Information retrieval

Ryan Tibshirani
Data Mining: 36-462/36-662

January 17 2013

What we use to do

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

Then: go to the library

Librarian

Card catalog

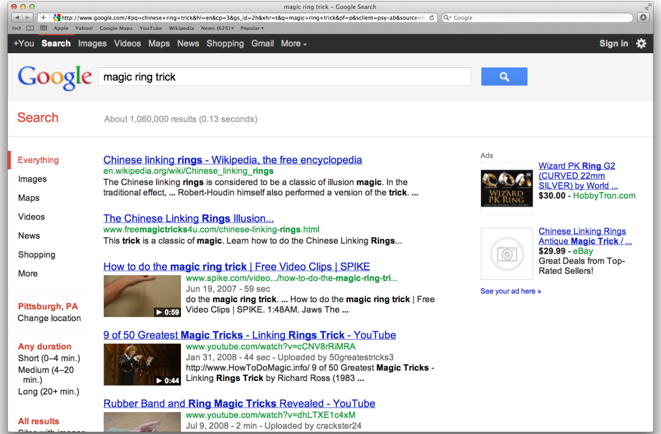
Title	**Magic : stage illusions and scientific diversions, including trick photography**	
Author	Hopkins, Albert Allis, 1869-1939.	
Publisher:	Arno,	
Pub date:	1977.	
Physical description	xii, 556 p. : ill. ; 25 cm.	
Item info:	1 copy available at Hunt Library.	
Holdings		Change Display
Hunt Library		
	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

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 representation

Bag-of-words representation of a document is very simple-minded: just list all the 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

Counting words

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

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: "Ryan loves statistics." and 2: "Jess hates, hates statistics!"

Query: "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	Jess	loves	Ryan	statistics
X_1	0	0	1	1	1
X_2	2	1	0	0	1
Y	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

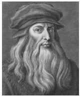
Documents: 8 Wikipedia articles, 4 about the TMNT Leonardo, Raphael, Michelangelo, and Donatello, and 4 about the painters of the same name

1

2

3

4



5

6

7

8

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

	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

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

- 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,

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

Back to our Wikipedia example

	dist/doclen	dist/l2len
doc 1 (tmnt leo)	0.385	1.373
doc 2 (tmnt rap)	0.378	1.322
doc 3 (tmnt mic)	0.378	1.319
doc 4 (tmnt don)	0.389	1.393
doc 5 (real leo)	0.390	1.405
doc 6 (real rap)	0.382	1.349
doc 7 (real mic)	0.381	1.325
doc 8 (real don)	0.393	1.411
query	0.000	0.000

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

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

(Future reference: IDF performs something like variable selection)

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

	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

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

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

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 also depends on the language. Apparently it is easier in English than it is in:

- German, a "fusional" language, e.g.,

  Hubschrauberlandeplatz = helicopter landing pad

- Turkish, an "agglutinative" language, e.g.,

  Turklestiremedigimizlerdensinizdir = maybe you are one of those whom we were not able to Turkify'

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

Rocchio's algorithm takes feedback from the user about relevance, and then refines the query and repeats the search

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?)

Text mining in R

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

E.g.,

```
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

Next time: PageRank

Taking advantage of the link structure of the web