Text processing and machine learning

Working with words:

Natural language processing
Information retrieval
Machine learning

Reminder of yesterday

- Foundations of network analysis
- Representing and describing networks
- Network measures
- Community detection
- Network visualisation
- Tools
- Projects: do you have one?

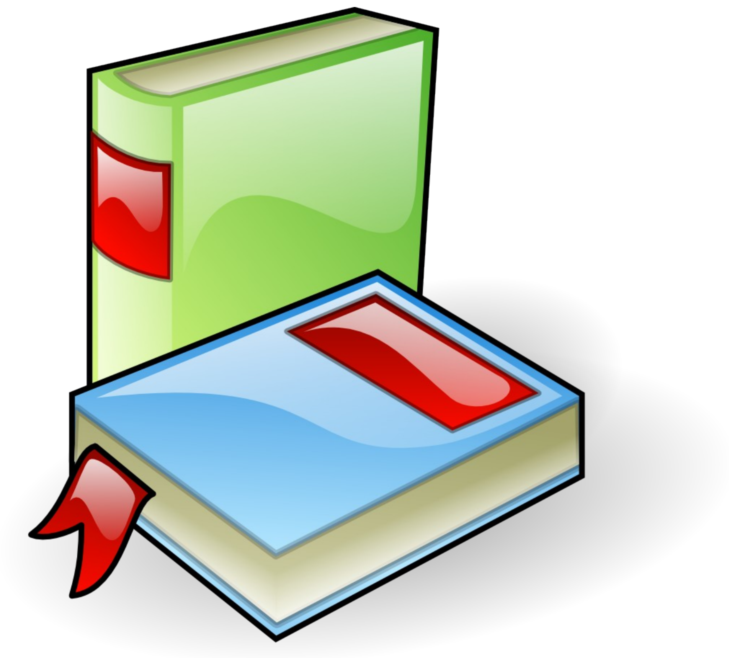
Working with words

- What are people talking about?
- How do people interact?
- What sort of language do they use?
- What are people happy about? Sad? Upset?
- Where are there disagreements?

Outline of today

- Natural language processing (NLP)
- Information retrieval (IR)
- Machine learning (ML)
- Practical session with NTLK and Python

Natural language processing



Overview

- What is natural language processing?
- Basics
- Segmentation
- Morphology
- Lexical normalisation
- Collocations and statistical models
- Part-of-speech tagging

What is NLP?

Computer processing of **natural** languages: the languages written or spoken by people.

- Messy; ambiguous; varied; changing
- We will focus on NLP for social media and social science questions

Text

Electronic text comes from lots of sources:

- Newswires, the web, chat rooms, business documents, email…
- Needs to be cleaned

In many formats:

- OCR, PDF, XML, HTML, binary formats, …

Word, **token**, **term**, **lexeme**

Part of speech (POS)

Tokenisation

We want to reduce **text** to **words** for processing.

```
i am pleased to be in beijing

I am pleased,i am in BJ

don't won't I'll he'd

O'Connor

New York

+61 (0) 2 6216 7065

:-)

22.50 元
```

H_2O

Segmentation

严守一把手机丢了

ภาษาเขียน

ພາສາລາວ

tiếng Việt

arbeidsongeschiktheidsverzekering

Donaudampfschiffahrtselektrizitätenhauptbetriebswerkbauunterbeamtengesellschaft

Segmentation

- 严守一 / 把 / 手机 / 丢了

  (Yan Shouyi lost his mobile phone)

- 严守 / 一把 / 手机 / 丢了

  *(*It strictly adheres to a lost mobile phone)*

- 严守 / 一把手 / 机丢了

  *(*Strictly number one machine lost)*

How can we do this?

- Treat every character as a word
- Always take the longest match
- Conditional random fields

  Iterate over all characters, finding most likely break points

  Probability of a break is determined by a weighted sum of “features”

  Features can include character identity and some amount of history

  This is a **probabilistic method**.

Morphology and stemming

We need to recognise different forms of words:

```
love this restaurant
loved this restaurant
loving these restaurants
```

- Not such a problem in Chinese(?)
- Sometimes a problem in English
- A real problem in some other languages

Stemming and lemmatisation

Stemming turns words into **stems**, which are the same regardless of inflection

- Stems need not be real words!, e.g. Porter stemmer:

sses → ss	caresses → caress
ies → i	ponies → poni
s → <empty>	cares → care
(m > 1) ation → ate	predication → predicate *nation → nate*

Lemmatisation turns words into **lemmas**, which are dictionary entries

Collocations

Two or more words that act as a unit, syntactically and semantically:

`mobile phone`, `weapons of mass destruction`, `broad daylight`, `kick the bucket`, `to run out`, `James Bond`

- Non-compositional
- Non-substitutable
- Non-modifiable

How to find them? (1)

- Just count?

```
of the
in the
to the
on the
for the
```

How to find them? (2)

- Just count?
- Count, but filter by POS?

```
New York (AN)
United States (AN)
Los Angeles (NN)
last year (AN)
Saudi Arabia (NN)
```

How to find them? (3)

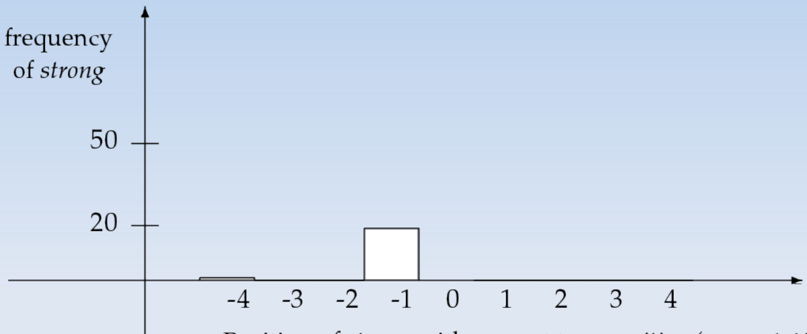
- Just count?
- Count, but filter by POS?
- Mean and variance of relative position?

… **`previous`** `fifteen` **`games`** …   (d=2)

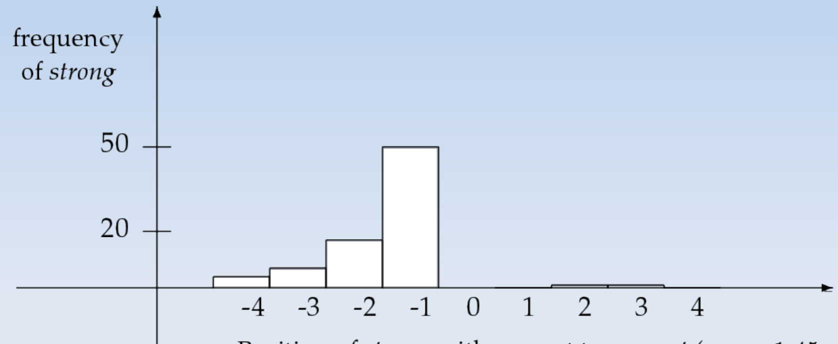
… **`previous`** **`games`** `were lost` … (d=1)

… **`games`** `in` **`previous`** `times` …   (d=-2)

Mean and variance

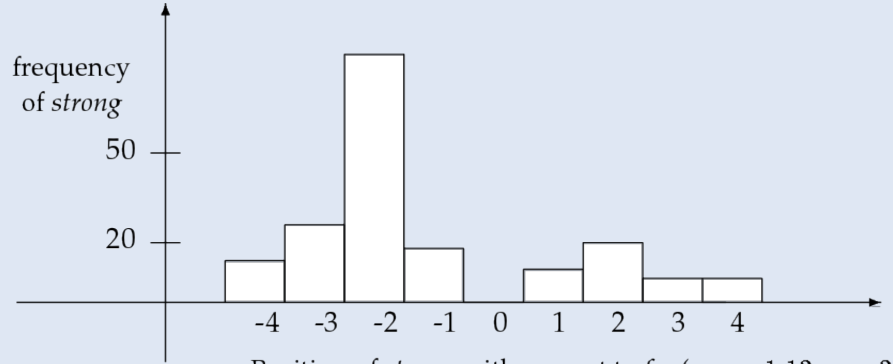


Position of *strong* with respect to *opposition* ($\mu = -1.15, \sigma = 0.67$).



Position of *strong* with respect to *support* ($\mu = -1.45, \sigma = 1.07$).

```
New York          (d≈1)
previous games    (d≈2)
minus points      (d≈3)
hundreds dollars  (d≈4)
```



Position of *strong* with respect to *for* ($\mu = -1.12, \sigma = 2.15$).

Probability

P(X) is the probability of some **event** X. $\overline{X}$=“not X”

P(X|Y) is the **conditional** probability of event X, given that the event Y occurs:

P(X|Y) = P(X and Y) / P(Y)

Two events are **independent** iff

P(X and Y) = P(X) P(Y)

…and therefore P(X|Y) = P(X) unless P(Y) = 0

How to find them? (4)

Statistical hypothesis testing:

- Have H_0, the **null** hypothesis: “no difference”
- Calculate $P(X \mid H_0)$
- If this is less than e.g. 5%, reject H_0

H_0: Words w_1 and w_2 are independent

- $P(w_1,w_2) = P(w_1)\ P(w_2)$

Part-of-speech tagging

We might also want to “tag” words with their **part of speech (POS)**. Why?

- Disambiguation
- Extracting noun phrases (subjects and objects)
- Information extraction
- Units of indexing and retrieval

How to tag?

- Static, most common class
- Hand-written rules
- Again, use probabilistic techniques

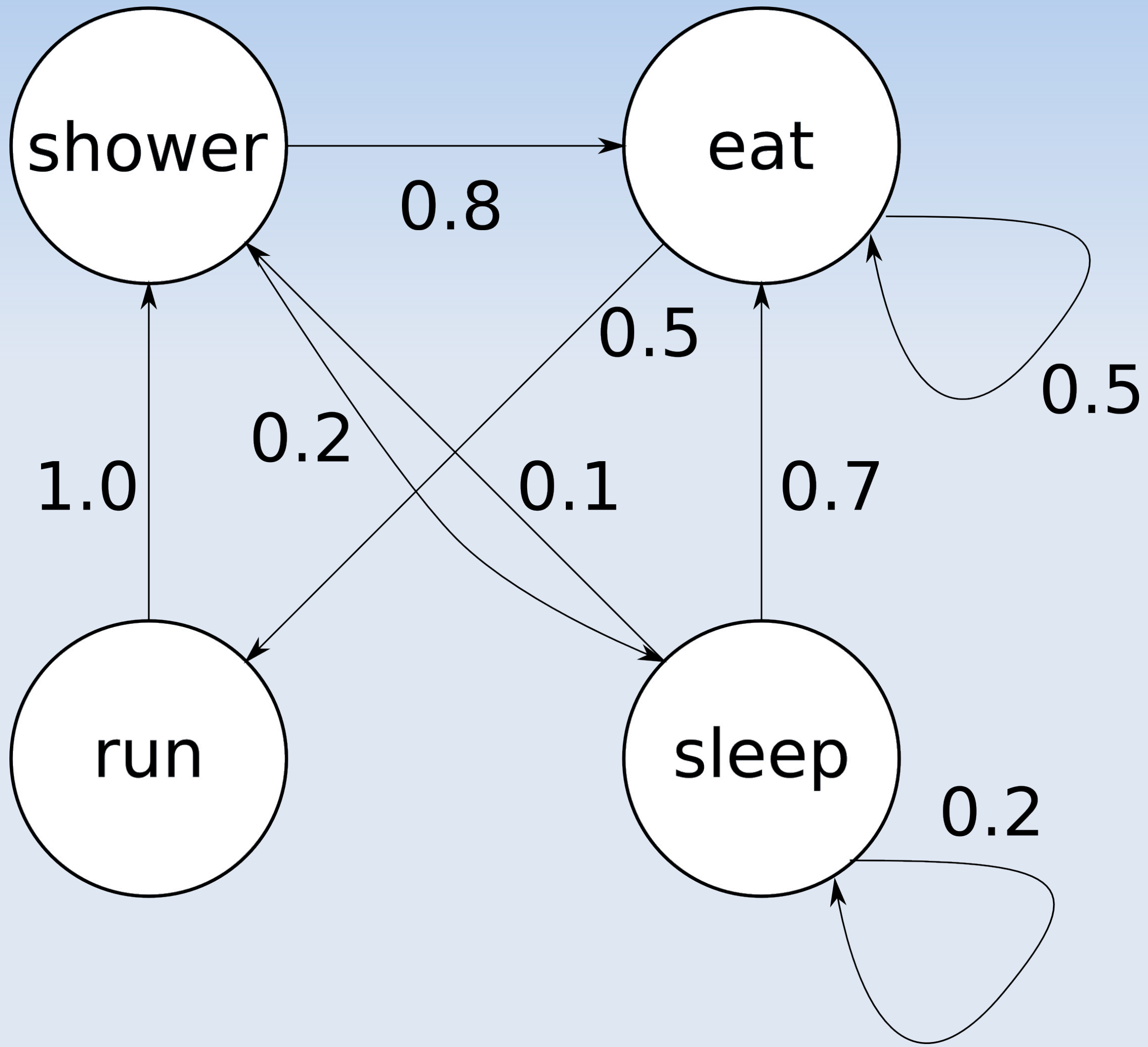
Markov Models (background)

We have:

- $S = \{S_1 \dots S_k\}$, a set of states
- A, a transition matrix, $a_{ij}=P(X_{t+1}=j \mid X_t=i)$
- Π, the initial state vector
- $X = (X_1 \dots X_t)$, a sequence of values from S

Models have **limited horizon** and are **stationary**.

Markov Models are NFAs



Hidden Markov Models

Now a twist: **emit outputs** as we leave a state, but do this probabilistically.

- Output is from a set K
- B describes the output from each edge:

  $b_{ijk} = P(O_t=k \mid X_t=s_i, X_{t+1}=s_j)$

A **Hidden Markov Model** = (S, K, Π, A, B)

- We can't see the states (X), just the output (O)

HMMs for POS tagging

- States (S) = parts of speech
- Output (O) = sequence of words

Question: **given O, what is the most likely X?**

- The Viterbi algorithm effectively computes this

We can get A and B from training data

Named entities

HMMs (and CRFs) can also be used to find **named entities**, e.g. the names of people, places, or corporations.

Features include:

- In English: orthography, sequences, prefixes
- In Chinese: lists of family names (and rules), marker words like 公司 or 市, bigrams

Some example code (1)

```
>>> import nltk
>>> from nltk.book import *
>>> text4.collocations()
Building collocations list
United States; fellow citizens; four years;
years ago; Federal Government; General
Government; American people; Vice President;
Old World; Almighty God; Fellow citizens;
Chief Magistrate; Chief Justice; God bless;
every citizen; Indian tribes; public debt;
one another; foreign nations; political
parties
```

See the nltk collocations module: http://nltk.org/api/nltk.html

Some example code (2)

```
>>> import nltk
>>> text = nltk.word_tokenize("once upon a time in a hole in the ground there lived a hobbit.")
>>> nltk.pos_tag(text)
[('once', 'RB'), ('upon', 'IN'), ('a', 'DT'), ('time', 'NN'), ('in', 'IN'), ('a', 'DT'), ('hole', 'NN'), ('in', 'IN'), ('the', 'DT'), ('ground', 'NN'), ('there', 'RB'), ('lived', 'VBN'), ('a', 'DT'), ('hobbit', 'NN'), ('.', '.')]
```

See the nltk tag package: http://nltk.org/api/nltk.tag.html

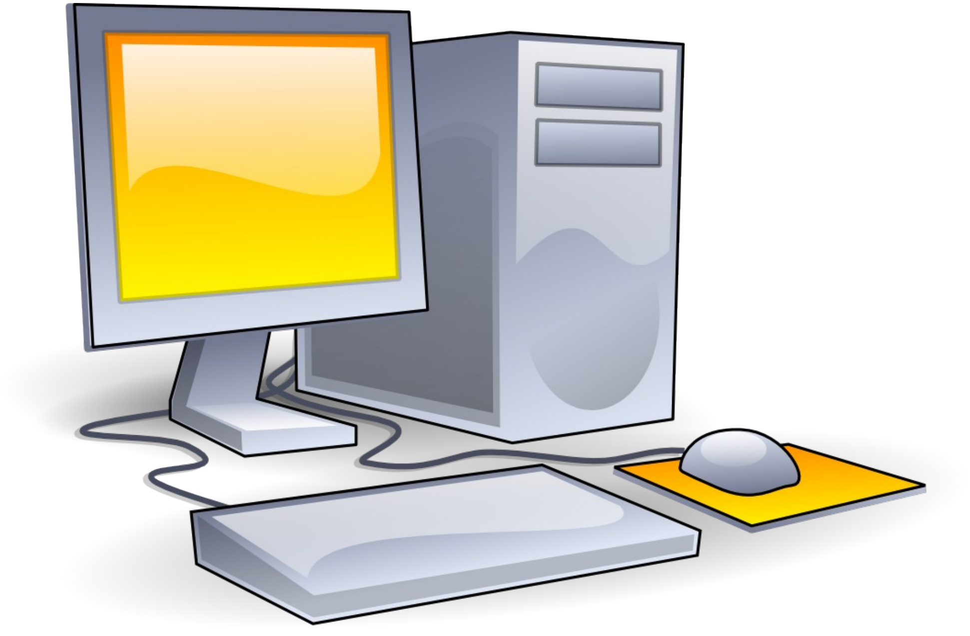
Some example code (3)

```
>>> s="Germany's representative to the
European Union is Herr Smith."

>>> tokenised = nltk.word_tokenize(s)

>>> tagged = nltk.pos_tag(tokenised)

>>> print nltk.ne_chunk(tagged)

Tree('S', [Tree('GPE', [('Germany', 'NNP')]),
("'s", 'POS'), ('representative', 'NN'),
('to', 'TO'), ('the', 'DT'),
Tree('ORGANIZATION', [('European', 'NNP'),
('Union', 'NNP')]), ('is', 'VBZ'),
Tree('PERSON', [('Herr', 'NNP'), ('Smith',
'NNP')]), ('.', '.')])
```

Summary

- To get from bytes to “meaning” we can use a variety of tools: we need to extract the text, tokenise, segment, and we might want to deal with morphology.
- We can find parts of speech and named entities.
- Statistical and probabilistic techniques help us deal with the noisy and changing nature of natural language.

Information retrieval



Overview

- What is information retrieval?
- Term occorrence matrix
- Ranked retrieval
- The vector-space model and tf.idf
- External evidence

What is information retrieval (IR)?

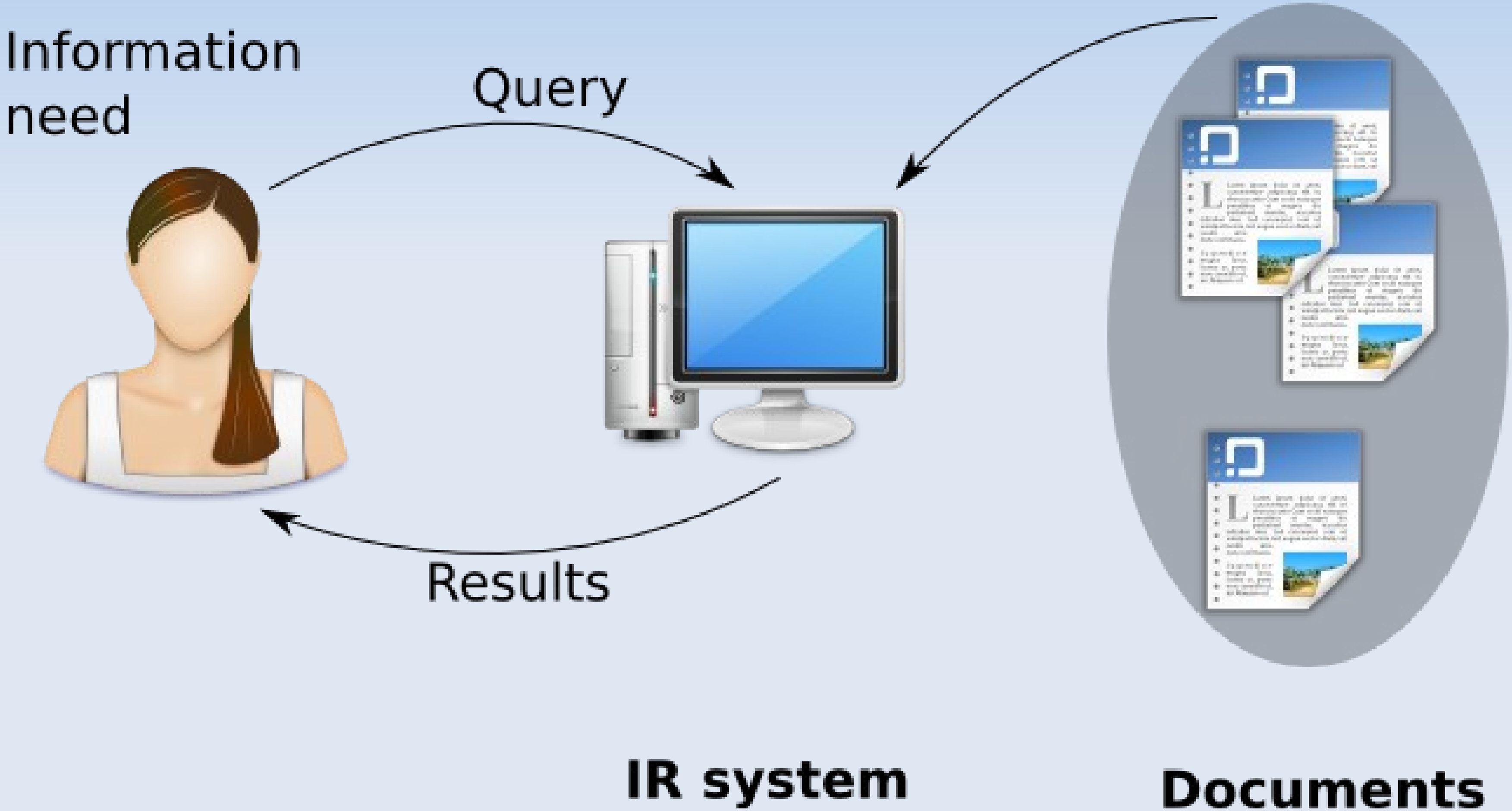
...finding material (usually documents)
of an unstructured nature (usually text)
that satisfies an information need
from within large collections
(usually stored on computers)

(Manning et al.)

What is information retrieval (IR)?

...in response to
an *underspecified information need*

What is information retrieval (IR)?



Information needs and queries

"I'm going to be in Beijing for a
few days in July and I'd like to
find something to do in my spare
time. Ideally it'd be walking
distance from my hotel (at …)
and I don't want
more than $50 alth
the theatre or something like that
I might spend more. Also, I've
already seen Tiananmen Square,
the Forbidden City, and the
Confucius Temple. Oh!, and I
enjoy discovering local food and

beijing attractions

How can we search?

To find documents with certain keywords, we can try:

- full text scanning
- document signatures
- term occurrence matrix

These are all ways to *represent documents.*

Term occurrence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	0	1
wowser	1	0	1	1	0	0

...

Term occurrence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	**1**	**1**	**0**	**0**	**0**	**1**
Brutus	1	1	0	1	0	0
Caesar	**1**	**1**	**0**	**1**	**1**	**1**
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	0	1
wowser	1	0	1	1	0	0
Antony AND Caesar	**1**	**1**	**0**	**0**	**0**	**1**

Term occurrence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	0	1
wowser	1	0	1	1	0	0
Brutus AND Caesar AND NOT Calpurnia						

Ranked retrieval

We can rank documents according to some **scoring function** and put the “best” at the top

Coordination level
Term weights
Vector space model

Scoring by coordination level

- initialise *scores* array to all 0
- for each query term *t* in the query *q*:
  - for each document *d* which includes *t*:
    - *scores*[*d*] += 1
- return top *k* entries of *scores*

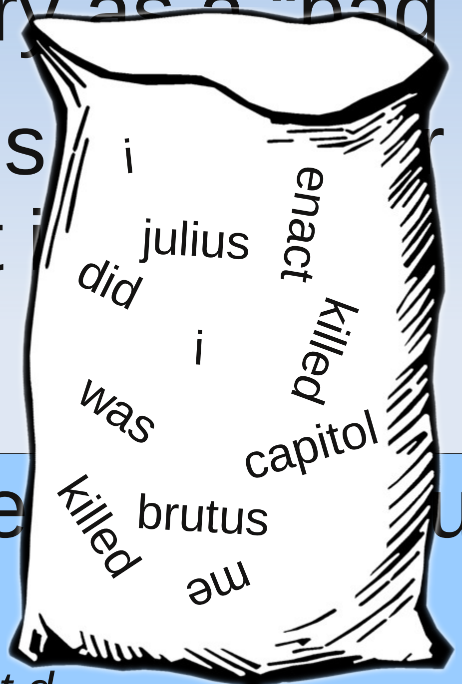
Scoring with term weights

- initialise *scores* array to all 0
- for each query term *t* in the query *q*:
  - for each document *d* which includes *t*:
    - *scores*[*d*] += weight of *t*
- return top *k* entries of *scores*

Term frequency

Step 1: We treat the query as a “bag of words”.

Step 2: We assume terms … r more often capture a more important i…



The **term frequency** of te… ument *d*,

$$\mathrm{tf}_{t,d}$$

is the number of times *t* appears in *d*.

Document frequency

tf gives us one measure of importance, but not all terms are the same—not even all common ones.

Step 3. We assume that terms which appear in fewer documents are more discriminating.

The **document frequency** of term *t*,

$$\mathrm{df}_t$$

is the number of documents *t* appears in.

Inverse document frequency

Then the **inverse document frequency** of term *t* is based on the reciprocal of df_t:

$$\mathrm{idf}_t = \log(N / \mathrm{df}_t)$$

tf.idf

$$\text{tf.idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

- ▲ High when *t* occurs many times in few documents.
- ▼ Low when *t* occurs in many documents.
- ► In between when *t* occurs a few times, and/or in many documents.

tf.idf weights

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	81	15.3	0	0	0	0.6
Brutus	7	8.6	0	0.3	0	0
Caesar	1	93	0	0.1	0.2	0.2
Calpurnia	0	4	0	0	0	0
Cleopatra	75	0	0	0	0	0
mercy	1.8	0	6.3	11	0	1
wowser	1	0	2	3	0	0

...

Simple scoring by tf.idf

- Initialise *scores* array to all 0
- For each query term *t* in the query *q*:
  - For each document *d* which includes *t*:
    - Calculate $\text{tf.idf}_{t,d}$
    - $Scores[d]$ += $\text{tf.idf}_{t,d}$
- Return top *k* entries of *scores*

Coffee break (30min)

tf.idf weights

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	81	15.3	0	0	0	0.6
Brutus	7	8.6	0	0.3	0	0
Caesar	1	93	0	0.1	0.2	0.2
Calpurnia	0	4	0	0	0	0
Cleopatra	75	0	0	0	0	0
mercy	1.8	0	6.3	11	0	1
wowser	1	0	2	3	0	0
...						

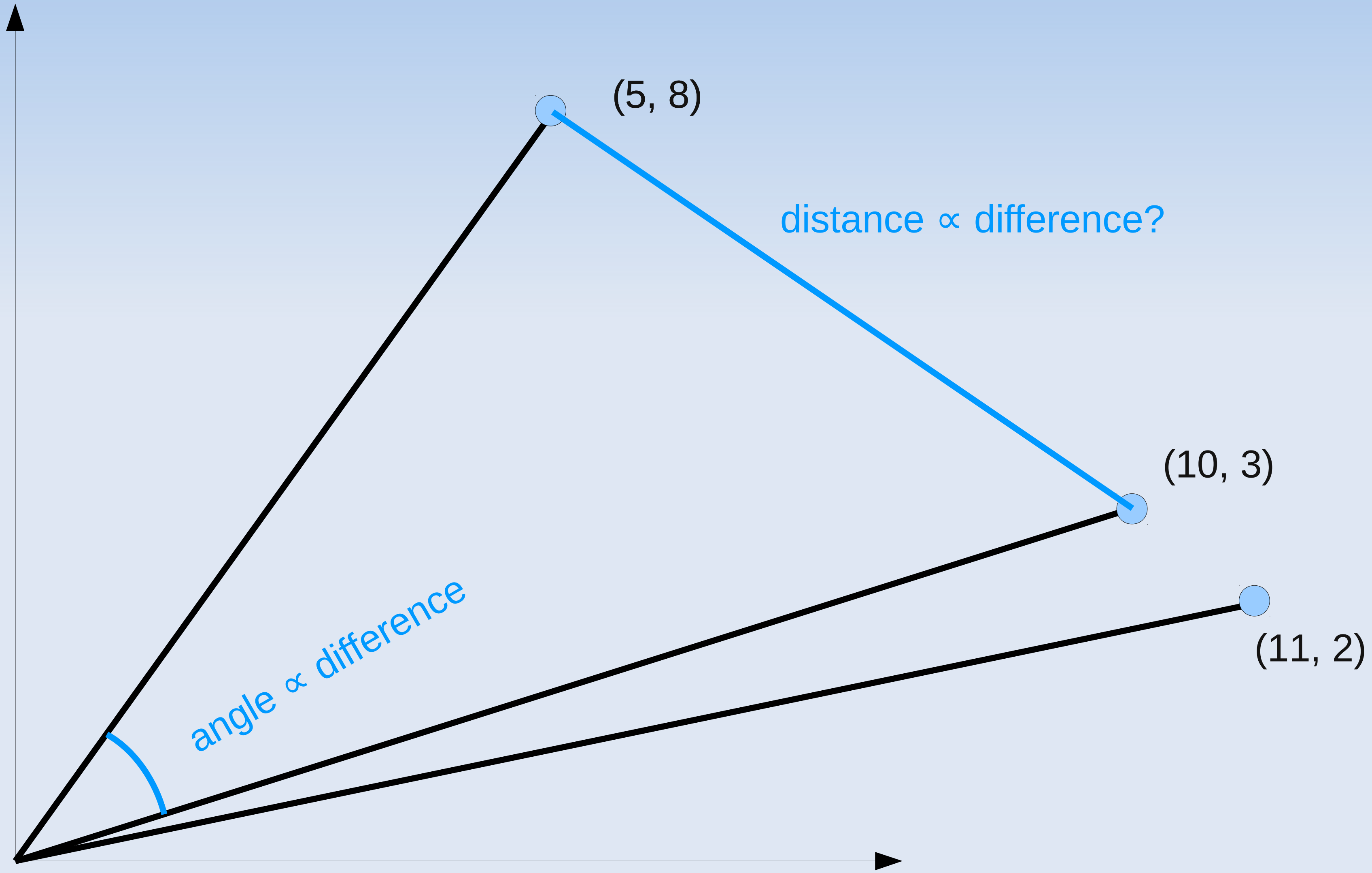
Vectors

$$\text{vector } \vec{x} = (x_1, x_2, x_3, \ldots)$$

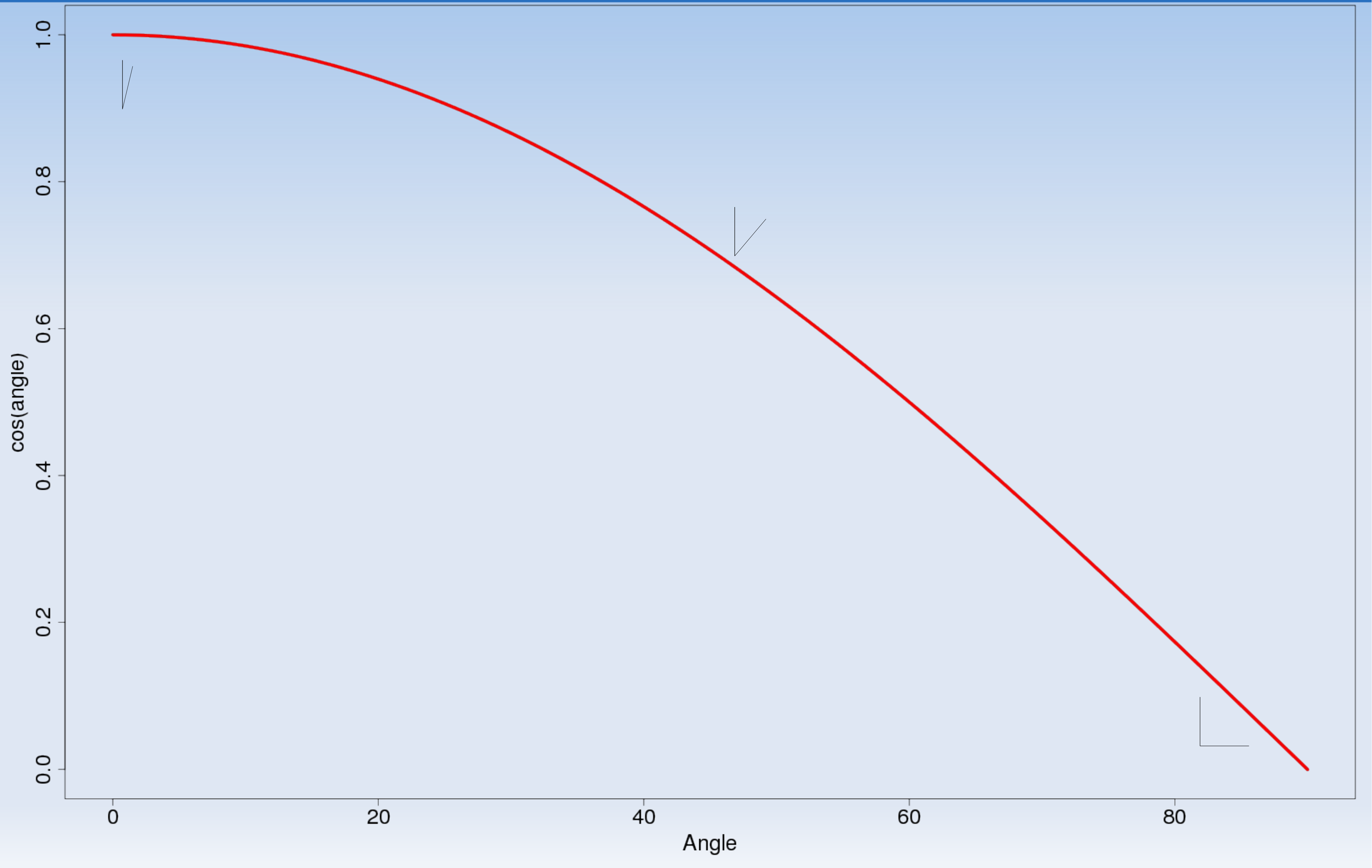
$$\text{document vector } \vec{d} = (w_{t1,d}, w_{t2,d}, w_{t3,d}, \ldots)$$

$$\vec{AC} = (81, 7, 1, 0, 75, 1.8, 1, \ldots)$$

Documents as vectors



Cosine



Cosine similarity

Vector dot product:

$$\vec{x} \cdot \vec{y} = \sum_i x_i y_i$$

Length normalisation:

$$\hat{x} = \frac{\vec{x}}{\|\vec{x}\|} \text{ and } \hat{y} = \frac{\vec{y}}{\|\vec{y}\|}$$

Similarity follows:

$$\mathrm{sim}(\vec{x}, \vec{y}) = \cos\theta = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|\|\vec{y}\|} = \hat{x} \cdot \hat{y}$$

Scoring by cosine similarity

- Initialise *scores* array to all 0
- For each query term *t* in the query *q*:
  - Calculate query weight $w_{t,q}$
  - For each document *d* which includes *t*:
    - Calculate term weight $w_{t,d}$
    - $Scores[d] \mathrel{+}= w_{t,d} w_{t,q}$
- Return top *k* entries of *scores*

Cluster hypothesis

Documents that are similar to each other are likely to be relevant to the same information need

→ Documents which are close in vector space are likely to describe the same topic

→ Documents which are close to a query are likely to be relevant to that query

Centrality

Incorporating centrality

Centrality is **query-independent evidence**: it is the same for any query.

Can simply combine this with **query-dependent** evidence such as probability of relevance, cosine distance, term counts, …

score(d,q) = α PageRank(d) + (1-α) similarity(d,q)

Other forms of evidence

- Trust in or authority of the host (or domain, or domain owner, or network block)
- Frequency or recency of updates
- URLs
- Language
- Centrality or other graph measures
- ...

Summary

- We often want to find a subset of documents according to **topic**, which we assume means **word(s)**.
- We can use Boolean models …
- … but vector-space models tend to work better
- We can weight terms, use similarity functions
- And we can mix in other forms of evidence

Practical session: get the data

Get a copy of `reuters.zip`

- Linux or Mac: put it in `/usr/share/nltk_data/corpora` or in `~/nltk_data/corpora`
- Windows: put it in `c:\nltk_data\corpora`

Test it: run python and type

```
>>> from nltk.corpus import reuters
>>> len(reuters.words())
1720901
```

Lunch break (90min)

Machine learning

Overview

- What is machine learning?
- Supervised and unsupervised methods
- Classification
- Clustering
- Dealing with high-dimensional data

What is machine learning?

Getting machines to learn from data:

- Finding patterns
- Recognising objects (or documents or…)

We will look at two common problems: **classification** and **clustering**.

(Un)supervised learning

- **Supervised learning:** Given some examples, with the “right answers” provided, learn how to generalise.
- **Unsupervised learning:** Given some examples (only), find some patterns in the data.

Classification

- We want to learn to **classify** documents.
- For example by topic, or language, or viewpoint, or…
- This is a **supervised** problem.

Classification methods

Manually? Doesn't scale!

Training set D:

⟨“I am happy”, *positive*⟩

⟨“Hate Mondays”, *negative*⟩

Classifier:

γ(“Happy it's a Monday”)=?

Instea

- Naï
- k-N
- Support vector machines

Given a document space X, classes C, training set D ⟨d,c⟩ ∈ D×C, learn a classifier γ: X→C

Probability (a reminder)

P(X) is the probability of some **event** X. $\overline{X}$=“not X”

P(X|Y) is the **conditional** probability of event X, given that the event Y occurs.

Bayes's rule: $\mathrm{P}(X|Y)=\dfrac{\mathrm{P}(Y|X)\,\mathrm{P}(X)}{P(Y)}$.

Bayes's rule in action

We want:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \mathrm{P}(c \mid d)$$

$$= \underset{c \in C}{\operatorname{argmax}} \; \frac{\mathrm{P}(d \mid c)\,\mathrm{P}(c)}{\mathrm{P}(d)}$$

$$= \underset{c \in C}{\operatorname{argmax}} \; \mathrm{P}(d \mid c)\,\mathrm{P}(c)$$

Conditional independence

We need to estimate P(*d* | *c*) , the probability of document *d* given class *c*. But how can we do this?

Assume features/terms are **independent:**

Independent: P(X and Y) = P(X) P(Y)

$$P(d|c) = P(\langle t_1, t_2, \ldots \rangle | c)$$

$$= \prod_{i=1..n} P(t_i | c)$$

Most likely class

Now we know the **maximum a prosteriori (MAP)** estimate for a document.

$$
\begin{aligned}
\gamma(d) &= \hat{c} \\
&= \underset{c \in C}{\operatorname{argmax}}\; \mathrm{P}(c \mid d) \\
&= \underset{c \in C}{\operatorname{argmax}}\; \mathrm{P}(c) \prod_{i=1..n} \mathrm{P}(t_i \mid c) \\
&= \underset{c \in C}{\operatorname{argmax}}\; \log(\mathrm{P}(c)) + \sum_{i=1..n} \log(\mathrm{P}(t_i \mid c))
\end{aligned}
$$

Naïve Bayes classifier

$$\gamma(d) = \underset{c \in C}{\operatorname{argmax}} \ \log(\mathrm{P}(c)) + \sum_{i=1..n} \log(\mathrm{P}(t_i | c))$$

Where:

- Each $\mathrm{P}(t_i \mid c)$ tells us what evidence t_i provides for class c
- $\mathrm{P}(c)$ tells us the relative frequency of c
- And we choose the class with the best evidence.

Parameters

How do we estimate all these probabilities?

Maximum likelihood estimates (MLEs):

"I like vegetables" → "healthy"
"I like ice-cream and sauce" → "unhealthy"
"I like vegetables and rice" → ???

mples

$$\hat{P}(t|c) = \frac{T_{t,c}}{\sum_{t' \in V} T_{t',c}}$$

Number of times t appears in class c

Whole vocabulary

$\hat{P}$("rice" | healthy) = ???

Zero!

For any word we haven't seen before, $T_{t,c}=0$

Therefore we will get zero probabilities P(*c* | *d*), for every class!

We are saying "the probability of seeing term *t* in class *c* is zero". Is this sensible?

We should allow this and "set aside" some of the probability space. **"Add-one smoothing":**

$$\hat{P}(t|c)=\frac{T_{t,c}+1}{\sum_{t'\in V}(T_{t',c}+1)}$$

Some example code (1)

```
>>> from nltk.corpus import movie_reviews
>>> import random
>>> documents = [(list(movie_reviews.words(fileid)), category)
...     for category in movie_reviews.categories()
...     for fileid in movie_reviews.fileids(category)]
>>> random.shuffle(documents)
```

For each category …
… get all the file IDs

For each file ID, list all the
words and make a pair
(list-of-words, category)

Some example code (2)

Load all the words in all the reviews, but only use the top 2000

```
>>> all_words = nltk.FreqDist(w.lower()
...   for w in movie_reviews.words())
>>> word_features = all_words.keys()[:2000]
>>> def document_features(document):
...   document_words = set(document)
...   features = {}
...   for word in word_features:
...     features['contains(%s)' % word] = (word in document_words)
...   return features
```

For each word in our top 2000, make a feature

Some example code (3)

```
>>> featuresets = [(document_features(d), c)
...   for (d,c) in documents]
>>> train_set, test_set = featuresets[100:], featuresets[:100]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> nltk.classify.accuracy(classifier, test_set)
0.8
```

For each document, use our new function to make a pair
(features, category)

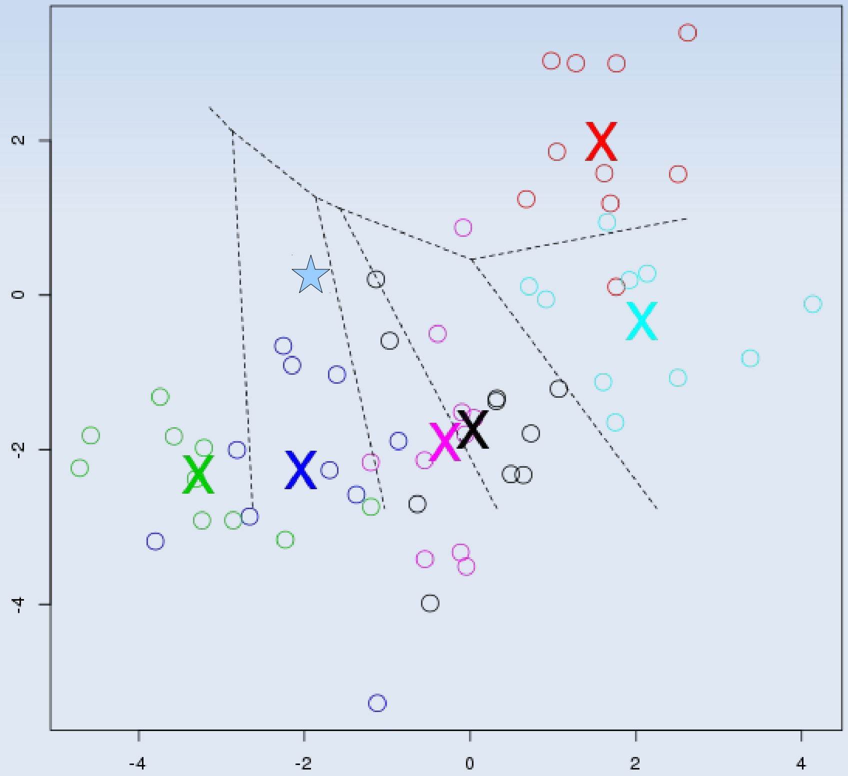
Split into test & training sets, train, then report accuracy

Classification in vector space

- Assume that documents in the same class form a region in vector space,
- …and that these regions don't overlap.

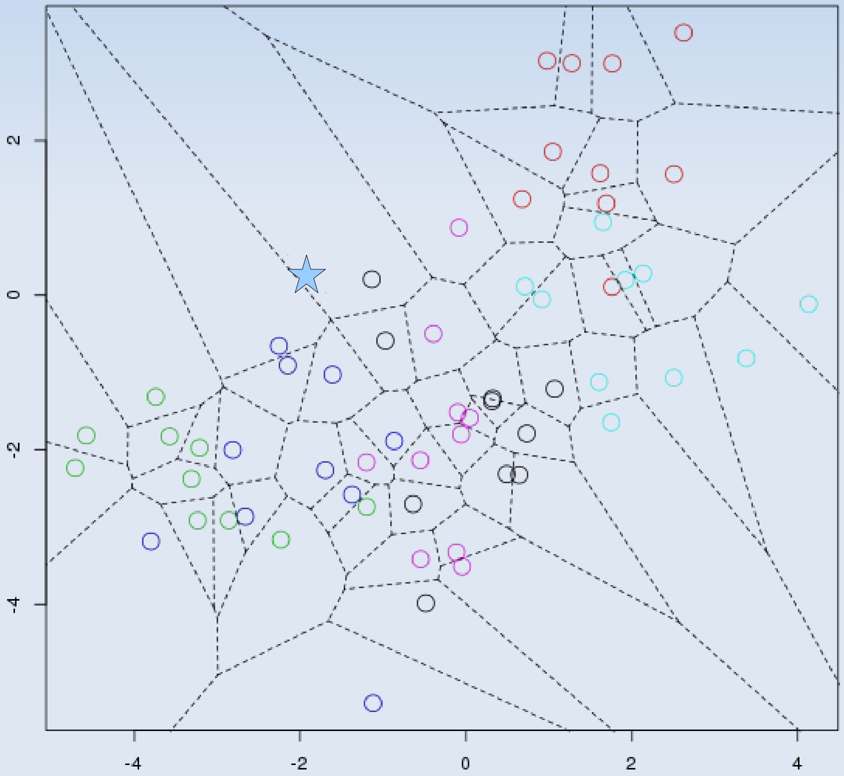
(This is the contiguity hypothesis.)

Rocchio



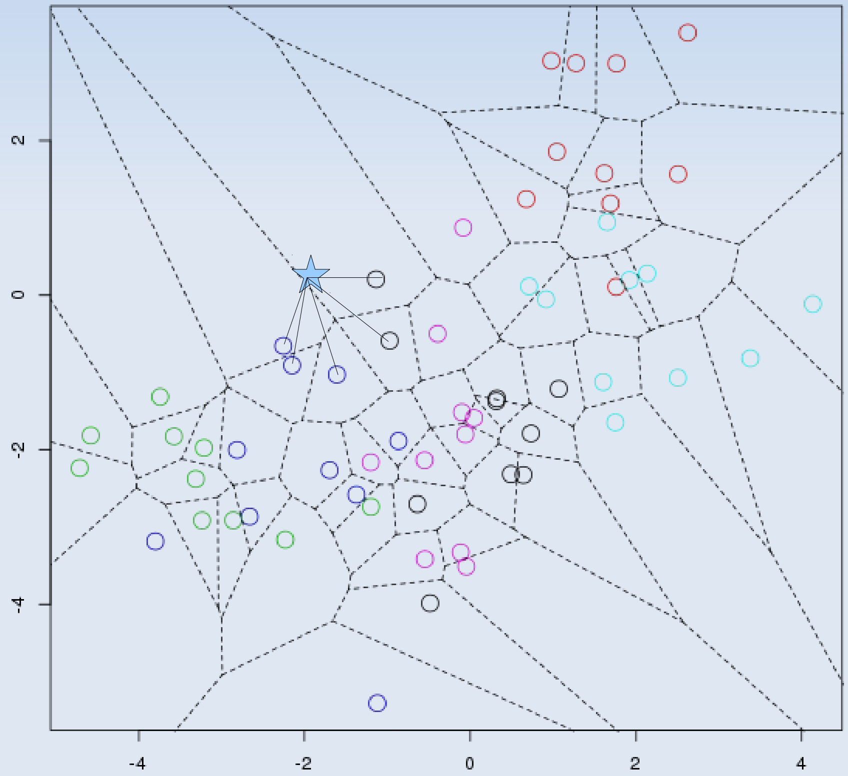
- Get the centroid of each class (mean value of each feature)
- Assign a new document *d* to the class of its closest centroid.
- Problems?

1-NN



- Assign each new document to the same class as its nearest neighbour.
- Problems?

k-NN



- Assign each new document to the majority class amongst its nearest *k* neighbours.
- k-NN tends to be a good choice.

Other choices

- Support vector machines (SVMs)

  Try to find a separating hyperplane

- Logistic regression

  Linear regression, but with response variable mapped to binary via logit function

More than two classes?

- One-of-*n*: an object is in exactly one class

  Run a classifier for each class and take the most probable

- Any-of-*n*: classes are independent

  Just run a classifier for each class

Evaluation

If we do have labelled data, we can hold some back: have a **training set** and a **testing set**.

Then we can compare our predictions with the true labels and ask: how often do we get it right?

- Can use a “confusion matrix”:

	Actual	
Predict	**Yes**	**No**
Yes	tp	fp
No	fn	tn

Measures

- Accuracy: (tp+tn) / (tp+fp+fn+tn)
- Precision, P: tp / (tp+fp)
- Recall, R: tp / (tp+fn)
- F1: (2PR) / (P+R)

	Actual	
Predict	**Yes**	**No**
Yes	tp	fp
No	fn	tn

Overfitting

A word of caution: never evaluate on the same data you trained on!

- We want to know how well it works on **new** data (how well it predicts outcomes)
- It is very easy to **overfit**, i.e. learn quirks of the training set instead of general rules

Typically we take about 10% as a **testing set**: don't look at it!

Can even repeat this (“cross-validation”).

Clustering

We have a lot of unlabelled data; are there natural groups?

- Are there animals that tend to live together?
- Are there countries with similar economies?
- Are there people who talk about the same things?

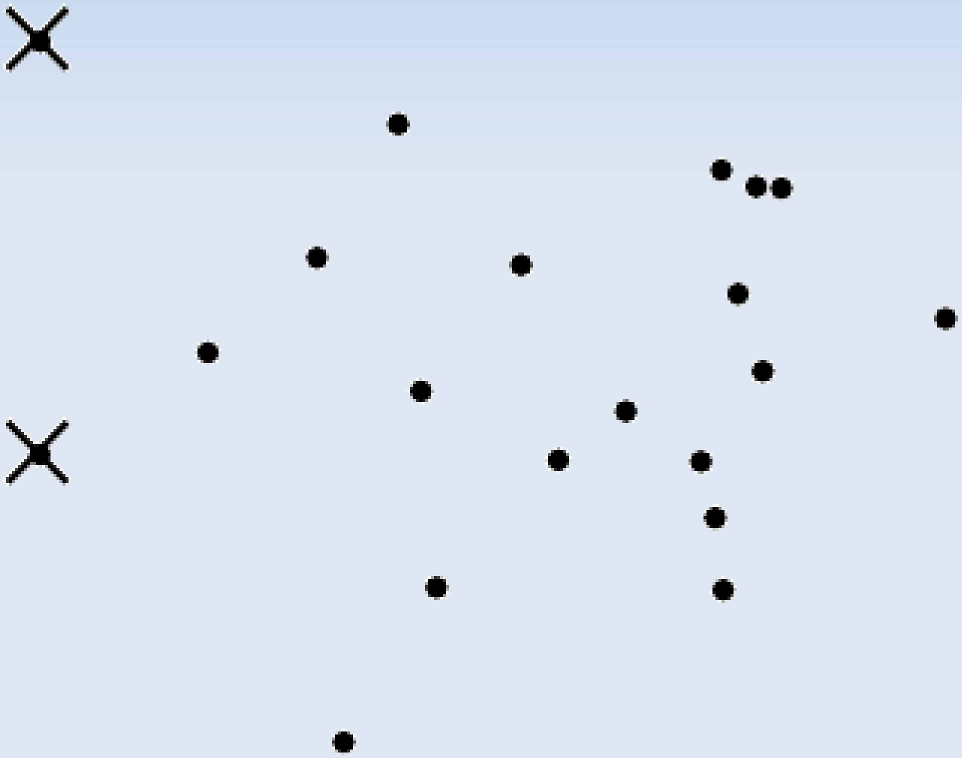
This is an **unsupervised** problem.

k-means

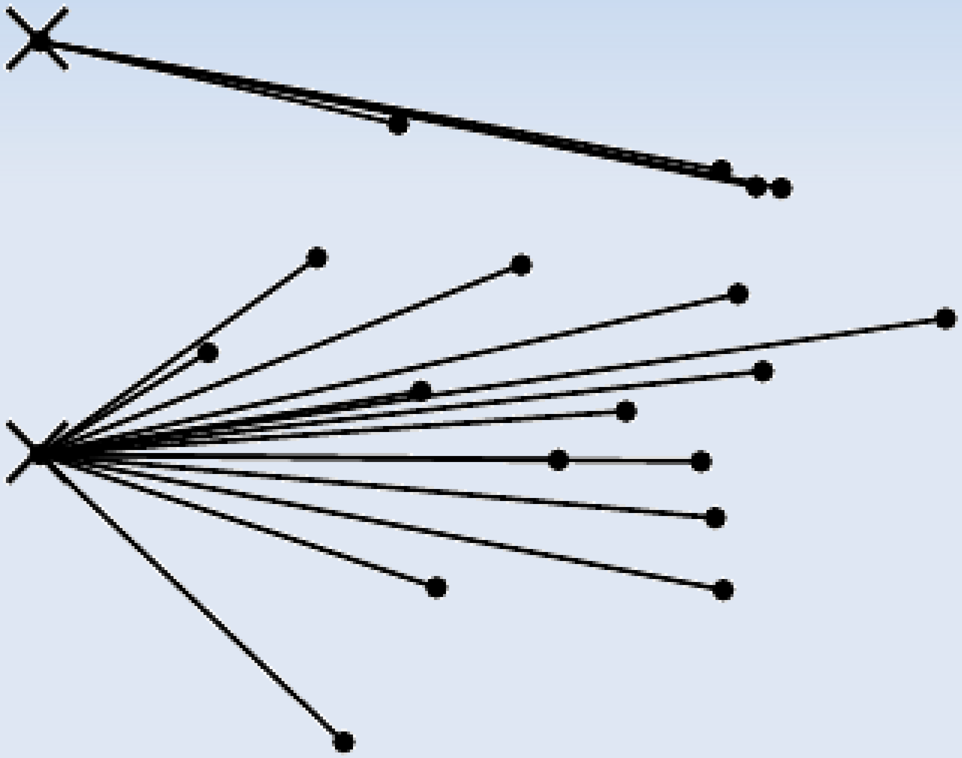
We need to know how many clusters we want.
Then:

- Choose starting points (centroids) for the clusters, randomly
- Assign each object to the nearest centroid
- Recalculate the centroids
- Repeat until convergence

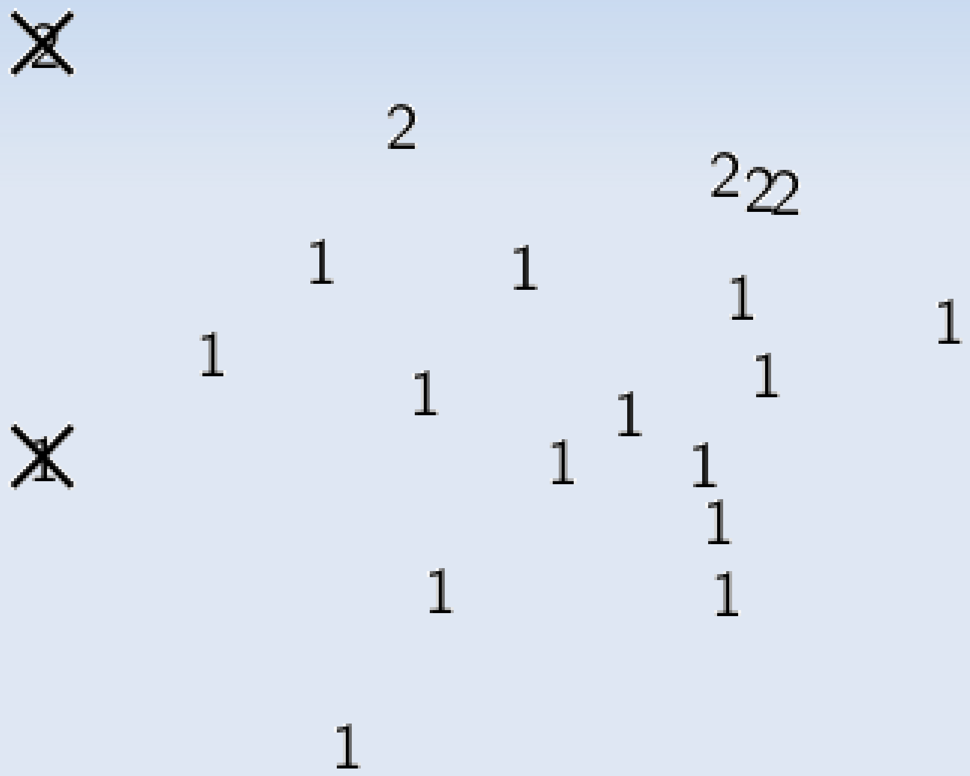
k-means



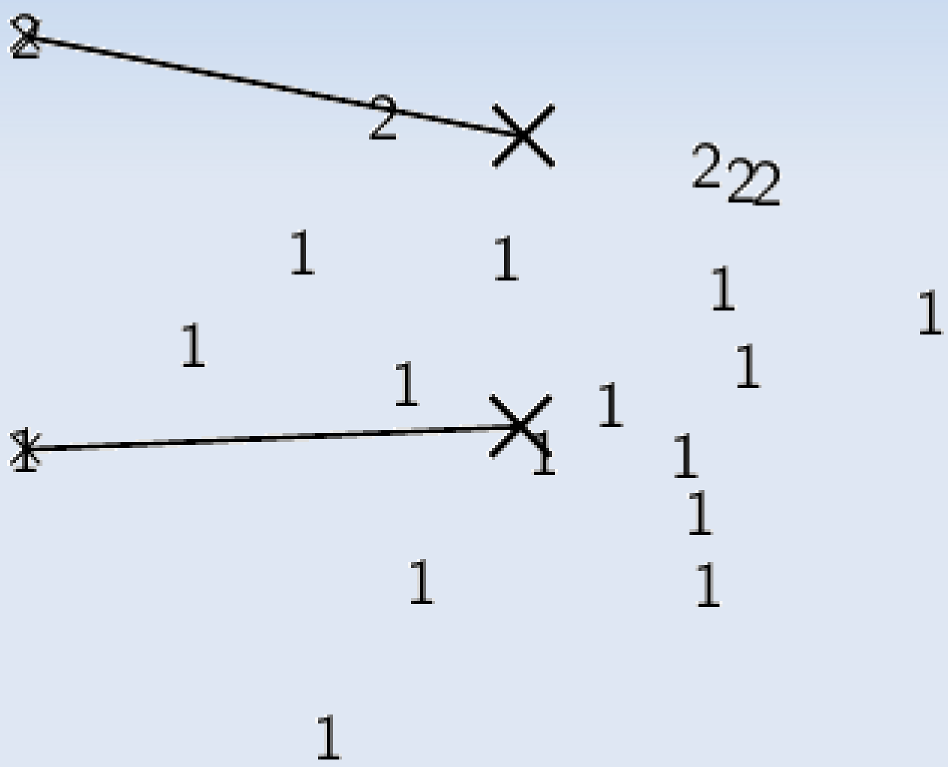
k-means



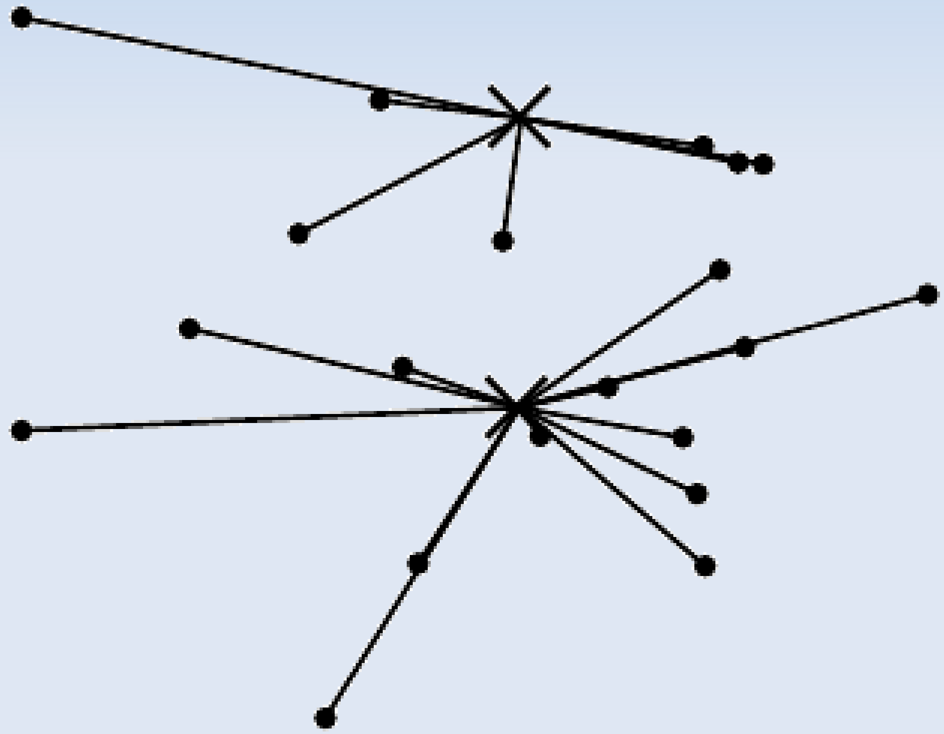
k-means



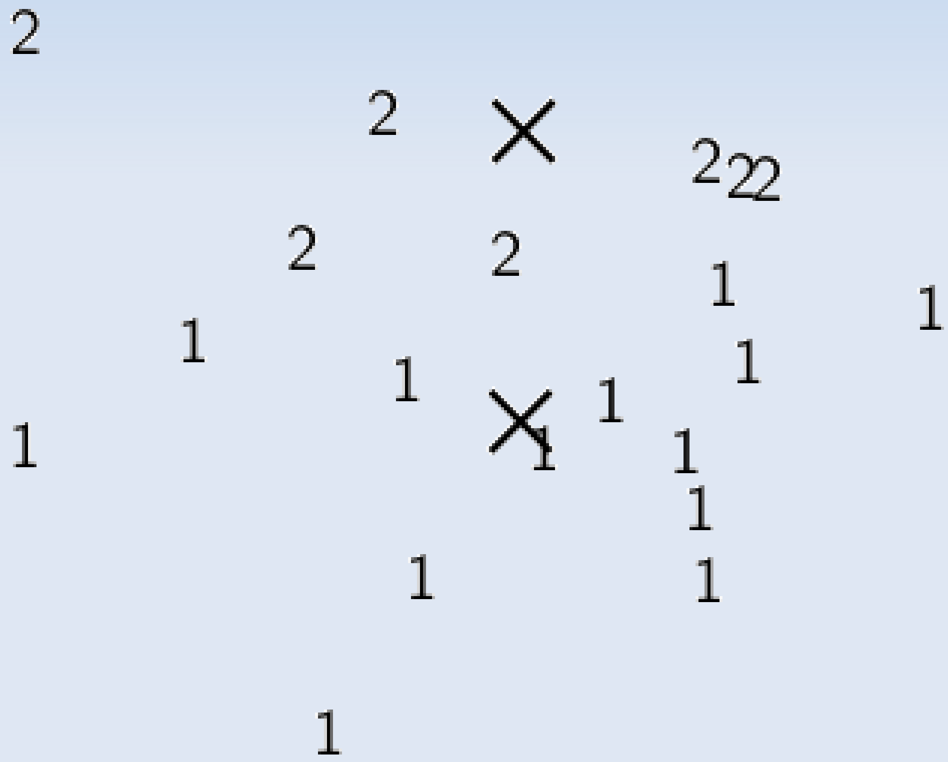
k-means



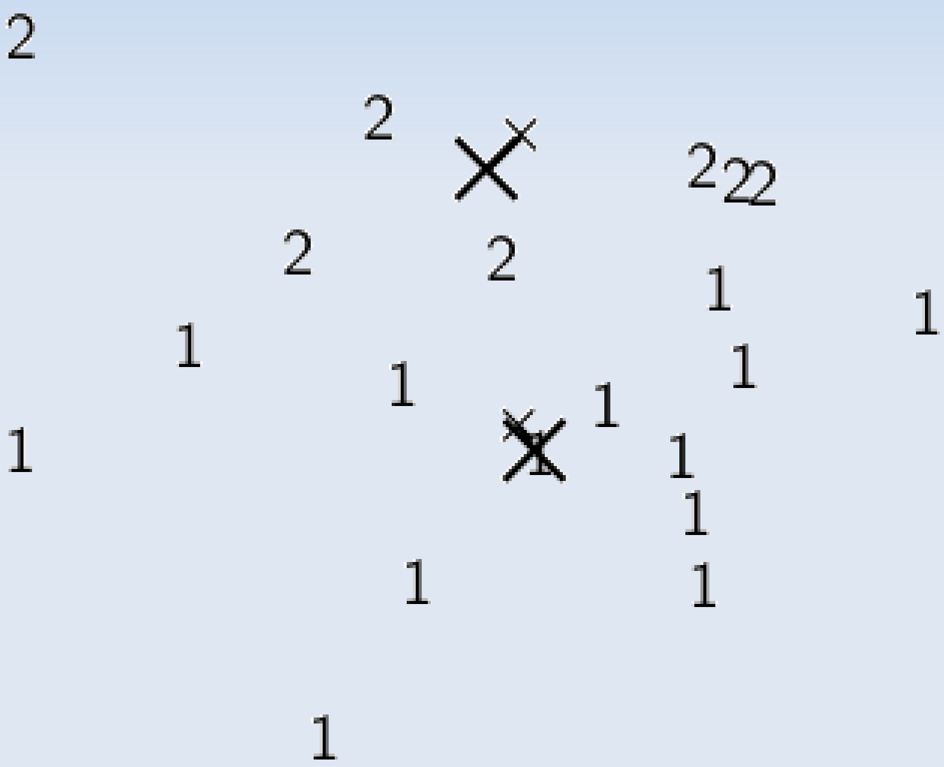
k-means



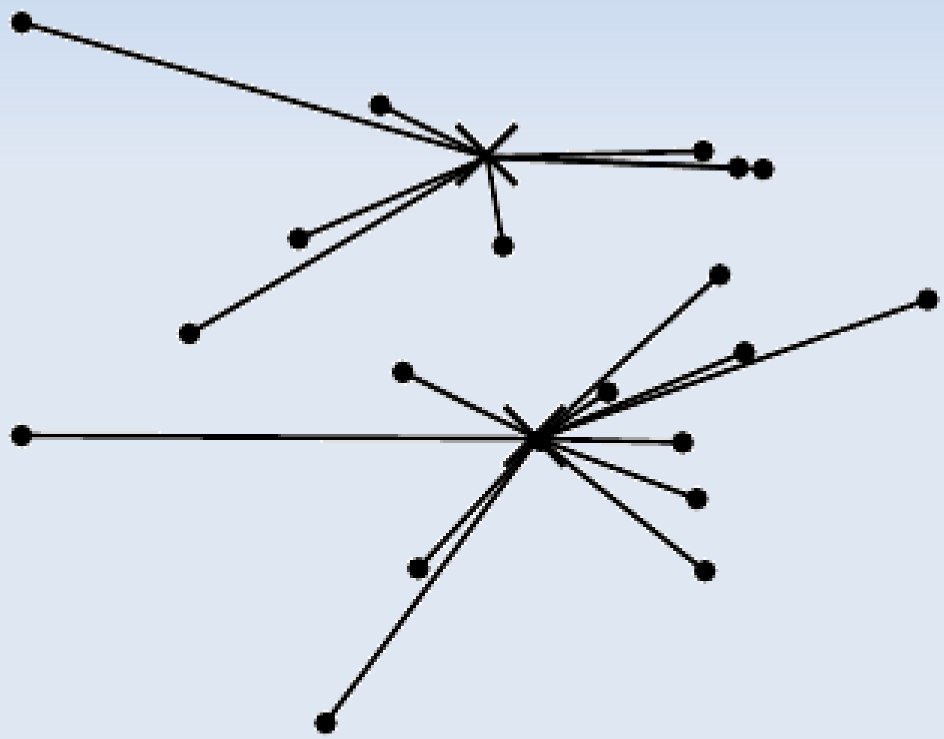
k-means



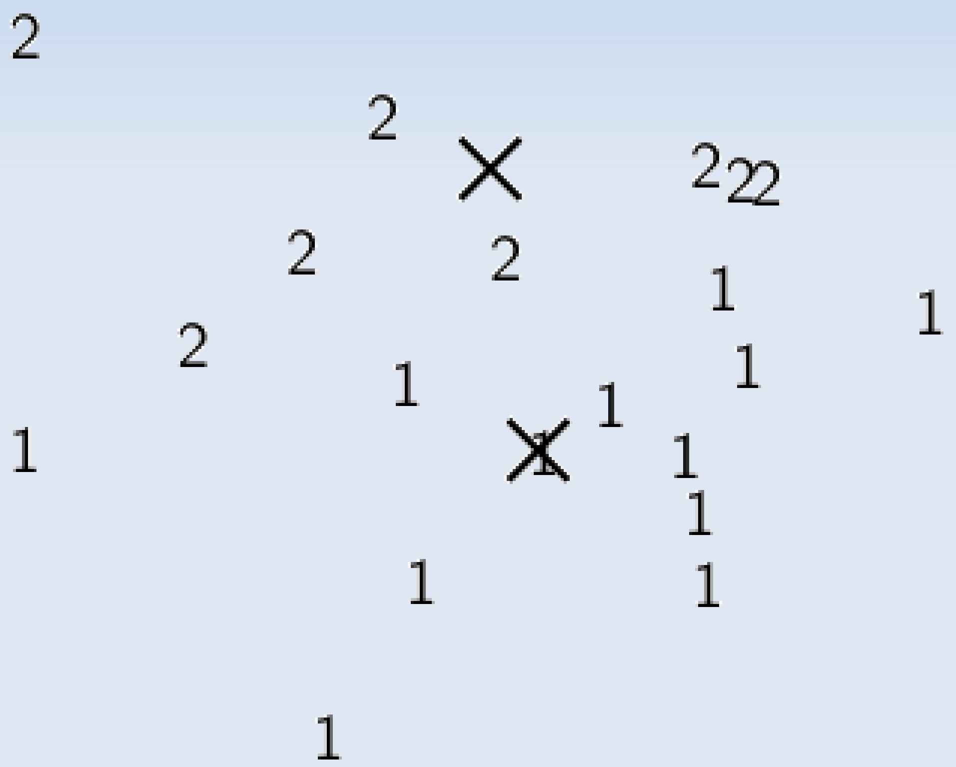
k-means



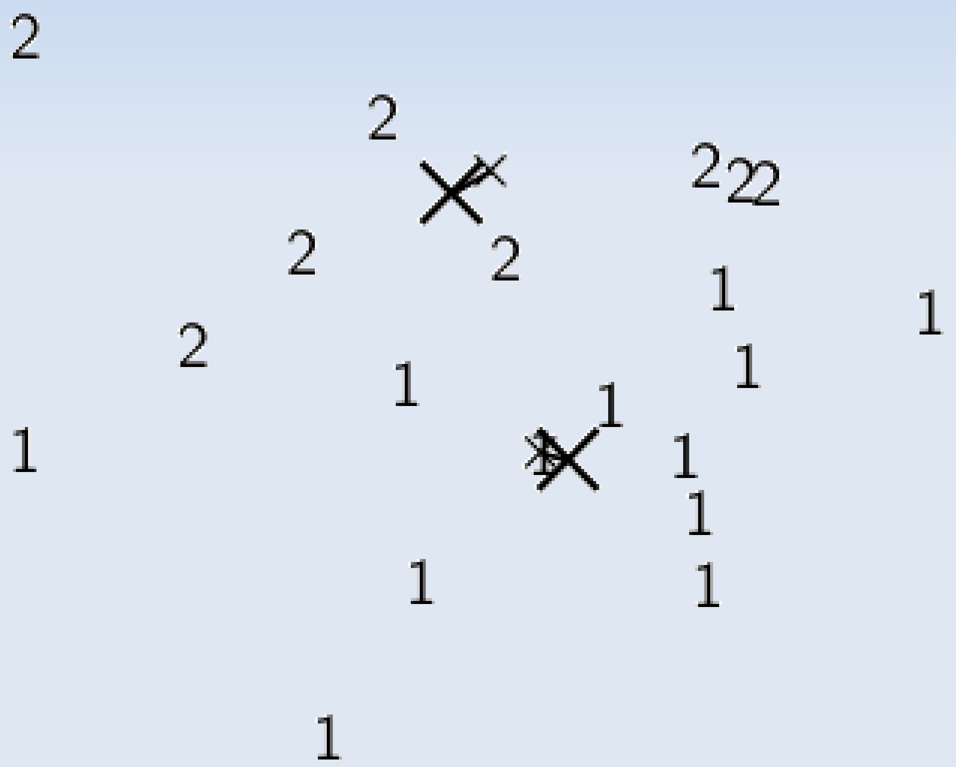
k-means



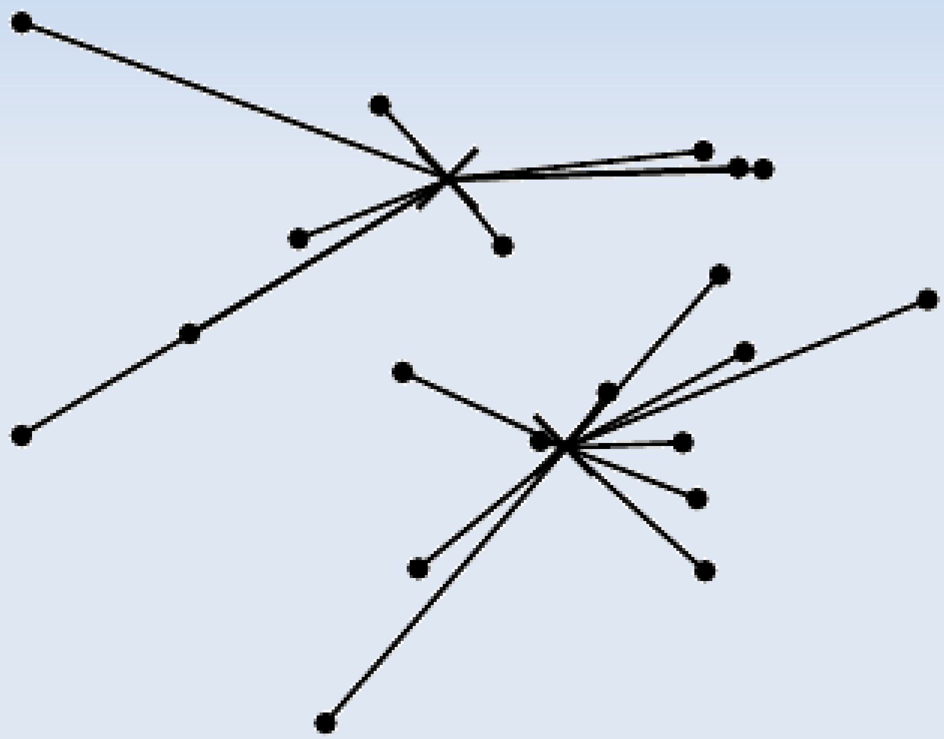
k-means



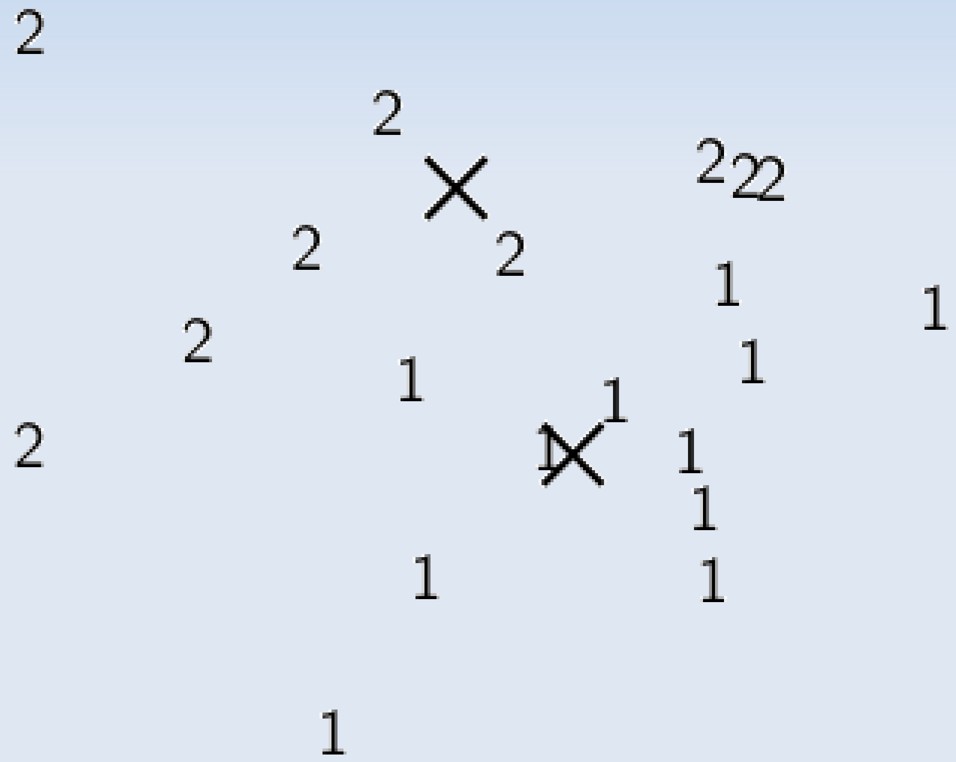
k-means



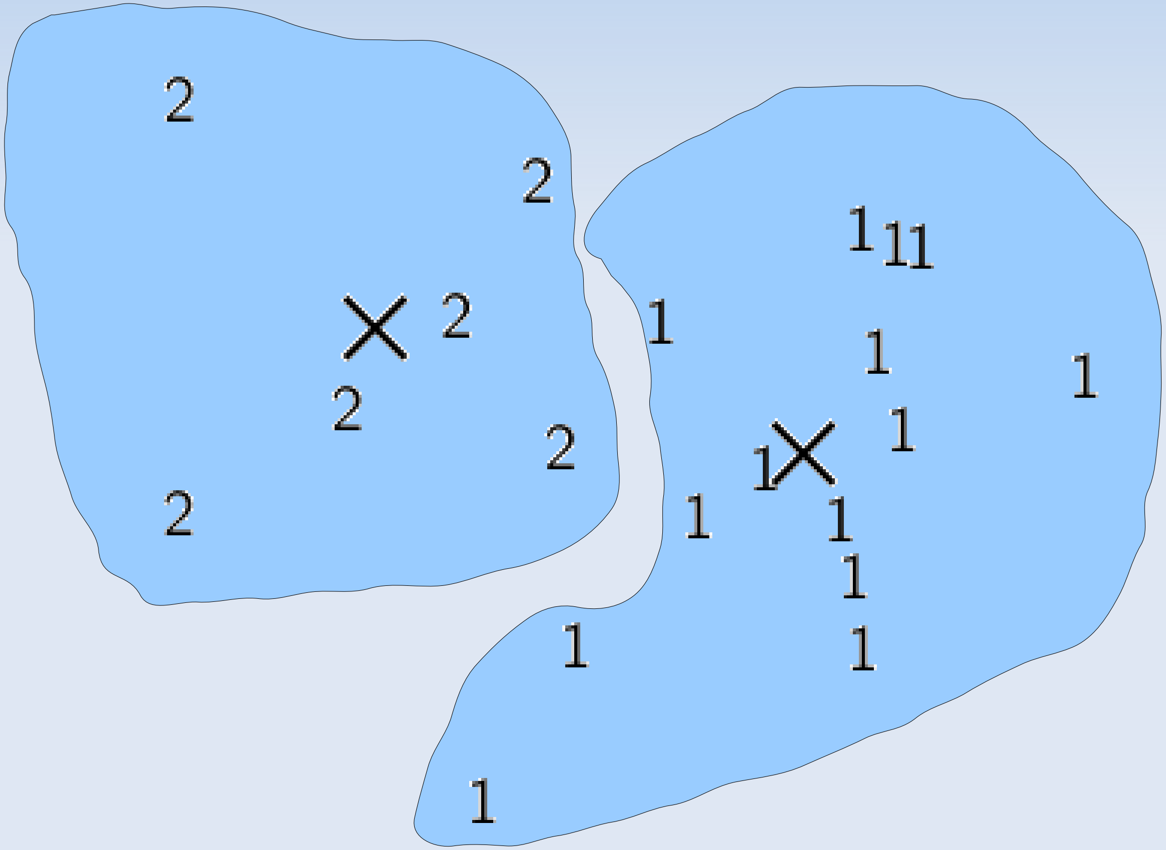
k-means



k-means



k-means



Dimensionality reduction

Our data can have **lots** of dimensions (e.g. tens or hundreds of thousands). Reducing this:

- Makes processing easier;
- Helps discover hidden or “latent” structure and common elements

We get “similar” **lower-dimension** data. This is similar to e.g. principal component analysis.

Overall aims

Reduce dimensions:

- 1st dimension explains the most variation
- 2nd the next most
- And so on for *k* dimensions

Control error:

- Minimise error, as distance: $\|A - \hat{A}\|$
- So that points close in original space are close in reduced space

Singular value decomposition

SVD is a **projection** onto a lower-dimensional space.

Term-topic matrix

$$A_{t \times d} = T_{t \times n}\ S_{n \times n}\ D_{d \times n}{}^{T}$$

Document-topic matrix

- T and D are orthonormal
- The SVD is unique
- Then just take the first *k* of *n* entries (they carry the most information)

Singular value decomposition

SVD is a **projection** onto a lower-dimensional space.

$$\hat{A}_{t \times d} = T_{t \times k} \, S_{k \times k} \, D_{d \times k}^{T}$$

An example (1)

$$A_{t\times d} = T_{t\times n}\, S_{n\times n}\, D_{d\times n}{}^{T}$$

$$A = \begin{array}{l} \\ \text{cosmonaut} \\ \text{astronaut} \\ \text{moon} \\ \text{car} \\ \text{truck} \end{array} \begin{array}{cccccc} d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{array}$$

From Manning and Schütze

An example (2)

$A_{t \times d} = T_{t \times n}\ S_{n \times n}\ D_{d \times n}{}^{T}$, n=5

$$S = \begin{pmatrix} 2.16 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.59 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.28 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.39 \end{pmatrix}$$

From Manning and Schütze

An example (3)

$A_{t \times d} = T_{t \times n}\ S_{n \times n}\ D_{d \times n}{}^{T}$, n=5, set k=2

$$T = \begin{array}{l} \\ \text{cosmonaut} \\ \text{astronaut} \\ \text{moon} \\ \text{car} \\ \text{truck} \end{array} \left(\begin{array}{ccccc} \text{dim 1} & \text{dim 2} & \text{dim 3} & \text{dim 4} & \text{dim 5} \\ -0.4 & -0.3 & 0.6 & 0.6 & 0.3 \\ -0.1 & -0.3 & -0.6 & 0.0 & 0.7 \\ -0.5 & -0.5 & -0.4 & 0.0 & -0.6 \\ -0.7 & 0.4 & 0.2 & -0.6 & 0.2 \\ -0.3 & 0.7 & -0.4 & 0.6 & -0.1 \end{array} \right)$$

From Manning and Schütze

An example (4)

$A_{t \times d} = T_{t \times n} \, S_{n \times n} \, D_{d \times n}^{T}$, n=5, set k=2

$$S_{2\times 2} D_{2\times 5} =$$

$$\begin{pmatrix} & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \\ \text{dim 1} & -1.62 & -0.60 & -0.04 & -0.97 & -0.71 & -0.26 \\ \text{dim 2} & -0.46 & -0.84 & -0.30 & 1.00 & 0.35 & 0.65 \end{pmatrix}$$

From Manning and Schütze

Uses of SVD

- **Reduce computation**
- **Latent semantic indexing**: use SVD to find a small number of topics, then index those (not words) for better retrieval
- **Document similarity**: let $B = S_{k \times k} D_{d \times k}$, then BB^T is document similarity on topics (not words)
- **Word similarity**: can do the same thing for words to find those which appear in the same places

Summary

- It's possible to use machine learning to find patterns behind data sets, including text data.
- **Classification** (supervised): k-NN or SVM are good choices.
- **Clustering** (unsupervised): k-means is a good choice.
- There are also unsupervised methods to **reduce the dimension** of your data and reveal hidden structure.

Summary of today

- Natural language processing
  - Segmentation, normalisation, stemming
  - Part-of-speech tagging, named entities
- Information retrieval
  - Term occurrence matrix
  - Term weights, tf.idf, vector space model
- Machine learning
  - Supervised/unsupervised, classification/clustering
  - Evaluation

Summary of today

- Natural language processing
- Bag-of-words
- Term weights
- Vector representations and dimensionality
- Independence, sparsity, and smoothing
- Probabilistic models and algorithms

Reminder of next three days

- Day 3: statistical network models
- Day 4: dynamic networks
- Day 5: hackathon and project showcase

Practical session: get the data

Get a copy of `reuters.zip`

- Linux or Mac: put it in `/usr/share/nltk_data/corpora` or in `~/nltk_data/corpora`
- Windows: put it in `c:\nltk_data\corpora`

Test it: run python and type

```
>>> from nltk.corpus import reuters
>>> len(reuters.words())
1720901
```

Coffee break

Thanks: Wray Buntine, Hanna Suominen, Hinrich Schütze

Practical session

Using python and nltk:

- Load the `reuters` corpus
- Build a classifier, e.g. Naïve Bayes, with words as binary features (see the slides)
  - Just look at 'grain', 'crude', and 'livestock' categories
- Evaluate it and look at the “best” features

Practical session

Now try other ideas for features, e.g.:

- Removing stopwords (use `nltk.corpus.stopwords.words('english')`)
- Stems, not words (use `nltk.stem.porter`)
- Only nouns or only named entities, not all words (use a POS tagger or a NE recogniser)

Do they help?

Look at the most informative features each time