Informatics 1

School of Informatics, University of Edinburgh

Data and Analysis

Part III

Unstructured Data

Alex Simpson

March 2012

Part III — Unstructured Data

Examples of Unstructured Data

- Plain text.

  There is structure, the sequence of characters, but this is *intrinsic* to the data, not imposed.
  We may wish to impose structure by, e.g., annotating (as in Part II).

- Bitmaps for graphics or pictures, digitized sound, digitized movies, etc.

  These again have *intrinsic* structure (e.g., picture dimensions).
  We may wish to impose structure by, e.g., recognising objects, isolating single instruments from music, etc.

- Experimental results.

  Here there may be structure in how represented (e.g., collection of points in n-dimensional space).
  But an important objective is to uncover implicit structure (e.g., confirm or refute an experimental hypothesis).

Topics

We consider two topics in dealing with unstructured data.

1. *Information retrieval*

   How to find data of interest in within a collection of unstructured data documents.

2. *Statistical analysis of data*

   How to use statistics to identify and extract properties from unstructured data (e.g., general trends, correlations between different components, etc.)

Information Retrieval

The *Information retrieval (IR) task*: given a query, find the documents in a given collection that are relevant to it.

Assumptions:

1. There is a large document collection being searched.
2. The user has a need for particular information, formulated in terms of a query (typically keywords).
3. The task is to find all and only the documents relevant to the query.

Example: Searching a library catalogue. Document collection to be searched: books and journals in library collection. Information needed: user specifies query giving details about author, title, subject or similar. Search program returns a list of (potentially) relevant matches.

Key issues for IR

Specification issues:

- Evaluation: How to measure the performance of an IR system.
- Query type: How to formulate queries to an IR system.
- Retrieval model: How to find the best-matching document, and how to *rank* them in order of relevance.

Implementation issues:

- Indexing: how to represent the documents searched by the system so that the search can be done efficiently.

The goal of this lecture is to look at the three *specification issues* in more detail.

Evaluation of IR

The performance of an IR system is naturally evaluated in terms of two measures:

- *Precision:* What proportion of the documents returned by the system match the original objectives of the search.
- *Recall:* What proportion of the documents matching the objectives of the search are returned by the system.

We call documents matching the objectives of the search *relevant documents*.

True/false positives/negatives

	Relevant	Non-relevant
Retrieved	true positives	false positives
Not retrieved	false negatives	true negatives

- *True positives (TP):* number of relevant documents that the system retrieved.
- *False positives (FP):* number of non-relevant documents that the system retrieved.
- *True negatives (TN):* number of non-relevant documents that the system did not retrieve.
- *False negatives (FN):* number of relevant documents that the system did not retrieve.

Defining precision and recall

	Relevant	Non-relevant
Retrieved	true positives	false positives
Not retrieved	false negatives	true negatives

Precision

$$\boldsymbol{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Recall

$$\boldsymbol{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Comparing 2 IR systems — example

Document collection with 130 documents.

28 documents relevant to some given query.

System 1: retrieves 25 documents, 16 of which are relevant

$$\mathrm{TP}_1 = 16,\quad \mathrm{FP}_1 = 25 - 16 = 9,\quad \mathrm{FN}_1 = 28 - 16 = 12$$

$$P_1 = \frac{\mathrm{TP}_1}{\mathrm{TP}_1 + \mathrm{FP}_1} = \frac{16}{25} = 0.64 \qquad R_1 = \frac{\mathrm{TP}_1}{\mathrm{TP}_1 + \mathrm{FN}_1} = \frac{16}{28} = 0.57$$

System 2: retrieves 15 documents, 12 of which are relevant

$$\mathrm{TP}_2 = 12,\quad \mathrm{FP}_2 = 15 - 12 = 3,\quad \mathrm{FN}_2 = 28 - 12 = 16$$

$$P_2 = \frac{\mathrm{TP}_2}{\mathrm{TP}_2 + \mathrm{FP}_2} = \frac{12}{15} = 0.80 \qquad R_2 = \frac{\mathrm{TP}_2}{\mathrm{TP}_2 + \mathrm{FN}_2} = \frac{12}{28} = 0.43$$

N.B. System 2 has higher precision. System 1 has higher recall.

Precision versus Recall

A system has to achieve both high precision and recall to perform well. It doesn't make sense to look at only one of the figures:

- Suppose a system returns every document in the collection: 100% recall, but low precision.
- Suppose a system returns just one document, but it is relevant: 100% precision, but possibly low recall.

Precision-recall tradeoff: Systems may be able to improve precision at the cost of recall, or increase recall at the cost of precision.

Whether precision or recall is more important depends on the application of the system.

F-score

The *F-score* is an evaluation measure that combines precision and recall.

$$F_\alpha = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

Here α is a *weighting factor* with $0 \leq \alpha \leq 1$.

High α means precision more important. Low α means recall is more important.

Often $\alpha = 0.5$ is used, giving the *harmonic mean* of P and R:

$$F_{0.5} = \frac{2PR}{P+R}$$

Using F-score to compare — example

We compare the examples on slide III: 10 using the F-score (with $\alpha = 0.5$).

$$F_{0.5}(\text{System}_1) = \frac{2P_1R_1}{P_1 + R_1} = \frac{2 \times 0.64 \times 0.57}{0.64 + 0.57} = 0.60$$

$$F_{0.5}(\text{System}_2) = \frac{2P_2R_2}{P_2 + R_2} = \frac{2 \times 0.80 \times 0.43}{0.80 + 0.43} = 0.56$$

The F-score (with this weighting) rates System 1 as better than System 2.

Query type

We shall only consider *simple queries* of the form:

- Find documents containing *word1*, *word2*, . . . , *wordn*

More specific tasks are:

- Find documents containing all the words *word1*, *word2* . . . *wordn*;
- or find documents containing as many of the words *word1*, *word2* . . . *wordn* as possible.

Going beyond these forms, queries can also be much more complex: they can be combined using boolean operations, look for whole phrases, substrings of words, look for matches of regular expressions, etc.

A retrieval model

If we look for all documents containing all words of the query — or all documents that contain some of the words of the query — then this may well result in a large number of documents, of widely varying relevance.

In this situation, it can help if IR systems can rank documents according to likely relevance.

There are many such ranking methods.

We focus on one, which uses the *vector space model*.

This model is the basis of many IR applications; it originated in the work of Gerard Salton and others in the 1970's, and is still actively developed.

In this course, we shall only use it in one particularly simple way.

The vector space model

Core ideas:

- Treat documents as points in a high-dimensional vector space, based on words in the document collection.
- The query is treated in the same way.
- The documents are ranked according to document-query similarity.

N.B. You do not need a detailed understanding of vector spaces to follow the working of the model.

The vector associated to a document

Suppose $\text{Term}_1, \text{Term}_2, \ldots, \text{Term}_n$ are all the different words occurring in the entire collection of documents $\text{Doc}_1, \text{Doc}_2, \ldots, \text{Doc}_K$.

Each document, Doc_i, is assigned an n-valued vector:

$$(m_{i1}, m_{i2}, \ldots, m_{in})$$

where m_{ij} is the number of times word Term_j occurs in document Doc_i.

Similarly, the query is assigned an n-valued vector by considering it as a document itself.

Example

Consider the document

Sun, sun, sun, here it comes

and suppose the only words in the document collection are: *comes*, *here*, *it*, *sun*.

The vector for the document is $(\mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{3})$

comes	here	it	sun
1	1	1	3

Similarly, the vector for the query *sun comes* is $(\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1})$

Document matrix

The frequency information for words in the document collection is normally precompiled in a *document matrix*.

This has:

- Columns represent the words appearing the document collection
- Rows represent each document in the collection.
- each entry in the matrix represents the frequency of the word in the document.

Document matrix — example

$$\begin{array}{c|ccccc|}
 & \text{Term}_1 & \text{Term}_2 & \text{Term}_3 & \ldots & \text{Term}_n \\
\hline
\text{Doc}_1 & 14 & 6 & 1 & \ldots & 0 \\
\text{Doc}_2 & 0 & 1 & 3 & \ldots & 1 \\
\text{Doc}_3 & 0 & 1 & 0 & \ldots & 2 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\text{Doc}_K & 4 & 7 & 0 & \ldots & 5 \\
\hline
\end{array}$$

N.B. Each row gives the vector for the associated document.

Vector similarity

We want to rank documents according to relevance to the query.

We implement this by defining a measure of *similarity* between vectors.

The idea is that the most relevant documents are those whose vectors are most similar to the query vector.

Many different similarity measures are used. A simple one that is conceptually appealing and enjoys some good properties is the *cosine* of the angle between two vectors.

Cosines (from school trigonometry)

Recall that the *cosine* of an angle θ is:

$$\frac{\text{adjacent}}{\text{hypotenuse}}$$

in a right-angled triangle with angle θ.

Crucial properties:

$$\cos(0) = 1 \qquad \cos(90^\circ) = 0 \qquad \cos(180^\circ) = -1$$

More generally, two n-dimensional vectors will have cosine: 1 if they are identical, 0 if they are orthogonal, and -1 if they point in opposite directions.

The value $\cos(x)$ *always* lies in the range from -1 to 1.

Vector cosines

Suppose $\vec{x}$ and $\vec{y}$ are n-value vectors:

$$\vec{x} = (x_1, \dots, x_n) \qquad \vec{y} = (y_1, \dots, y_n)$$

Their *cosine* (that is, the cosine of the angle between them) is calculated by:

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

Here $\vec{x} \cdot \vec{y}$ is the *scalar product* of vectors $\vec{x}$ and $\vec{y}$, while $|\vec{x}|$ is the length (or *norm*) of the vector $\vec{x}$.

Vector cosines — example

Continuing the example from slide 11.18, suppose:

$$\vec{x} = (1, 1, 1, 3) \qquad \vec{y} = (1, 0, 0, 1)$$

Then:

$$\begin{aligned} \vec{x} \cdot \vec{y} &= 1 + 0 + 0 + 3 = 4 \\ |\vec{x}| &= \sqrt{1 + 1 + 1 + 9} = \sqrt{12} \\ |\vec{y}| &= \sqrt{1 + 0 + 0 + 1} = \sqrt{2} \end{aligned}$$

So

$$\cos(\vec{x}, \vec{y}) = \frac{4}{\sqrt{12} \times \sqrt{2}} = \frac{2}{\sqrt{6}} = 0.82$$

to two significant figures.

Ranking documents

Suppose $\vec{y}$ is the query vector, and $\vec{x_1}, \ldots, \vec{x_K}$ are the K document vectors.

We calculate the K values:

$$\cos(\vec{x_1}, \vec{y}), \quad \ldots \quad , \cos(\vec{x_K}, \vec{y})$$

Sorting these, the documents with the highest cosine values when compared to the query $\vec{y}$ are the best match, and those with the lowest cosine values are counted as least suitable.

N.B. On this slide $\vec{x_1}, \ldots, \vec{x_K}$ are K (potentially) different vectors, each with n values.

Discussion of cosine measure

The cosine similarity measure, as discussed here, is very crude.

- It only takes word frequency into account, not position or ordering
- It takes all words in the document collection into account (whether very common "stop" words which are useless for IR, or very uncommon words unrelated to the search)
- All words in the document collection are weighted equally
- It ignores document size (just the angles between vectors not their magnitude are considered)

Nevertheless, the cosine method can be refined in various ways to avoid these problems. (This is beyond the scope of this course.)

Other issues

- Precision and recall, as defined, only evaluate the set of documents returned, they do not take *ranking* into account. Other more complex evaluation measures can be introduced to deal with ranking (e.g., *precision at a cutoff*).
- We have not considered the efficient implementation of the search for documents matching a query. This is often addressed using a purpose-built index such as an *inverted index* which indexes all documents using the words in the document collection as keys.
- Often useful ranking methods make use of information extraneous to the document itself. E.g., Google's *pagerank* method evaluates documents according to their degree of *connectivity* with the rest of the web (e.g., number of links to page from other pages).

These are important issues, but are beyond the scope of this course.