Advanced ML

– Computing similarities in high dimensional spaces –

Eric Gaussier

Univ. Grenoble Alpes

UFR-IM^{2}AG, LIG, MIAI@Grenoble Alpes

eric.gaussier@imag.fr

Table of content

A simple and general representation: vectors

Most data are represented as real-valued (sometimes binary, presence/absence information) vectors (temperature, rotation speed, consumption, ...)

4-dimensional vector $\begin{pmatrix} -0,76 \\ 5,48 \\ -28997 \\ 3076,23 \end{pmatrix}$ (binary, pres./abs.) $\begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$

Document (texts or images) indexing is an operation that produces vector space representations

Indexing steps

1. Segmentation
    - Segment a text into words:

      the importance of retrieving the good information
      the, importance, of, retrieving, the, good, information

      7 words but only 6 word types; depending on languages, may require a dictionary
2. Stop-word removal (stop-word list)
3. Normalization
    - Upper/lower-case, inflected forms, lexical families
    - Lemmatization, stemming

$\rightarrow$ Bag-of-words: *importance, good, retriev, inform*

Vector space representation

- The set of all word types constitute the vocabulary of a collection. Let p be the size of the vocabulary and N be the number of documents in the collection $\rightarrow$ p-dimensional vector space (each axis corresponds to a word type)
- Each document **x** is represented by a vector the coordinates of which correspond to:
  - Presence/absence or number of occurrences of the word type in the doc: $x_i = \mathrm{tf}_i^x$ (number of times i occurs in **x**)
  - Normalized number of occurrences: $x_i = \frac{\mathrm{tf}_i^x}{\sum_{i=1}^M \mathrm{tf}_i^x}$
  - *tf*idf*:
    $$x_i = \frac{\mathrm{tf}_i^x}{\sum_{i=1}^M \mathrm{tf}_i^x} \underbrace{\log \frac{N}{\mathrm{df}_i}}_{\mathrm{idf}_i}$$
    where df_i is the number of docs in which word (type) i occurs (idf: inverse document frequency)

Similarity and distance between vectors (1)

The vector space representation gives access to the mathematical material associated to vector spaces: distances, similarity measures, dimensionality reduction, ...

Notation: $(\mathbf{x}, \mathbf{x}') \in \mathbb{R}^p,\ \mathbf{x} = \begin{pmatrix} x_1 \\ \cdots \\ x_p \end{pmatrix},\ \mathbf{x}' = \begin{pmatrix} x'_1 \\ \cdots \\ x'_p \end{pmatrix}$

Dot product and cosine

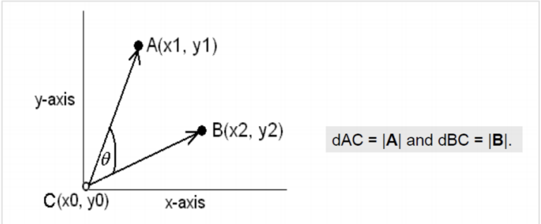
$$\mathbf{x} \cdot \mathbf{x}' = \sum_{i=1}^{p} x_i\, x'_i, \quad \cos(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x} \cdot \mathbf{x}'}{||\mathbf{x}||_2\ ||\mathbf{x}'||_2} = \frac{\mathbf{x}}{||\mathbf{x}||_2} \cdot \frac{\mathbf{x}'}{||\mathbf{x}'||_2}$$

with the 2-norm $||\mathbf{x}||_2 = \sqrt{\mathbf{x} \cdot \mathbf{x}}$

Similarity and distance between vectors (2)

Illustration

The cosine ($\in [-1; 1]$) corresponds to the cosine of the angle between two vectors. It is maximal when the two vectors are colinear



dAC = |**A**| and dBC = |**B**|.

$$\mathrm{Sim}(A, B) = \text{cosine } \theta = \frac{A \bullet B}{|A||B|} = \frac{x1 * x2 + y1 * y2}{(x1^2 + y1^2)^{1/2} \; (x2^2 + y2^2)^{1/2}}$$

Similarity and distance between vectors (3)

$$\mathbf{x} = \begin{pmatrix} 0.7 \\ -3.1 \\ 0 \\ 73 \end{pmatrix}, \; \mathbf{x}' = \begin{pmatrix} 5 \\ -7 \\ 18 \\ 0 \end{pmatrix}$$

- $\mathbf{x} \cdot \mathbf{x}' = 0.7 \times 5 + (-3.1) \times (-7) + 0 \times 18 + 73 \times 0 = 25.2$
- $||\mathbf{x}||_2 = \sqrt{0.7^2 + (-3.1)^2 + 0 + 73^2} \approx 73.07$
- $\cos(\mathbf{x}, \mathbf{x}') \approx 0.17$

Similarity and distance between vectors (4)

$$\text{Pres./abs. : } \mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \ \mathbf{x}' = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

- $\mathbf{x} \cdot \mathbf{x}' = 1 \times 1 + 0 \times 1 + 1 \times 1 + 1 \times 0 = 2$
  Number of elements in common
- $||\mathbf{x}||_2 = ||\mathbf{x}'||_2 = \sqrt{3}$
- $\cos(\mathbf{x}, \mathbf{x}') = \frac{2}{3}$

Similarity and distance between vectors (4)

Pres./abs. : $\mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \; \mathbf{x}' = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$

- $\mathbf{x} \cdot \mathbf{x}' = 1 \times 1 + 0 \times 1 + 1 \times 1 + 1 \times 0 = 2$
  Number of elements in common
- $||\mathbf{x}||_2 = ||\mathbf{x}'||_2 = \sqrt{3}$
- $\cos(\mathbf{x}, \mathbf{x}') = \frac{2}{3}$

Similarity and distance between vectors (4)

Pres./abs. : $\mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \ \mathbf{x}' = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$

- $\mathbf{x} \cdot \mathbf{x}' = 1 \times 1 + 0 \times 1 + 1 \times 1 + 1 \times 0 = 2$
  Number of elements in common
- $||\mathbf{x}||_2 = ||\mathbf{x}'||_2 = \sqrt{3}$
- $\cos(\mathbf{x}, \mathbf{x}') = \frac{2}{3}$

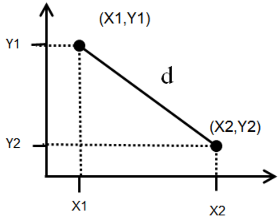
Similarity and distance between vectors (5)

Euclidean distance between two vectors

$$d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2 = \sqrt{\sum_{i=1}^{p} (x_i - x'_i)^2}$$

On the previous exemple:

$$||\mathbf{x} - \mathbf{x}'||_2 = \sqrt{(1-1)^2 + (0-1)^2 + (1-1)^2 + (1-0)^2} = \sqrt{2}$$



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Interlude

1. IR: queries and documents are both represented as vectors; documents are ranked according to their similarity (typically dot product) wrt a query
2. Classification (supervised learning): using a set of already classified objects (training set), assign a new object to a predefined category
   - Find k nearest neighbors in set of already classified objects
   - Select the most represented category (k-Nearest Neighbors, k-NN)
3. Clustering (unsupervised learning): k-Means for example

Table of content

Computing a dot product (simple case)

- Each vector can be represented by a p dimensional array containing the coeffcients/coordinates/weights of the vector; it is straightforward to write an algorithm that computes the dot product between two such arrays $\mathbf{x}$ and $\mathbf{x}'$.

```
scal = 0.0
for $i$=1 to $p$ do
    scal $+=$ $x_i \, x'_i$
end
```

- Assuming we have N objects, what is the complexity of computing the dot product between one object and all the others?

$\rightarrow \mathcal{O}(Np)$

Computing a dot product (simple case)

- Each vector can be represented by a p dimensional array containing the coeffcients/coordinates/weights of the vector; it is straightforward to write an algorithm that computes the dot product between two such arrays $\mathbf{x}$ and $\mathbf{x}'$.

```
scal = 0.0
for i=1 to p do
    scal += $x_i \, x'_i$
end
```

- Assuming we have N objects, what is the complexity of computing the dot product between one object and all the others?

$\rightarrow \mathcal{O}(Np)$

Computing a dot product (simple case)

- Each vector can be represented by a *p* dimensional array containing the coeffcients/coordinates/weights of the vector; it is straightforward to write an algorithm that computes the dot product between two such arrays **x** and $\mathbf{x}'$.

```
scal = 0.0
for i=1 to p do
    scal $+=$ $x_i\, x'_i$
end
```

- Assuming we have *N* objects, what is the complexity of computing the dot product between one object and all the others?

$\rightarrow \mathcal{O}(Np)$

Computing a dot product (simple case)

- Each vector can be represented by a *p* dimensional array containing the coeffcients/coordinates/weights of the vector; it is straightforward to write an algorithm that computes the dot product between two such arrays $\mathbf{x}$ and $\mathbf{x}'$.

```
scal = 0.0
for i=1 to p do
    $\text{scal} += x_i \, x'_i$
end
```

- Assuming we have N objects, what is the complexity of computing the dot product between one object and all the others?

$\rightarrow \mathcal{O}(Np)$

Computing dot product in high-dimensional spaces (1)

- The above computation is too slow when p is large, say above 10^4 which is typical of large textual collections.
- Such collections are sparse however, meaning that a document contains very few words.
  - $p \approx 10^4$ is the number of words in the collection; a document $\mathbf{x} \in \mathbb{R}^p$
  - Document $\mathbf{x} \in \mathbb{R}^p$ contains around 10^2 different words
  - Most dimensions of $\mathbf{x}$ are null!
- Need for sparse representations

Computing dot product in high-dimensional spaces (2)

Example of a sparse representation

$$\text{document } \mathbf{x} \begin{cases} \text{int l} & \text{(\# of words in doc.)} \\ \text{ArrTerms int[l]} & \text{(indices of words in doc.,} \\ & \quad \text{in increasing order)} \\ \text{ArrWeigh float[l]} & \text{(weights of words)} \\ \ldots & \end{cases}$$

Computing dot product in high-dimensional spaces (3)

Example

- Vocabulary: {language, programming, C, python, java, objects, compilation} (indexed from 1 to 7).
- Document **x** contains *programming*, which occurs 7 times, and *C*, which occurs 3 times.
- Weight: normalized number of occurrences

$$\mathbf{x} = \begin{cases} l = 2 \\ ArrTerms[0] = 2, \; ArrTerms[1] = 3 \\ ArrWeigh[0] = 0.7, \; ArrWeigh[1] = 0.3 \\ \ldots \end{cases}$$

Computing dot product in high-dimensional spaces (4)

Algorithm - dot product between $\mathbf{x}$ and $\mathbf{x}'$

```
scal $= 0.0$, $i = 0$, $j = 0$
while (i < (x.l-1) && j < (x'.l-1)) do
    if x.ArrTerms[i] < x'.ArrTerms[j] then
        i++
    else if x.ArrTerms[i] > x'.ArrTerms[j] then
        j++
    else
        scal $+= x.ArrWeigh[i]\, x'.ArrWeigh[j]$
        i++, j++
    end
end
```

Complexity: $\mathcal{O}(x.l + x'.l) = \mathcal{O}(l_m)$ where l_m average size of a document (roughly 10^2 in large textual collections)

Computing dot product in high-dimensional spaces (5)

Illustration

x.ArrTerms:	2	7	11	237
x.ArrWeigh:	0.1	2.3	0.765	1.09

x'.ArrTerms:	1	11	789
x'.ArrWeigh:	8.34	0.17	1.23

Inverted file (1)

It is also possible to accelerate the comparison between one document (query) and all documents of the collections as in many cases a given word only occurs in few documents (another sparsity property). The trick is to use an *inverted file* which provides, for each word, the list of documents (again sorted in increasing order of the index of the doc) containing the word.

$$\text{mot } i \left\{ \begin{array}{ll} \text{int l} & \text{(\# of docs)} \\ \text{TabDocs int[l]} & \text{(indices of docs, increasing order)} \\ \ldots & \end{array} \right.$$

Remark Useful for measures for which the contribution of terms not present in a document is null (dot product and cosine but not Euclidean distance)

Inverted file (2)

Algorithm - dot product between **x** and all other documents

1. For each word in **x**, retrieve all docs in $x.TabDocs[]$; put them in list L
2. Compute dot product between **x** and all docs in L (and not all docs of the collection!)

Complexity: Let L_m denote the average number of docs in which a word occurs

$$\Rightarrow \mathcal{O}(l_m L_m l_m) = \mathcal{O}(L_m l_m^2)$$

As $L_m \approx 10^3$ or 10^4, gain of 10^7 or 10^6 compared to $\mathcal{O}(pN)$ with $N \approx 10^9$ and $p \approx 10^5$.

Building the inverted file

In a static collection, 3 main steps:

1. Extraction of pairs of indices *(word, doc)*; single pass over collection
2. Sorting above pairs according to word id, then doc id
3. Merging pairs to get, for each word, the list of docs containing it

These steps raise no problem when the collections are sufficiently small to fit in memory

What about big collections?

When the memory is not sufficient - The BSBI algorithm (1)

One stores intermediate files on disk, prior to merge them in a single inverted file

1. Extraction of pairs of word and doc ids and storage on disk (global file)
2. The global file is read block by block, each block fitting in memory; for each block, construct an inverted file and store it on disk (partial inverted file)
3. Merging all partial inverted files to create a global inverted file

Associated algorithm: *Blocked sort-based indexing* (BSBI)

When the memory is not sufficient - The BSBI algorithm (2)

The inversion in BSBI consists in sorting (word,doc) od pairs according to word ids then doc ids (complexity typically in $\mathcal{O}(T \log T)$ where T is the number of pairs). The merging step uses the same trick as the one used to compute the dot product:

Illustration (1)

Toy example with 5 words ($w_1, ..., w_5$) and 8 docs ($d_1, ..., d_8$).
Only 3 pairs (word id, doc id) can fit in memory.

Global file:

$w_1 : d_1$	$w_2 : d_4$	$w_2 : d_1$
$w_3 : d_8$	$w_1 : d_3$	$w_4 : d_7$
$w_5 : d_5$	$w_2 : d_2$	$w_1 : d_6$

Illustration (2)

Reading global file block by block and sorting to partial inverted files.

$$\begin{array}{|c|c|c|}\hline w_1 : d_1 & w_2 : d_4 & w_2 : d_1 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|}\hline w_1 : d_1 & w_2 : d_1 & w_2 : d_4 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|}\hline w_3 : d_8 & w_1 : d_3 & w_4 : d_7 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|}\hline w_1 : d_3 & w_3 : d_8 & w_4 : d_7 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|}\hline w_5 : d_5 & w_2 : d_2 & w_1 : d_6 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|}\hline w_1 : d_6 & w_2 : d_2 & w_5 : d_5 \\ \hline \end{array}$$

Illustration (3)

Merging partial inverted files.

$w_1 : d_1$	$w_2 : d_1$	$w_2 : d_4$

$w_1 : d_3$	$w_3 : d_8$	$w_4 : d_7$

$w_1 : d_6$	$w_2 : d_2$	$w_5 : d_5$

$\Rightarrow$ Write $(w_1 : d_1)$ to global inverted file. The partial inverted files become:

$w_2 : d_1$	$w_2 : d_4$

$w_1 : d_3$	$w_3 : d_8$	$w_4 : d_7$

...

$w_1 : d_6$	$w_2 : d_2$	$w_5 : d_5$

Table of content

Conclusion

- Simple but efficient algorithms (mandatory to deal with large sparse collections)
- Straightforward to parallelize (additional gain)
- Efficient algorithms also exist for computing approximations of similarity value (MinHash [1] for detecting duplicate web pages)

[0] C. Manning, P. Raghavan, H. Schütze, ”Introduction to Information Retrieval”, 2008 (https://nlp.stanford.edu/IR-book/information-retrieval-book.html)

[1] A. Roder, "On the resemblance and containment of documents", In Proc. of Compression and Complexity of Sequences, 1997