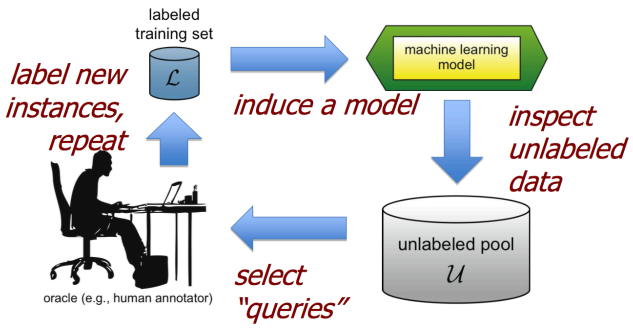

membership query synthesis
model generates a query de novo
stream-based selective sampling
instance space or input distribution
sample an instance
model decides to query or discard
pool-based active learning
sample a large pool of instances
$\mathcal{U}$
model selects the best query
query is labeled by the oracle

Most common in NLP applications

You can think about generating examples and asking for them. Another approach you can think about is data streaming.

Pool-Based Active Learning Cycle

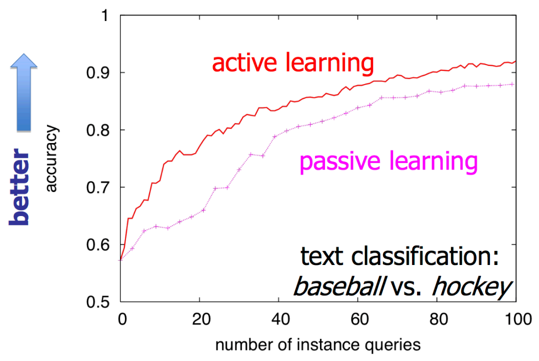
There are several scenarios in which active learners may pose queries, and there are also several different query strategies that have been used to decide which instances are most informative. In this section, two illustrative examples in the pool-based active learning setting (in which queries are selected from a large pool of unlabeled instances U) are presented using an uncertainty sampling query strategy (which selects the instance in the pool about which the model is least certain how to label).



This figure illustrates the pool-based active learning cycle. A learner may begin with a small number of instances in the labeled training set L, request labels for one or more carefully selected instances, learn from the query results, and then leverage its new knowledge to choose which instances to query next. Once a query has been made, there are usually no additional assumptions on the part of the learning algorithm. The new labeled instance is simply added to the labeled set L, and the learner proceeds from there in a standard supervised way.

⇨ I have a small label set, I learn a model. We use that model to make predictions on unlabelled data. In active learning we ask a question to someone about what the true label is.

You show a plot, a number of queries on the x axes and accuracy. Usually you get such a curve.
With active learning the algorithm decides which examples it takes. And usually for the same number of instances you have a better accuracy than passive learning. This is used a lot.



IBM — Sentiment analysis for blogs; Noisy relabeling – *Prem Melville*

SIEMENS — Biomedical NLP & IR; Computer-aided diagnosis – *Balaji Krishnapuram*

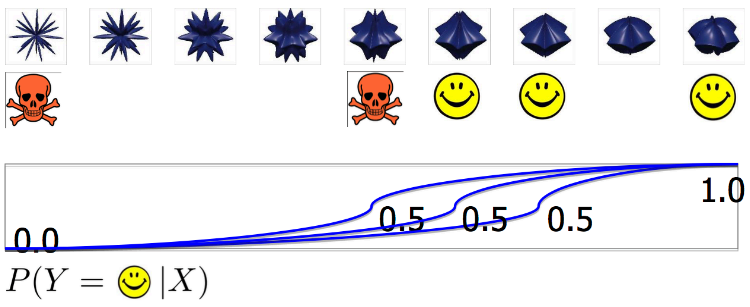
Microsoft — MS Outlook voicemail plug-in [Kapoor et al., IJCAI'07]; "A variety of prototypes that are in use throughout the company." – *Eric Horvitz*

Google — "While I can confirm that we're using active learning in earnest on many problem areas… I really can't provide any more details than that. Sorry to be so opaque!" – *David Cohn*

3.2 How to select queries?

We assume that we received a small sample of unlabeled examples and a set of labeled examples. The goal is to reduce the training cost, that is to say, the cost of getting labels.
Key question: which examples do we want to use?

⇨ Let's try generalizing our binary search method using a probabilistic classifier. We apply something like logistic regression.

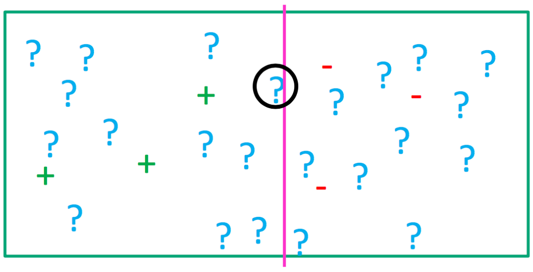


Query the examples the learner is most uncertain about.

- Closest to 0,5 probability
- Closest to decision surface

Look at examples close to the decision boundary since these cause the boundary to move.

If we have possible and negative examples, look at examples that are closest to the line. We take one, and we ask someone to give it a label. Here, it is given a negative label. We retrain the classifier, giving a new decision boundary. We again pick an example close to the boundary, give it a label, and we do this again and again, etc.

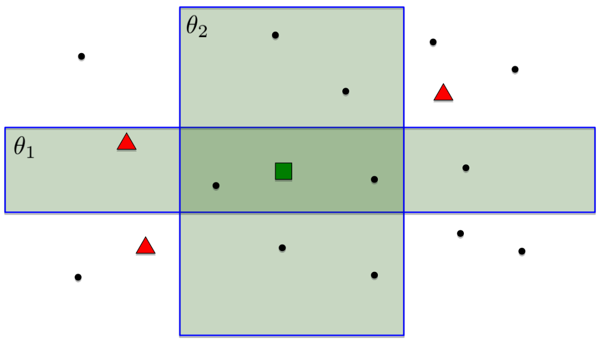


The intuition here is that we should first observe labels of points close to the decision boundary.

3.3 Query-By-Committee (CBC)

Train a committee C = {q1, q2, ..., qC} of classifiers on the labeled data in L.
Query instances in U for which the committee is in most disagreement. Try to find models that are consistend with the training data, reducing the model version space.

Key idea: reduce the model version space. It expedites search for a model during training



We have 4 labeled instances: 3 triangles and a square. Train a model such that everything inside rectangles is of class squares and everything outside is of class triangle.

How to build a committee:

- "Sample" models from $P(q|L)$ [Dagan & Engelson, ICML'95; McCallum & Nigam, ICML'98]
- Standard ensembles (e.g., bagging, boosting) [Abe & Mamitsuka, ICML'98]

How to measure disagreement

- « XOR » committee classifications
- View vote distribution as probabilities, use uncertainty measures (e.g., entropy)

3.4 Alternative Query Types

So far, we assumed queries are instances. <u>Ex</u>: for document classification the learner queries documents.

Can the learner do better by asking different types of questions? Questions not about instances but about different features for instance.

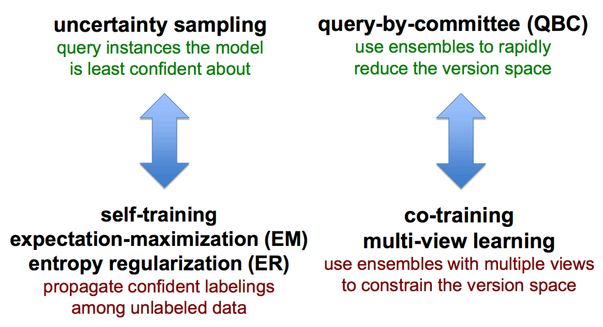
- Multiple-instance active learning
- Feature active learning: try to get feedback about which features are more useful than others. You can then get better results.
  In NLP tasks, we can often intuitively label features.
  - The feature word "puck" indicates the class hockey
  - The feature word "strike" indicates the class baseball.

Tandem learning exploits this by asking both instance-label and feature-relevance queries. <u>Ex</u>: "is puck an important discriminative feature?"

4. Summary

Both try to attack the same problem: making the most of unlabeled data U. Getting labels is expensive. Each attacks from a different direction:

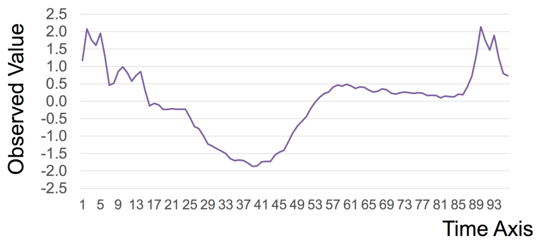
- **Semi-supervised** learning **exploits** what the model thinks it knows about unlabeled data. Propagates information about what it knows. Propagate information from label data to unlabeled data.
- **Active** learning **explores** the unknown aspects of the unlabeled data. Look at what is uncertain.


uncertainty sampling
query instances the model is least confident about
query-by-committee (QBC)
use ensembles to rapidly reduce the version space
self-training
expectation-maximization (EM)
entropy regularization (ER)
propagate confident labelings among unlabeled data
co-training
multi-view learning
use ensembles with multiple views to constrain the version space

CHAPTER 10: TIME SERIES

1. Time Series Intro

Time series are a collection of measurements made sequentially in time. Usually people only want to measure one variable when looking at time series.



Time series faces different challenges

- Lots and lots of data: videos for instance.
- Autocorrelation: current value depends on previously observed values.
- Data is messy: you usually collect time series from sensors.
  - Different sampling rates and ranges (e.g., clipping in accelerometer data)
  - Noise
  - Missing values (e.g., sensor drops)
  - Etc.

Standard Pre-processing: Z-Normalization

Usually when working with collected data from sensors you want to do some sort of pre-processing.

Given: S = (s1,...,sn)

Let each $s_i = \frac{s_i - \mu_S}{\sigma_S}$

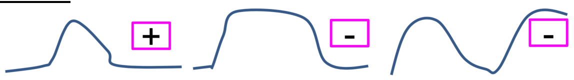
μ_S is the average value of the variable in S and performs offset scaling

σ_S is standard deviation of the variable in S and performs amplitude scaling

2. Time Series Classification

One of the obvious tasks you can do with time series is classification.

Given: a set of time series with known labels.



Goal: make prediction about future time series. You are given a new time series measurement and you want to predict the class of this instance.

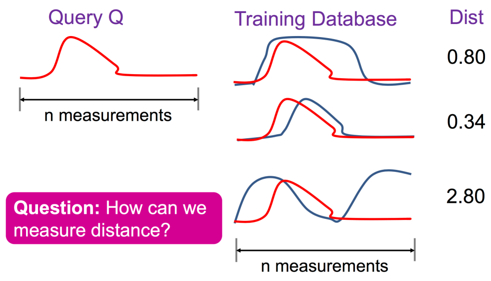
Assumption: labels are assigned to entire sequence. We do not necessarily know what value/set of values are responsible for the label. I don't know which measurement cause the time series to be labelled positive or negative.

Idea 1: Define features
The first obvious approach you can do is to define features. So, given a time series, convert it to a feature vector representation. There are several ways to do so:

- Use statistics: Min, max, etc.
- Transformations: discrete wavelet transform, etc.
- Entropy
- …

⇨ Apply standard supervised learners to data to make a prediction.

Idea 2: Nearest Neighbours



Compare three-time series to my training set and measure distance similarity between them. Just predict the label based on the smallest distance.

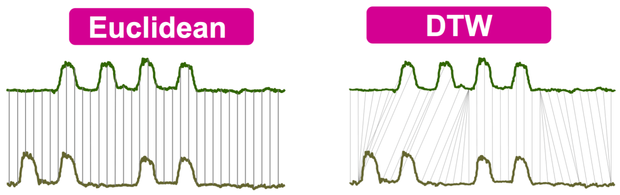
⇨ How can we measure the distance between time series?

- **Euclidean distance**
  Given: Two time series of equal length
  $Q = (q_1, q_2, \dots q_n)$
  $S = (s_1, \dots s_n)$
  Take the Euclidean distance by comparing each measurement in each time series to the ones of the other time series.

$$D(Q,S) = \sqrt{\sum_{i=1}^{n} (q_i - s_i)^2}$$

⇨ Can we capture this alignment? One standard approach for doing this is called **Dynamic Time Warping (DTW)**
Basic idea: Allow one-to-many mapping. Each measurement in time series q gets exactly one measurement in time series s. It allows us to have some non-linearity in the data.
Constraints on mapping:

- Must match the beginning and end of sequence
- Monotonicity: cannot go backwards in time
- Continuity: no gaps.
- (optional) Warping window: w: |i-j| ≤ w

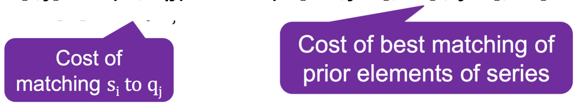


Compute DTW with Dynamic programming. There is no restriction that both sequences should have the same length. Again we have the same problem set-up, having two time series S and Q.

Given: S = (S_1, … S_n) and Q = (q_1, …, q_m)

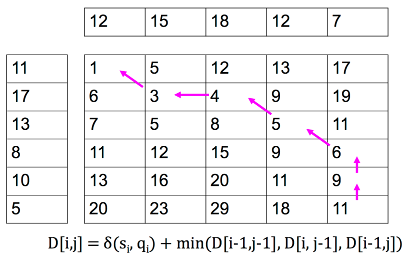
Do:

- Let D = n x m matrix
- Let δ(si, qj) be the distance between si and qj
- D[i,j] = δ(si, qj) + min(D[i-1,j-1], D[i, j-1], D[i-1,j])


Cost of matching s_i to q_j
Cost of best matching of prior elements of series

If w is given, then only compute D[i,j] if |i-j| ≤ w

⇨ An Example is given in the slides.

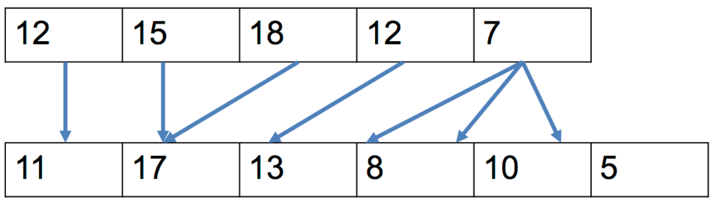


	12	15	18	12	7
11	1	5	12	13	17
17	6	3	4	9	19
13	7	5	8	5	11
8	11	12	15	9	6
10	13	16	20	11	9
5	20	23	29	18	11

D[i,j] = δ(s_i, q_j) + min(D[i-1,j-1], D[i, j-1], D[i-1,j])

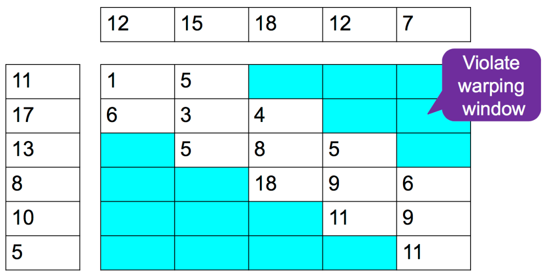
Trace back. We want to start from the right corner. Trace back. We know that 5 will be matched to 7, so 10 can me matched to 7 and 8 will be matched to 7. 13 is matched to 12. Etc.

This allows a mapping between both sequences.



This is a good way to measure the distance between both sequences.

If we have a working window of 1, we get the following matrix. Blue entries are the ones violating the constraints.


Violate warping window

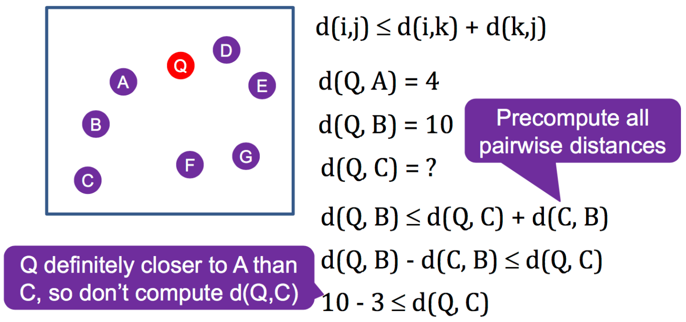
	12	15	18	12	7
11	1	5	🟦	🟦	🟦
17	6	3	4	🟦	🟦
13	🟦	5	8	5	🟦
8	🟦	🟦	18	9	6
10	🟦	🟦	🟦	11	9
5	🟦	🟦	🟦	🟦	11

$$D[i,j] = \delta(s_i, q_j) + \min(D[i-1,j-1], D[i, j-1], [i-1,j])$$

One thing to know is that DTW is not a Metric … Because

- DTW violates the triangle inequality
- Metrics are usually preferred as many efficiency tricks exploit the triangle inequality
  - However, many "tricks" exist for DTW
  - Can compute it very quickly
- Also, most of the time DTW acts like a metric. <u>Ex</u>: > 99% of randomly sampled triples will satisfy the triangle inequality for DTW.

Exploiting Triangle Inequality


Q
D
A
E
B
F
G
C
Precompute all pairwise distances
Q definitely closer to A than C, so don't compute d(Q,C)

$$d(i,j) \le d(i,k) + d(k,j)$$

$$d(Q, A) = 4$$

$$d(Q, B) = 10$$

$$d(Q, C) = ?$$

$$d(Q, B) \le d(Q, C) + d(C, B)$$

$$d(Q, B) - d(C, B) \le d(Q, C)$$

$$10 - 3 \le d(Q, C)$$

Setting the Warping Constraint

The Warping constraint has a big influence on results and "best" value is problem dependent.

- Set via a tuning set
- Usually small values (e.g., < 20) are fine
- As training set size gets larger, expect that less warping is needed. Why? So, the more example I have, I will expect to have to do less warping.

Implementing DTW

- Dynamic programming DTW: N^2 time and space per pair compared
- Many efficiency tricks exist that scale this to very large data sets (i.e, linear time)
  - Early abandoning of computation
  - Exploiting wrapping constraint

- Etc.

So far, we assume time series measures only one variable. However, multiple series measure multiple variables: S = (s1,...,sn) where each si = (si,1,...,si,d). The question is, how can we deal with multiple dimensional variables?
There exist two ways to extend DTW.

- **Independent**: sum each dimension separately. Align first dimensions of two sequences, then second dimension, third one, etc. So we essentially assume dimensions to be independent. DTW(S,Q) = DTW(S1,Q1)+...+DTW(Sd,Qd)
- **Dependent**: single matrix and compute score along all dimensions. e.g., δ(si, qi) is d dimensional Euclidean distance.

Selecting independent or dependent metric is domain dependent:

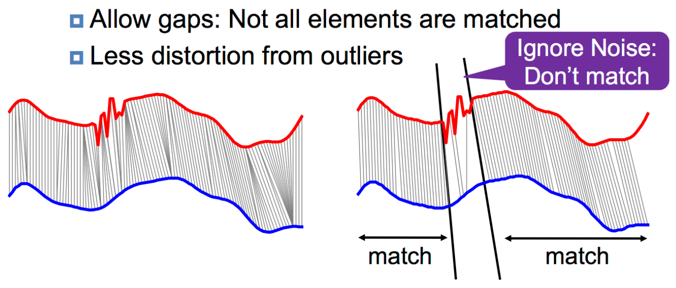
- **Independent**: Acceleration on foot and hand
- **Dependent**: Player location on soccer field

Beware of curse of dimensionality!

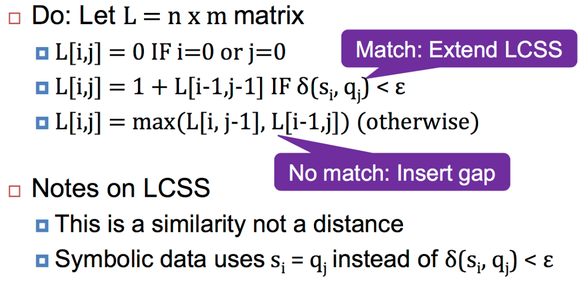
- DTW probably only useful if d < 10 (or perhaps even smaller)
- If d > 10, probably need to drop some dimensions.

Allowing for Gaps: Longest Common Subsequence (LCSS)

Potential issues with DTW: all points are matched so outliers can distort score.
Longest common subsequence which allows gaps, meaning that not all points are matched. We will ignore some noise in the data.


- Allow gaps: Not all elements are matched
- Less distortion from outliers

Computing LCSS: Dynamic Programming
Given: S = (s1,...,sn) and Q= (q1,...,qm)


- Do: Let L = n x m matrix
  - L[i,j] = 0 IF i=0 or j=0
  - L[i,j] = 1 + L[i-1,j-1] IF $\delta(s_i, q_j) < \varepsilon$
  - L[i,j] = max(L[i, j-1], L[i-1,j]) (otherwise)
- Notes on LCSS
  - This is a similarity not a distance
  - Symbolic data uses $s_i = q_j$ instead of $\delta(s_i, q_j) < \varepsilon$

Match: Extend LCSS
No match: Insert gap

Notes on LCSS

- This is a similarity not a distance

- Symbolic data uses si = qj instead of δ(si, qj) < ε

Why is it interesting to study something like DTW?

- DTW is common: Medicine, bioinformatics, gesture recognition, music analysis, etc.
- Many large-scale empirical evaluations show that nearest neighbour with DTW is hard to beat
  - DTW is very simple and easy to implement
  - Good advice: Try simple thing first
- Ideas such as dynamic programming, etc. translate to other problems.

3. Symbolic Representations

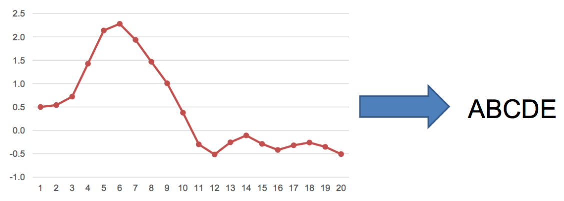
⇨ Why a symbolic representation?

- Enables new analysis by using symbolic approaches: plot them for instance.
- Symbols tend to be easier for people to interpret than numbers
- Compress the data

SAX representation

Given: A time series S = (s1,...,sn) , an window size w, and an alphabet size a

Do: Convert S to a symbolic sequence of length n/w which involves an alphabet of a symbols. So we will divide or sequences into parts.



How does this approach work? First we assume the data is normalized, then we apply the Piece aggregate approximation.

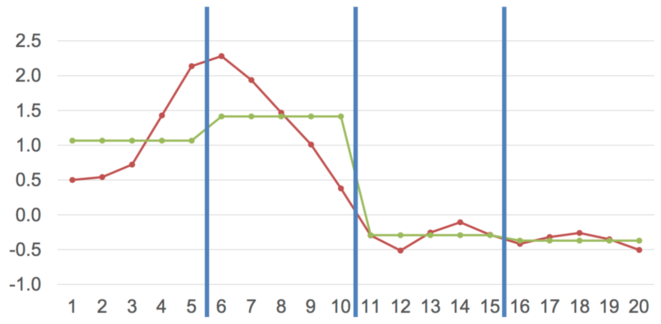
⇨ Z-normalize the time series

Apply the piece aggregate approximation (PAA)

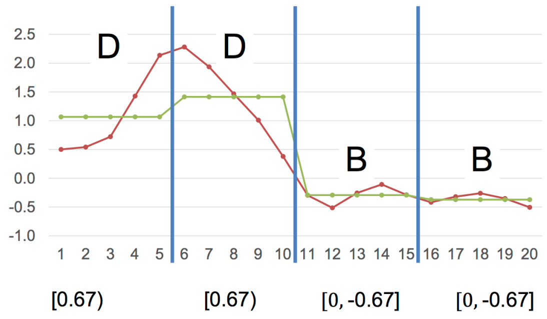
- Divide the series into n/w windows
- Compute the average value in each window and replace each observed value in the window with the average

Assign a symbol based on ranges of PAA values

Convert PAA representation to a string



- Exploit that the time series is z normalized
- Divide the area under the standard normal into equal size regions
- Assign one symbol for each region



Anything that falls into a particular gets the corresponding symbol.

⇨ Why SAX?

- Converted time series to a string: Can apply fancy algorithms (e.g., from bioinformatics)
- Makes visualization easier (borrowing ideas from bioinformatics)
- Most indexing schemes work best with symbols: Can use extensible hashing with SAX

4. Applications to sports

Three new types of Data

- **Event stream:** Events with time and location
- **Athlete monitoring**: GPS, accelerometer, etc.
- **Optical tracking**: X, Y locations of players

⇨ These are all time series

Two tasks

- **Rating players:** Assign a rating to each action a player performs in a match
- **Understands strategy:** Discover patterns form player tracking data

STARSS

Given: Event stream with type and location of all events (e.g., passes and shots)
Do: Assign rating to each action by assigning a value to each action.
Approach

1. Split matches in phases
2. Rate phases
3. Distribute phase rating over individual actions
4. Aggregate players' ratings over season

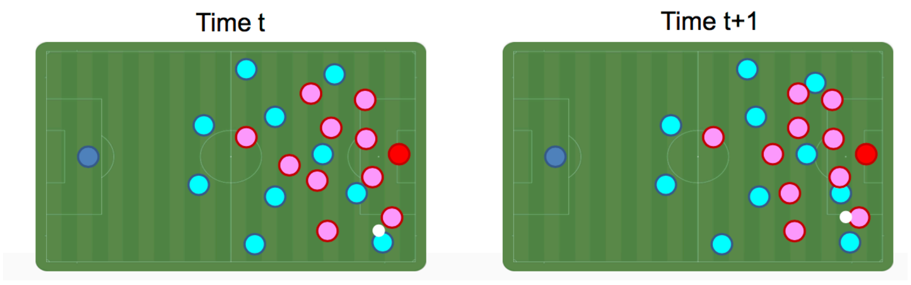
Rating phases:
1. Find k most similar phases (e.g., 100)
2. Of these, count how many result in a goal (e.g., 6)

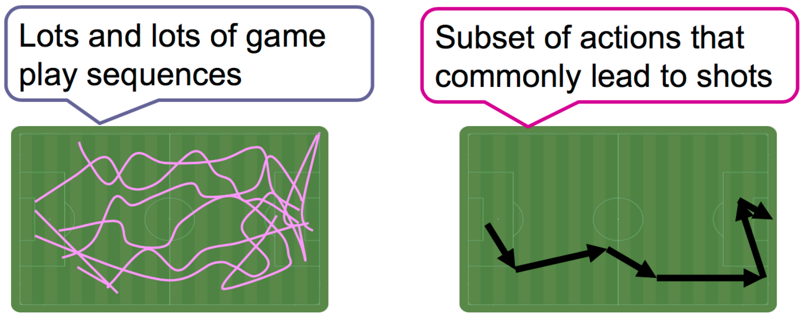
$$Rating(phase) = \frac{6\ goals}{100\ similar\ phases} = 0.06$$

⇨ Distribute phase rating across its constituent actions. Actions at the end are more important: Exponential Decay.

Discover Offensive Strategies in Football Matches

Given: Event stream with type and location of all events (e.g., passes and shots)
Locations of all players and the ball (10 hz sample)
Find: Typical offensive strategies




Lots and lots of game play sequences
Subset of actions that commonly lead to shots

- Film study is time consuming
- Automation can help speed this up
- Computers good at finding patterns in large data sets

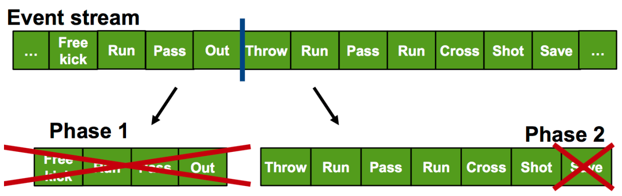
Challenges:

- Relationships and how they change over time are important
  - Space
  - Interactions between players
- Order of events is important
- May want to generalize over players involved
- Exact same sequence of events unlikely to occur multiple times

Important Steps

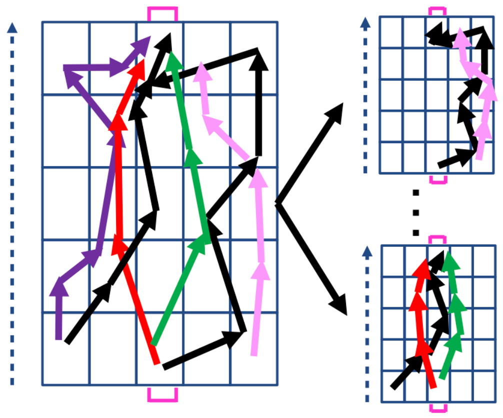
1. Data cleaning
    - Outliers and incorrect values
        - Valid field coordinates
        - Player and ball movements seem "possible"
    - Teams switch direction at half time: Normalize data such tat team always attacks the same goal.
    - Account for changes in data (e.g., position switches, new players, etc.

2. Event stream pre-processing



3. Clustering data
   Three benefits
    - Teams employ multiple strategies
    - Generalize from a specific location
    - Subsequent step more computationally efficient

   Divide phases into different groups such that the phases in a group are "similar".



4. Identifying important strategies: within each cluster, find frequently occurring subsequences. See slides for more details.

5. To conclude

To conclude, time series occur in multiple domains: sports, healthcare, music, mathematics, etc. Sports provides many rich sources of data form historical sources, sensors provide detailed data. This relatively simple analysis has made a huge impact on several sports.

1-NN approach with DTW is a very strong approach for classifying time series.

- DTW is one-to-many mapping
- Can be efficiently computed

Can convert time series into a symbolic sequence.

Key challenges include

- Volume of data
- Capturing structure: Spatial, temporal, etc.