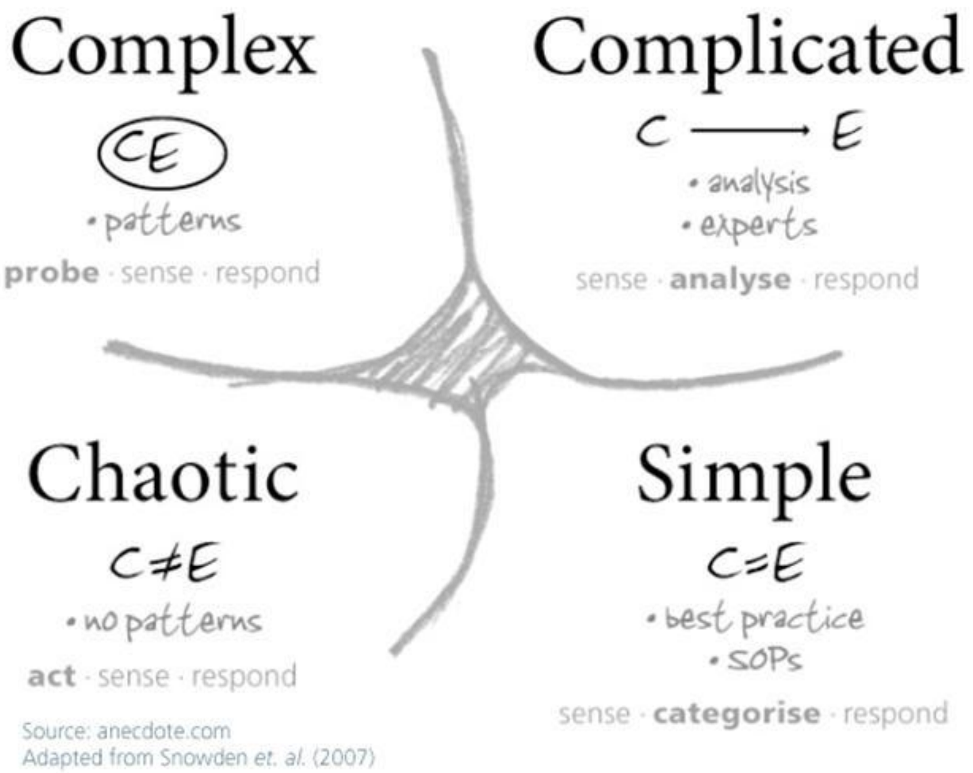
Knowledge Technologies Notes

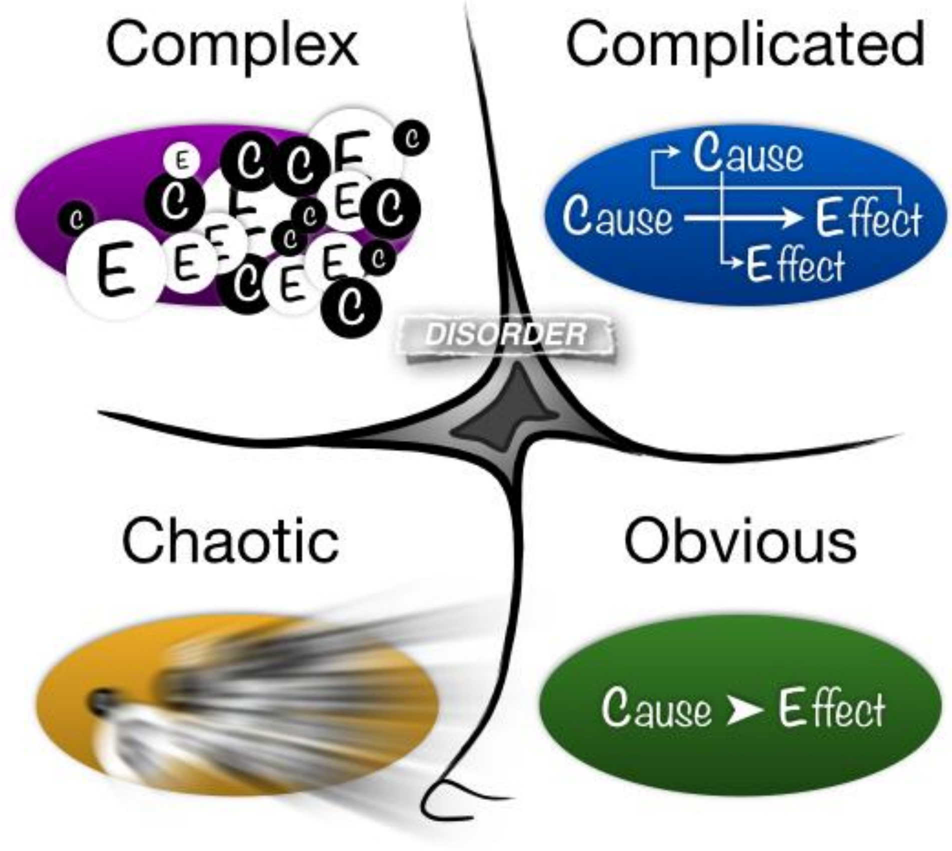
1. Introduction to Knowledge Technologies

Types of Problems (Cynefin Model)

- Simple: the relationship between cause and effect is clear, the approach is to Sense - Categorise - Respond and we can apply best practice. These problems are deterministic in the sense of solving
- Complicated: the relationship between cause and effect requires analysis or some other form of investigation and/or the application of expert knowledge, the approach is to Sense - Analyze - Respond and we can apply good practice. These problems are solved using Knowledge Technologies
- Complex: the relationship between cause and effect can only be perceived in retrospect, but not in advance, the approach is to Probe - Sense - Respond and we can sense emergent practice and use it to learn from experience
- Chaotic: there is no obvious relationship between cause and effect at systems level, the approach is to Act - Sense -Respond and we can discover novel practice
- Disorder: the state of not knowing what type of causality exists, in which state people will revert to their own comfort zone making a decision. We may not learn much from this experience



Source: anecdote.com
Adapted from Snowden *et. al.* (2007)



Cynefin Examples

- *My car key is simple*
  It took me about three seconds to understand how my car key works. OK, maybe that's not quite correct. Mine has a battery in it. If I take it apart it might take me another three hours to understand its details. But yeah, I'm smart, I'll manage.
- *My car is complicated*
  It would take me years to understand how my car works. And I don't intend to. But if I did, then some day in the far future I would know with certainty the purpose of each mechanism and each electrical circuit. I would fully understand how to control it, and I would be able to take my car apart and reassemble it, driving it exactly as I did before. In theory, of course. In practice, only real men do things like that.
- *Car traffic is complex*
  I can travel up and down the same street for twenty years, and things would be different every time. There is no way to fully understand and know what happens around me on the road when I drive, how other drivers operate their vehicles, and how the people in the streets interact. I can make guesses, and I can gain experience in predicting outcomes. But I will never know for sure.
- *Car traffic in Lagos (Nigeria) is chaotic*
  When things get too complex, they easily become chaotic. Traffic in Lagos is so bad, it is not even predictable. Poor infrastructure and planning, heaps of waste, pollution, lack of security, floods, and many more problems make it one of the worst places in the world to be, as a simple car driver.

Examples of Concrete Tasks

- The task is well-defined, we can assess whether the solution is correct. The data is transformed in a mechanical way or leads to a mechanical action
- Compute a missile trajectory
- Crack a code (decryption without the decryption key)
- Do accountancy over financial data
- Operate a camera (focus, exposure), store the image
- Guide a cutting tool, operate an assembly line
- Map mouse movements to cursor movements

Examples of Knowledge Tasks

- The data is irregular or unreliable, or the outcome is not well-defined. Context is critical
- Compression of an image
- List of answers to a typical web query
- Translation between languages
- Identification of what a health treatment
- Finding an “optimal” route between two locations
- Deciding what movie to watch

Thinking Tools for Knowledge Technologies

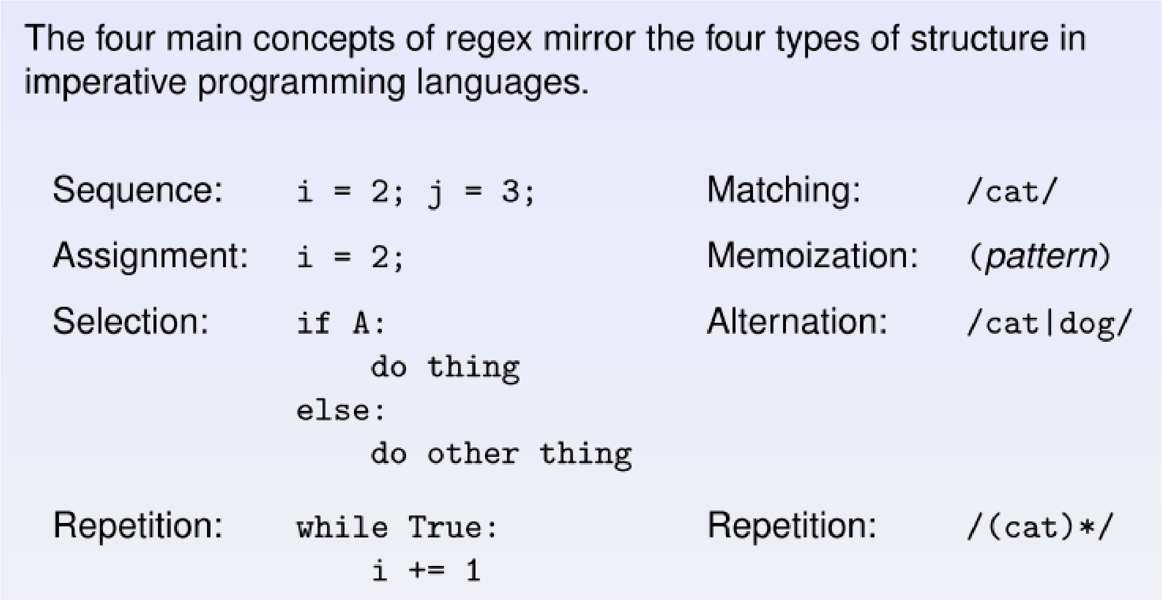
- Consider effectiveness rather than correctness. (Can a document ranking possibly be “correct”?). Sometimes even correctness cannot be defined and/or asserted
- Consider what sort of output a human would produce, given enough time and memory
- Identify what can be quantified, and what approximations we can make to help

2. Exact String Search

Regular Expressions

- Regular expressions (regex, regexp) are patterns that match character strings
- The foundation of regex is literal matching: /knowledge/ where each character matches itself, and matches are case sensitive
- { } [ ] ( ) ^ $. | * + ? $ \ are known as metacharacters and need to be escaped by a backslash (\) to be used in a literal match
- Substrings are uninterpreted; they are not assumed to be whole words or have any specific semantics, so /lane/ will match “planet”
- . matches any single character, e.g. /.n.wl.d../ matches 'knowledge'
- [] matches anything in the brackets
- ^ is an anchor to the start of a string or line
- $ is an anchor to the end of a string or line, e.g. /\$(US|AU)/ matches '$US' or '$AU'
- | expresses disjunction, e.g. /a|b|c/ matches any of 'a', 'b', or 'c'
- * indicates zero or more of the immediately preceding element (greedy)
- ? indicates zero or one of the immediately preceding element (greedy)
- + indicates one of more of the immediately preceding element (greedy)
- {n} matches exactly n of the preceding element

- {m,n} matches between m and n inclusive of the preceeding element (can leave out either to specify m or more or up to n)
- [^] negates anything in an character class, e.g. /[^A-Za-z]/ matches any non-alpha character
- \d is the same as [0-9]
- \D is negation of above
- \w is the same as [a-zA-Z0-9_]
- \W is negation of above
- \s is the same as [\ \t\r\b\f]
- \S is negation of above
- ( ) stores the contents as a variable which can be references later with a back-reference, e.g. /([a-zA-Z]+) +\1/ matches any sequence of one or more letters repeated twice in a row



The four main concepts of regex mirror the four types of structure in imperative programming languages.

Sequence:	`i = 2; j = 3;`	Matching:	`/cat/`
Assignment:	`i = 2;`	Memoization:	`(`*pattern*`)`

Selection:	`if A:`
	`    do thing`
	`else:`
	`    do other thing`
		Alternation:	`/cat|dog/`

Repetition:	`while True:`
	`    i += 1`
		Repetition:	`/(cat)*/`

Naive Matching

- Simply test all the possible placements of the matching query to the string
- `bbanbantbanterbalanca`
- `ba`
- `bbanbantbanterbalanca`
- ` bant`
- `bbanbantbanterbalanca`
- `  b`
- `bbanbantbanterbalanca`
- `   b`
- `bbanbantbanterbalanca`
- `    bante`
- This method requires up to q (length of query) comparisons for t characters in the string

Boyer-Moore Matching

- Suppose instead we start the comparisons at the end of the query
- If the mismatch is ‘a’ then we must shift at least 4 characters to the right (to get to the next 'a' in the query string)

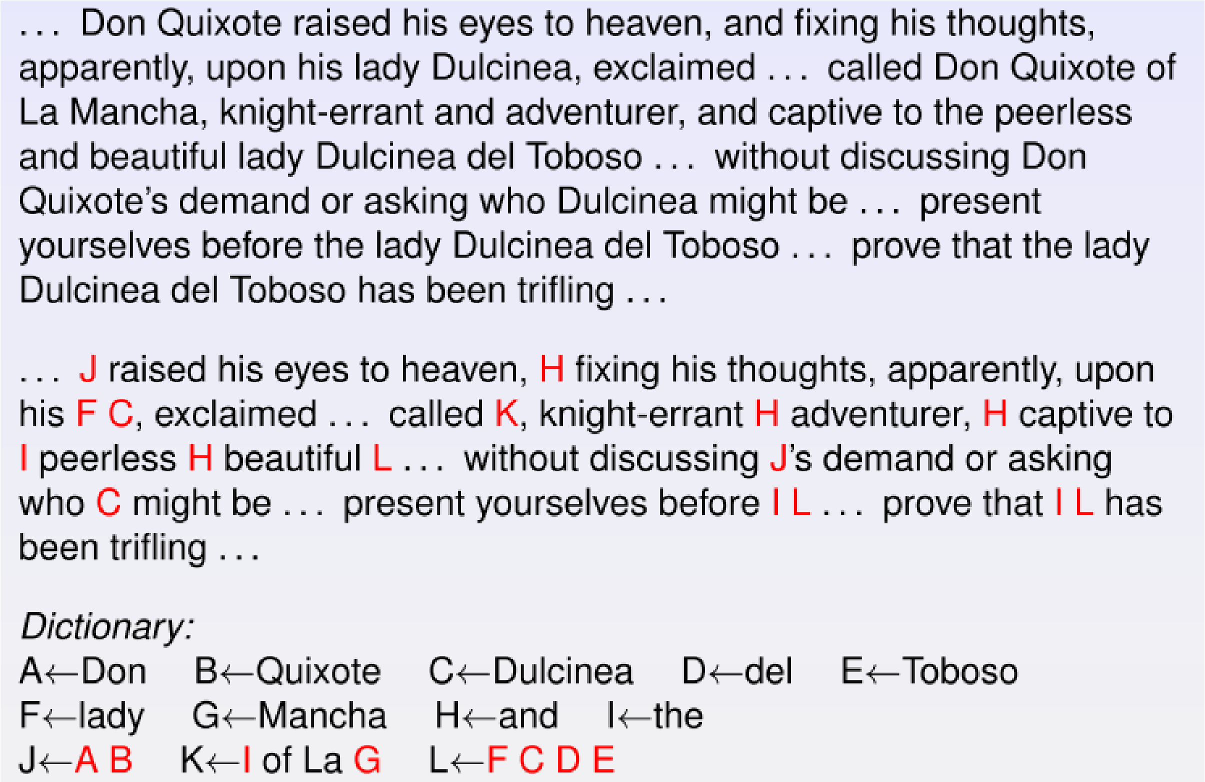
-
  ```
  bbanbantbanterbalanca
  banter
  ```

-
  ```
  bbanbantbanterbalanca
      banter
  ```

-
  ```
  bbanbantbanterbalanca
          banter
  ```

Codebook Text Compression

- Compression can be seen as a mechanism for efficient representation of repetition
- This can be done by construction of a codebook to replace common phrases



. . . Don Quixote raised his eyes to heaven, and fixing his thoughts, apparently, upon his lady Dulcinea, exclaimed . . . called Don Quixote of La Mancha, knight-errant and adventurer, and captive to the peerless and beautiful lady Dulcinea del Toboso . . . without discussing Don Quixote's demand or asking who Dulcinea might be . . . present yourselves before the lady Dulcinea del Toboso . . . prove that the lady Dulcinea del Toboso has been trifling . . .

. . . J raised his eyes to heaven, H fixing his thoughts, apparently, upon his F C, exclaimed . . . called K, knight-errant H adventurer, H captive to I peerless H beautiful L . . . without discussing J's demand or asking who C might be . . . present yourselves before I L . . . prove that I L has been trifling . . .

Dictionary:

A←Don B←Quixote C←Dulcinea D←del E←Toboso
F←lady G←Mancha H←and I←the
J←A B K←I of La G L←F C D E

LZ Compression and Search

- Consider an alphabet of four characters, {e h r t}, and the string heherttherehere
- Pretend that the alphabet is written to the left of the string (the divider is not counted as a character in the below method, start in front of the new character(s))
- Encode each substring by a pointer to the left to an occurrence of the same string, where each pointer consists of d distance to the left and a number of characters to copy
- So we begin with: ehrt|heherttherehere

-
  ```
  ehrt(3,1)(5,1)(2,2)(6,1)(6,1)(1,1)(5,3)(2,1)(4,4)
  ehrt|  h    e   he    r    t    t   her   e  here
  ```
- The pointers and counts can be very compact, maybe 6 to 16 bits each. In effect the pointers be seen as creating a dictionary embedded in the string. Some strings can get very substantially compressed, for ecample:

-
  ```
  aaaaaaaaaaaaaaaaaa…
  a|(1,1)(2,2)(4,4)…
  ```

- But which string to search for? Is it better to search for the longest match, or the nearest one? Remember that the larger the number to be encoded, the more bits are required
- The longer is the string, the more matches will be possible, and a thorough search for a best solution would make compression impossibly slow. Instead, various heuristics are used
- The main heuristic is to do no search at all and instead build a dictionary of string substitutions made so far
- The initial dictionary consists of all single letters. Only dictionary entries can be matched, and the longest match is always taken. When a substitution such as ang is made in tangled, the entry angl is added to the dictionary

3. Approximate String Search

Closest Match

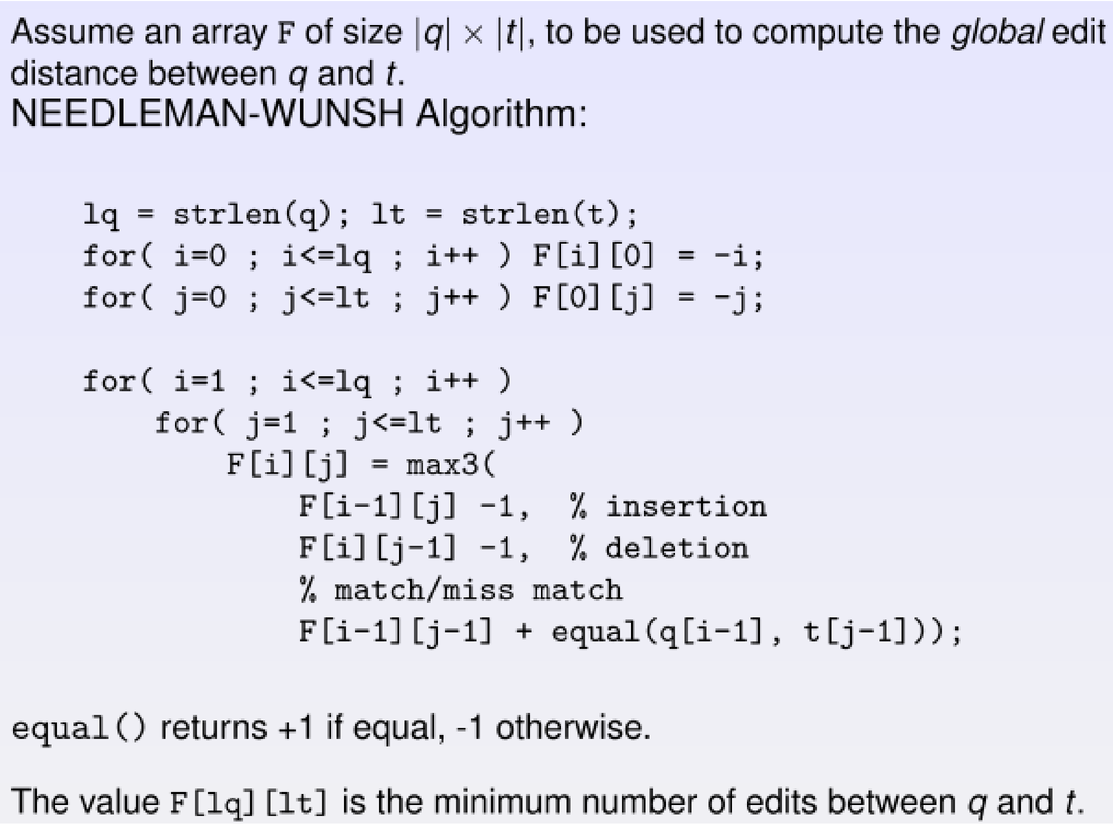
- A common computational task is to find the nearest match to a given string
- Example uses: spelling correction, name matching, query repair, phonetic matching, data cleansing, genomics
- Thorough or exhaustive solutions (in some contexts) require unrealistic computational resources, so we need to find workable approximations
- Context matters: not just language and alphabet, but the reason that the search is taking place, what the user is trying to achieve

Neighbourhood Search

- Given a query string that is not in the dictionary, the task is to find the nearest neighbours
- One approach is to generate all neighbours, and see if they are present: enumerate all of the single-character deletions, insertions, and replacements
- For example, for an alphabet of four letters {e h r t}, the word *thr* has the immediate neighbours {hr tr th ethr hthr rthr tthr tehr thhr trhr tthr ther thhr thrr thtr thre thrh thrr thrt ehr hhr rhr ter trr ttr the thh tht}
- The number of neighbours is given by $N_1 = n + A(n + 1)(A - 1)n$, where n is the query length over alphabet of size A
- With N_2, however, the cost quickly becomes unmanageably high
- There is a trade-off between false negatives and false positives. That is, there is a need to consider effectiveness.

Edit Distances

- The simplest edit distance is the number of insertions, deletions, and substitutions needed to turn one string into another
- Measures such as edit distances are attractive because matches are scored, so answers can be ranked
- Note that the index i codes for the vertical position. Movements up/down indicate a deletion from the query, and across represent insertions into the query



Assume an array `F` of size $|q| \times |t|$, to be used to compute the *global* edit distance between *q* and *t*.

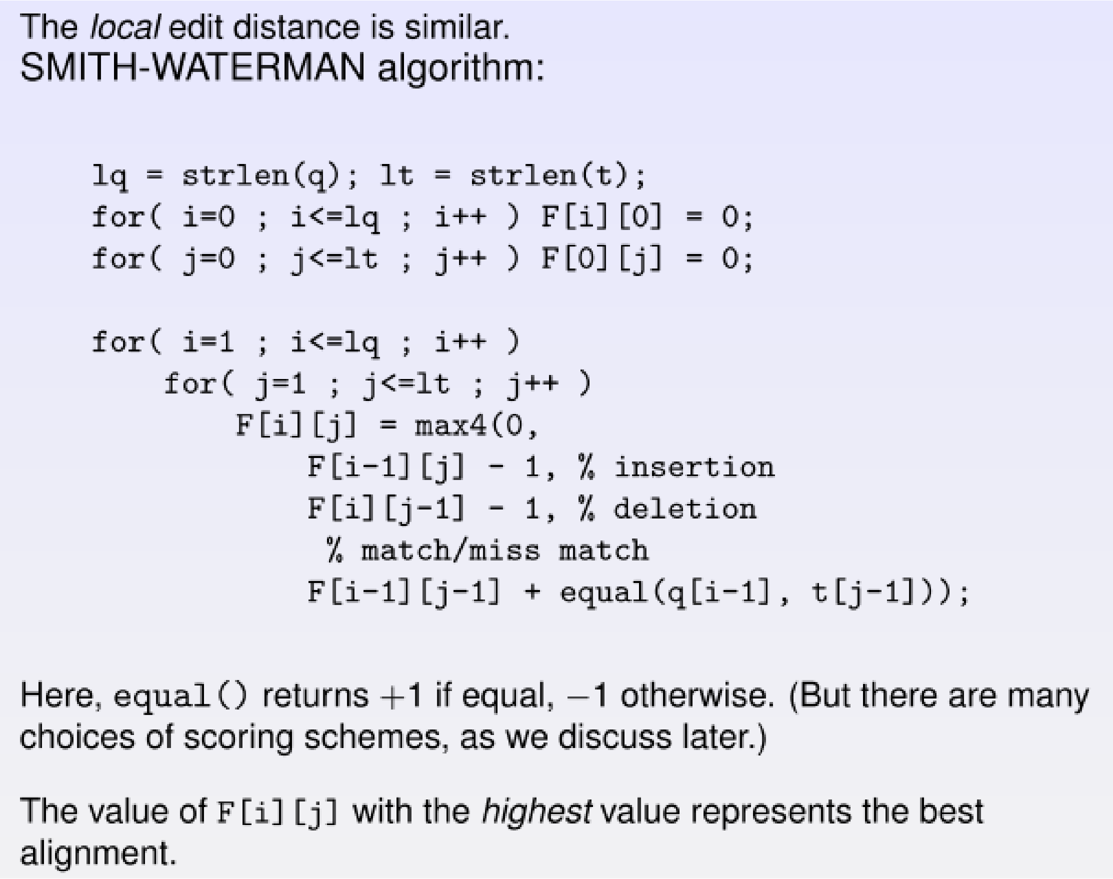
NEEDLEMAN-WUNSH Algorithm:

```
lq = strlen(q); lt = strlen(t);
for( i=0 ; i<=lq ; i++ ) F[i][0] = -i;
for( j=0 ; j<=lt ; j++ ) F[0][j] = -j;

for( i=1 ; i<=lq ; i++ )
    for( j=1 ; j<=lt ; j++ )
        F[i][j] = max3(
            F[i-1][j] -1,  % insertion
            F[i][j-1] -1,  % deletion
            % match/miss match
            F[i-1][j-1] + equal(q[i-1], t[j-1]));
```

`equal()` returns +1 if equal, -1 otherwise.

The value `F[lq][lt]` is the minimum number of edits between *q* and *t*.



The *local* edit distance is similar.

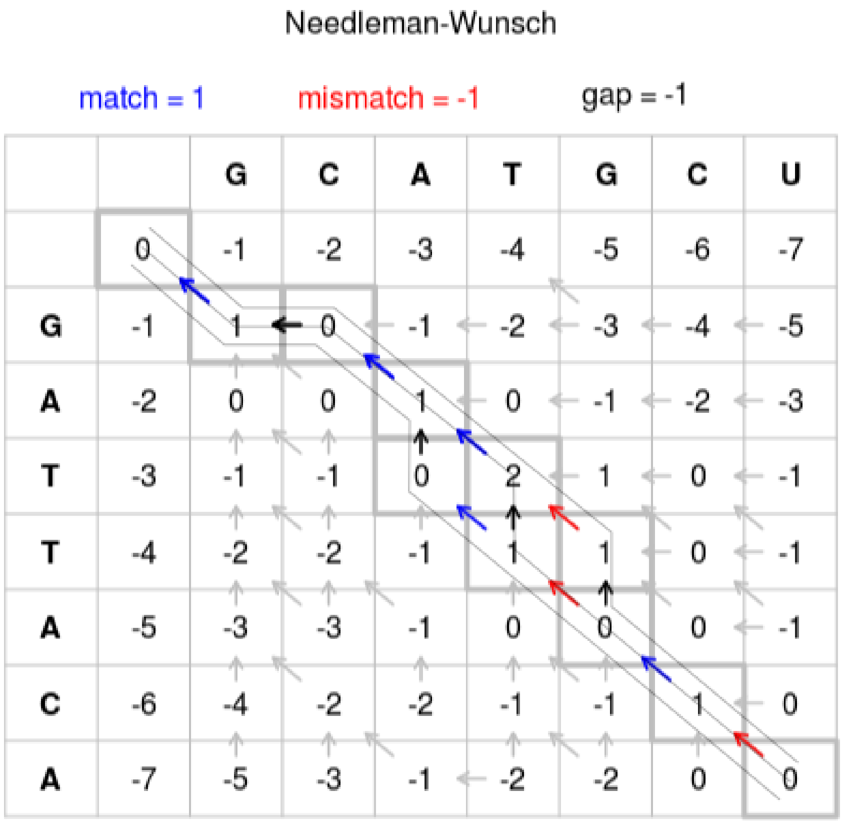
SMITH-WATERMAN algorithm:

```
lq = strlen(q); lt = strlen(t);
for( i=0 ; i<=lq ; i++ ) F[i][0] = 0;
for( j=0 ; j<=lt ; j++ ) F[0][j] = 0;

for( i=1 ; i<=lq ; i++ )
    for( j=1 ; j<=lt ; j++ )
        F[i][j] = max4(0,
            F[i-1][j] - 1, % insertion
            F[i][j-1] - 1, % deletion
             % match/miss match
            F[i-1][j-1] + equal(q[i-1], t[j-1]));
```

Here, `equal()` returns $+1$ if equal, -1 otherwise. (But there are many choices of scoring schemes, as we discuss later.)

The value of `F[i][j]` with the *highest* value represents the best alignment.



Needleman-Wunsch

match = 1 mismatch = -1 gap = -1

		G	C	A	T	G	C	U
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	0	1	0	-1	-2	-3
T	-3	-1	-1	0	2	1	0	-1
T	-4	-2	-2	-1	1	1	0	-1
A	-5	-3	-3	-1	0	0	0	-1
C	-6	-4	-2	-2	-1	-1	1	0
A	-7	-5	-3	-1	-2	-2	0	0

N-grams

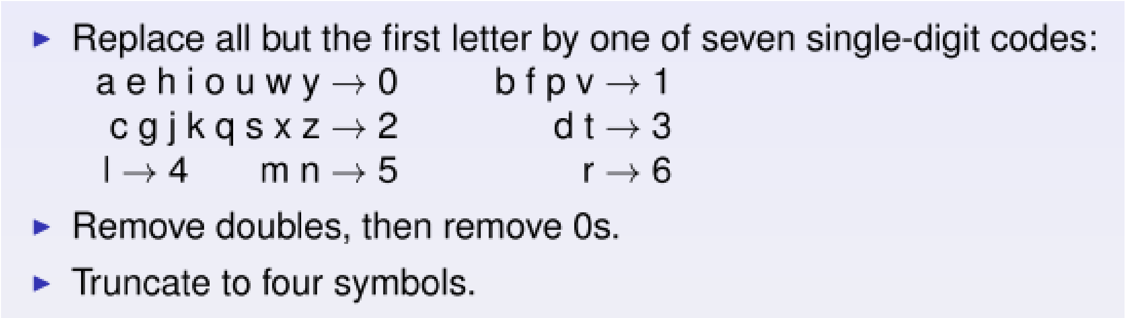
- Define $G_n(s)$ to be the set of all substrings of s of length n, so for example: $G_2(\text{Gorbachev}) = \{\text{Go}, \text{or}, \text{rb}, \text{ba}, \text{ac}, \text{ch}, \text{he}, \text{ev}\}$
- The n-gram distance of strings t and s is defined to be:

$$|G_n(s)| + |G_n(t)| - 2|G_n(s) \cap G_n(t)|$$

- The smaller the n-gram distance, the closer the match. Cost is approximately O(|s| + |t|), compared to O(|s| · |t|) for an edit distance
- Unlikely to produce useful scores for long strings or small alphabets

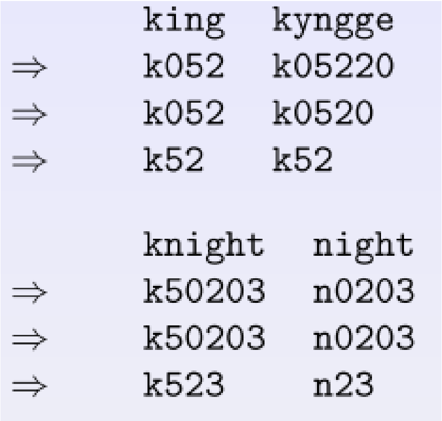
Soundex

- Useful when the spelling of a word (particularly names) is highly variable
- Soundex transforms a string into a 4-character code that represents its sound



- Replace all but the first letter by one of seven single-digit codes:

  a e h i o u w y → 0   b f p v → 1
  c g j k q s x z → 2   d t → 3
  l → 4   m n → 5   r → 6
- Remove doubles, then remove 0s.
- Truncate to four symbols.



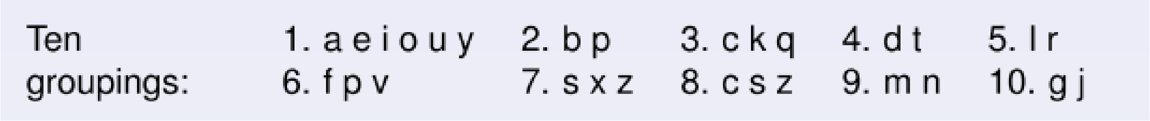
```
       king     kyngge
⇒      k052     k05220
⇒      k052     k0520
⇒      k52      k52

       knight   night
⇒      k50203   n0203
⇒      k50203   n0203
⇒      k523     n23
```

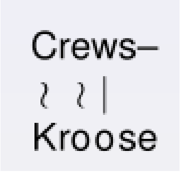
- There are 6734 distinct Soundex codes

Lexicographic Methods

- Strings that sound alike are often spelt alike, so it is plausible to suppose that lexicographic string comparison method will be effective at finding strings that sound alike
- Editex is a lexigraphic method which modifies the standard edit distance, giving variable penalties for insertion or substitution of letters depending on what class they are in
- So this leads to a system that detects both exact and approximate letter matches



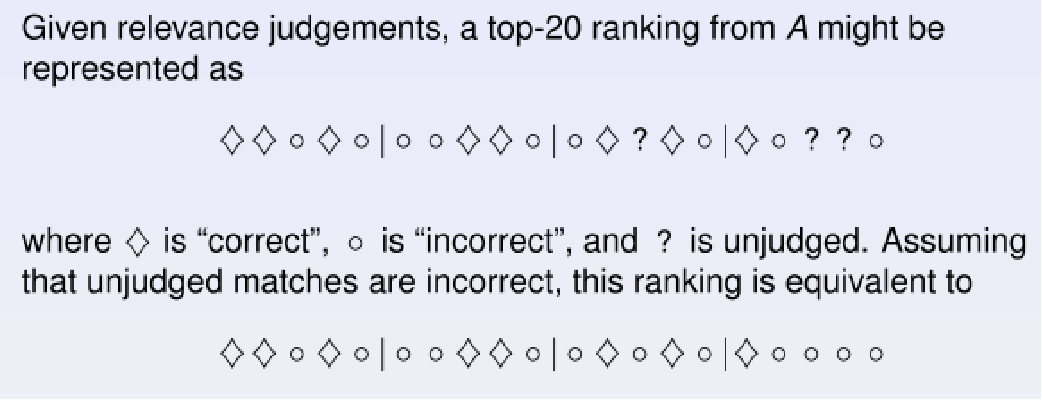
Ten	1. a e i o u y	2. b p	3. c k q	4. d t	5. l r
groupings:	6. f p v	7. s x z	8. c s z	9. m n	10. g j



Crews–
≀ ≀ |
Kroose

Performance

- Given a query, a matching technique A returns a ranked list of answers



Given relevance judgements, a top-20 ranking from *A* might be represented as

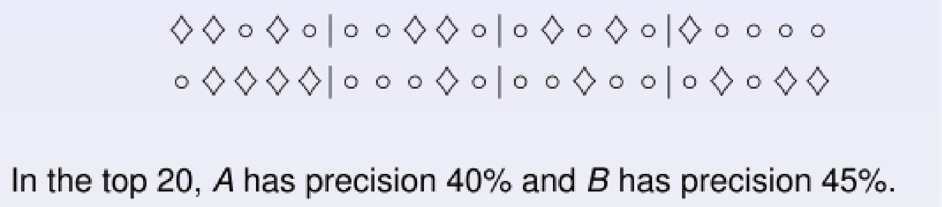
◇ ◇ ○ ◇ ○ | ○ ○ ◇ ◇ ○ | ○ ◇ ? ◇ ○ | ◇ ○ ? ? ○

where ◇ is “correct”, ○ is “incorrect”, and ? is unjudged. Assuming that unjudged matches are incorrect, this ranking is equivalent to

◇ ◇ ○ ◇ ○ | ○ ○ ◇ ◇ ○ | ○ ◇ ○ ◇ ○ | ◇ ○ ○ ○ ○

- There is no objective criteria that can be used to assess a scoring method! Choice of measure is based on arguments from properties such as expected user behaviour
- Precision: the proportion of retrieved matches that are judged correct
- Recall: the proportion of the correct matches that were retrieved

- We can measure precision fairly easily (if we have a human judge), but recall is essentially impossible to measure because we don't know what was not discovered



◇ ◇ ○ ◇ ○ | ○ ○ ◇ ◇ ○ | ○ ◇ ○ ◇ ○ | ◇ ○ ○ ○ ○

○ ◇ ◇ ◇ ◇ | ○ ○ ○ ◇ ○ | ○ ○ ◇ ○ ○ | ○ ◇ ○ ◇ ◇

In the top 20, *A* has precision 40% and *B* has precision 45%.

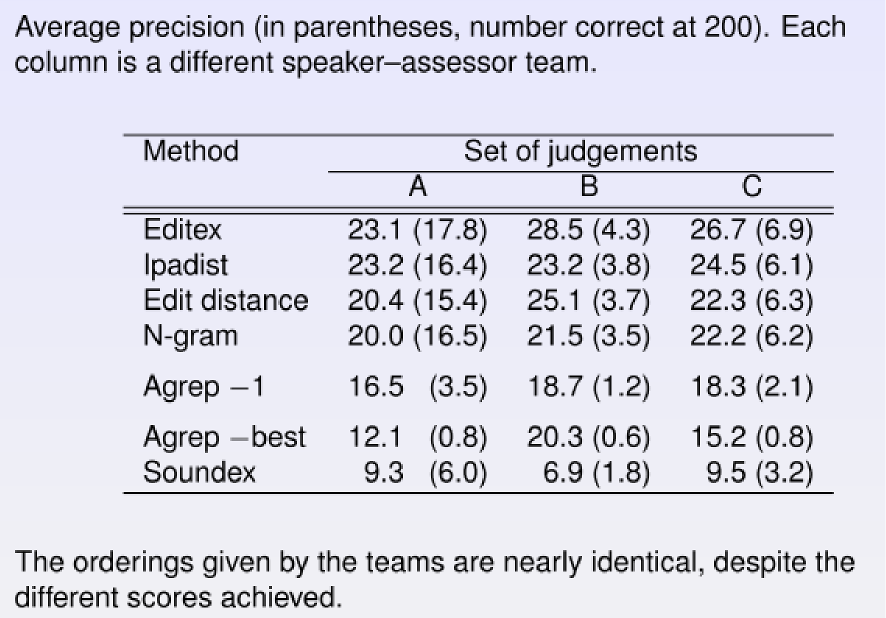
- Average precision is a way of calculating a weighted average of precisions at each position in the ranking, with higher ranks counting for more
- If there are known to be R relevant items and d items in a ranking, with $r_i = 1$ if the ith item in the ranking is relevant, we have:

$$AP = \frac{1}{R}\sum_{i=1}^{d}\left(\frac{r_i}{i}\cdot\sum_{j=1}^{i} r_j\right)$$

- Different performance measures are not necessarily consistent with each other
- Given the diversity of retrieval methods used by different search engines, it is easy to have A > B for one query and A < B for another. For this reason, we use a sample of queries (10 is too few, 50 is adequate, more is preferable) and average the per-query effectiveness
- We then need to use a statistical test to ensure that the two samples are indeed different (to some level of confidence), rather than simply different by chance

Baselines and Benchmarks

- A baseline is used to determine how any proposed method is doing compared to 'dumb and simple' methods, which can often work surprisingly well
- Baselines are also valuable in getting a sense for the difficulty of a given task – sometimes a simple method is so good that there isn’t much scope for a clever method to do better
- Baseline: naive or straightforward method that we would expect a rich (or principled, or . . .) method to better
- Benchmark – established “best practice” technique that we are pitching our method against



Average precision (in parentheses, number correct at 200). Each column is a different speaker–assessor team.

Method	Set of judgements		
	A	B	C
Editex	23.1 (17.8)	28.5 (4.3)	26.7 (6.9)
Ipadist	23.2 (16.4)	23.2 (3.8)	24.5 (6.1)
Edit distance	20.4 (15.4)	25.1 (3.7)	22.3 (6.3)
N-gram	20.0 (16.5)	21.5 (3.5)	22.2 (6.2)
Agrep −1	16.5 (3.5)	18.7 (1.2)	18.3 (2.1)
Agrep −best	12.1 (0.8)	20.3 (0.6)	15.2 (0.8)
Soundex	9.3 (6.0)	6.9 (1.8)	9.5 (3.2)

The orderings given by the teams are nearly identical, despite the different scores achieved.

4. Document Retrieval

Defining Information retrieval

- Information retrieval (IR) is “the subfield of computer science that deals with storage and retrieval of documents” (not databases)
- There is an emphasis on the user. IR systems can be characterized as mechanisms for finding documents that are of value to an individual
- The meaning or content of a document is of more interest than the specific words used to express the meaning
- Typical kinds of document collection include web pages, newspaper articles, intranets, academic publications, company reports, all documents on a PC, research grant applications, parliamentary proceedings, bibliographic entries, historical records, electronic mail, and court transcripts
- In the context of IR, “documents” include text, images, music, speech, handwriting, video, and genomes

Data retrieval versus information retrieval

- In conventional data retrieval systems, prior to storage data is transformed into a representation suitable for manipulation by an algebraic query language
- Information that is ambiguous or atypical cannot be represented or queried
- Users may not agree on the value of a particular document
- A data retrieval system is used to retrieve items based on facts that describe them:
    - “Get articles from The Age dated 15/7/2002”
    - “Fetch articles filed by Piotr Kulowsky in Kursograd”
    - “Get the article entitled ‘Alta Vista Searching for Success’”
- An IR system is used to retrieve items based on their meaning:
    - “Find articles that argue for better public transport”
    - “Is Bosnia a good holiday destination?”
    - “Get articles about different kinds of dementia”

History of Information Retrieval

- The study of IR originated in schools of librarianship, during the nineteenth century
- The Dewey Decimal system (1876) can be seen, in retrospect, as a contribution to information retrieval
- Investigation of computerized IR began during the 1950s, and grew slowly until about 1990, driven by the accumulation of information in electronic form: medical files, corporate repositories, large-scale bibliographies
- Search engines based on IR techniques are a key computing technology
- In some applications – email management, personal document management – IR systems are beginning to replace file systems, and the traditional role of curator is being marginalized. Thus IR is an example of a unifying technology that is replacing a diversity of prior approaches

Information Needs and Query Answers

- An IR system is used by someone because they have an information need they wish to resolve. Information needs can be highly specific, but may be difficult to articulate or explain (to a human or a search system)
- For example: what are the best travel destinations in Northumberland?
- If the query is unsuccessful, the user may reformulate it, thus many different queries can represent the same information need
- Consider the query “intel processor” under the web, news, groups, images, video, shopping, and scholar tabs provided by Google. A different information need is meant in each case
- There is no clear formal way to define what an 'answer' to a query is. Just because a document contains all the keywords doesn't mean that it is useful
- Relevance can be defined as: a document is relevant (that is, on the right topic) if it contains knowledge that helps the user to resolve the information need

Evidence of Similarity

- Document retrievals are often ranked in accordance with measured similarity
- Some words in a search are clearly more significant than others. The query “volcano” might well find relevant documents by itself, but the query “south” is highly unlikely to do so, if what we want is 'active south American volcano'
- Additional heuristics that are used:
  - Choose documents with the query terms in the title or URL
  - Look for occurrences of the query terms as phrases, not just individually
  - Choose pages that were created recently
  - Choose authoritative pages with large numbers of incoming links.
  - Choose pages with appropriately labelled incoming links
  - Use expansion terms/synonyms
- Query expansion (QE) is the process of reformulating a seed query to improve retrieval performance in information retrieval operations. In web search, query expansion involves evaluating a user's input and expanding the search query to match additional documents
- Monotonicity requirements
  - Less weight is given to terms that appear in many documents
  - More weight is given to terms that appear many times in a document
  - Less weight is given to documents that have many terms

Vector-Space Model

- Early approaches were quite ad hoc and not very theoretical, so not very useful for research
- Models are used throughout science to unify observations, make predictions, and provide direction. IR is no exception. One such model is the vector-space model
- Suppose there are n distinct indexed terms in some collection. Each document d can then be thought of as a vector of weights describing the importance of each term in the document:

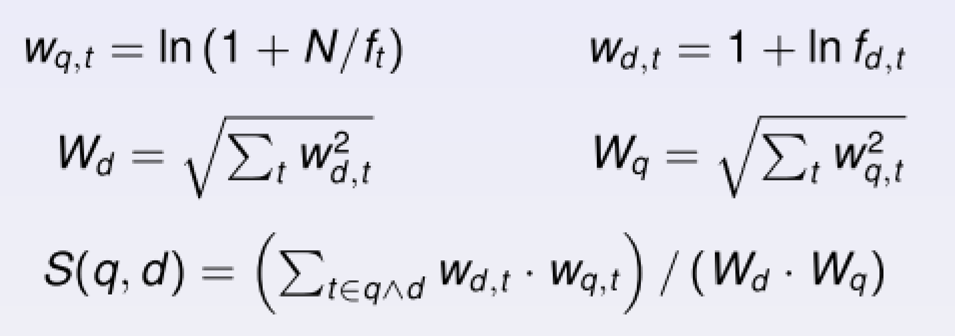
$$\left(w_{d,1}, w_{d,2}, \dots, w_{d,n}\right)$$

- Most w values will be zero, because most documents only contain a tiny proportion of a collection’s terms

- A vector locates a document (or, equivalently in this context, a query) as a point in n-space. The angle of the vector is determined by the relative weight of each term
- In the vector space model, the space is orthogonal – that is, the terms are treated as if they occur independently. This is obviously false. Some words (say “Saudi”) positively influence the likelihood of observing others (“Arabia”, “prince”, “oil”)
- As only the angle between two vectors really is important, we can estimate similarity by:

$$S(q,d) = \frac{\sum_t w_{d,t} \cdot w_{q,t}}{\sqrt{\sum w_{d,t}^2}\sqrt{\sum w_{q,t}^2}}$$

- To make these computations we need to know:
    - $f_{d,t}$: the frequency of term t in document d
    - $f_{q,t}$: the frequency of term t in the query q
    - f_t: the number of documents in the collection containing the term t
    - N: the number of documents in the collection
    - n: the number of indexed terms in the collection
    - $F_t = \sum_d f_{d,t}$: the number of occurences of t in the collection
    - $F = \sum_t F_t$: the number of occurrences of all terms in the collection



$$w_{q,t} = \ln(1 + N/f_t) \qquad w_{d,t} = 1 + \ln f_{d,t}$$

$$W_d = \sqrt{\textstyle\sum_t w_{d,t}^2} \qquad W_q = \sqrt{\textstyle\sum_t w_{q,t}^2}$$

$$S(q,d) = \left(\textstyle\sum_{t \in q \wedge d} w_{d,t} \cdot w_{q,t}\right) / (W_d \cdot W_q)$$

- We wish to find documents d that have high occurrences $f_{d,t}$ of terms t that are rare in the collection as a whole (low f_t). It is also preferable that the document is short

Term-Oriented Language Models

- A message M composed of distinct symbols $w_1, w_2, \dots, w_n$ with each symbol having a frequency of f_i has a total length of $|M| = \sum_i f_i$
- According to information theory, the minimum length needed to encode such as message is to allocate $-\log_2 \frac{f_i}{|M|}$ bits per symbol (more common symbols have fewer bits)
- The entropy of the message is the theoretical minimum length of the message in the context of the amount of information it contains. It is given by:

$$E = \sum_i -f_i \log_2 \frac{f_i}{|M|}$$

- A related approach is to smooth the frequency of each term in the document with the frequency of the same term overall, and multiply this over all the terms found in both the document and the query. The term μ is a 'mixing term' indicating how much the document and background are smoothed together:

$$S(q,d) = \prod_{t \in q \wedge d} \left(\frac{|d|}{|d| + \mu} \cdot \frac{f_{d,t}}{|d|} + \frac{\mu}{|d| + \mu} \cdot \frac{F_t}{F} \right) = \sum_{t \in q \wedge d} \log \left(\frac{f_{d,t}}{|d| + \mu} + \frac{\mu}{|d| + \mu} \cdot \frac{F_t}{F} \right)$$

$$= \sum_{t \in q} \log \left(\frac{\mu}{|d| + \mu} \frac{F_t}{F} \right) \left(\frac{f_{d,t}}{\mu} \frac{F}{F_t} + 1 \right)$$

$$= \sum_{t \in q} \log \left(\frac{\mu}{|d| + \mu} \right) + \sum_{t \in q} \log \left(\frac{F_t}{F} \right) + \sum_{t \in q} \log \left(\frac{f_{d,t}}{\mu} \frac{F}{F_t} + 1 \right)$$

Since $\log\left(\frac{F_t}{F}\right)$ does not depend on d we can drop it, as we only care about ranking

$$S(q,d) = \sum_{t \in q} \log \left(\frac{\mu}{|d| + \mu} \right) + \sum_{t \in q} \log \left(\frac{f_{d,t}}{\mu} \frac{F}{F_t} + 1 \right)$$

$$S(q,d) = |q| \log \left(\frac{\mu}{|d| + \mu} \right) + \sum_{t \in q} \log \left(\frac{f_{d,t}}{\mu} \frac{F}{F_t} + 1 \right)$$

The TREC Experiments

- The National of Standards and Technology established the large-scale TREC framework in 1992 to compare search engines in a systematic, unbiased way
- In a typical year, the document pools are (a) 2 gigabytes of newswire-type data, or about 0.5 million documents, and (b) 100 gigabytes of web data, or about 7 million documents
- On the newswire data there was 50 queries, and about 30 groups participated with 61 systems, each reporting the top 1000 documents for each query
- The top 100 answers for each system were pooled, giving about 3,000 documents per query or 150,000 documents overall
- Humans assessed each of the 150,000 documents for relevance to the queries, finding an average of about 70 relevant documents per query

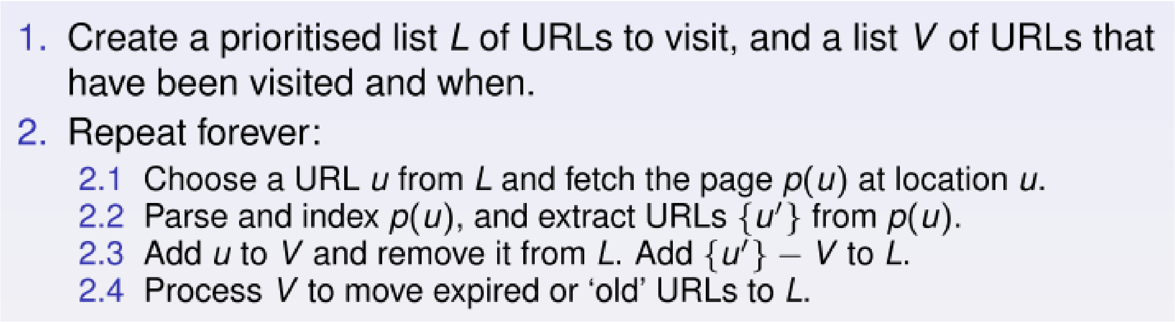
5. Web Search

Four Aspects of Searching

- Crawling: the data to be searched needs to be gathered from the web
- Parsing: the data then needs to be translated into a canonical form
- Indexing: data structures must be built to allow search to take place efficiently
- Querying: the data structures must be processed in response to queries

Crawling

- Before a document can be queried, the search engine must know that it exists. On the web, this is achieved by crawling
- Crawlers attempt to visit every page of interest and retrieve them for processing and indexing. Basic challenge: there is no central index of URLs of interest
- The observation that allows effective harvesting of the web is that it is a highly linked graph. Thus, we assume that if a web page is of interest, there will be a link to it from another page. Hence, given a sufficiently rich set of starting points, every interesting site on the web will be reached eventually



```
1. Create a prioritised list $L$ of URLs to visit, and a list $V$ of URLs that have been visited and when.
2. Repeat forever:
    2.1 Choose a URL $u$ from $L$ and fetch the page $p(u)$ at location $u$.
    2.2 Parse and index $p(u)$, and extract URLs $\{u'\}$ from $p(u)$.
    2.3 Add $u$ to $V$ and remove it from $L$. Add $\{u'\} - V$ to $L$.
    2.4 Process $V$ to move expired or 'old' URLs to $L$.
```

- Other problems:
    - Some websites return the same content at a new URL with each visit
    - Some pages never return status 'done' on access
    - Some regions and content providers have low bandwidth
    - Avoid crawler traps: for example, a 'next month' link on a calendar can potentially be followed until the end of time
    - Ensure that dynamic pages are visited sufficiently frequently

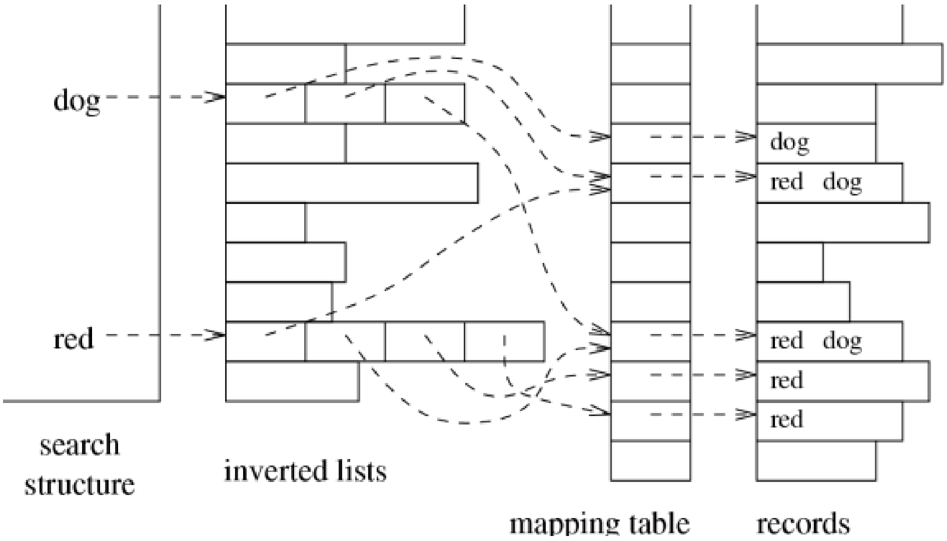
Document Parsing

- Once a document has been fetched, it must be parsed. The words in the document are extracted and added to a data structure recording which documents contain which words
- Metadata is often the first thing checked, as it can provide information about the document type, format, character encoding, etc. Metadata is generally not itself indexed, as it is not visible to users
- Tokenisation is the process of breaking up a web page into a series of tokens, which could be phrases or single words. Decisions must be made about how to treat hypenation, compounding, possessives, canonicalisation of dates, numbers and spelling, abbreviations
- Historically, search engines discarded both stop words ('content-free' terms such as the, or, and so on), but they now generally appear to be indexed too
- Stemming is an attempt to undo the processes that lead to word formation, stripping away affixes like 'in' and 'ive' in 'inexpensive' so as just to index the core semantic root tokens (which need not themselves be words). One approach is to write a series of cascading rules to be applied. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation
- Zoning refers to examining the content of a site based upon its region of the page, such as title, anchor text, headings, captions, etc. Web search engines typically calculate weights for each of these zones, hence the observed behaviour of web search engines to favour pages that have the query terms in titles

Indexing

- Fast query evaluation makes use of an index: a data structure that maps terms to the documents that contain them. For example, the index of a book maps a few key terms to page numbers
- The only practical index structure for text query evaluation is the inverted file: a collection of lists, one per token, recording the identifiers of the documents containing that term
- Search structure: a framework which lists all tokens and the number of documents containing each one, along with a pointer to the start of the corresponding inverted list

- Inverted list: one for each token, contains the identifiers of each document containing the term, as well as the frequency $f_{d,t}$ of term t in document d
- Total size of an index database is large, maybe 40% of the original data, but not prohibitive



Querying

- To evaluate a general Boolean query:
  - Fetch the inverted list for each query term
  - Use intersection of lists to resolve AND
  - Use union of lists to resolve OR
  - Take the complement of a list to resolve NOT
  - Ignore within-document frequencies
- To evaluate a query under the cosine measure:
  - 1. Allocate an accumulator A_d for each document d, and set $A_d = 0$
  - 2. For each query term t
    - 2.1 Calculate the weight of the term $w_{q,t}$ and fetch the inverted list for t
    - 2.2 For each item (document identifier) in the inverted list, calculate the weight of the document for that term $w_{d,t}$, and set $A_d = A_d + w_{q,t} \times w_{d,t}$
  - 3. Read the W_d values and for each $A_d > 0$ normalise by setting $A_d = A_d / W_d$
  - 4. Identify the r greatest A_d values and return the corresponding documents
- That is, starting with a set of N zeroed accumulators, use the lists to update the accumulators term by term. Then use the document lengths to normalize
- With the standard query evaluation algorithm, most accumulators are non-zero and an array is the most efficient structure. However, most accumulator values are trivially small, with the only matching terms being one or more common words
- If, instead, only low f_t (rare) terms are allowed to create accumulators, the number of accumulators is greatly reduced. A simple mechanism is to impose a limit L on the number of accumulators
- This can be done in two ways: have a 'limit' L such that if $|A| < L$ no further accumulators are created, or a 'threshold' S such that if $w_{q,t} \times w_{d,t} < S$ no accumulator is created

Phrase Queries

- Many queries evaluate successfully if treated as a phrase, that is, the words must be adjacent in the retrieved text
- Thus, it makes sense to give such pages a higher score than pages in which the words are separated
- The problem with phrase searching is that the number of distinct phrases grows far more rapidly than the number of distinct terms
- There are three main strategies for phrase query evaluation:
    - Process queries as bag-of-words, so that the terms can occur anywhere in matching documents, then post-process to eliminate false matches
    - Add word positions to the index entries, so the location of each word in each document can be used during query evaluation, so now each entry for each term in the inverted list consists of is $\langle d, f_{d,t}, p_1, \ldots, p_{f_{d,t}} \rangle$, where p_i give positions (in terms of word counts) of appearances of t in d
    - Use some form of phrase index or word-pair index so that they can be directly identified without using the inverted index
- Proximity is an a variant, imprecise form of phrase querying. Favour documents where the terms are near to each other. Search for "phrases" where the terms are within a specified distance of each other

Link Analysis

- In general search, each document in considered independently. In web search, a strong piece of evidence for a page's importance is given by links, in particular how many other pages have links to this page
- PageRank is a particularly important algorithm used by google to rank search results
- Basic intuition of PageRank: each web document has a fixed number of credits associated with it, a portion of which it redistributes to documents it links to, while it also receives credits from pages that point to it. The final number of credits the page is left with determines its pagerank
- The process used to calculate the π(d) values is based on the notion of "random walks" with the option to 'teleport' to a random page with fixed probability
- PageRank has a reputation for being critical to the performance of Google, and has attracted a great deal of research interest
- Analyses of Google searches shows that in most cases the importance of PageRank is low

Further Heuristics

- Anchor text: the text used in hypertext links, treated as a special zone and often very important (e.g. try a search for 'click here')
- Click-throughs: note which pages people actually visit by counting click-throughs
- Hand-fixing: manually alter the behavior of common queries, cache the answers to common queries, and index selected specific phrases
- Parallelisation: divide the collection among multiple servers, each of which has an index of its documents; have separate servers for crawling and index construction.
- Outsource: accept feeds from dynamic data providers such as booksellers, newspapers, and microblogging sites, and integrate diverse data resources, such as maps and directories

6. Introduction to Machine Learning

Goals of Data Mining

- To handle large, high dimensional and complex data (relational, time series, graphical, text)
- To discover interesting patterns that can help understand underlying process
- To discover knowledge that is useful/actionable
- Increasingly necessarily because of proliferation of data sources and insufficient space to store all the new data, as well as insufficient human brainpower to examine it all manually
- Applications in fields of bioinformatics, commerce, health, engineering, information retrieval, modelling complex systems, decision support systems, spam detection, language technologies

Prediction Methods

- Use some variables to predict unknown or future values of other variables
- Classification: predict an unknown class given known attribute variables. Applications: direct marketing (predict who will buy based on demographic data), fraud detection (predict dishonest transactions), classification of galaxies from image properties
- Regression: predict the value of a continuous varialbe given the values of other variables. Applications: sales revenue prediction based on advertising spending, prediction of weather as a function of wind, humidity, temperature, etc
- Deviation Detection: outlier detection, anomaly detection

Description Methods

- Find human-interpretable patterns that describe the data
- Clustering: given a set of data points and a similarity measure, find clusters in the data such that data in one cluster is similar to data in other clusters. Applications: market segmentation, document classification and categorisation
- Association Rule Discovery: given a set of records, produce dependency rules to predict occurrence of an item based on occurrence of other items. Applications: marketing and sales promotion (what people will buy), inventory management (what parts needed)
- Sequential Pattern Discovery: given a set of objects, with its own timeline of events, find rules that predict strong sequential dependencies among different events. Applications: telecommunications alarm logs, computer system errors

7. Naïve Bayes Classification

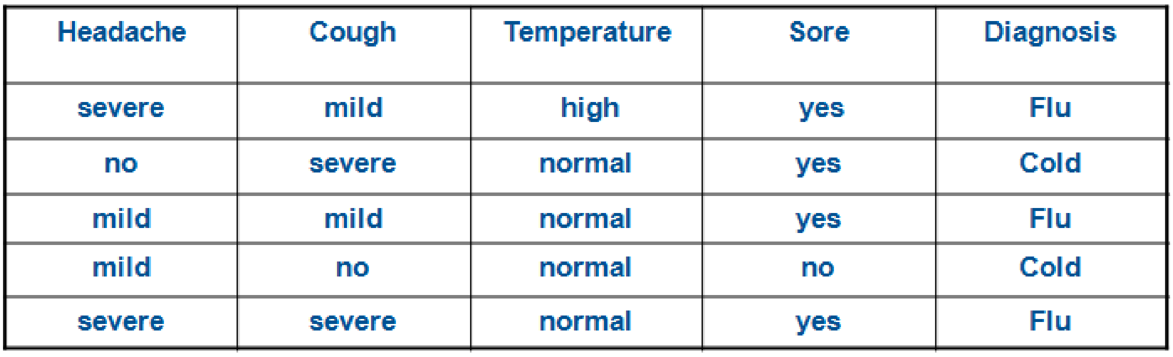
The Object of Classification

- In a collection of training data, each item of the data contains a set of attributes (features), at least one of the attributes is the class
- The goal is to build a model for class attributes (dependent variables) as a function of the values of other attributes (independent variables)
- The discovered model (function) is used to predict the label of a previously unseen item
- For determining how good the discovered model is a test set is used. Usually, the given data set is partitioned into two disjoint training and test sets. The training set is used to build the model and the test set is used to validate it

The Naïve Bayes Classifier

- Given a set of K classes $C_1, C_2, \dots, C_k$ and a dataset of tupes $X_i = (x_1, x_2, \dots, x_n)$ for n attribute variables, we want to be able to predict given an unclassified X what class it belongs to
- The naïve bayes classifier uses the bayesian updating formula to determine the probability of each class for any parameter vector X:
$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$
- Since $P(X)$ is constant for all classes and only relative probabilities matter, it is ignored
- The prior $P(C_i)$ is easily calculated from the training date: $P(C_i) = \frac{n_i}{n}$
- The likelihood is the hard part. To facilitate calculation, we make the 'naïve' assumption that each attribute is conditionally independent of every other. Thus we have the formula:
$$P(X|C_i) = P(x_1|C_1) \times P(x_2|C_2) \times \dots \times P(x_n|C_n)$$
- Note that if any value comes out zero, we use e = small value (can use less than 1/n where n is the number of training instances), so as the avoid the whole product going to zero

Example Calculation



Headache	Cough	Temperature	Sore	Diagnosis
severe	mild	high	yes	Flu
no	severe	normal	yes	Cold
mild	mild	normal	yes	Flu
mild	no	normal	no	Cold
severe	severe	normal	yes	Flu

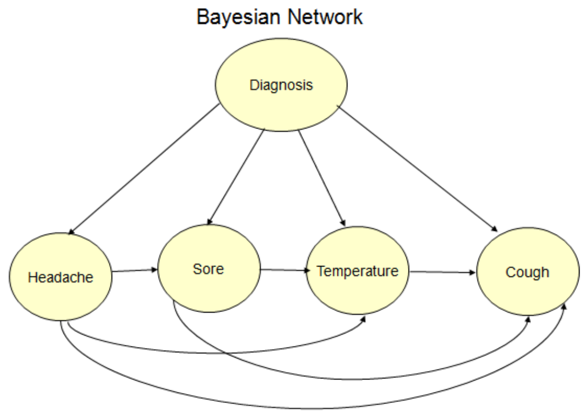
P(FLU) = 3/5	**P(Cold) = 2/5**
P(Headache = severe | Flu) = 2/3	P(Headache = severe | Cold) = 0/2 = e
P(Headache = mild | Flu) = 1/3	P(Headache = mild | Cold) = 1/2
P(Headache = no | Flu) = 0/3 = e	P(Headache = no | Cold) =1/2
P(Sore = severe | Flu) = 1/3	P(Sore = severe | Cold) = 1/2
P(Sore = mild | Flu) = 2/3	P(Sore = mild | Cold) = 0/2 ~ e
P(Sore = no | Flu) = 0/3 = e	P(Sore = no | Cold) = 1/2
P(Temperature = High | Flu) = 1/3	P(Temperature = High | Cold) = 0/2 = e
P(Temperature = Normal | Flu) = 2/3	P(Temperature = Normal | Cold) = 2/2
P(Cough = yes | Flu) = 3/3	P(Cough = yes | Cold) = 1/2
P(Cough = no | Flu) = 0/3 =e	P(Cough = no | Cold) = 1/2

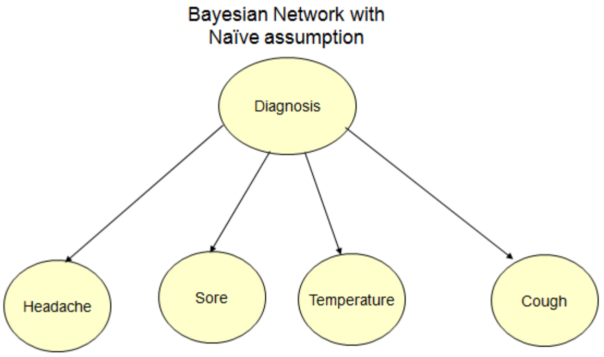
$$
\begin{aligned}
&P(Flu|Headache\ =\ sev, Sore\ =\ no, Temp\ =\ nor, Cough\ =\ yes) = P(Flu|X)\\
&= P(X|Flu) \times P(Flu)\\
&= P(severe|Flu)P(no\ sore|Flu)P(normal|Flu)P(yes\ cough|Flu) \times P(Flu)\\
&= \left(\frac{2}{3}\right)(e)\left(\frac{2}{3}\right)(1) \times \left(\frac{3}{5}\right)\\
&= \frac{12}{45}e\\
&= 0.27e
\end{aligned}
$$

$$
\begin{aligned}
&P(Cold|Headache\ =\ sev, Sore\ =\ no, Temp\ =\ nor, Cough\ =\ yes) = P(Cold|X)\\
&= P(X|Cold) \times P(Cold)\\
&= P(severe|Cold)P(no\ sore|Cold)P(normal|Cold)P(yes\ cough|Cold) \times P(Flu)\\
&= \left(\frac{1}{2}\right)\left(\frac{1}{2}\right)(1)\left(\frac{1}{2}\right) \times \left(\frac{2}{5}\right)\\
&= \frac{2}{40}\\
&= 0.05
\end{aligned}
$$

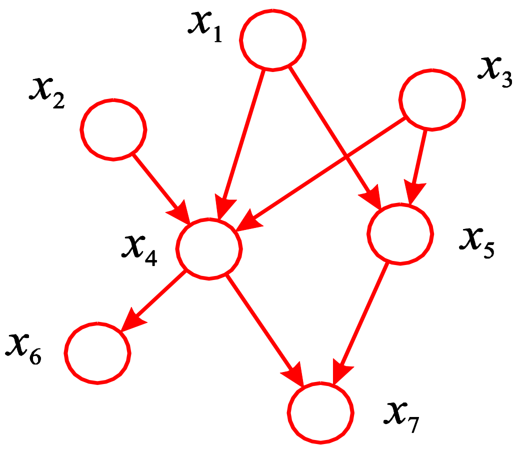
Since $P(Flu|X) < P(Cold|X)$, we assign X to Cold.

The Naïve Assumption





Note that in probabilistic graphical models, we only need to include direct conditional dependencies:



$$
P(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = P(x_1)P(x_2)P(x_3)P(x_4|x_1, x_2, x_3)P(x_5|x_1, x_3)P(x_6|x_4)P(x_7|x_4, x_5)
$$

8. Decision Trees

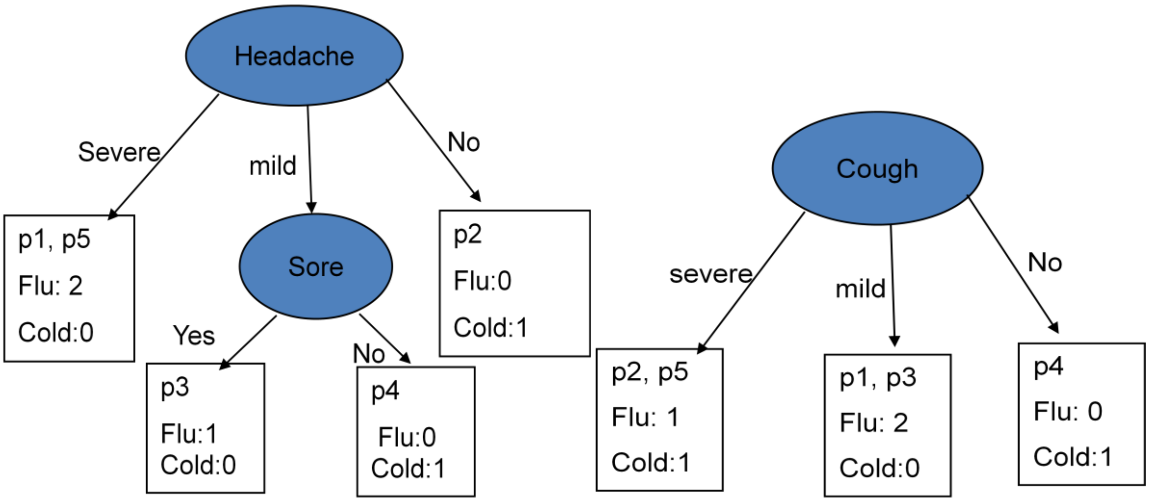
Introduction to Trees

- Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value
- The usual idea is to extract from the data one rule for each leaf node, generally chosen in a way so as to split the data up as quickly as possible (like 20 questions)
- The difficult questions to answer relate to how to build an optimal decision tree, as many variations are possible. In particular, how to choose attribute values at each decision node? How to choose number of branches at each node and attribute values for partitioning the data? When to stop the growth of the tree?
- Optimal construction of a Decision Tree is NP hard, so instead we use various heuristics
- A common method is to choose an attribute to partition the data at the node, such that each new partition is as homogeneous as possible
- We can stop the growth of the tree if all the leaf nodes are largely dominated by a single class (that is the leaf nodes are nearly pure)

Example Trees

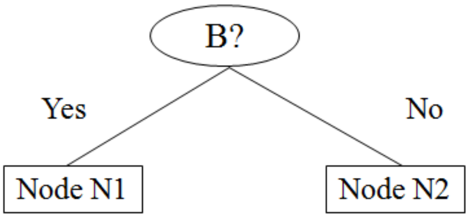
Patient#	**Headache**	**Cough**	**Temperature**	**Sore**	**Diagnosis**
p1	**severe**	**mild**	**high**	**yes**	**Flu**
P2	**no**	**severe**	**normal**	**yes**	**Cold**
P3	**mild**	**mild**	**normal**	**yes**	**Flu**
P4	**mild**	**no**	**normal**	**no**	**cold**
p5	**severe**	**severe**	**normal**	**yes**	**Flu**

Two different example decision trees for the same data:



Computing the GINI Index

- The GINI coefficient is one of the most common measures of node impurity
- The GINI for node t is calculated as: $GINI(t) = 1 - \sum_i [p(i|t)]^2$, where $p(i|t)$ is the relative frequency of class i at node t
- Maximum value of Gini index = (1 - 1/n) where n is the number of classes, and occurs when records are equally distributed among all classes (worse case, most impure, least information)
- Minimum is (0.0) when all records belong to one class, implying most interesting information or most pure or most homogeneous
- When a node p is split into k partitions (children), the quality of split is computed as: $GINI(split) = \sum_{t=1}^{k} \frac{n_t}{n} GINI(t)$, basically a weighted average of GINIs of each child node t, and n is the number of records at the parent node



	Parent
C1	**6**
C2	**6**
Gini = 0.500	

Gini(N1)
$= 1 - (5/7)^2 - (2/7)^2$
= 0.408

Gini(N2)
$= 1 - (1/5)^2 - (4/5)^2$
= 0.32

	N1	**N2**
C1	**5**	**1**
C2	**2**	**4**
Gini=0.371		

Gini(Children)
= 7/12 * 0.408 +
5/12 * 0.32
= 0.371

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

Computing Entropy

- An alternative splitting criteria is based on information gain, as measured by entropy
- The entropy is computed as: $ENTROPY(t) = -\sum_i p(i|t) \log p(i|t)$, where $p(i|t)$ is the relative frequency of class i at node t
- Maximum entropy = log n, where n is the number of records in each class, implying minimal information as each class has the same number of records at this node
- Minimum entropy = (0.0) where all records belong to one class, implying max information
- When a node p is split into k partitions (children), the quality of split is computed as: $ENTROPY: GAIN(split) = ENTROPY(p) - \left(\sum_{t=1}^{k} \frac{n_i}{n} ENTROPY(t)\right)$, which gives the change in entropy from the parent node to that weighted entropy of the child nodes, n_t being the number of records at each child node t

Computing Classification Error

- Yet another alternative measure of splitting is classification error
- $ERROR(t) = 1 - \max_i p(i|t)$, where $p(i|t)$ is the relative frequency of class i at node t
- Maximum error = 1-1/n, when records are equally distributed among all classes
- Minimum error = (0.0) when all records belong to one class

Comparative Example

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Gini = 1 – P(C1)^2 – P(C2)^2 = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Gini = 1 – (1/6)^2 – (5/6)^2 = 0.278

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Gini = 1 – (2/6)^2 – (4/6)^2 = 0.444

$$\boxed{GINI(t) = 1 - \sum_j [p(j|t)]^2}$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Entropy = – (1/6) log$_2$ (1/6) – (5/6) log$_2$ (1/6) = 0.65

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Entropy = – (2/6) log$_2$ (2/6) – (4/6) log$_2$ (4/6) = 0.92

$$\boxed{Entropy(t) = -\sum_j p(j|t) \log_2 p(j|t)}$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

C1	**2**
C2	**4**

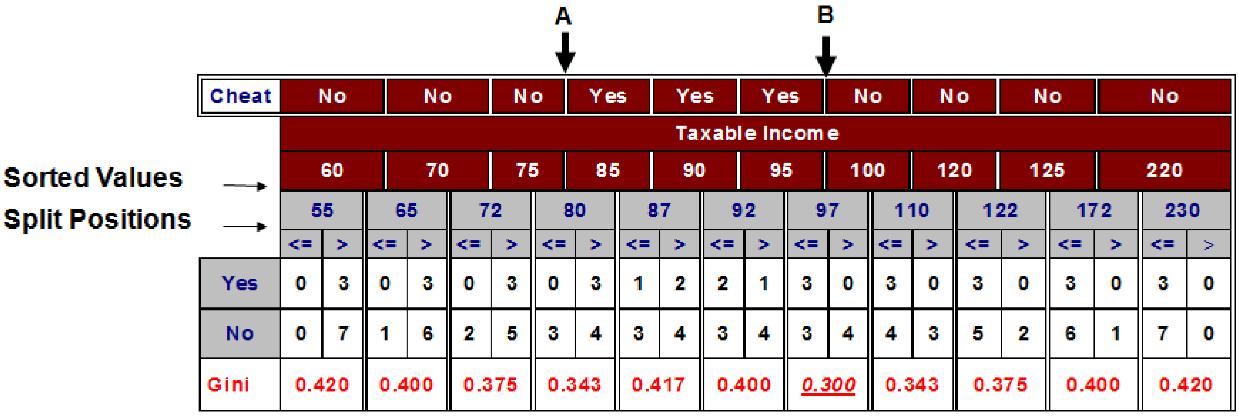
P(C1) = 2/6 P(C2) = 4/6

Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

$$\boxed{Error(t) = 1 - \max_i P(i|t)}$$

Continuous Variables

- For continuous variables, use binary decisions to split the possible values into two discrete classes, greater than or less than some cutoff value
- To do this, first sort the attribute on values, then linearly scan these values, updating the gini index at points where class label changes. Once this process is complete, choose the split position that has the least gini index


A
B
Sorted Values
Split Positions

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Taxable Income	60	70	75	85	90	95	100	120	125	220

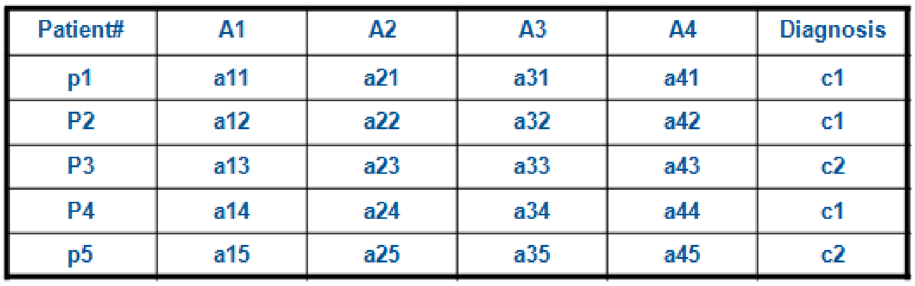
	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		0.420	

Tree Induction

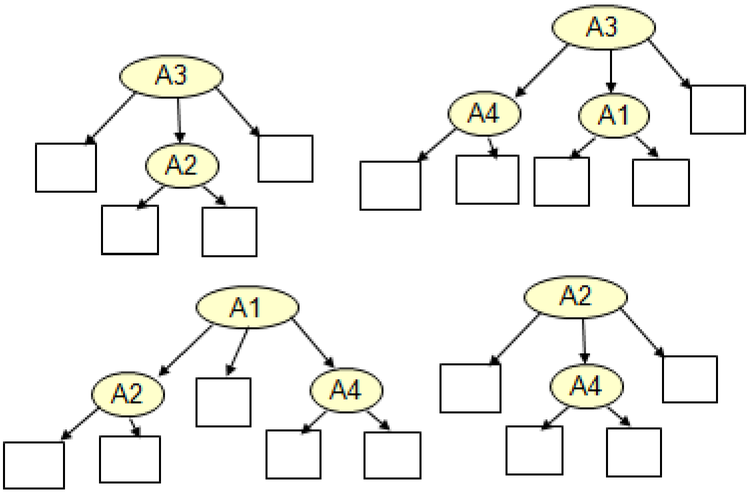
- A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in recursive partitioning
- This process is called top-down induction of decision trees, and is an example of a greedy algorithm
- The basic approach is at each step to select an attribute and tree split which optimizes some particular criteria, generally GINI index, entropy, or misclassification rate
- The recursion is completed when the subset at a node has all the same value of the target variable (class), or when splitting no longer adds value to the predictions. Sometimes we can also stop expanding when all records have similar attribute values (as a heuristic)
- To use the tree for predictive classification, simply traverse the branch of the decision tree from the root node matching the corresponding attribute values of the test record. Once a terminal node is reached, make the decision based on whichever class has the highest frequency at that node (ideally the terminal node will be pure)

Bagging

- A single decision tree cannot capture full information available in the data. This is especially true for data sets with very large number of dimensions
- One solution to this involves bagging, which involves building a number of similar decision trees by sampling the training data
- For example, for a data set containing N records we sample N records randomly with replacement (this is called bootstrapping). For each sample we build a decision Tree, so we have a number of such trees, each of which may make slightly different predictions
- At the time of classification we collect all the decisions from the decision trees and apply majority rule to make the final decision - generally this method works well



Patient#	A1	A2	A3	A4	Diagnosis
p1	a11	a21	a31	a41	c1
P2	a12	a22	a32	a42	c1
P3	a13	a23	a33	a43	c2
P4	a14	a24	a34	a44	c1
p5	a15	a25	a35	a45	c2



Random Forests

- This is an extension of the bagging technique, where the construction of each tree is different to the standard decision tree method
- At each node we first randomly select "m" features (attributes) out of all the features unlike the standard method which considers all the non-chosen features so far and chooses the best attribute to split the node among these attributes
- 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
- The independence of each tree depends on the small values of m, as smaller m means the chosen splitting variable at each node is more likely different to other trees
- Unlike standard Decision Tree method, there is no need to prune the constructed trees due to the randomness imposed both on the selection of data and the tree construction
- Random Forests are shown to be one of the best classification methods experimentally

9. Evaluating Machine Learning Methods

Evaluating Decision Trees

- All rules are mutually exclusive this implies only one rule will be applicable at the time of decision making
- Easy to extract the rules from traversing the decision tree
- Too restrictive in terms of number rules learned
- The rules may be too specific and can be very complex

Evaluating Naïve Bayes

- Naïve Bayes is very simple to build, extremely fast to make decisions, and easy to change the probabilities when the new data becomes available
- Scales easily for large number of dimensions (100s) and data sizes
- Easy to explain the reason for the decision made - clear interpretation
- One should apply NB first before launching into more sophisticated classification techniques

Issues for Classification

- Overfitting: for decision trees we need to make sure it does not over fit the data, as then the model will not generalise. One way to do this is to have a stopping criterion that makes sure that if the GINI index is smaller than some threshold value we stop splitting the node.
- Missing Values: missing values can be predicated by using the distribution of the attribute values, especially if we find two highly correlated attributes we can use this correlation to predict the missing value
- Costs of Classification: we need to optimize the decision based on cost involved using confusion matrix

Evaluation Metrics

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Sensitivity = \frac{TP}{TP + FN}$$

Accuracy with respect to positive cases also called true positive rate

$$Specificity = \frac{TN}{TN + FP}$$

Accuracy with respect to negative cases

$$\mathrm{Re}call = \frac{TP}{TP + FN}$$

$$\Pr ecision = \frac{TP}{TP + FP}$$

$$F1\_Score = \frac{2\,\mathrm{Re}call * \Pr escision}{\mathrm{Re}call + precision}$$

Cross-Validation

- Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set
- Leave-One-Out: if we have N data points for which we know the labels, we systematically choose each data point as test case and the rest as training data, which means we have to train the system N times and the average performance is computed. Avoids sampling bias and gives better accuracy for predictions (as more data used for training), however infeasible for large datasets
- Ten Fold cross validation: partition the data into ten parts, training the system on each set of nine sequentially, using the remaining one as a test. The average performance is then computed. Not as good as leave-one-out owing to sampling bias in division into ten groups, however generally much quicker to perform
- Holdout validation: a subset of observations is chosen randomly from the initial sample to form a testing set, and the remaining observations are retained as the training data

Additional Notes

- The LZ algorithm only takes more than one character if that character has been used at least once (i.e. won't do so straight from the alphabet)