TÉCNICO LISBOA

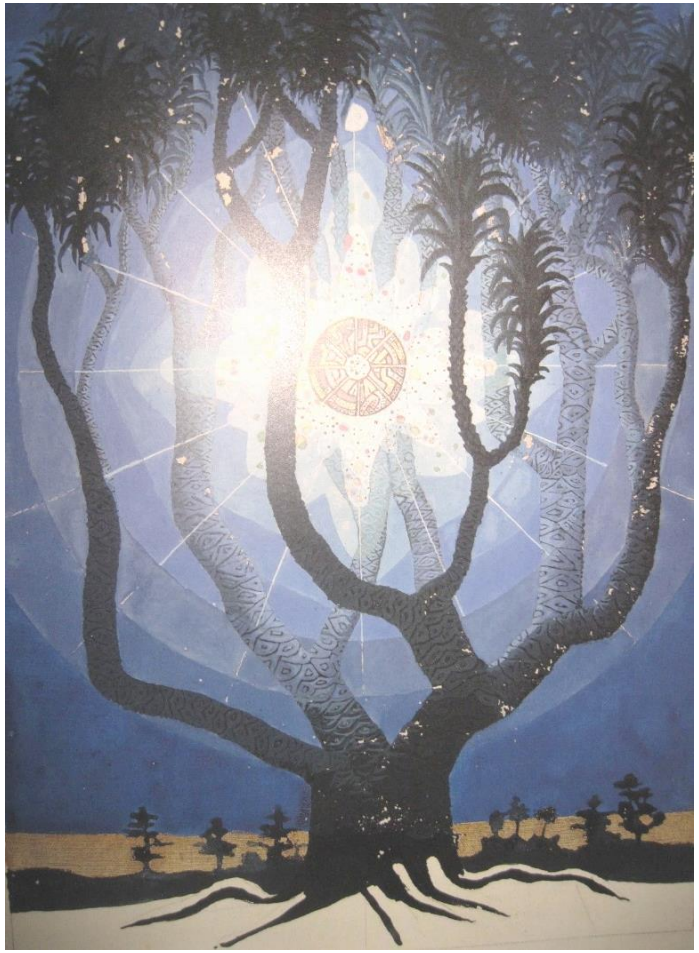
IR models: the vector space model

Acknowledgments: Standford NLP group

PRI 2023/24

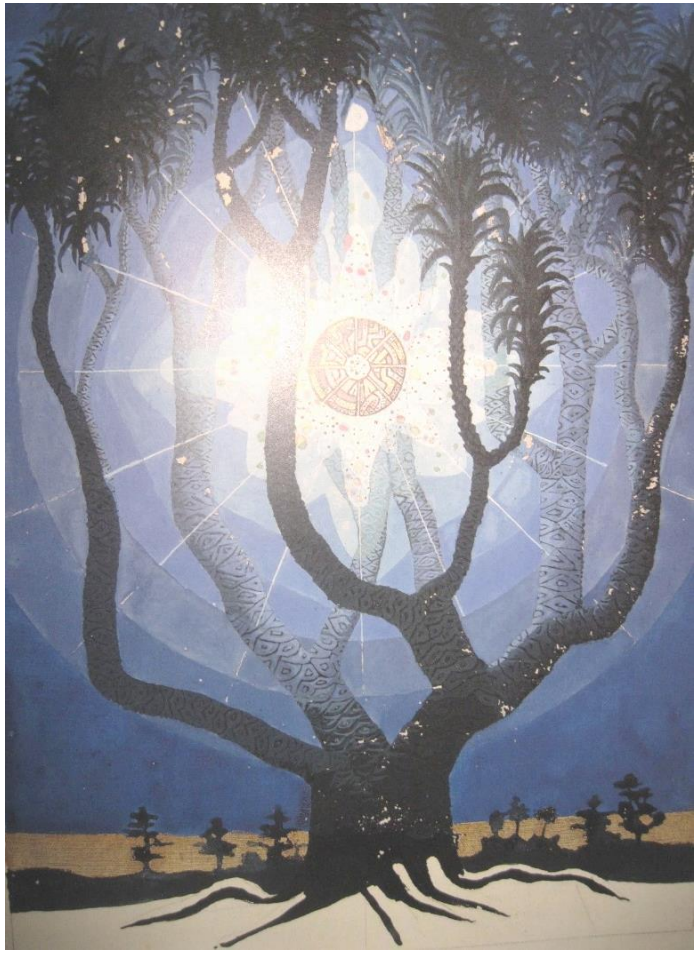
Rui Henriques
rmch@tecnico.ulisboa.pt

Outline



- **Ranked retrieval**
  - problems of Boolean model
  - ranked retrieval models
  - Jaccard-based scoring
- **Term frequency model**
  - bag of words
  - log-frequency weighting
- **TF-IDF model**
  - IDF weighting
  - TF-IDF weighting
- **Vector spaces**
  - formalizing vector space
  - cosine similarity
- **TF-IDF variants**
- **Final remarks**

Outline



- **Ranked retrieval**
  - **problems of Boolean model**
  - **ranked retrieval models**
  - **Jaccard-based scoring**
- Term frequency model
  - bag of words
  - log-frequency weighting
- TF-IDF model
  - IDF weighting
  - TF-IDF weighting
- Vector spaces
  - formalizing vector space
  - cosine similarity
- TF-IDF variants
- Final remarks

Ranked retrieval

- Thus far… queries have all been **Boolean**
  - documents either matching or don't a logical query
  - extension to ***phrase*** *queries* and ***proximity*** *queries* (using positional indexes)
  - queries can be expanded to *tolerate* ***errors***

- **Good** for:
  - expert users with precise understanding of their needs and the collection
  - applications (can automatically consume thousands of results)

- **Yet** most users:
  - incapable of writing Boolean queries (or entailing too much thinking)
  - do not want to wade through 1000s of results

Boolean search problem: *feast* or *famine*

- Boolean queries often result in either too few (≈0) or too many (1000s) results
  - query 1: "*standard user dlink 650*" ⇒ 200,000 hits
  - query 2: "*standard user dlink 650 no card found*" ⇒ 0 hits

- Using a Boolean model:
  - skill to come up with a query that produces a manageable number of hits
    - AND gives too few; OR gives too many
  - how to navigate results? Where to start?

Ranked retrieval models

- How to solve this handicap?
  - **Ranked document retrieval**
    - given a query, the IR system returns an ordering over the (top) documents in the collection
    - rather than retrieving a bulk of documents satisfying a query expression
  - **Free text querying**
    - query in natural language (NL)… rather than a logical expression

- Two separate choices
  - yet in practice, ranked retrieval are normally associated with free text queries and vice versa. Google:
    - short queries as both logical and NL
    - (long) NL queries internally encoded as both text and logical expression

Ranked retrieval: scoring as its basis

- In ranking: *feast or famine is not an issue*
  - large result sets:
    - we just show the top *k* (e.g. 10) results
    - principle: do not overwhelm the user
  - few result sets not a problem here (documents can still be scored and returned)

- *Premise*: more relevant results are ranked higher than less relevant results
  - this requires **scoring**
    - 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 (should be)
- *Today* we will be looking to different scores
  - scores based on ***matching terms*** (e.g. Jaccard)
  - scores based on ***term frequency*** (e.g. log term frequency)
  - scores sensitive to the ***expected frequency*** (e.g. TF-IDF)
- In the *next lecture* we will look to alternative models (e.g. language models)

Jaccard-based scoring

- A commonly used measure of overlap of two sets A and B

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

  - jaccard(A,A) = 1
  - jaccard(A,B) = 0 if $A \cap B = 0$
  - A and B don't have to be the same size
  - always assigns a number between 0 and 1

- **Exercise**
  - query-document match score that Jaccard coefficient computes the documents below?
    - **query**: *ides of march*
    - **document 1**: *caesar died in march* $\Rightarrow$ JACCARD(q, d) = 1/6
    - **document 2**: *the long march* $\Rightarrow$ JACCARD(q, d) = ?

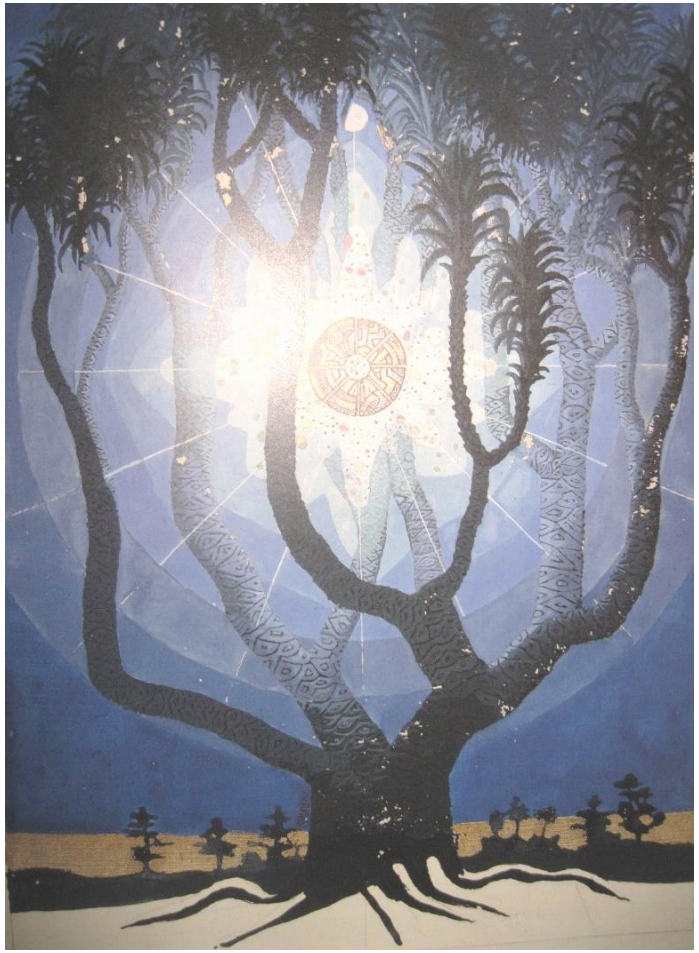
Issues with Jaccard for scoring

- It doesn't consider ***term frequency*** (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms
  - Jaccard is unable to consider this information
- We need a more sophisticated way of normalizing for length
  - Later in this lecture…

$$|A \cap B|/\sqrt{|A \cup B|}$$

… instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalization

Outline



- Ranked retrieval
  - problems of Boolean model
  - ranked retrieval models
  - Jaccard-based scoring
- **Term frequency model**
  - **bag of words**
  - **log-frequency weighting**
- TF-IDF model
  - IDF weighting
  - TF-IDF weighting
- Vector spaces
  - formalizing vector space
  - cosine similarity
- TF-IDF variants
- Final remarks

Recall: binary term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth …
ANTHONY	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	1	1
WORSER	1	0	1	1	1	0
. . .						

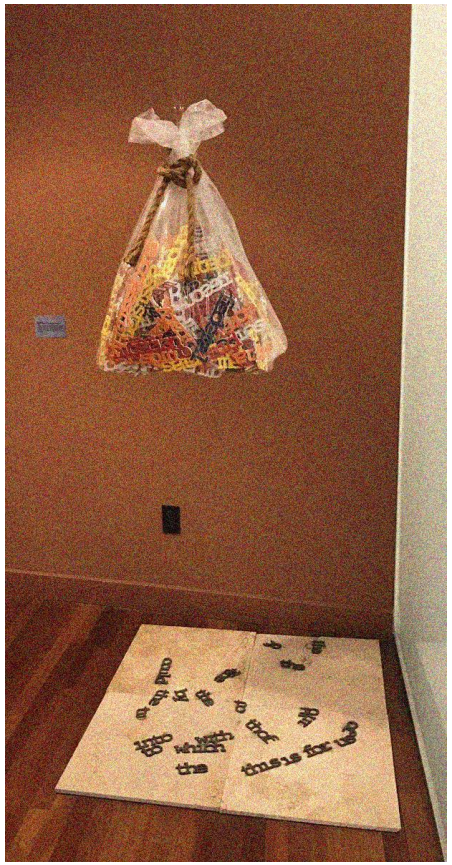
Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Term-document count matrices

	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 now represented as a count vector $\in \mathbb{N}^{|V|}$

Bag of words model



- vector representation does not consider the ordering of words in a document
  - *"John is quicker than Mary"* and "*Mary is quicker than John"* have the same vectors!
  - this is called the **bag of words** model
- although this appears to be a step back
  - the positional index is still able to distinguish these two documents!
- we will look at "recovering" positional information in the next lecture

  … for now: bag of words model

Term frequency: pros and cons

- The term frequency $\mathbf{tf}_{t,d}$ 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. How?
- Raw term frequency is not what we want:
  - a document with 10 occurrences of the term is more relevant than a document with 1 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

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

  - 0 → 0
  - 1 → 1
  - 2 → 1.3
  - 10 → 2
  - 1000 → 4
  - …

- Score for a document-query pair: sum over terms *t* in both *q* and *d*
  - the score is 0 if none of the query terms is present in the document

Exercise

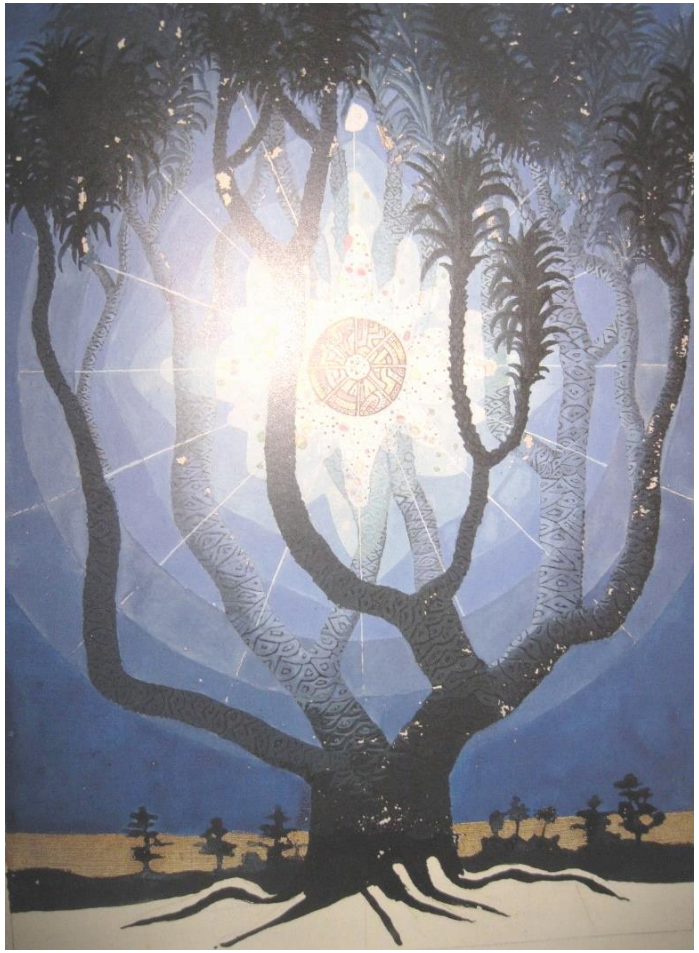
- Compare the *Jaccard matching score* and *log frequency matching score* for the following query-document pairs:
  - **q1**: [information on cars]

    d1: "*all you've ever wanted to know about cars*"
  - **q2**: [information on cars]

    d2: "*information on trucks, information on planes, information on trains*"
  - **q3**: [red cars and red trucks]

    d3: "*cops stop red cars more often*"

Outline



- Ranked retrieval
  - problems of Boolean model
  - ranked retrieval models
  - Jaccard-based scoring
- Term frequency model
  - bag of words
  - log-frequency weighting
- **TF-IDF model**
  - **IDF weighting**
  - **TF-IDF weighting**
- Vector spaces
  - formalizing vector space
  - cosine similarity
- TF-IDF variants
- Final remarks

Document *versus* collection frequency: rare terms

- **Rare terms** are more informative than frequent terms
  - recall *stop words*!
  - *log frequency weighting* unable to distinguish rare from 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 high weights for rare terms like ARACHNOCENTRIC

- Frequent terms are less informative than rare terms
  - consider frequent terms (e.g., *high, increase, line*)
  - a document containing such term is more likely to be relevant than a document that does not…

    … but it's not a sure indicator of relevance!

Document frequency

- In addition to term frequency (the frequency of the term in the document) ...
  ... we also want to use the frequency of the term in the collection for weighting and ranking
  – ***principle***: term score decreases (increases) with increased (decreased) document frequency

- Given a term, the document frequency is the number of documents where the term occurs in
  – $\mathbf{df}_t$ is the document frequency of term *t*
    – $\mathbf{df}_t$ is an inverse measure of the **informativeness** of *t*
    – $\mathbf{df}_t \leq N$ (where N is the number of documents in the collection)

idf weight

- We define the idf (inverse document frequency) of *t* by

$$\text{idf}_t = \log_{10} \frac{N}{\text{df}_t}$$

  - we use log (N/df_t) instead of N/df_t to "dampen" the effect of idf (similarly to log term weighting)
  - note that we use the log transformation for both term frequency and document frequency
  - in practice, the base of the *log* is immaterial

- Effect of idf on ranking:
  - does idf have an effect on ranking for one-term queries, like "iPhone"?
  - idf has no effect on ranking one term queries!
    - idf affects the ranking of documents for queries with at least two terms

Example

- Compute idf_t knowing N = 1M

term	df_t	idf_t
calpurnia	1	?
animal	100	?
sunday	1000	?
fly	10,000	?
under	100,000	?
the	1,000,000	?

$$idf_t = \log_{10} \frac{1{,}000{,}000}{df_t}$$

  - solution: 6, 4, 3, 2, 1, 0

- For the query **capricious person**, idf weighting makes occurrences of **capricious** count much more in the final document ranking than occurrences of **person**

Collection vs. document frequency

word	collection frequency	document frequency
INSURANCE	10440	3997
TRY	10422	8760

Why document frequency?

- ***Collection frequency*** of *t*: number of *t* tokens in the collection
- *Document frequency* of *t*: number of documents *t* occurs in
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
- This example suggests that df (and idf) is better for weighting than cf (and icf)

tf-idf weighting

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

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

- Best known weighting scheme in information retrieval
  - *note*: the **–** in "tf**–**idf" is a hyphen, not a minus sign
  - alternative names: tf.idf, tf x idf

- Principle:
  - increases with the number of occurrences within a document (term frequency)
  - increases with the rarity of the term in the collection (inverse document frequency)

tf-idf document-query score

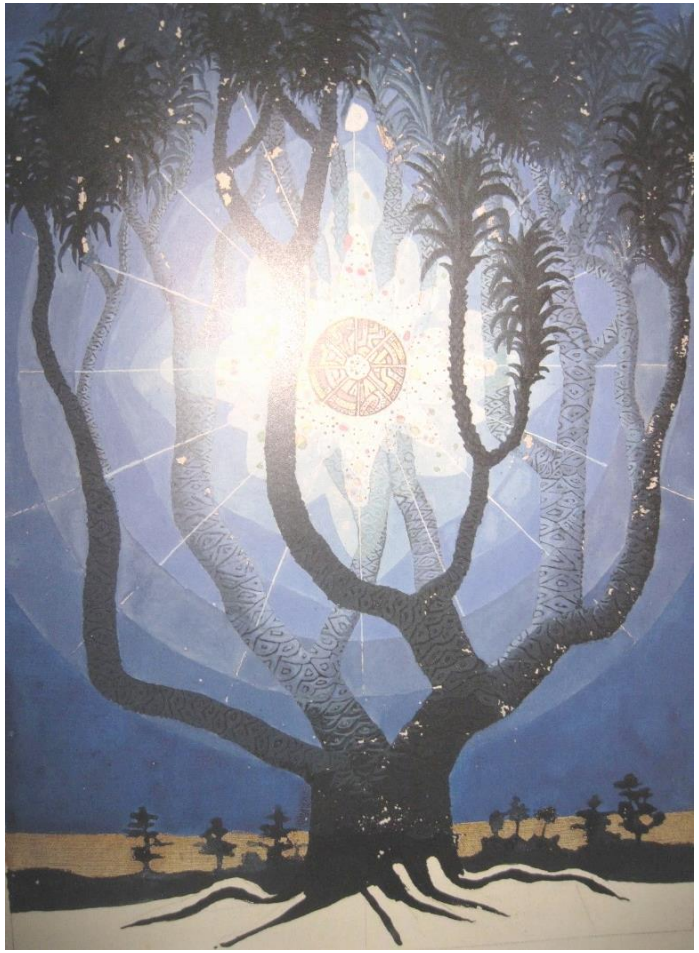
- Simple sum of tf-idf scores for each query term

$$\text{score}(q, d) = \sum_{t \in q \cap d} \text{tf–idf}_{t,d}$$

- Many variants
  - how “tf” is computed (with/without logs)
  - whether the terms in the query are also weighted
  - …

- *Exercise*: let us consolidate term, collection and document frequency
  - relationship between df and cf?
  - relationship between tf and cf?
  - relationship between tf and df?

Outline



- Ranked retrieval
  - problems of Boolean model
  - ranked retrieval models
  - Jaccard-based scoring
- Term frequency model
  - bag of words
  - log-frequency weighting
- TF-IDF model
  - IDF weighting
  - TF-IDF weighting
- **Vector spaces**
  - **formalizing vector space**
  - **cosine similarity**
- TF-IDF variants
- Final remarks

Count matrix

	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
. . .						

Recall: each document is represented as a count vector $\in \mathbb{N}^{|V|}$

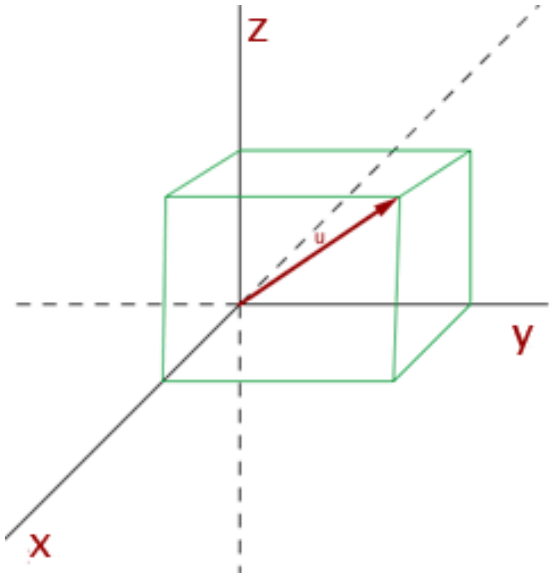
Binary → count → weight matrix

	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 as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

Documents as vectors

- Each document is now represented as a ***point*** or ***vector*** in a multi-dimensional space
  - point or vector defined by the tf-idf weights $\in \mathbb{R}^{|V|}$
- So we have a $|V|$-dimensional vector space
  - terms as **axes** of the space
  - **documents** as **points** or **vectors** in this space
- Characteristics of the vector space
  - **high-dimensional**: tens of millions of dimensions when you apply this to a web search engine
  - vectors are **sparse**: most entries are zero



Queries as vectors

- Key idea 1: do the same for queries
  - represent **queries** as points or **vectors** in the space
- Key idea 2: rank documents according to their proximity to the query in this space
  - proximity = similarity of vectors
  - proximity ≈ inverse of distance

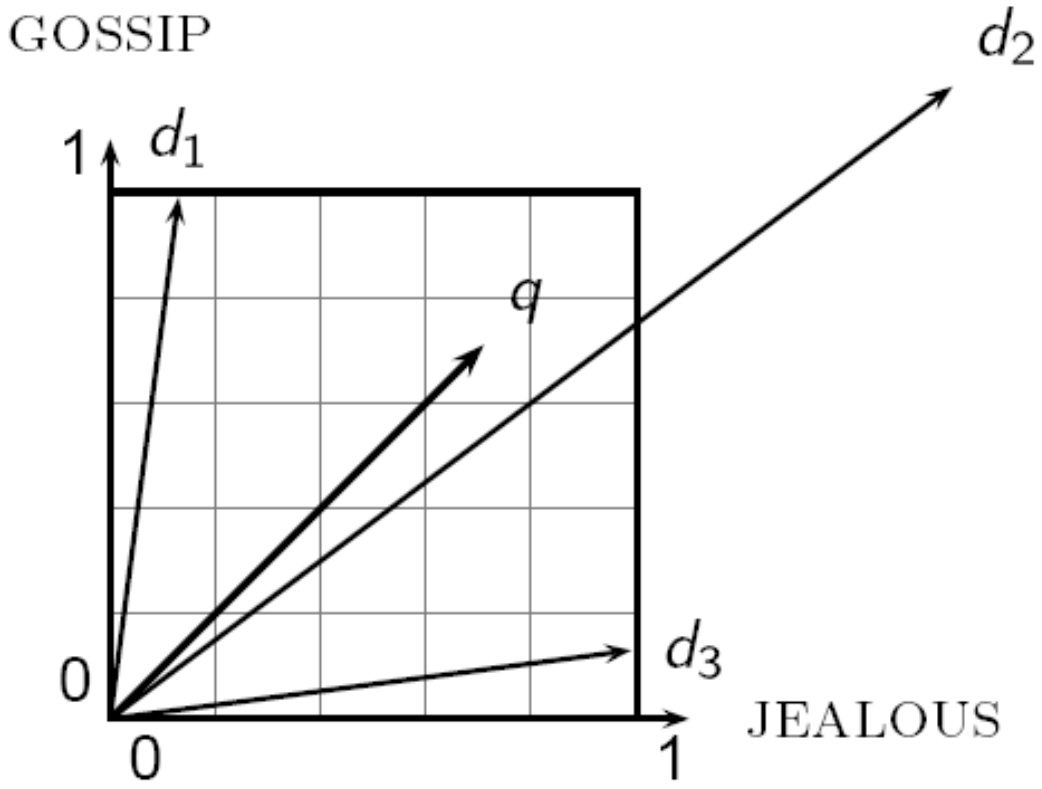
- *Recall*: we want to get away from the you are-either-in-or-out Boolean model
  … *instead*: rank more relevant documents higher than less relevant documents

- How?
  - first cut: distance between two vectors (pair query-document)?
    - distance between the end points of the two vectors?
    - Euclidean distance?

Formalizing vector space proximity

- Euclidean distance is a bad idea ...
  ... because Euclidean distance is large for vectors of different lengths

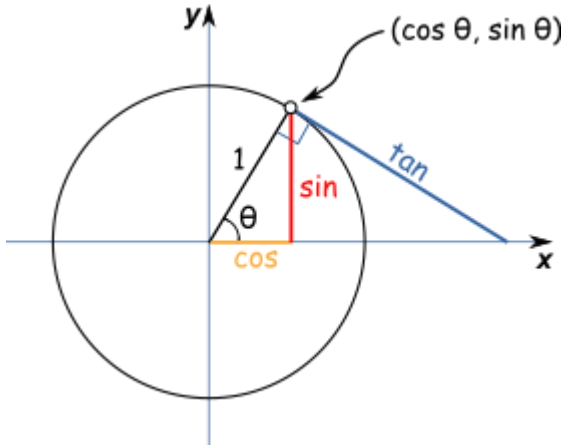
- The Euclidean distance between q and d_2 is large even though the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar



Use angle instead of distance

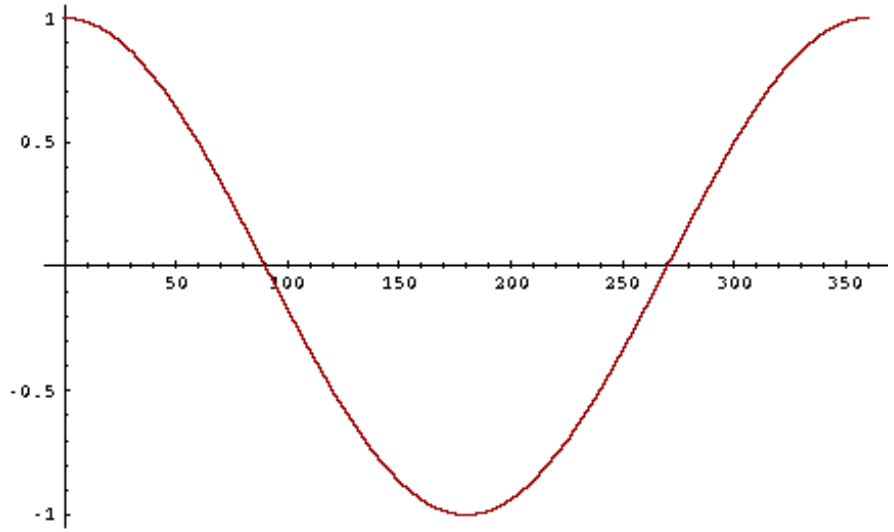
- Rank documents according to angle with query
  - experiment
    - take a document *d* and append it to itself
    - call this document *d*′ (d′ is twice as long as d)
  - "semantically" *d* and *d*′ have the same content
  - the angle between the two documents is 0, corresponding to maximal similarity ...
    ... even though the Euclidean distance between the two documents can be quite large

- *Key idea*: rank documents according to angle with query



From angles to cosines

- the following two notions are equivalent.
  - rank documents in *decreasing* order of the angle between query and document
  - rank documents in *increasing* order of `cosine(query,document)`
- cosine is a monotonically decreasing function for the interval [0°, 180°]



But how – *and why* – should we be computing cosines?

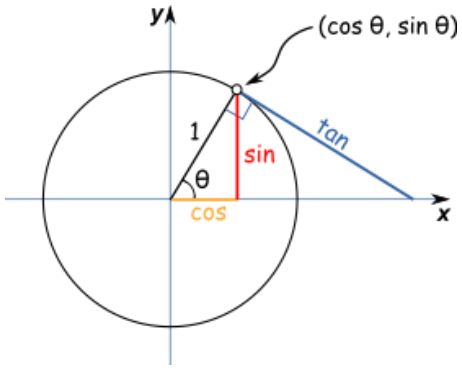
Length normalization

- 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

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

  - this maps vectors onto the unit sphere ...

$$\mathbf{x} = (1,1,1),\ \mathbf{x}' = \frac{\mathbf{x}}{\|\mathbf{x}\|_2} \approx (0.577, 0.577, 0.577),\ \|\mathbf{x}'\|_2 = 1$$

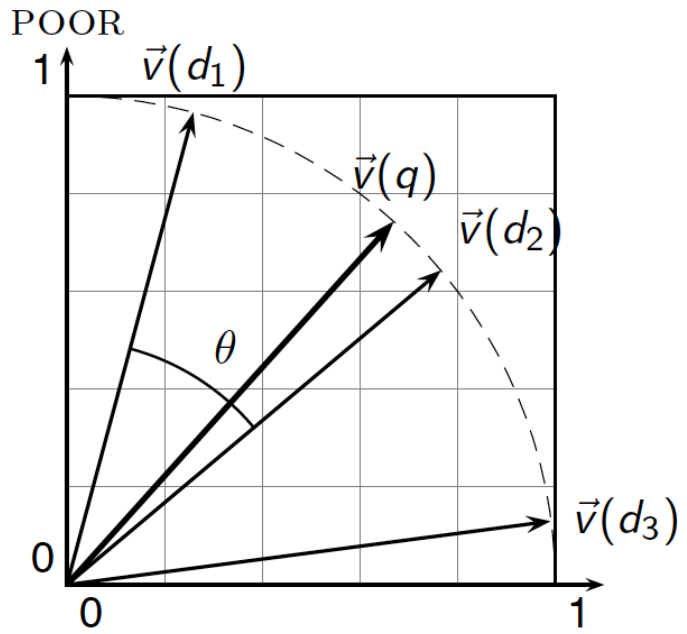


- As a result: longer documents and shorter documents yield weights of the same order of magnitude
  - effect on q and d_2 from earlier slide: similar vectors after length-normalization

Cosine similarity

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

- 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 $\vec{d}$
- cos($\vec{q}$, $\vec{d}$) is the cosine similarity of $\vec{q}$ and $\vec{d}$



For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$\cos\left(\vec{q},\vec{d}\right) = \vec{q}\cdot\vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

Cosine: example

- How similar are these novels?
  - SaS: Sense and Sensibility
  - PaP: Pride and Prejudice
  - 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

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

Cosine: example

- How similar are these novels?
  - cos(SaS,PaP) ≈ 0.789 ∗ 0.832 + 0.515 ∗ 0.555 + 0.335 ∗ 0.0 + 0.0 ∗ 0.0 ≈ 0.94.
  - cos(SaS,WH) ≈ 0.79
  - cos(PaP,WH) ≈ 0.69
  - Why do we have **cos(SaS,PaP) > cos(SaS,WH)**?

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 and **cosine normalization**

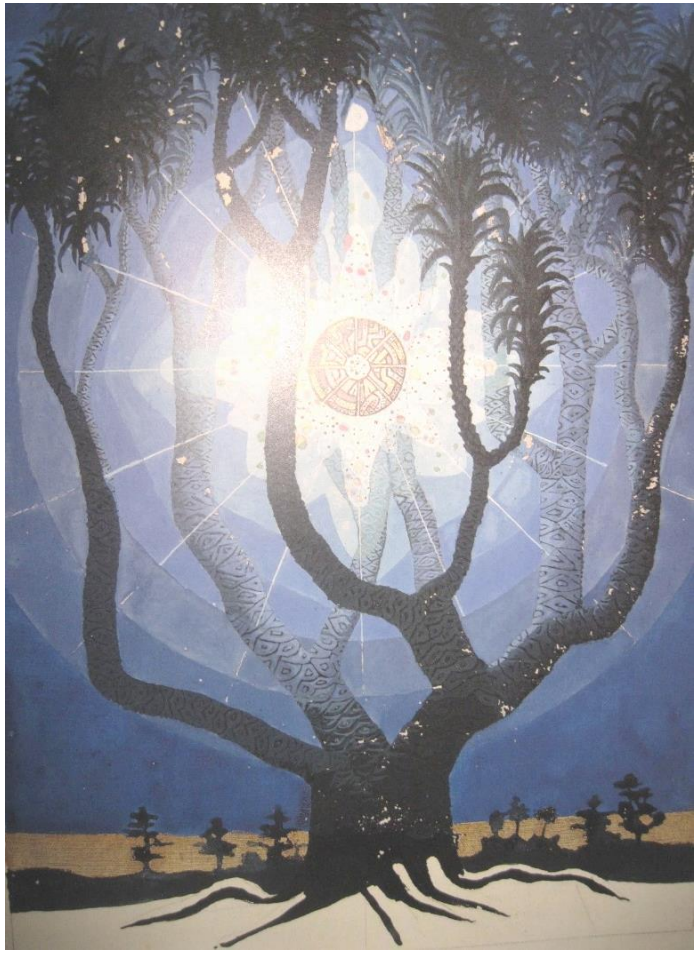
term	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

Computing scores in inverted indices

- How to efficiently compute cosine using an **inverted index**?
  - as well as alternative similarity metrics on vector spaces?

```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d] += w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

Outline



- Ranked retrieval
  - problems of Boolean model
  - ranked retrieval models
  - Jaccard-based scoring
- Term frequency model
  - bag of words
  - log-frequency weighting
- TF-IDF model
  - IDF weighting
  - TF-IDF weighting
- Vector spaces
  - formalizing vector space
  - cosine similarity
- **TF-IDF variants**
- **Final remarks**

tf-idf has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

Why is the base of the log in idf immaterial?

Weighting may differ in *queries* vs *documents*

- many search engines allow for **different weighting schema** for **queries** vs. **documents**

- **SMART notation:**
  - ***ddd.qqq*** using the acronyms from the previous table
  - defines the combination in use in an engine

- a very standard weighting scheme is: **lnc.ltn**
  - *document*: logarithmic tf (l as first character), no idf and cosine normalization
  - *query*: logarithmic tf (l in leftmost column), idf (t in second column), no normalization …

a bad idea?

Recommended variants

weighting scheme	document term weight	query term weight
1	$f_{i,j} * \log \frac{N}{n_i}$	$(0.5 + 0.5 \frac{f_{i,q}}{max_i \ f_{i,q}}) * \log \frac{N}{n_i}$
2	$1 + \log f_{i,j}$	$\log(1 + \frac{N}{n_i})$
3	$(1 + \log f_{i,j}) * \log \frac{N}{n_i}$	$(1 + \log f_{i,q}) * \log \frac{N}{n_i}$

- **Exercise**
  - using **lnc.ltn** weighting scheme
  - compute similarity between
    - *query*: “best car insurance”
    - *document*: “car insurance auto insurance”

TF-IDF exercise

- Using **lnc.ltn** weighting scheme
  - *query*: “best car insurance”
  - *document*: “car insurance auto insurance”

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n’lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Mid-results:

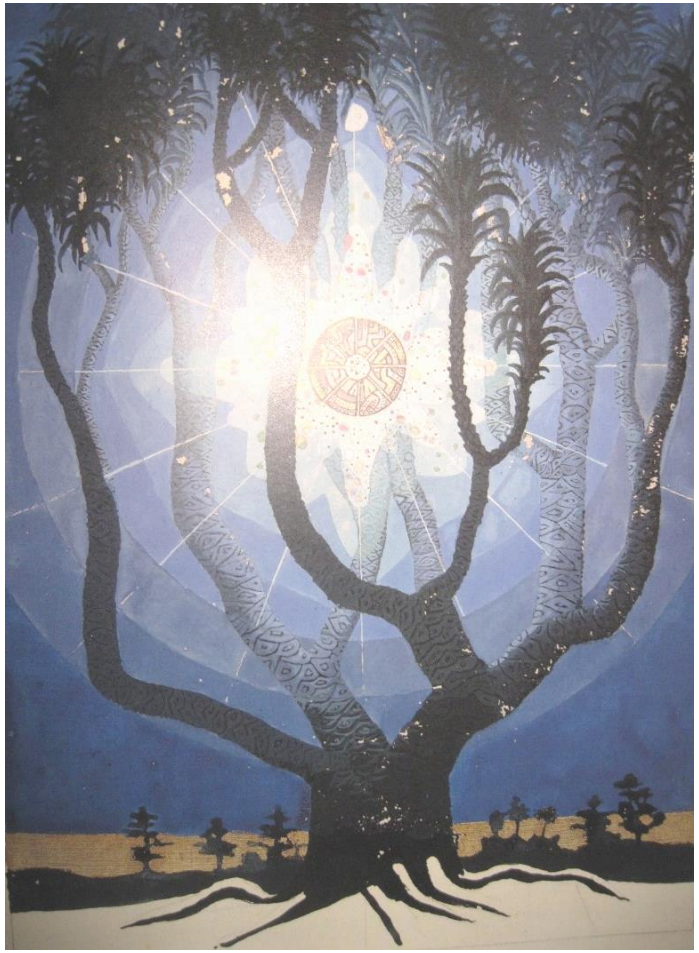
doc length $\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$

1/1.92 ≈ 0.52

1.3/1.92 ≈ 0.68

cosine(*q*,*d*) = 0 + 0 + 1.04 + 2.04 = 3.08

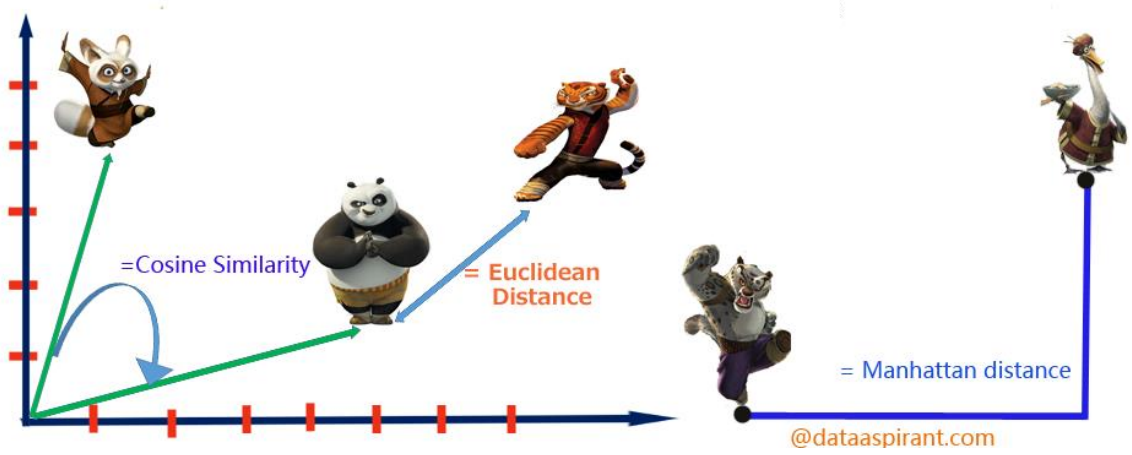
Outline



- **Ranked retrieval**
  - problems of Boolean model
  - ranked retrieval models
  - Jaccard-based scoring
- **Term frequency model**
  - bag of words
  - log-frequency weighting
- **TF-IDF model**
  - IDF weighting
  - TF-IDF weighting
- **Vector spaces**
  - formalizing vector space
  - cosine similarity
- **TF-IDF variants**
- **Final remarks**

Final remarks

- *Covered topics*
  - the importance of going beyond the Boolean model
  - the vector space model
    - ranked retrieval
    - tf-idf: best known traditional ranking scheme
    - query and documents as tf-idf vectors
    - cosine to measure query-document similarity



- *Pointers*
  - MIR book: chapter 4
  - IIR book: chapter 6
  - term weighting and cosine similarity tutorial for SEO folk http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html

Thank You

rmch@tecnico.ulisboa.pt