Lecture 1: Similarity Searching and Information Retrieval

36-350, Data Mining

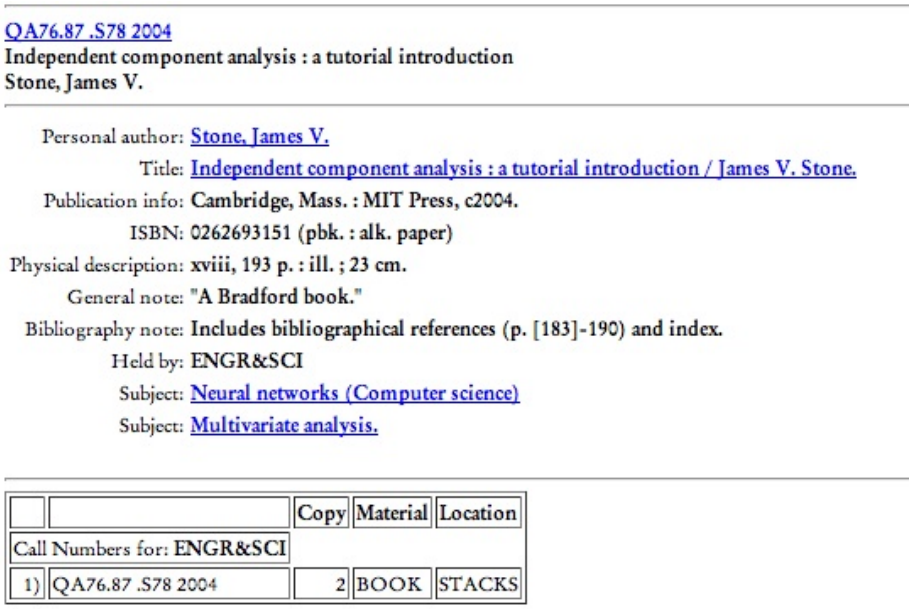
26 August 2009

READINGS: *Principles of Data Mining*, ch. 1, and sections 14.1 and 14.3.0–14.3.1.

One of the fundamental problems with having a lot of data is finding what you're looking for. This is called **information retrieval**.

The oldest approach is to have people create data about the data, **metadata**, to make it easier to find relevant items. Library catalogues are like this (Figure 1): people devise detailed category schemes for books, magazines, etc. For instance, a book has a title, one or more authors (possibly "anonymous" or pseudonyms), a publisher, a place of publication, a date of publication, possibly an ISBN, and its contents belong to one or more subject topics, with a lot of work going in to designing the set of subject-matter topics. A magazine doesn't have an author, it has multiple volumes with multiple dates, and it has an ISSN instead of an ISBN, if it has a number like that at all. (Nowadays we'd call this sort of scheme an **ontology**.) Having fixed on a scheme like this, people would actually examine the objects, decide which categories they belong to, and write down that information along with their location on the shelves, and then copy this information so that it appeared in multiple places — by title, by author, by subject, etc. This worked OK for a few thousand years, but it needs people, who are slow and expensive and don't scale. It's not feasible for searching census records, or purchase histories at an online store, or the Web.

The next oldest approach is Boolean queries: things like "all census records of Presbyterian or Methodist plumbers in Rhode Island with at least two but no more than five children". The first electronic data-processing machines were invented about 120 years ago to do searches like this. The advantages are that it's very easy to program, it doesn't need a lot of human intervention, and sometimes it's exactly what you want to do, say if you're taking a census. But there are disadvantages: most people don't think in Boolean queries; it works best when it's dealing with **structured** data, like census records, not **unstructured** data like most documents on the Web; and it's got no sense of *priority*. Suppose you're loking for a used car, thinking of buying a 2001-model-year Saturn, and want to know what problems they're prone to. Imagine doing a Boolean search of the whole Web for "Saturn AND 2001 AND problems". *Some*



QA76.87 .S78 2004
Independent component analysis : a tutorial introduction
Stone, James V.

Personal author: Stone, James V.
Title: Independent component analysis : a tutorial introduction / James V. Stone.
Publication info: Cambridge, Mass. : MIT Press, c2004.
ISBN: 0262693151 (pbk. : alk. paper)
Physical description: xviii, 193 p. : ill. ; 23 cm.
General note: "A Bradford book."
Bibliography note: Includes bibliographical references (p. [183]-190) and index.
Held by: ENGR&SCI
Subject: Neural networks (Computer science)
Subject: Multivariate analysis.

		Copy	Material	Location
Call Numbers for: ENGR&SCI				
1)	QA76.87 .S78 2004	2	BOOK	STACKS

Figure 1: Example of **metadata** (from `cameo.library.cmu.edu`). This is information *about* the book, which is not part of its actual contents. Notice that this is **structured**: every book in the catalogue will have a title, an author (possibly "anonymous"), one or more subjects, a location, etc. All of this information is humanly-generated.

Figure 2: Example of the structured meta-data provided with the *New York Times Annotated Corpus*. Notice in particular the subject classifications. From `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2008T19`. (Text and annotations copyright by the New York Times Co., used here for educational purposes per the terms of the licensing agreement.)

of those documents will be just what you want, but you'll also get a lot about the planet Saturn, about the novel *2001: A Space Odyssey* (set at Saturn), and so on.

This is where **searching by similarity** comes in. Suppose you can find *one* Web page which is about problems in 2001-model Saturns. We're going to see today how you can tell your computer "find me more pages like this". We will see later how you can avoid the step of initially lucking into the first page, and how you can use the same sort of trick to search other kinds of data — say images on the Web, or hospital patient records, or retail transactions, or telephone-call records.

To illustrate these ideas concretely, we're going to use a part of the *New York Times Annotated Corpus* (Sandhaus, 2008). This consists of 1.8×10^6 stories from the *Times*, from 1987 to 2007, which have been hand-annotated with metadata about their contents. (See Figure 2.) We are going to look at methods for information retrieval which do *not* use the metadata, but having it there will help us determine how well those methods are working. You will get to use a small part of this data set in the homework.[1]

[1] The metadata and annotations are in a language called XML. You will *not* have to learn it.

1 Representation

The crucial first step is to chose a representation of the data. There are multiple considerations here.

- The representation ought to be something our methods can work with easily.
- The representation ought to be something we can easily generate from the raw data.
- The representation ought to highlight the important, helpful aspects of the data, and suppress others. (If the representation *didn't* ignore some aspects of the data, it would be the data again.)

One of the things we'll see is that getting a good representation — trading these concerns off against each other — is at least as important as picking the right algorithms.

For today, we are searching for documents by similarity, so our data are natural-language documents.

We could try to represent the *meaning* of the documents. Look Figure 2 again. The abstract reads "Chairs of the 1920s and 30s are featured at Johnson & Hicks, new home furnishing store in TriBeCa". We could try to represent the meaning here in something like logical notation:

```
exhibit.of(chairs(age-1920--1930),Johnson&Hicks,now)
is.a(Johnson&Hicks,store,type="home furnishing")
location.of(Johnson&Hicks,TriBeCa)
begin.date(Johnson&Hicks,now)
```

and then we'd probably want to go on to tell the system that chairs are a kind of furniture, where TriBeCa is, that chairs that old are a kind of vintage product, etc. If we could do this, there are tons of methods for automated logical reasoning which we could use to find stories about displays of contemporary chairs, or vintage lamps, etc. The snag is that extracting this sort of meaning *from the text alone*, without using human readers, turns out to be insanely hard. People in artificial intelligence have been saying it'll happen within the next twenty years for more than fifty years; so we'll wish them good luck and leave them to their work.

A less ambitious sort of representation which still tries to get more or less explicitly at meanings would be to draw up a big list of categories, and then go over the documents and check of which categories they fall in to. This is basically what the annotating meta-data does for the *Times* corpus. Social scientists sometimes do this and call it **coding** the documents; they'll often code for the emotional tone or **valence** as well. (This tends to be done with smaller, more focused sets of documents, so they can get away with smaller sets of categories.) Once we have this, our representation is a bunch of discrete variables, and there are lots of methods for analyzing discrete variables. But,

again, extracting this sort of information *automatically* from the text is too hard for this to be useful to us.[2] In fact, the best automatic methods for this sort of thing use the bag-of-words representation which is coming up next.

1.1 Textual features

We don't know enough about how people extract meanings from texts to be able to automate it, but we do know that we extract meanings from texts — and different meanings from different texts.[3] This suggests that we should be able to use **features** or aspects of the text as proxies or imperfect signs of the actual meanings. And the text is right there in the computer already, so this should be easy. We just have to decide which textual features to use.

Think of what goes on in trying to distinguish between Saturn the brand of car, Saturn the planet, Saturn the mythological figure, Saturn the type of rocket, etc. Documents about all of these will of course contain the word "Saturn", so looking for that word wouldn't help us tell them apart. However, the *other* words in the document will tend to differ. "Saturn" together with words like "automobile", "wheels", "engine", "sedan" tends to indicate the car, whereas words like "rings", "hydrogen", "orbit", "Titan", "Voyager", "telescope", "Cassini", etc., indicate the planet. This suggests a *very* simple representation of the document: all the different words it contains.

That representation is actually a little *too* simple to work well in practice. The classic **bag-of-words** (BoW) representation is to list all of the distinct words in the document together with how often each one appears. The name comes from imagining printing out the text, cutting the paper into little pieces so each word is on its own piece, and then throwing all the pieces into a bag. This is a definitely set of purely textual features (one per distinct word), and it's not hard for us to calculate it automatically from the data. What remains to be seen is whether we can actually do useful work with this representation.

Vectors There are actually two different ways you could try to code up the bag-of-words, at least as I've described it so far. To illustrate this, let's look at the full text of the story featured in Figure 2 and see how to turn it into a bag of words.

> Lisa Weimer, right, opened her home furnishings store, Johnson & Hicks, in TriBeCa in September, and discovered her passion for chairs, especially those from the 1920's and 30's. "I love simple, clean lines and the richness of woods," said Ms. Weimer, who was once a home furnishings buyer at Bergdorf Goodman.
>
> A pair of French 1930's walnut chairs with checkerboard backs, above, are $8,500; steel folding chairs from the 1930's, originally

[2] However, we will see later that it is sometimes possible to trick people into doing this coding work for us for free.

[3] Conversely, different people extract different meanings from the *same* texts, raising another set of issues, which I will ignore.

X	1920s	1930s	a	above
9	1	3	3	1
and	are	at	backs	bergdorf
4	3	3	1	1
buyer	chairs	checkerboard	clean	discovered
1	4	1	1	1
especially	ferry	folding	for	franklin
1	1	1	2	1
french	from	furnishings	goodman	her
2	2	2	1	2
hicks	home	hudson	i	in
2	2	1	1	2
information	is	johnson	lines	lisa
1	1	2	1	1
love	ms	of	on	once
1	1	2	1	1
opened	originally	pair	passion	richness
1	1	1	1	1
right	said	september	simple	steel
2	1	1	1	1
store	street	tables	the	there
1	2	1	3	1
those	tribeca	used	walnut	was
1	1	1	1	1
weimer	who	with	woods	
2	1	1	1	

Table 1: Counts of the distinct words in the story, mapping all numbers to "X".

> used on a French ferry, are $575 each; tubular steel dining chairs upholstered in Ultrasuede, right, are 12 for $14,000. There are 500 chairs, and 100 tables. Johnson & Hicks is at 100 Hudson Street at Franklin Street. Information: (212) 966-4242.

Throwing away punctuation, and treating all numbers as just "X", we get the word-counts in Table 1

There are (at least) two different data structures we could use to store this information. One is a list of **key-value pairs**, also known as an **associative array**, a **dictionary** or a **hash**. The keys here would be the words, and the associated values would be the number of occurrences, or count, for each word. If a word does not appear in a document, that word is not one of its keys. Every document would have, in principle, its own set of keys. The order of the keys is entirely arbitrary; I printed them out alphabetically above, but it really doesn't matter.

It turns out, however, that it is a lot more useful to implement bags of words as **vectors**. Each component of the vector corresponds to a different word in the total **lexicon** of our document collection, in a fixed, standardized order. The value of the component would be the number of times the word appears, possibly including zero.

We use this vector bag-of-words representation of documents for two big reasons:

- There is a huge pre-existing technology for vectors: people have worked out, in excruciating detail, how to compare them, compose them, simplify them, etc. Why not exploit that, rather than coming up with stuff from scratch? (How would you measure the distance between two associate arrays?)
- In practice, it's proved to work pretty well.

To illustrate, I have taken ten random documents from the *Times* corpus — five of them about music, five about the arts (excluding music) — and turned them all into bag-of-words vectors.[4] There were 700 distinct words in these ten documents (excluding "singletons" which only appeared in a single text). This means that each document is represented by a vector with 700 components — that we have 700 features. Obviously I can't show you these vectors, but I will show a few of these components (Table 2).

Things to notice:

- The data take the form of a matrix. Each row corresponds to a distinct **case** (or instance **instance**, **unit**, **subject**, ...) — here, a document — and each column to a distinct feature. Conventionally, the number of cases is n and the number of features is p. It is no coincidence that this is the same format as the data matrix $\mathbf{X}$ in linear regression.
- Look for contrasts between the documents about music and those about art. Some of them ("camera", "hit", "melody") you could probably have guessed, if you'd thought about it — but what's going on with "husband" and "wife", or "imagined"?
- The word "a" seems to appear more in stories about art than in stories about music. Why might this be? Do we really want to pay attention to it?
- Does it really make sense to distinguish here between "photographs" and "photography"?

2 Measuring Similarity

Right now, we are interested in saying which documents are similar to each other because we want to do search by content. But measuring **similarity**

[4] You will get these documents, and more, in the first problem set.

	a	against	but	camera	gallery	hit	husband	images	imagined
music.1	13	0	3	0	0	0	0	0	0
music.2	18	0	7	0	0	2	0	0	0
music.3	33	0	2	0	3	1	0	0	0
music.4	28	0	11	0	0	1	0	0	0
music.5	10	0	0	0	1	0	0	0	0
art.1	20	0	3	2	0	0	1	0	0
art.2	51	0	9	1	4	0	0	2	1
art.3	55	1	6	11	1	0	2	8	0
art.4	64	2	7	0	0	0	0	0	2
art.5	11	1	1	0	0	0	0	2	0

	instruments	melody	new	old	photographs	photography	songs	wife
music.1	3	1	0	0	0	0	0	0
music.2	0	0	1	1	0	0	0	0
music.3	0	0	2	1	0	0	3	0
music.4	0	0	2	0	0	0	0	1
music.5	0	1	2	1	0	0	1	0
art.1	0	0	1	0	0	1	0	1
art.2	0	0	3	3	1	4	0	1
art.3	1	0	5	2	0	3	0	2
art.4	0	0	1	0	0	0	0	2
art.5	0	0	0	0	1	1	0	0

Table 2: Bag-of-words vectors for five randomly selected stories classified as "music", and five classified as "art" (but not music), from the *Times* corpus. The table shows a selection of the 700 features.

— or equivalently measuring **dissimilarity** or **distance** — is fundamental to data mining. Most of what we will do will rely on having a sensible way of saying how similar to each other different objects are, or how close they are in some geometric setting. Getting the right measure of closeness will have a huge impact on our results.

This is where representing the data as vectors comes in so handy. We already know a nice way of saying how far apart two vectors are, the ordinary or **Euclidean distance**, which we can calculate with the Pythagorean formula:

$$\|\vec{x} - \vec{y}\| \equiv \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

where x_i, y_i are the i^{th} components of $\vec{x}$ and $\vec{y}$. Remember that for bag-of-words vectors, each distinct word — each entry in the lexicon — is a component or feature.

(The **Euclidean length** or **Euclidean norm** of any vector is

$$\|\vec{x}\| \equiv \sqrt{\sum_{i=1}^{p} x_i^2}$$

so the distance between two vectors is the norm of their difference $\vec{x} - \vec{y}$. Equivalently, the norm of a vector is the distance from it to the origin, $\vec{0}$.)

Now, there are other ways of measuring distance between vectors. Another possibility is the **taxicab** or **Manhattan** distance

$$\sum_{i=1}^{p} |x_i - y_i|$$

It's a perfectly good distance metric; it just doesn't happen to work so well for our applications.

2.1 Normalization

Just looking at the Euclidean distances between document vectors doesn't work, at least if the documents are at all different in size. Instead, we need to **normalize** by document size, so that we can fairly compare short texts with long ones. There are (at least) two ways of doing this.

Document length normalization Divide the word counts by the total number of words in the document. In symbols,

$$\vec{x} \mapsto \frac{\vec{x}}{\sum_{i=1}^{p} x_i}$$

Notice that all the entries in the normalized vector are non-negative fractions, which sum to 1. The i^{th} component is thus the probability that if we pick a word out of the bag at random, it's the i^{th} entry in the lexicon.

Euclidean length normalization Divide the word counts by the Euclidean length of the document vector:

$$\vec{x} \mapsto \frac{\vec{x}}{\|\vec{x}\|}$$

For search, normalization by Euclidean length tends to work a bit better than normalization by word-count, apparently because the former de-emphasizes words which are rare in the document.

Cosine "distance" is actually a similarity measure, not a distance:

$$d_{\cos}\,\vec{x}, \vec{y} = \frac{\sum_i x_i y_i}{\|\vec{x}\|\|\vec{y}\|}$$

It's the cosine of the angle between the vectors $\vec{x}$ and $\vec{y}$.

3 Practice the Criterion of Truth

I've been pretty free with saying that things work or they don't work, without being at all concrete about what I mean by "working". We are going to see many, many different ways of elaborating on "working", but for right now, a first cut is to look at how often the most-similar document is in the wrong class. (This obviously relies on our having access to pre-assigned classes.)

	Best match by similarity measure		
	Euclidean	Euclidean + word-count	Euclidean + length
music.1	art.5	art.4	art.4
music.2	art.1	music.4	music.4
music.3	music.4	music.4	art.3
music.4	music.2	art.1	art.3
music.5	art.5	music.3	music.3
art.1	music.1	art.4	art.3
art.2	music.4	art.4	art.4
art.3	art.4	art.4	art.4
art.4	art.3	art.3	art.3
art.5	music.1	art.3	art.3
error count	6	2	3

Table 3: Closest matches for the ten documents, as measured by the distances between bag-of-words vectors, and the total error count (number of documents whose nearest neighbor is in the other class).

This shows that normalization definitely helps, but the difference here between normalizing by word-count and normalizing by Euclidean length is small, and if anything in the opposite direction from what I promised. That promise will, however, be fulfilled as we go forward, starting next time when we look at how to do search and classify documents.

Exercises

Do not turn these in, but do think them through.

1. Why is it called the "Manhattan metric" or "taxicab metric"? (Think before looking this up.)

2. Why do we need to normalize bag-of-word vectors to compare documents with different sizes?

3. Why does normalization by Euclidean length de-emphasize a document's rare words more strongly than normalization by word count? *Hint:* think about the relationship between $\sum_i |x_i|$ and $\|\vec{x}\|$.

4. Why is the cosine distance "the cosine of the angle between the two vectors"?

5. Explain how the cosine distance is related to the Euclidean-length normalized distance between two vectors.

References

Sandhaus, Evan (2008). "The New York Times Annotated Corpus." Electronic database. URL `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2008T19`.