Intelligent Data Analysis
Lecture Notes on
Document Mining

Peter Tiňo

1 Representing Textual Documents as Vectors

Our next topic will take us to seemingly very different data spaces - those of textual documents. We will outline how to perform analysis in such spaces. Our key step will be representation of documents as (potentially very high-dimensional) vectors. This will be done through identifying co-ordinates with *terms*. Each term is a specific word or phrase, such as "equation", "solution", "sun" etc. We then represent a document as a vector by specifying the value ('strength') of each co-ordinate (term) for that document.

Let $\mathcal{D} = \{d^1, d^2, \ldots, d^N\}$ be our document set (library) and $\mathcal{T} = \{t_1, t_2, \ldots, t_T\}$ be the set of terms. In a realistic scenario, $|\mathcal{T}| = T$ can be a large number and we will need to pick the relevant terms that best represent the variety of documents in our dataset.

We want to transform the i-th document $d^i i$ into a vector:

$$d^i \mapsto \mathbf{x}^i = (x_1^i, x_2^i, \ldots, x_T^i)^T \in \mathbb{R}^T$$

Each $x_k^i \geq 0$ is a 'weight' corresponding to the term t_k. It quantifies how 'important' the term is in the document d_i. How should we calculate this weight? We propose a couple of ideas:

i) x_k^i is binary - is the term in the document or not? This is a bit simple, but can be efficient to calculate.

ii) x_k^i represents the frequency of the term t_k in the document d^i, denoted N_k^i. This is better, but shorter and longer documents will have values of x_k^i that are not directly comparable.

iii) $x_k^i = \frac{N_k^i}{|d^i|}$ is the relative frequency of the term, where $|d^i|$ denotes the length of the document. The undesirable influence of the document length on x_k^i is now reduced.

Finally we realize that, of course, if a term is frequent in a given document, than it is important for it. But what if that term is also frequent in almost all other documents in our

dataset (library)? Intuitively, in such a case the term would not characterize the document very well - it will not make the document "stand out" among the other documents in the library. Hence, given a document, we would like to assign a term high weight if it is *both* frequent in the document *and* it makes the document unique - e.g. it occurs sparsely across other documents. We have the following observations:

1) The more frequent term t_k is in document d^i, the more it is representative of its content;

2) The more documents t_k occurs in, the 'less discriminating' it is.

We can quantify 1) by using the *relative frequency* (probability) of the term in the document,

$$\frac{N_k^i}{|d^i|} = Pr(t_k \mid d^i) = P(t_k). \tag{1}$$

For 2), we need to know the proportion of documents in the document set (library) $\mathcal{D}$ that contain t_k. This can be computed by

$$\frac{N_k}{N} = Pr(t_k \mid \mathcal{D}) = Q(t_k), \tag{2}$$

where N_k is the number of documents that contain t_k and $N = |\mathcal{D}|$ is the library size. It is important to note that both probability distributions P and Q are defined over the same (term) set $\mathcal{T}$. We say that they share the same support. Combining (1) and (2), we can find an appropriate way to compute x_k^i:

$$x_k^i = P(t_k) \cdot \log_2 \frac{1}{Q(t_k)} = \frac{N_k^i}{|d^i|} \cdot \log_2 \frac{N}{N_k}. \tag{3}$$

This method is known as *term frequency inverse document frequency*, or TFIDF for short. Indeed, for term t_k to have high importance in document d^i, it needs to be frequent in d^i (high term frequency $P(t_k)$) *and* infrequent in the library (low $Q(t_k)$, high inverse document frequency). Intuitively, the log function squashes high values of $1/Q(t_k)$, as it really does not matter, provided the term is quite unique in the library, whether $Q(t_k)$ is as small as 0.001, or 0.0001... We will now justify the TFIDF expression in the framework of Information Theory.

2 Information theoretic view of TFIDF

Suppose Alice has a 4-sided *fair* dice and wants to relay what value was landed on the dice to Bob. She cannot do this directly as Bob is overseas, but she can speak to him through some communication channel that accepts binary messages. We can use the following coding scheme to represent each singular event when we transmit it through the channel:

Landing a One $-$ 00
Landing a Two $-$ 01
Landing a Three $-$ 10
Landing a Four $-$ 11

Note that in this case there seems to be a relationship between the number of bits needed and the probability $\frac{1}{4}$ of the possible events that can occur (under the assumption of uniform distribution (fair dice). In particular,

$$\log_2 \frac{1}{\frac{1}{4}} = \log_2 4 = 2 \text{ bits.}$$

Using similar reasoning, if Alice used an 8-sided fair dice, then she would need 3 bits of information to describe all eight possible events. Again,

$$\log_2 \frac{1}{\frac{1}{8}} = \log_2 8 = 3 \text{ bits.}$$

In the general case (not uniform distribution), consider a random variable X and the associated probability $P(x)$ of an event x. If $P(x)$ is high and x actually happens, we are not surprised to see x happening and hence do not learn much from realizing that x was observed. On the other hand, if $P(x)$ is small and x happens, we are surprised and consider it a useful bit of information. Hence, the information (or the amount of statistical surprise) in the event x is inversely proportional to its probability of occurrence, $\frac{1}{P(x)}$. It can be shown that, like in the examples above, the optimal message length (in bits) to use in order to convey the information that x happened is

$$\log_2 \frac{1}{P(x)} \text{ bits.} \tag{4}$$

We now draw many realizations x from the random variable X and each time pass the information about the observed event x through a communication channel using $\log_2 \frac{1}{P(x)}$ bits. What is the average number of bits per realization when coding all of the events? This is just the expected value of (4):

Definition 2.1. For a random variable X distributed with $P(x)$, let A be the event set. The *entropy* of X is defined as

$$H(X) = \mathbb{E}_P \left[\log_2 \frac{1}{P(x)} \right] = \sum_{x \in A} P(x) \cdot \log_2 \frac{1}{P(x)}.$$

Now consider the following scenario: Suppose that for a random variable X, distributed with $P(x)$, when designing a code for the events $x \in A$, we wrongly assume X is distributed according to a probability distribution $Q(x)$, $Q \neq P$. Hence each event $x \in A$ will be coded with a message of length

$$\log_2 \frac{1}{Q(x)} \text{ bits,} \tag{5}$$

instead of the optimal one $\log_2 \frac{1}{P(x)}$. But the events are actually generated with probability P, meaning that our average code length is now the expected value of $\log_2 \frac{1}{Q(x)}$ with respect to the distribution P:

Definition 2.2. [1] Let P and Q be two probability distributions over the same event set A. The *cross-entropy* from P to Q is defined as

$$H^c(P,Q) = \mathbb{E}_P\left[\log_2 \frac{1}{Q(x)}\right] = \sum_{x\in A} P(x)\cdot \log_2 \frac{1}{Q(x)}. \tag{6}$$

Of course, since we built the code using a wrong distribution Q (instead of P), the average code length will be higher:

$$H^c(P,Q) \geq H(P),$$

with equality, if (and only if) $P = Q$. The value of $H^c(P,Q)$ thus reflects (quantifies) how different are the two distributions P and Q. This difference is sometimes expressed as a normalized quantity, i.e. the number of *additional* bits we need to use as a consequence of mis-specifying Q for P: $H^c(P,Q) - H(P)$. This quantity is known as K-L divergence from P to Q:

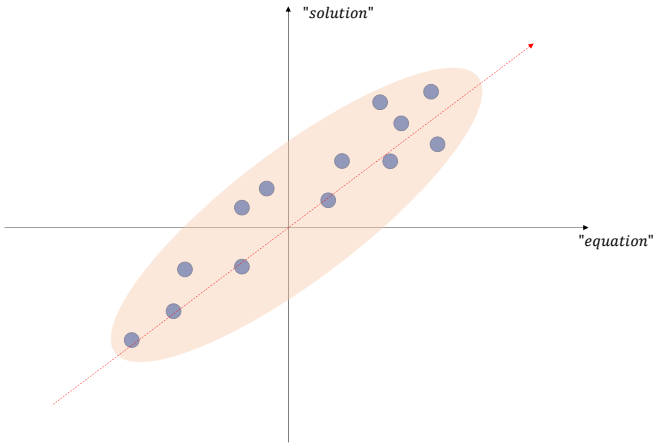
$$\begin{aligned} D_{KL}[P||Q] &= \mathbb{E}_P\left[\log_2 \frac{1}{Q(x)}\right] - \mathbb{E}_P\left[\log_2 \frac{1}{P(x)}\right] \\ &= \sum_{x\in A} P(x)\cdot \log_2\left(\frac{1}{Q(x)}\right) - \sum_{x\in A} P(x)\cdot \log_2\left(\frac{1}{P(x)}\right) \\ &= \sum_{x\in A} P(x)\cdot \left[\log_2\left(\frac{1}{Q(x)}\right) + \log_2 P(x)\right] \\ &= \sum_{x\in A} P(x)\cdot \log_2 \frac{P(x)}{Q(x)}. \end{aligned}$$

Using (1), (2) and comparing the TFIDF expression (3) with (6), we see that TFIDF weights can be viewed as contributions to the cross-entropy from the term distribution in the particular document to the term distribution across the whole library.

3 'Concepts' and Co-occurring Terms

In the previous sections we discussed how to transform a document set $\mathcal{D} = \{d^1, d^2, \ldots, d^N\}$ into vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N \in \mathbb{R}^T$ according to some keywords or terms that characterise the documents. This leaves us with the problem of how to deal with thousand terms (typically, $T \approx 1000$) and a library containing a huge number of documents. This is where we apply what we learnt in principal component analysis. To demonstrate why, suppose that we want to search for terms "solution" and "equation" in our library, calculating the weights for each document and plotting the data (vector representations of documents) as shown:

[1] see `https://en.wikipedia.org/wiki/Cross_entropy` for details



It is obvious that we see some sort of structure in the document set. In particular, the terms "solution" and "equation" have high *co-occurrence*, implying some sort of 'semantics' behind the documents, represented by the red dashed line. For example, a detected high co-occurrence of the terms "solution" and "equation" in the documents suggests the existence of an abstract semantic topic (*concept*) in the documents that can be characterised as *"solving mathematical equations"*. Analogously, high co-occurrence of terms "cloud", "rain", "sun" etc. points to an abstract concept of "*weather*" discussed in the documents. So, by applying PCA we are able to reduce our term set into the 'semantic' or 'concept' space spanned by the dominant eigenvectors of the data covariance matrix. Let c be the number of 'concepts' (dominant eigenvectors) we find within the data, where usually $c << T$ suffice to preserve substantial data variation.

$$\mathcal{D} = \{d^1, d^2, \ldots, d^N\} \xrightarrow{\text{TFIDF}} \mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N \in \mathbb{R}^T \xrightarrow{\text{PCA}} \tilde{\mathbf{x}}^1, \tilde{\mathbf{x}}^2, \ldots, \tilde{\mathbf{x}}^N \in \mathbb{R}^c$$

4 Querying in Latent Semantic Analysis (LSA)

Latent semantic analysis projects document vectors into the concept space, as discussed above. Typing few keywords, we form a mini query document that is then projected onto the concept space. Then we ask how 'similar' it is compared to the other documents expressed in the concept space. The degree of similarity can be quantified e.g. through cos of the angle between the query document and a document in the library (both, of course, projected to the c-dimensional concept space).

1) For our document set $\mathcal{D} = \{d^1, d^2, \ldots, d^N\}$, transform each document into a vector representation using TFIDF.

2) Using the co-variance matrix $\mathbf{C}$ of these vectors, find the eigenvectors and their corresponding eigenvalues:

$$\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_T \in \mathbb{R}^T$$
$$\lambda_1, \lambda_2, \ldots, \lambda_T \in \mathbb{R}_{\geq 0}.$$

3) Pick the $c < T$ most significant vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_c$ such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_c$. These direction vectors will represent the 'concepts' within our document set.

4) If we let $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_c]$ be our concept matrix, project the documents onto the concept space using the expression

$$\tilde{\mathbf{x}}^j = \mathbf{U}^T \mathbf{x}^j.$$

5) Project the query document represented as a vector $\mathbf{d}_Q \in \mathbb{R}^T$ onto the concept space:

$$\tilde{\mathbf{d}}_Q = \mathbf{U}^T \mathbf{d}_Q.$$

6) To search for documents similar to $\tilde{\mathbf{d}}_Q \in \mathbb{R}^c$ in our reduced space, we find the angle between a document $\tilde{\mathbf{x}}^i$ and the query $\tilde{\mathbf{d}}_Q$:

$$\cos \alpha = \frac{\left(\tilde{\mathbf{d}}_Q\right)^T \tilde{\mathbf{x}}^i}{||\tilde{\mathbf{d}}_Q|| \cdot ||\tilde{\mathbf{x}}^i||}$$

7) Return documents that have $\cos \alpha$ greater than some pre-defined threshold.