

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 6: Scoring, Term Weighting, The Vector Space Model

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2008.05.20

Definitions

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
- **Type** – The same as a term in most cases: an equivalence class of tokens.

Recall: Inverted index construction

- Input:

Friends, Romans, countrymen. So let it be with Caesar ...

- Output:

friend roman countryman so ...

- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. “King of Denmark”
- Most web search engines index stop words.

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* → *be*
- Example: *car, cars, car's, cars'* → *car*
- Example: *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form (the [lemma](#)).
- Inflectional morphology (*cutting* → *cut*) vs. derivational morphology (*destruction* → *destroy*)

Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional **and** derivational
- Example for derivational: *automate*, *automatic*, *automation* all reduce to *automat*

Porter stemmer: A few rules

Rule

SSSES → SS

IES → I

SS → SS

S →

Example

caresses → caress

ponies → poni

caress → caress

cats → cat

Outline

- 1 Recap
- 2 Term frequency
- 3 tf-idf weighting
- 4 The vector space

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher.
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in $[0, 1]$ – to each document
- This score measures how well document and query “match”.

Query-document matching scores

- We need a way of assigning a score to a query/document pair.
- Let's start with a one-term query.
- If the query term does not occur in the document: score should be 0.
- The more frequent the query term in the document, the higher the score

From now on, we will use the frequencies of terms

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is represented by a **count vector** $\in \mathbb{N}^{|V|}$.

Bag of words model

- We do not consider the **order** of words in a document.
- *John is quicker than Mary* and *Mary is quicker than John* are represented the same way.
- This is called a **bag of words model**.

Term frequency tf

- The term frequency $tf_{t,d}$ of term t in document d is defined as the **number of times that t occurs in d** .
- We want to use tf when computing query-document match scores.
- But how?

Term frequency tf

- The term frequency $tf_{t,d}$ of term t in document d is defined as the **number of times that t occurs in d** .
- We want to use tf when computing query-document match scores.
- But how?
- Raw term frequency is not what we want.
- A document with 10 occurrences of the term is more relevant than a document with one occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log frequency weighting

- The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \text{tf}_{t,d} & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.

Log frequency weighting

- The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \text{tf}_{t,d} & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d :

$$\text{matching-score} = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

Log frequency weighting

- The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \text{tf}_{t,d} & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d :
matching-score = $\sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Outline

- 1 Recap
- 2 Term frequency
- 3 tf-idf weighting**
- 4 The vector space

Document frequency

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is **rare** in the collection (e.g., ARACHNOCENTRIC)
 - A document containing this term is very likely to be relevant.
 - → We want a **high weight for rare terms** like ARACHNOCENTRIC.

Document frequency

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is **rare** in the collection (e.g., ARACHNOCENTRIC)
 - A document containing this term is very likely to be relevant.
 - → We want a **high weight for rare terms** like ARACHNOCENTRIC.
- Consider a term in the query that is **frequent** in the collection (e.g., HIGH, INCREASE, LINE)
 - A document containing this term is more likely to be relevant than a document that doesn't, but it's not a sure indicator of relevance.
 - → **For frequent terms**, we want positive weights for words like HIGH, INCREASE, and LINE, but **lower weights** than for rare terms.

Document frequency

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is **rare** in the collection (e.g., ARACHNOCENTRIC)
 - A document containing this term is very likely to be relevant.
 - → We want a **high weight for rare terms** like ARACHNOCENTRIC.
- Consider a term in the query that is **frequent** in the collection (e.g., HIGH, INCREASE, LINE)
 - A document containing this term is more likely to be relevant than a document that doesn't, but it's not a sure indicator of relevance.
 - → **For frequent terms**, we want positive weights for words like HIGH, INCREASE, and LINE, but **lower weights** than for rare terms.
- We will use document frequency to factor this into computing the matching score.
- The document frequency is **the number of documents in the collection that the term occurs in**.

idf weight

- df_t is the document frequency, the number of documents that t occurs in.
- df is an inverse measure of the **informativeness** of the term.
- We define the **idf weight** of term t as follows:

$$idf_t = \log_{10} \frac{N}{df_t}$$

- idf is a measure of the **informativeness** of the term.

Examples for idf

Compute idf_t using the formula: $\text{idf}_t = \log_{10} \frac{1,000,000}{\text{df}_t}$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.



$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$$

- Best known weighting scheme in information retrieval
- Note: the “-” in tf-idf is a hyphen, not a minus sign!

Summary: tf-idf

- Assign a tf-idf weight for each term t in each document d :
$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$$
- N : total number of documents
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Outline

- 1 Recap
- 2 Term frequency
- 3 tf-idf weighting
- 4 The vector space

Binary \rightarrow count \rightarrow weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35	
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0	
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0	
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0	
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0	
MERCY	1.51	0.0	1.90	0.12	5.25	0.88	
WORSER	1.37	0.0	0.11	4.15	0.25	1.95	
...							

Each document is now represented by a **real-valued vector** of tf-idf weights $\in \mathbb{R}^{|V|}$.

Documents as vectors

- Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.
- So we have a $|V|$ -dimensional real-valued vector space.
- Terms are **axes** of the space.
- Documents are **points** or **vectors** in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- This is a very sparse vector - most entries are zero.

Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query

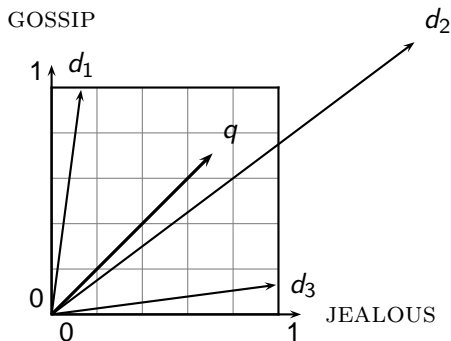
How do we formalize vector space similarity?

- First cut: distance between two points
- (= distance between the end points of the two vectors)
- Euclidean distance?

How do we formalize vector space similarity?

- First cut: distance between two points
- (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea ...
- ... because Euclidean distance is **large** for vectors **of different lengths**.

Why distance is a bad idea



The Euclidean distance of \vec{q} and \vec{d}_2 is large although the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.

Use angle instead of distance

- Rank documents according to angle with query
- Thought experiment: take a document d and append it to itself. Call this document d' .
- “Semantically” d and d' have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity.
- The Euclidean distance between the two documents can be quite large.

From angles to cosines

- The following two notions are equivalent.
 - Rank documents according to the **angle** between query and document in decreasing order
 - Rank documents according to **cosine**(query,document) in increasing order
- Cosine is a monotonically decreasing function of the angle for the interval $[0^\circ, 180^\circ]$

Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – here we use the L_2 norm:

$$||x||_2 = \sqrt{\sum_i x_i^2}$$

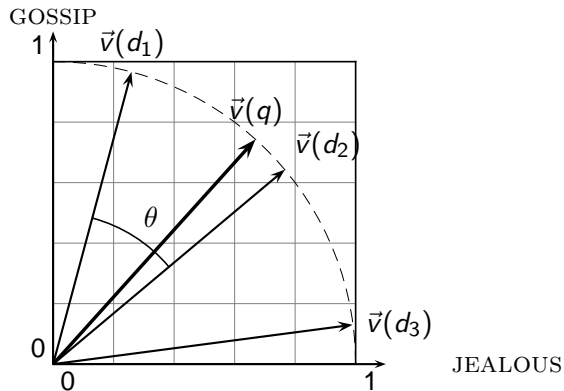
- This maps vectors onto the unit sphere ...
- ...since after normalization: $||x||_2 = \sqrt{\sum_i x_i^2} = 1.0$
- As a result, longer documents and shorter documents have weights of the same order of magnitude.
- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have **identical vectors** after length-normalization.

Cosine similarity between query and document

$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- q_i is the tf-idf weight of term i in the query.
- d_i is the tf-idf weight of term i in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of \vec{q} and \vec{d} .
- This is the cosine similarity of \vec{q} and \vec{d} or, equivalently, the cosine of the angle between \vec{q} and \vec{d} .

Cosine similarity illustrated



Cosine: Example

How similar are
the novels? SaS:
Sense and
Sensibility, PaP:
Pride and
Prejudice, and
WH: Wuthering
Heights?

term frequencies (counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

Cosine: Example

term frequencies (counts)

log frequency weighting

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

(To simplify this example, we don't do idf weighting.)

Cosine: Example

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

log frequency weighting
& cosine normalization

term	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

Cosine: Example

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

log frequency weighting
& cosine normalization

term	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

- $\cos(\text{SaS}, \text{PaP}) \approx 0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.
- $\cos(\text{SaS}, \text{WH}) \approx 0.79$
- $\cos(\text{PaP}, \text{WH}) \approx 0.69$
- Why do we have $\cos(\text{SaS}, \text{PaP}) > \cos(\text{SaS}, \text{WH})$?

Summary: Ranked retrieval in the vector space model

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents with respect to the query
- Return the top K (e.g., $K = 10$) to the user

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 13: Text Classification & Naive Bayes

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2008.06.10

Outline

- 1 Text classification
- 2 Naive Bayes
- 3 Evaluation of TC
- 4 NB independence assumptions

Formal definition of TC: Training

Given:

- A document space \mathbb{X}
 - Documents are represented in this space, typically some type of high-dimensional space.
- A fixed set of classes $\mathbb{C} = \{c_1, c_2, \dots, c_J\}$
 - The classes are human-defined for the needs of an application (e.g., spam vs. non-spam).
- A training set \mathbb{D} of labeled documents with each labeled document $\langle d, c \rangle \in \mathbb{X} \times \mathbb{C}$

Using a learning method or learning algorithm, we then wish to learn a classifier γ that maps documents to classes:

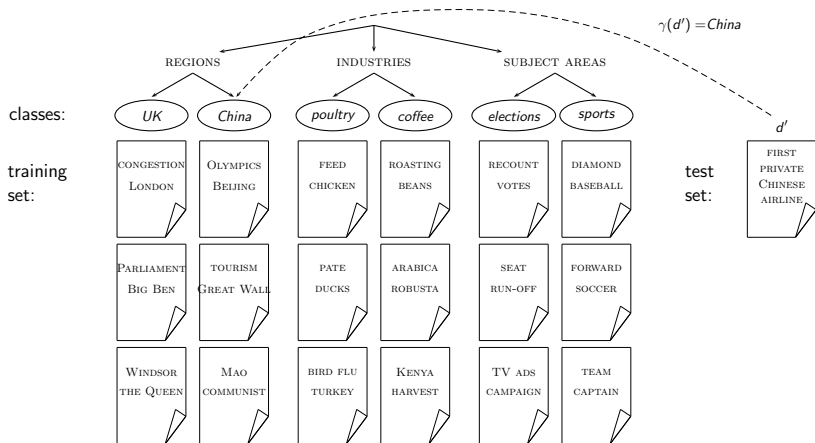
$$\gamma : \mathbb{X} \rightarrow \mathbb{C}$$

Formal definition of TC: Application/Testing

Given: a description $d \in \mathbb{X}$ of a document

Determine: $\gamma(d) \in \mathbb{C}$, that is, the class that is most appropriate for d

Topic classification



Many search engine functionalities are based on classification.

Examples?

Another TC task: spam filtering

From: ''' <takworldld@hotmail.com>
Subject: real estate is the only way... gem oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for
similar courses

I am 22 years old and I have already purchased 6 properties
using the
methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

=====
Click Below to order:
<http://www.wholesaledaily.com/sales/nmd.htm>
=====

Applications of text classification in IR

- Language identification (classes: English vs. French etc.)
- The automatic detection of spam pages (spam vs. nonspam, example: googel.org)
- The automatic detection of sexually explicit content (sexually explicit vs. not)
- Sentiment detection: is a movie or product review positive or negative (positive vs. negative)
- Topic-specific or *vertical* search – restrict search to a “vertical” like “related to health” (relevant to vertical vs. not)
- Machine-learned ranking function in ad hoc retrieval (relevant vs. nonrelevant)
- Semantic Web: Automatically add semantic tags for non-tagged text (e.g., for each paragraph: relevant to a vertical like health or not)

Outline

- 1 Text classification
- 2 Naive Bayes
- 3 Evaluation of TC
- 4 NB independence assumptions

The Naive Bayes classifier

- The Naive Bayes classifier is a probabilistic classifier.
- We compute the probability of a document d being in a class c as follows:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

The Naive Bayes classifier

- The Naive Bayes classifier is a probabilistic classifier.
- We compute the probability of a document d being in a class c as follows:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

- $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c
- $P(t_k|c)$ as a measure of how much evidence t_k contributes that c is the correct class.
- $P(c)$ is the prior probability of c .
- n_d is the number of tokens in document d .

Maximum a posteriori class

- Our goal is to find the “best” class.
- The best class in Naive Bayes classification is the most likely or maximum a posteriori (MAP) class c_{map} :

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg \max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

- We write \hat{P} for P since these values are estimates from the training set.

Derivation of Naive Bayes rule

We want to find the class that is most likely given the document:

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} P(c|d)$$

Apply Bayes rule $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$:

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} \frac{P(d|c)P(c)}{P(d)}$$

Drop denominator since $P(d)$ is the same for all classes:

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} P(d|c)P(c)$$

Too many parameters / sparseness

$$\begin{aligned}c_{\text{map}} &= \arg \max_{c \in \mathbb{C}} P(d|c)P(c) \\ &= \arg \max_{c \in \mathbb{C}} P(\langle t_1, \dots, t_k, \dots, t_{n_d} \rangle | c)P(c)\end{aligned}$$

Why can't we use this to make an actual classification decision?

Too many parameters / sparseness

$$\begin{aligned}c_{\text{map}} &= \arg \max_{c \in \mathbb{C}} P(d|c)P(c) \\ &= \arg \max_{c \in \mathbb{C}} P(\langle t_1, \dots, t_k, \dots, t_{n_d} \rangle | c)P(c)\end{aligned}$$

Why can't we use this to make an actual classification decision?

- There are too many parameters $P(\langle t_1, \dots, t_k, \dots, t_{n_d} \rangle | c)$, one for each unique combination of a class and a sequence of words.
- We would need a very, very large number of training examples to estimate that many parameters.
- This is the problem of **data sparseness**.

Naive Bayes conditional independence assumption

To reduce the number of parameters to a manageable size, we make the **Naive Bayes conditional independence assumption**:

$$P(d|c) = P(\langle t_1, \dots, t_{n_d} \rangle | c) = \prod_{1 \leq k \leq n_d} P(X_k = t_k | c)$$

We assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(X_k = t_k | c)$.

Maximum a posteriori class

- Our goal is to find the “best” class.
- The best class in Naive Bayes classification is the most likely or maximum a posteriori (MAP) class c_{map} :

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg \max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

Taking the log

- Multiplying lots of small probabilities can result in floating point underflow.
- Since $\log(xy) = \log(x) + \log(y)$, we can sum log probabilities instead of multiplying probabilities.
- Since log is a monotonic function, the class with the highest score does not change.
- So what we usually compute in practice is:

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k | c)]$$

Parameter estimation

- How to estimate parameters $\hat{P}(c)$ and $\hat{P}(t_k|c)$ from training data?

Parameter estimation

- How to estimate parameters $\hat{P}(c)$ and $\hat{P}(t_k|c)$ from training data?
- Prior:

$$\hat{P}(c) = \frac{N_c}{N}$$

- N_c : number of docs in class c ; N : total number of docs

Parameter estimation

- How to estimate parameters $\hat{P}(c)$ and $\hat{P}(t_k|c)$ from training data?
- Prior:

$$\hat{P}(c) = \frac{N_c}{N}$$

- N_c : number of docs in class c ; N : total number of docs
- Conditional probabilities:

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

- T_{ct} is the number of tokens of t in training documents from class c (includes multiple occurrences)

To avoid zeros: Add-one smoothing

- Add one to each count to avoid zeros:

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

- B is the number of different words (in this case the size of the vocabulary: $|V| = M$)

Naive Bayes: Summary

- Estimate parameters from training corpus using add-one smoothing
- For a new document, for each class, compute sum of (i) log of prior and (ii) logs of conditional probabilities of the terms
- Assign document to the class with the largest score

Example: Data

	docID	words in document	in $c = \textit{China}$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Example: Parameter estimates

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\bar{c}) = 1/4$

Conditional probabilities:

$$\hat{P}(\text{CHINESE}|c) = (5 + 1)/(8 + 6) = 6/14 = 3/7$$

$$\hat{P}(\text{TOKYO}|c) = \hat{P}(\text{JAPAN}|c) = (0 + 1)/(8 + 6) = 1/14$$

$$\hat{P}(\text{CHINESE}|\bar{c}) = (1 + 1)/(3 + 6) = 2/9$$

$$\hat{P}(\text{TOKYO}|\bar{c}) = \hat{P}(\text{JAPAN}|\bar{c}) = (1 + 1)/(3 + 6) = 2/9$$

The denominators are $(8 + 6)$ and $(3 + 6)$ because the lengths of text_c and $\text{text}_{\bar{c}}$ are 8 and 3, respectively, and because the constant B is 6 as the vocabulary consists of six terms.

Example: Classification

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$

$$\hat{P}(\bar{c}|d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

Thus, the classifier assigns the test document to $c = \textit{China}$.
The reason for this classification decision is that the three occurrences of the positive indicator `CHINESE` in d_5 outweigh the occurrences of the two negative indicators `JAPAN` and `TOKYO`.

Outline

- 1 Text classification
- 2 Naive Bayes
- 3 Evaluation of TC
- 4 NB independence assumptions

Violation of Naive Bayes independence assumptions

- The independence assumptions do not really hold of documents written in natural language.
- Conditional independence:

$$P(\langle t_1, \dots, t_{n_d} \rangle | c) = \prod_{1 \leq k \leq n_d} P(X_k = t_k | c)$$

- Examples for why this assumption is not really true?

Why does Naive Bayes work?

- Naive Bayes can work well even though conditional independence assumptions are **badly** violated.
- Example:

	c_1	c_2	class selected
true probability $P(c d)$	0.6	0.4	c_1
$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k c)$	0.00099	0.00001	
NB estimate $\hat{P}(c d)$	0.99	0.01	c_1

Why does Naive Bayes work?

- Naive Bayes can work well even though conditional independence assumptions are **badly** violated.
- Example:

	c_1	c_2	class selected
true probability $P(c d)$	0.6	0.4	c_1
$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k c)$	0.00099	0.00001	
NB estimate $\hat{P}(c d)$	0.99	0.01	c_1

- Double counting of evidence causes underestimation (0.01) and overestimation (0.99).
- Classification is about predicting the correct class and **not** about accurately estimating probabilities.
- Correct estimation \Rightarrow accurate prediction.
- But not vice versa!

Naive Bayes is not so naive

- Naive Bayes has won some bakeoffs (e.g., KDD-CUP 97)
- More robust to nonrelevant features than some more complex learning methods
- More robust to concept drift (changing of definition of class over time) than some more complex learning methods
- Better than methods like decision trees when we have **many equally important features**
- A good dependable baseline for text classification (but not the best)
- Optimal if independence assumptions hold (never true for text, but true for some domains)
- Very fast
- Low storage requirements

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 16: Flat Clustering

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2008.06.24

Outline

- 1 Recap
- 2 Introduction**
- 3 Clustering in IR
- 4 *K*-means
- 5 Evaluation
- 6 How many clusters?

What is clustering?

- Clustering is the process of grouping a set of documents into clusters of similar documents.
- Documents within a cluster should be similar.
- Documents from different clusters should be dissimilar.
- Clustering is the most common form of **unsupervised** learning.
- Unsupervised = there are no labeled or annotated data.

Classification vs. Clustering

- Classification: supervised learning
- Clustering: unsupervised learning
- Classification: Classes are human-defined and part of the input to the learning algorithm.
- Clustering: Clusters are inferred from the data without human input.

Classification vs. Clustering

- Classification: supervised learning
- Clustering: unsupervised learning
- Classification: Classes are human-defined and part of the input to the learning algorithm.
- Clustering: Clusters are inferred from the data without human input.
 - However, there are many ways of influencing the outcome of clustering: number of clusters, similarity measure, representation of documents, ...

Outline

- 1 Recap
- 2 Introduction
- 3 Clustering in IR**
- 4 *K*-means
- 5 Evaluation
- 6 How many clusters?

The cluster hypothesis

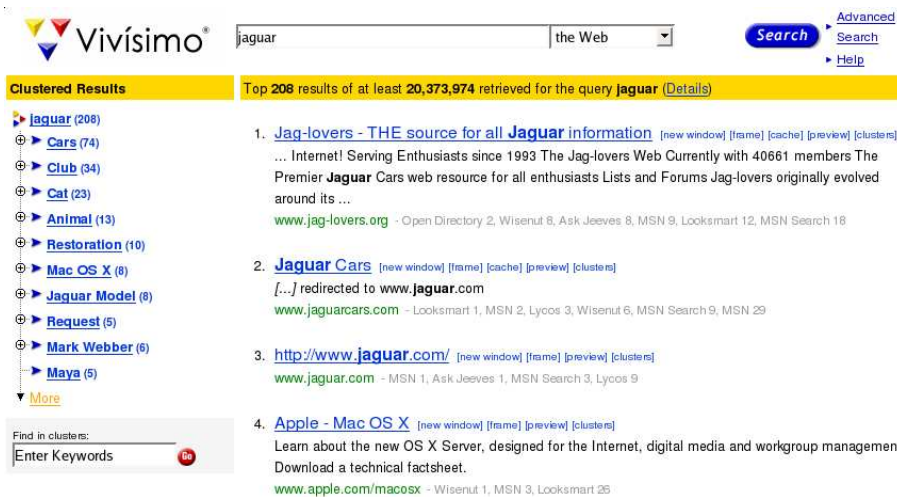
Cluster hypothesis. Documents in the same cluster behave similarly with respect to relevance to information needs.

All applications in IR are based (directly or indirectly) on the cluster hypothesis.

Applications of clustering in IR

Application	What is clustered?	Benefit	Example
Search result clustering	search results	more effective information presentation to user	
Scatter-Gather	(subsets of) collection	alternative user interface: "search without typing"	
Collection clustering	collection	effective information presentation for exploratory browsing	McKeown et al. 2002, http://news.google.com
Language modeling	collection	increased precision and/or recall	Liu&Croft 2004
Cluster-based retrieval	collection	higher efficiency: faster search	Salton 1971


Search result clustering for better navigation



Vivísimo® [Search](#) [Advanced Search](#) [Help](#)

Clustered Results Top 208 results of at least 20,373,974 retrieved for the query **jaguar** ([Details](#))

- ▶ [jaguar](#) (208)
 - ▶ [Cars](#) (74)
 - ▶ [Club](#) (34)
 - ▶ [Cat](#) (23)
 - ▶ [Animal](#) (13)
 - ▶ [Restoration](#) (10)
 - ▶ [Mac OS X](#) (8)
 - ▶ [Jaguar Model](#) (8)
 - ▶ [Request](#) (5)
 - ▶ [Mark Webber](#) (6)
 - ▶ [Maya](#) (5)
 - ▼ [More](#)

Find in clusters:
 


- [Jag-lovers - THE source for all Jaguar information](#) [new window] [frame] [cache] [preview] [clusters]
 ... Internet! Serving Enthusiasts since 1993 The Jag-lovers Web Currently with 40661 members The Premier **Jaguar** Cars web resource for all enthusiasts Lists and Forums Jag-lovers originally evolved around its ...
[www.jag-lovers.org](#) - Open Directory 2, Wisenut 8, Ask Jeeves 8, MSN 9, Looksmart 12, MSN Search 18
- [Jaguar Cars](#) [new window] [frame] [cache] [preview] [clusters]
 [...] redirected to [www.jaguar.com](#)
[www.jaguarcars.com](#) - Looksmart 1, MSN 2, Lycos 3, Wisenut 6, MSN Search 9, MSN 29
- [http://www.jaguar.com/](#) [new window] [frame] [preview] [clusters]
[www.jaguar.com](#) - MSN 1, Ask Jeeves 1, MSN Search 3, Lycos 9
- [Apple - Mac OS X](#) [new window] [frame] [preview] [clusters]
 Learn about the new OS X Server, designed for the Internet, digital media and workgroup management.
 Download a technical factsheet.
[www.apple.com/macosx](#) - Wisenut 1, MSN 3, Looksmart 26

Global navigation: Yahoo

YAHOO! DIRECTORY Search: ☐ the Web | ☒ the Directory | ☐ this category

Society and Culture

Directory > Society and Culture

 **Culture**
www.Dealtime.com SPONSOR RE

Shop and save on Magazines.

CATEGORIES ([What's This?](#))

Most Popular Society and Culture

- [Crime](#) (5453) NEW!
- [Cultures and Groups](#) (11025) NEW!
- [Environment and Nature](#) (8558) NEW!
- [Families](#) (1215)
- [Food and Drink](#) (9776) NEW!
- [Holidays and Observances](#) (3333)
- [Issues and Causes](#) (4842)
- [Mythology and Folklore](#) (984)
- [People](#) (16351)
- [Relationships](#) (595)
- [Religion and Spirituality](#) (37533)
- [Sexuality](#) (2812) NEW!

Additional Society and Culture Categories

- [Advice](#) (48)
- [Chats and Forums](#) (27)
- [Cultural Policy](#) (10)
- [Death and Dying](#) (394)
- [Disabilities](#) (1293)
- [Employment and Work@](#)
- [Etiquette](#) (54)
- [Events](#) (27)
- [Fashion@](#)
- [Gender](#) (21)
- [Home and Garden](#) (1080) NEW!
- [Magazines](#) (164)
- [Museums and Exhibits](#) (6052)
- [Pets@](#)
- [Reunions](#) (228)
- [Social Organizations](#) (338)
- [Web Directories](#) (6)
- [Weddings](#) (371)

SITE LISTINGS By Popularity | [Alphabetical](#) | ([What's This?](#)) Site

- Note: Yahoo/MESH are **not** examples of clustering.
- But they are well known examples for using a global hierarchy for navigation.
- Global navigation based on clustering:
 - Cartia
 - Themescapes
 - Google News

Flat vs. Hierarchical clustering

- Flat algorithms
 - Usually start with a random (partial) partitioning of docs into groups
 - Refine iteratively
 - Main algorithm: *K*-means

Flat vs. Hierarchical clustering

- Flat algorithms
 - Usually start with a random (partial) partitioning of docs into groups
 - Refine iteratively
 - Main algorithm: *K*-means
- Hierarchical algorithms
 - Create a hierarchy
 - Bottom-up, agglomerative
 - Top-down, divisive

Flat algorithms

- Flat algorithms compute a partition of N documents into a set of K clusters.
- Given: a set of documents and the number K
- Find: a partition in K clusters that optimizes the chosen partitioning criterion
- Global optimization: exhaustively enumerate partitions, pick optimal one
 - Not tractable
- Effective heuristic method: K -means algorithm

Outline

- 1 Recap
- 2 Introduction
- 3 Clustering in IR
- 4 K-means**
- 5 Evaluation
- 6 How many clusters?

K-means

- Objective/partitioning criterion: minimize the average squared difference from the centroid

K-means

- Objective/partitioning criterion: minimize the average squared difference from the centroid
- Recall definition of centroid:

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

where we use ω to denote a cluster.

K-means

- Objective/partitioning criterion: minimize the average squared difference from the centroid
- Recall definition of centroid:

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

where we use ω to denote a cluster.

- We try to find the minimum average squared difference by iterating two steps:
 - reassignment: assign each vector to its closest centroid
 - recomputation: recompute each centroid as the average of the vectors that were assigned to it in reassignment

Outline

- 1 Recap
- 2 Introduction
- 3 Clustering in IR
- 4 *K*-means
- 5 Evaluation**
- 6 How many clusters?

What is a good clustering?

- Internal criteria
 - Example of an internal criterion: RSS in *K*-means
- But an internal criterion often does not evaluate the actual utility of a clustering in the application.

What is a good clustering?

- Internal criteria
 - Example of an internal criterion: RSS in *K*-means
- But an internal criterion often does not evaluate the actual utility of a clustering in the application.
- Alternative: External criteria
 - Evaluate with respect to a human-defined classification

External criteria for clustering quality

- Based on a gold standard data set, e.g., the Reuters collection we also used for the evaluation of classification
- Goal: Clustering should reproduce the classes in the gold standard
- (But we only want to reproduce how documents are divided into groups, not the class labels.)
- First measure for how well we were able to reproduce the classes: **purity**

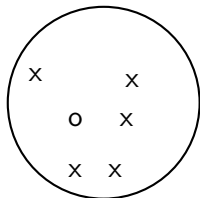
External criterion: Purity

$$\text{purity}(\Omega, C) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

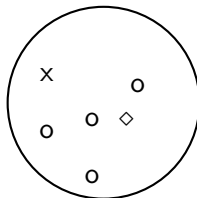
- $\Omega = \{\omega_1, \omega_2, \dots, \omega_K\}$ is the set of clusters and $C = \{c_1, c_2, \dots, c_J\}$ is the set of classes.
- For each cluster ω_k : find class c_j with most members n_{kj} in ω_k
- Sum all n_{kj} and divide by total number of points

Example for computing purity

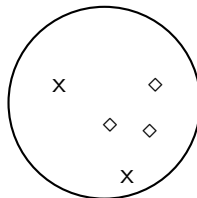
cluster 1



cluster 2



cluster 3



Majority class and number of members of the majority class for the three clusters are: x, 5 (cluster 1); o, 4 (cluster 2); and \diamond , 3 (cluster 3). Purity is $(1/17) \times (5 + 4 + 3) \approx 0.71$.

Rand index

- Definition: $RI = \frac{TP+TN}{TP+FP+FN+TN}$

Rand index

- Definition: $RI = \frac{TP+TN}{TP+FP+FN+TN}$
- Based on 2x2 contingency table:

	same cluster	different clusters
same class	true positives (TP)	false negatives (FN)
different classes	false positives (FP)	true negatives (TN)

Rand index

- Definition: $RI = \frac{TP+TN}{TP+FP+FN+TN}$
- Based on 2x2 contingency table:

	same cluster	different clusters
same class	true positives (TP)	false negatives (FN)
different classes	false positives (FP)	true negatives (TN)

- $TP+FN+FP+TN$ is the total number of pairs.
- There are $\binom{N}{2}$ pairs for N documents.
- Example: $\binom{13}{2} = 136$ in o/◇/x example
- Each pair is either positive or negative (the clustering puts the two documents in the same or in different clusters) ...
- ...and either “true” (correct) or “false” (incorrect): the clustering decision is correct or incorrect.

As an example, we compute RI for the o/◇/x example. We first compute TP + FP. The three clusters contain 6, 6, and 5 points, respectively, so the total number of “positives” or pairs of documents that are in the same cluster is:

$$TP + FP = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40$$

Of these, the x pairs in cluster 1, the o pairs in cluster 2, the ◇ pairs in cluster 3, and the x pair in cluster 3 are true positives:

$$TP = \binom{5}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} = 20$$

Thus, $FP = 40 - 20 = 20$.

FN and TN are computed similarly.

Rand measure for the o/◇/x example

	same cluster	different clusters
same class	TP = 20	FN = 24
different classes	FP = 20	TN = 72

RI is then $(20 + 72)/(20 + 20 + 24 + 72) \approx 0.68$.

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 17: Hierarchical Clustering

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

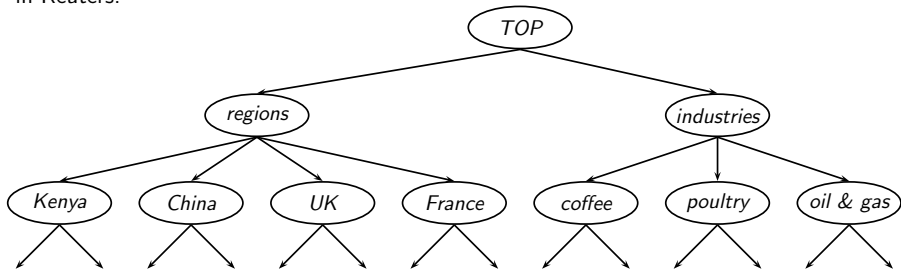
2008.07.01

Outline

- 1 Recap
- 2 Introduction**
- 3 Single-link/Complete-link
- 4 Centroid/GAAC
- 5 Variants
- 6 Labeling clusters

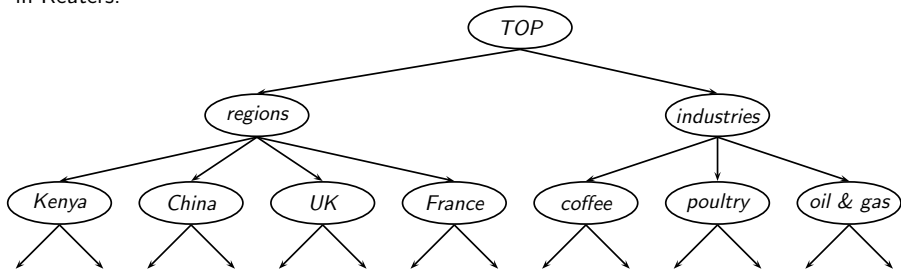
Hierarchical clustering

Our goal in hierarchical clustering is to create a hierarchy like the one we saw earlier in Reuters:



Hierarchical clustering

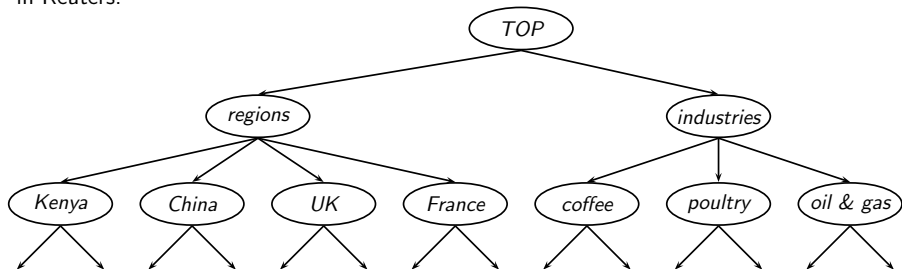
Our goal in hierarchical clustering is to create a hierarchy like the one we saw earlier in Reuters:



We want to create this hierarchy **automatically**.

Hierarchical clustering

Our goal in hierarchical clustering is to create a hierarchy like the one we saw earlier in Reuters:

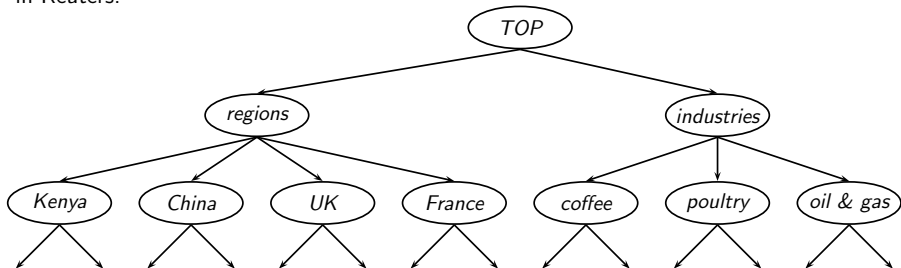


We want to create this hierarchy **automatically**.

We can do this either **top-down** or **bottom-up**.

Hierarchical clustering

Our goal in hierarchical clustering is to create a hierarchy like the one we saw earlier in Reuters:



We want to create this hierarchy **automatically**.

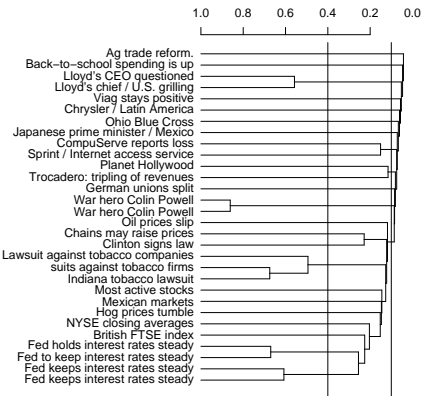
We can do this either **top-down** or **bottom-up**.

The best known bottom-up method is **hierarchical agglomerative clustering**.

Hierarchical agglomerative clustering (HAC)

- Assumes a similarity measure for determining the similarity of two **clusters** (up to now: similarity of documents).
- We will look at four different cluster similarity measures.
- Start with each document in a separate cluster
- Then repeatedly merge the two clusters that are most similar
- Until there is only one cluster
- The history of merging forms a binary tree or hierarchy.
- The standard way of depicting this history is a **dendrogram**.

A dendrogram



- The history of mergers can be read off from bottom to top.
- The horizontal line of each merger tells us what the similarity of the merger was.
- We can cut the dendrogram at a particular point (e.g., at 0.1 or 0.4) to get a flat clustering.