An Introduction to 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

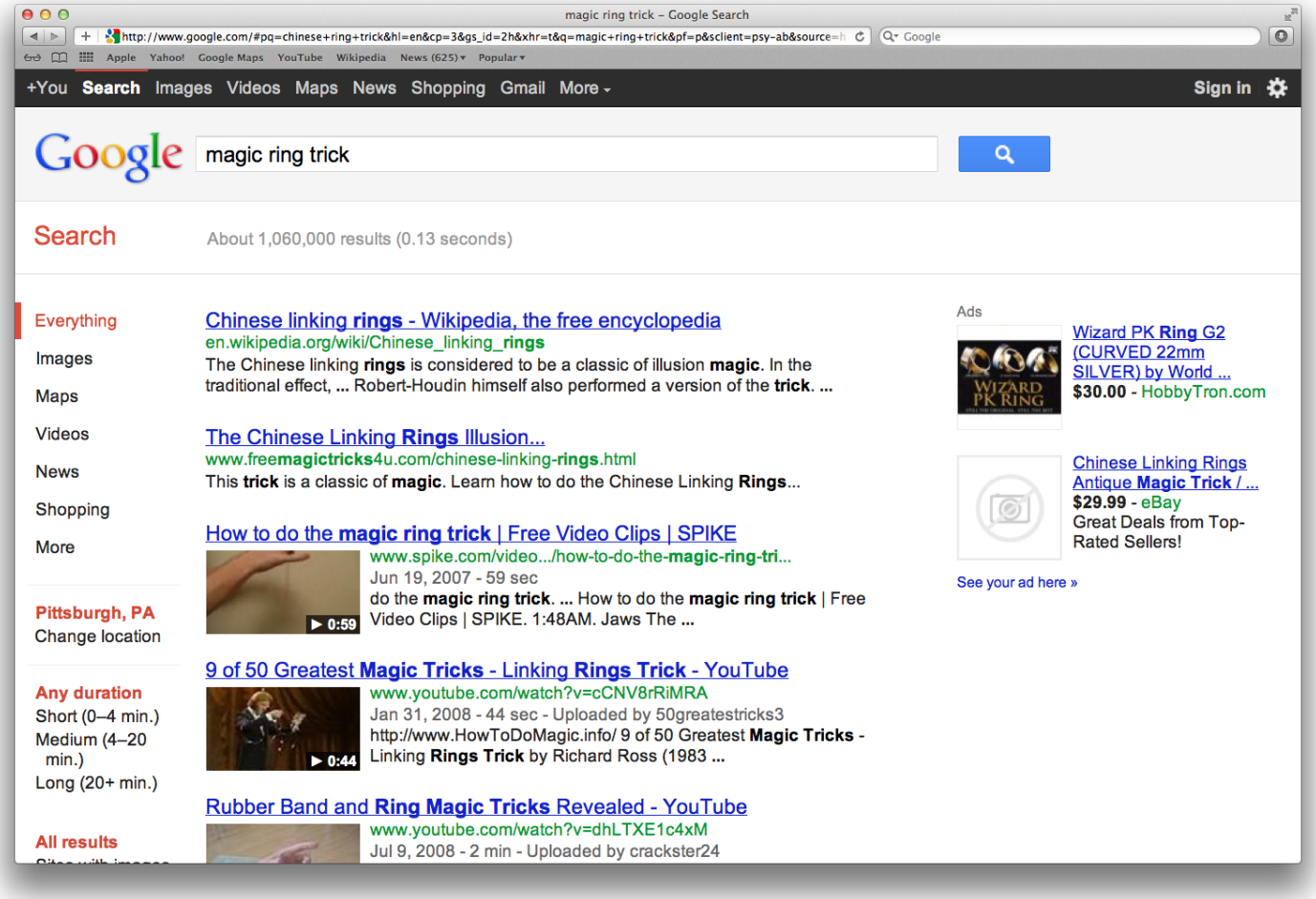
	CopyMaterial	Location
GV1547 .H74 1967	1Book	STACKS-3 (Stacks 3rd floor)

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

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 $w = 1, \ldots W$ (e.g., in alphabetical order), and the documents $d = 1, \ldots D$ (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 $X_d = (X_{d1}, \ldots X_{dW})$ 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 $Y = (Y_1, \ldots Y_W)$ 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

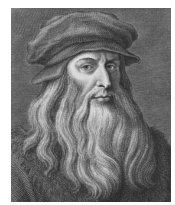
- 4 about the TMNT Leonardo, Raphael, Michelangelo, and Donatello and
- 4 about the painters of the same name

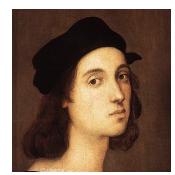

1

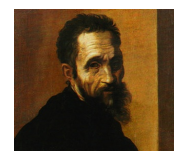

2

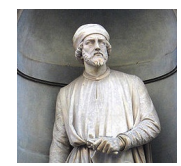

3


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

```
# load the text mining package
library(tm)
```

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

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

```
# load our Wikipedia data (8 articles)
load(file="docs.Rdata")
```

tm package

- We're going to take advantage of the `tm` package in `R`.
- In order to grab the Wikipedia text stored in `docs.Rdata`, we'll use the `VectorSource` function.
- We first take our documents and put them into a corpus. Next, we'll take the corpus and transform these into a document term matrix.
- Please see `https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf` for commands available in this package

Create a corpus of documents

```
# 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'
```

```
head(corp)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed):
## Content:  documents: 6
```

Create the document term matrix (DTM)

```
# create the DTM and perform some minor cleaning
dtm = DocumentTermMatrix(corp, control=list(tolower=TRUE,
removePunctuation=TRUE,removeNumbers=TRUE))
```

The DTM

Here we get a sample of terms and can see a sparse representation of the DTM. Converting the DTM into a matrix, will tranform it to a dense matrix (and this is more memory intensive to store for large matrices).

```
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 9, terms: 6844)>>
## Non-/sparse entries: 12449/49147
## Sparsity           : 80%
## Maximal term length: 36
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs and for his leonardo michelangelo raphael that the was with
##    1 103  13  65       82             4      24   36 235  31   33
##    2  61  11  45       14             7      45   13 135  23   19
##    3 112  21  79       18            77      23   40 254  46   31
##    4  63  14  42        6             4      11   16 178  16   21
##    5 364  83 160      210             9       6  105 739 128   94
##    6 230  74 156       10            17     100   41 514 106   48
##    7 135  36  74        2           159       2   37 447  85   18
##    8  59  30  15        0             0       0    9 161  18   14
##    9   0   0   0        0             1       1    0   0   0    0
```

Convert to Matrix

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 Euclidan distance (un-normalized).
- Use the `scale(x, center = TRUE, scale = TRUE)` function to help us!

```
# Inspect the columns of row 9 (query).
head(mydtm[9,])
```

```
##   --louvre --national   --uffizi   -convent   -giorgio    -louvre
##          0          0          0          0          0          0
```

```
# Intermediate Check
dim(scale(mydtm,center=mydtm[9,],scale=FALSE)^2)
```

```
## [1]    9 6844
```

```
# Compute Euclidean distance (vectorized version)
dist = sqrt(rowSums((scale(mydtm,center=mydtm[9,],scale=FALSE)^2)))
head(dist)
```

```
##        1        2        3        4        5        6
## 309.4398 185.1621 330.9577 220.1817 927.5020 646.2360
```

Bigger Example

Let's visualize this in a matrix, so we can analze the results

```
# the query
q = c("but","cool","dude","party","michelangelo","raphael","rude")
# make a matrix of the query and distance
mat = cbind(mydtm[,q],dist)
colnames(mat) = c(q,"dist")
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
## 7  10    0    0     0           159       2    0 527.1783
## 8   2    0    0     0             0       0    0 196.1199
## 9   1    1    1     1             1       1    1   0.0000
```

Bigger Example

Here, we output the un-normalized Euclidean 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?

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!
- That is, we want to make sure we're taking into the length of each document so that we can do more fair comparisons.
- 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,

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

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

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

- One can show emprically or theoretically that normalizing by ℓ_2 length tends to better de-emphasize words that are rare (not common) 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,],scale=F)^2))
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,],scale=F)^2))
mat.l2 = cbind(mydtm.l2[,q],dist.l2)
colnames(mat.l2) = c(q,"dist.l2")
```

Back to Wikipedia example

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

Back to Wikipedia example

```
wikipedia
```

```
##                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
```

How do we interpret our results?

1. According to the document length distance, doc 2 (tmnt rap) is closest to our query since it's distance 0.377 is smallest.
2. According to the Euclidean distance, doc 3 (tmnt mic) is closest to our query since it's distance 1.319 is smallest.

Can we do better than this? (Notice that many of the distances are very close in magnitude).

What's next

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 weighting

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

- For each word w, let n_w be the number of documents that contain the word w.
- 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 it's effectively tossed out of the representation.

(Future reference: IDF performs something like variable selection).

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

Wikipedia (with IDF and Document Length Norm)

```
# Weight the DTM using IDF
 dtm.idf = DocumentTermMatrix(corp,
    control=list(tolower=TRUE,
    removePunctuation=TRUE,
    removeNumbers=TRUE,weighting=weightTfIdf))
mydtm.idf.dl = as.matrix(dtm.idf)
mat.idf.dl = cbind(mydtm.idf.dl[,q],
sqrt(colSums((t(mydtm.idf.dl[1:9,])-mydtm.idf.dl[9,])^2)))
colnames(mat.idf.dl) = c(q,"dist")
```

For more information about `weighting`, see `https://www.rdocumentation.org/packages/tm/versions/0.7-5/topics/weightTfIdf`

Wikipedia (with IDF and Euclidean Norm)

```
mydtm.idf.l2 = mydtm.idf.dl * rowSums(mydtm) / sqrt(rowSums(mydtm^2))
mat.idf.l2 = cbind(
mydtm.idf.l2[,q],
sqrt(colSums((t(mydtm.idf.l2[1:9,])-mydtm.idf.l2[9,])^2)))
colnames(mat.idf.l2) = c(q,"dist")
```

Wikipedia (with IDF)

```
wikipedia.idf = cbind(mat.idf.dl[,8], mat.idf.l2[,8])
colnames(wikipedia.idf) = c("dist/doclen/idf", "dist/l2len/idf")
rownames(wikipedia.idf) = paste(rownames(wikipedia),
c("(tmnt leo)","(tmnt rap)","(tmnt mic)","(tmnt don)",
"(real leo)","(real rap)","(real mic)","(real don)"))
```

Wikipedia (with IDF)

```
wikipedia.idf
```

```
##                          dist/doclen/idf dist/l2len/idf
## 1 (tmnt leo) (tmnt leo)       0.6225164       1.704168
## 2 (tmnt rap) (tmnt rap)       0.6215259       1.708162
## 3 (tmnt mic) (tmnt mic)       0.6200730       1.679458
## 4 (tmnt don) (tmnt don)       0.6228930       1.712952
## 5 (real leo) (real leo)       0.6223648       1.693183
## 6 (real rap) (real rap)       0.6224201       1.702610
## 7 (real mic) (real mic)       0.6223114       1.687019
## 8 (real don) (real don)       0.6243245       1.744077
## 9 (tmnt leo) (tmnt leo)       0.0000000       0.000000
```

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.

You may wish to read more about this on your own or take more advanced machine learning classes.

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