Information Retrieval

STA 325, Supplemental Material

Information Retrieval

One of the fundamental problems with having a lot of data is finding what you're looking for.

This is called **information retrieval**!

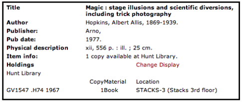
What we used to do

I want to learn about that magic trick with the rings!

Then: go to the library

Librarian

Card catalog



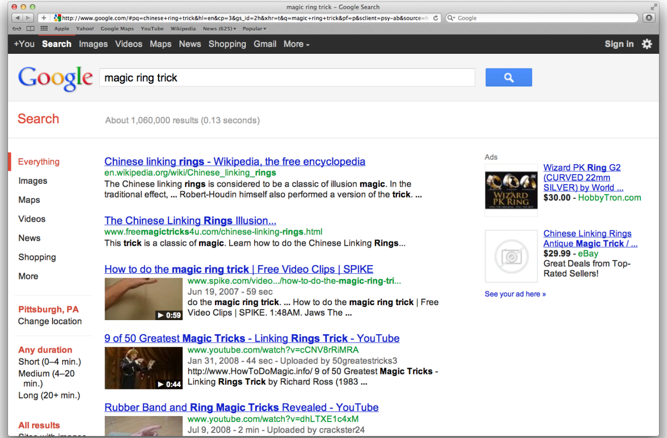
Title	**Magic : stage illusions and scientific diversions, including trick photography**	
Author	Hopkins, Albert Allis, 1869-1939.	
Publisher:	Arno,	
Pub date:	1977.	
Physical description	xii, 556 p. : ill. ; 25 cm.	
Item info:	1 copy available at Hunt Library.	
Holdings		Change Display
Hunt Library		
	CopyMaterial	Location
GV1547 .H74 1967	1Book	STACKS-3 (Stacks 3rd floor)

Metadata

Slow and expensive . . .

What we do now

Now: search the web!



How did Google do this?

Information retrieval and representations

Information retrieval: given a set of documents (e.g., webpages), our problem is to pull up the k most similar documents to a given query (e.g., “magic ring trick”)

First step is to think of a way of representing these documents. We want our representation to:

- Be easy to generate from the raw documents, and be easy to work with
- Highlight important aspects of the documents, and suppress unimportant aspects

There is kind of a **trade-off** between these two ideas

Try using the meaning of documents


Chinese linking rings – Wikipedia, the free encyclopedia

http://en.wikipedia.org/wiki/Chinese_linking_rings

Apple Yahoo! Google Maps YouTube Wikipedia News (625) Popular

Log in / create account

Article Discussion Read Edit Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Chinese linking rings

From Wikipedia, the free encyclopedia

The **Chinese linking rings** is considered[by whom?] to be a classic of illusion magic. In the traditional effect, solid metal rings appear to link and unlink, pass through each other, and form chains and other complex patterns and configurations. The rings may even be handed out to audience members for examination. Sometimes an audience member is invited onto the stage to perform alongside the magician as part of a 'do as I do' routine.

It was believed[by whom?] that Ching Ling Foo was one of the early performers of the linking rings in the form known today. Speculation about the rings' origin has been traced to Turkey, Egypt and the Middle East and as far back as the 1st century. A painting by Giacomo Mantegazza in 1876 shows a harem girl holding a set of rings above her head.

Christian Farla performs Linking Rings routine

French magician Philippe (Jacques Andre Noel Talon) was one of the first recorded magicians to use the rings. During the second half of his show, he donned an Oriental robe and conical cap. He took 8-inch (200 mm) solid metal rings and had the audience examine them. He claimed that he learned them from a troupe of Chinese jugglers in Britain. After the rings were returned, Philippe proceeded to link and unlink them. From there he made combination of glittering circles into complicated designs. According to Robert-Houdin, he said that for a finale, Philippe blew on them and they fell separately on the floor.[1] Robert-Houdin himself also performed a version of the trick.

Recent magicians who are known for their performances with the linking rings include Dai Vernon, Richard Ross, Jack Miller, Michael Skinner, Whit Haydn, Ian Ray - "The Genie Ali

What if we tried to represent the meaning of documents? E.g.,

```
type.of.trick = sleight of hand;
date.of.origin = 1st century;
place.of.origin = Turkey, Egypt;
name.origin = Chinese jugglers
in Britain; ...
```

This would be good in terms of our second idea (useful and efficient data reduction), but not our first one (extremely hard to generate, and even hard to use!)

Bag-of-words (BoW) representation

Bag-of-words representation of a document is very simple-minded: just list all the **distinct words** and **how many times they appeared**. E.g.,

```
magic = 29; ring = 34; trick = 6; illusion = 7; link = 9; ...
```

Very easy to generate and easy to use (first idea), but is it too much of a reduction, or can it still be useful (second idea)?

Idea: by itself "ring" can take on a lot of meanings, but we can learn from the other words in the document besides"ring". E.g.,

- Words "perform", "illusion", "gimmick", "Chinese", "unlink", "audience", "stage" suggest the right type of rings
- Words "diamond", "carat", "gold", "band", "wedding", "engagement", "anniversary" suggest the wrong type

Bag of Words

The name comes from the following:

1. Print out the text
2. Cut the text (paper) into little pieces so each word is on its own piece
3. Now throw the pieces of paper into a bag.

This is **literally a bag of words**. And it is a set of purely textual features (one per distinct word).

Will a BoW be useful for us? Let's find out!

Counting words

Recall problem: given a query and a set of documents, find the k documents most similar to the query

Counting words

Counting words:

- First make a list of all of the words present in the documents and the query
- Index the words $w = 1, \dots W$ (e.g., in alphabetical order), and the documents $d = 1, \dots D$ (just pick some order)
- For each document d, count how many times each word w appears (could be zero), and call this X_{dw}.
- The vector $X_d = (X_{d1}, \dots X_{dW})$ gives us the word counts for the dth document
- Do the same thing for the query: let Y_w be the number of times the wth word appears, so the vector $Y = (Y_1, \dots Y_W)$ contains the word counts for the query

Simple example

Documents:

1: "Beka loves statistics." and 2: "Noah hates, hates statistics!"

Y: "hates statistics"

$D = 2$ documents and $W = 5$ words total. For each document and query, we count the number of occurences of each word:

	hates	Beka	loves	Noah	statistics
X_1	0	1	1	0	1
X_2	2	0	0	1	1
Q	1	0	0	0	1

This is called the **document-term** matrix

Distances and similarity measures

We represented each document X_d and query Y in a convenient vector format.

Now how to measure similarity between vectors, or equivalently, dissimilarity or distance?

Measures of distance between n-dimensional vectors X, Y:

- The ℓ_2 or Euclidean distance is
$$\|X - Y\|_2 = \sqrt{\sum_{i=1}^{n} (X_i - Y_i)^2}$$
- The ℓ_1 or Manhattan distance is
$$\|X - Y\|_1 = \sum_{i=1}^{n} |X_i - Y_i|$$

Basic idea: find k vectors X_d with the smallest $\|X_d - Y\|_2$

(Note: ℓ_1 distance doesn't work as well here)

Bigger Example

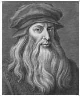
Documents: 8 Wikipedia articles

- 4 about the TMNT Leonardo, Raphael, Michelangelo, and Donatello and
- 4 about the painters of the same name


1

2

3

4

5

6

7

8

Query: "Raphael is cool but rude, Michelangelo is a party dude!"

Bigger Example

- Data is scraped from Wikipedia
- Load the data

```
library(tm)
```

```
## Warning: package 'tm' was built under R version 3.4.4
```

```
## Loading required package: NLP
```

```
load(file="docs.Rdata")
```

Bigger Example

Pre-process and create document term matrix (DTM)

```
# make a corpus of documents
corp = VCorpus(VectorSource(docs))
```

```
## Warning in as.POSIXlt.POSIXct(Sys.time(), tz = "GMT"): u
## 'zone/tz/2018e.1.0/zoneinfo/America/Toronto'
```

```
# create the DTM and clean
dtm = DocumentTermMatrix(corp, control=list(tolower=TRUE,
removePunctuation=TRUE,removeNumbers=TRUE))
```

Bigger Example

What does the DTM look like?

```
dtm
```

```
## <<DocumentTermMatrix (documents: 9, terms: 6844)>>
## Non-/sparse entries: 12449/49147
## Sparsity           : 80%
## Maximal term length: 36
## Weighting          : term frequency (tf)
```

Bigger Example

Let's convert the `dtm` into a matrix and see what it looks like!

```
mydtm = as.matrix(dtm)
mydtm[,100:110]
```

```
##     Terms
## Docs adams adaptation adapted adaption added adding addington addition
##    1     0          1       0        1     0      1         0        0
##    2     0          2       0        0     0      1         0        0
##    3     0          0       0        1     0      1         0        0
##    4     0          0       0        0     0      2         0        0
##    5     1          0       0        0     0      1         0        0
##    6     0          1       1        0     2      0         0        0
##    7     0          0       0        0     1      0         1        1
##    8     0          0       0        0     0      0         0        0
##    9     0          0       0        0     0      0         0        0
##     Terms
## Docs additional additionally addressed
##    1          2            0         0
##    2          2            0         0
##    3          3            1         0
##    4          2            0         0
##    5          2            0         0
##    6          2            0         1
##    7          1            0         1
##    8          2            0         0
```

Bigger Example

Let's now compute the un-normalized distance.

```r
# Compute unnormalized distance
q = c("but","cool","dude","party","michelangelo","raphael","rude")
dist = sqrt(rowSums((scale(mydtm,center=mydtm[9,],scale=F)^2)))
mat = cbind(mydtm[,q],dist)
colnames(mat) = c(q,"dist")
head(mat)
```

```
##   but cool dude party michelangelo raphael rude     dist
## 1  19    0    0     0            4      24    0 309.4398
## 2   8    1    0     0            7      45    1 185.1621
## 3   7    0    4     3           77      23    0 330.9577
## 4   2    0    0     0            4      11    0 220.1817
## 5  17    0    0     0            9       6    0 927.5020
## 6  36    0    0     0           17     100    0 646.2360
```

Bigger Example

Here, we output the un-normalized distance into a table.

	but	cool	dude	party	michelangelo	raphael	rude	dist
doc 1	19	0	0	0	4	24	0	309.453
doc 2	8	1	0	0	7	45	1	185.183
doc 3	7	0	4	3	77	23	0	330.970
doc 4	2	0	0	0	4	11	0	220.200
doc 5	17	0	0	0	9	6	0	928.467
doc 6	36	0	0	0	17	101	0	646.474
doc 7	10	0	0	0	159	2	0	527.256
doc 8	2	0	0	0	0	0	0	196.140
query	1		1 1	1	1	1	1	0.000

Does this matrix make sense given that the document lengths vary?
Let's think about this some more.

Varying document lengths and normalization

Different documents have different lengths. Total word counts:

doc 1	doc 2	doc 3	doc 4	doc 5	doc 6	doc 7	doc 8	query
3114	1976	3330	2143	8962	6524	4618	1766	7

- Wikipedia entry on Michelangelo the painter is almost twice as long as that on Michelangelo the TMNT (6524 vs 3330 words).
- And query is only 7 words long!
- We should normalize the count vectors X_d and Y in some way.

Varying document lengths and normalization

- Document length normalization: divide X by its sum,

$$X \leftarrow X / \sum_{w=1}^{W} X_w$$

- ℓ_2 length normalization: divide X by its ℓ_2 length,

$$X \leftarrow X / \|X\|_2$$

- apparently, normalizing by ℓ_2 length tends to better de-emphasize words that are rare in the document.

Back to Wikipedia example

```
# Document length normalization
mydtm.dl = mydtm/rowSums(mydtm)
dist.dl = sqrt(rowSums((scale(mydtm.dl,center=mydtm.dl[9,]
mat.dl = cbind(mydtm.dl[,q],dist.dl)
colnames(mat.dl) = c(q,"dist.dl")

# l2 length normalization
mydtm.l2 = mydtm/sqrt(rowSums(mydtm^2))
dist.l2 = sqrt(rowSums((scale(mydtm.l2,center=mydtm.l2[9,]
mat.l2 = cbind(mydtm.l2[,q],dist.l2)
colnames(mat.l2) = c(q,"dist.l2")
```

Back to Wikipedia example

```r
# Compare two normalization schemes
a = cbind(mat.dl[,8],mat.l2[,8])
colnames(a) = c("dist/doclen","dist/l2len")
rownames(a) = paste(rownames(a),
c("(tmnt leo)","(tmnt rap)","(tmnt mic)","(tmnt don)",
"(real leo)","(real rap)","(real mic)","(real don)"))
```

Back to our Wikipedia example

```
a
```

```
##                dist/doclen dist/l2len
## 1 (tmnt leo)    0.3852639   1.373039
## 2 (tmnt rap)    0.3777607   1.321860
## 3 (tmnt mic)    0.3781185   1.319045
## 4 (tmnt don)    0.3887840   1.393432
## 5 (real leo)    0.3905483   1.404963
## 6 (real rap)    0.3820675   1.349479
## 7 (real mic)    0.3812152   1.324745
## 8 (real don)    0.3935294   1.411485
## 9 (tmnt leo)    0.0000000   0.000000
```

Great!

So far we've dealt with varying document lenghts. What about some words being more helpful than others? Common words, especially, are not going to help us find relevant documents

Common words and IDF weighting

Someone asked in class last time about selectively paying less attention to certain words, especially common words, and more to the rest. This is an excellent notion.

To deal with common words, we could just keep a list of words like “the‘‘, “this”,”that”, etc. to exclude from our representation. But this would be both too crude and time consuming.

Inverse document frequency (IDF) weighting is smarter and more efficient

- For each word w, let n_w be the number of documents that contain this word.
- Then for each vector X_d and Y, multiply wth component by $\log(D/n_w)$.

If a word appears in every document, then it gets a weight of zero, so effectively tossed out of the representation.

IDF

1. Advantages: can get rid of unimportant variables.
2. Disadvantages: could throw out too many features, with small collection of docs. Also, can downweight or throw out informative features, e.g., all of our articles could likely contain Michelangelo, but how many times it appears is important!

Putting it all together

Think of the document-term matrix:

	word 1	word 2	...	word W
doc 1				
doc 2				
$\vdots$				
doc D				

- Normalization scales each row by something (divides a row vector X by its sum $\sum_{i=1}^{W} X_i$ or its ℓ_2 norm $\|X\|_2$)
- IDF weighting scales each column by something (multiplies the wth column by $\log(D/n_w)$)
- We can use both, just normalize first and then perform IDF weighting

Back to our Wikipedia example, again

```
{\footnotesize
\begin{verbatim}
                 dist/doclen/idf dist/l2len/idf
doc 1 (tmnt leo)           0.623          1.704
doc 2 (tmnt rap)           0.622          1.708
doc 3 (tmnt mic)           0.620          1.679
doc 4 (tmnt don)           0.623          1.713
doc 5 (real leo)           0.622          1.693
doc 6 (real rap)           0.622          1.703
doc 7 (real mic)           0.622          1.690
doc 8 (real don)           0.624          1.747
query                      0.000          0.000
\end{verbatim}}
```

Oops! This didn't work as well as we might have hoped. Why?

(Hint: our collection only contains 8 documents and 1 query . . .)

Limitations of IDF

Recall that the IDF weight of w is

$$\text{IDF}(w) =: log(N/n_w)$$

.

Note that if w appears in only a few documents, it will get a weight of about log(N), and all documents containing w will tend to be close to each other, which is what we see in the Wikipedia example.

Stemming

Stemming takes derived forms of words (like “cars”, “flying”) and reduces them to their stem (“car”, “fly”). Doing this well requires linguistic knowledge (so the system doesn’t think the stem of “potatoes” is “potatoe”, or that “gravity” is the same as “grave”), and it can even be harmful.

Stemming

Having words “connect”, “connects”, “connected” “connecting”, “connection”, etc. in our representation is extraneous. Stemming reduces all of these to the single stem word “connect”

Can a simple list of rules provide perfect stemming?

It seems not: consider “connect” and “connectivity"“, but"relate" and “relativity”; or “sand” and “sander”, but “wand” and “wander”.

Stemming

Stemming also depends on the language. Let's consider Turkish.

For example, Turkish is what is known as an "aggulutinative" language, in which grammatical units are "glued together" to form compound words whose meaning would be a whole phrase or sentence in English.

For example, gelemiyebelirim, "I may be unable to come," yapabilecekdiyseniz, "if you were going to be able to do," or calistirilmamaliymis, "supposedly he ought not to be made to work."

German does this too, but not so much.

This causes problems with Turkish-language applications, because many sequences-of-letters-separated-by-punctuation are effectively unique.

Feedback

People are usually better at confirming the relevance of something that's been found, rather than explaining what they're looking for in the first place

Queries are users trying to explain what the user is looking for (to a computer), so this can be pretty bad.

An important idea in data mining is that people should do things at which they are better than computers and vice versa

Feedback and Rocchio's algorithm

Rocchio's algorithm takes feedback from the user about document relevance and then refines the query and repeats the search giving more weight to what user's like, and less to what user's don't like.

Feedback and Rocchio's algorithm

1. User gives an initial query Y
2. Computer returns documents it believes to be relevant, and user divides these into sets: revelant R and not relevant NR
3. Computer updates the query string as

$$Y \leftarrow \alpha Y + \frac{\beta}{|R|} \sum_{X_d \in R} X_d - \frac{\gamma}{|NR|} \sum_{X_d \in NR} X_d$$

4. Repeat steps 2 and 3

We have to choose constants $\alpha, \beta, \gamma > 0$ (interpretations?)

Notation:

- $|R|$ and $|NR|$ are the number of relevant and non-relevant documents
- α say how much continuity there is between the old search and the new search.
- β and γ guage our preference for recall (we find more relevant documents) versus precision (more of what we find is relevant).

Text mining in R

Helpful methods implemented in the package tm, available on the CRAN repository

```
dtm <- DocumentTermMatrix(corp,
  control=list(tolower=TRUE,
  removePunctuation=TRUE,
  removeNumbers=TRUE,
  stemming=TRUE,
  weighting=weightTfIdf))
```

Recap: information retrieval

In information retrieval we have a collection of documents and a query (this could just be one of our documents), and our goal is to find the k most relevant documents to the query

Achieved by using a bag-of-words representation, where we just count how many times each word appears in each document and the query

This gives us a document-term matrix. We can hence return the k documents whose word count vectors are closest to the query vector (these are rows of the matrix) in terms of ℓ_2 distance

Important extensions include normalization (row scaling) and IDF weighting (column scaling) the document-term matrix, before computing distances. Other extensions: stemming, feedback