Chapter 13

Automatic Key Word and Key Sentence Extraction

Due to the explosion of the amount of textual information available, there is a need to develop automatic procedures for text summarization. A typical situation is when a Web search engine presents a small amount of text from each document that matches a certain query. Another relevant area is the summarization of news articles.

Automatic text summarization is an active research field with connections to several other areas, such as information retrieval, natural language processing, and machine learning. Informally, the goal of text summarization is to *extract content from a text document and present the most important content to the user in a condensed form and in a manner sensitive to the user's or application's need* [67]. In this chapter we will have a considerably less ambitious goal: we will present a method for automatically extracting key words and key sentences from a text. There will be connections to the vector space model in information retrieval and to the concept of pagerank. We will also use nonnegative matrix factorization. The presentation is based on [114]. Text summarization using QR decomposition is described in [26, 83].

13.1 Saliency Score

Consider a text from which we want to extract key words and key sentences. As an example we will take Chapter 12 from this book. As one of the preprocessing steps, one should perform stemming so that the same word stem with different endings is represented by one token only. Stop words (cf. Chapter 11) occur frequently in texts, but since they do not distinguish between different sentences, they should be removed. Similarly, if the text carries special symbols, e.g., mathematics or mark-up language tags (HTML, LaTeX), it may be necessary to remove those.

Since we want to compare word frequencies in different sentences, we must consider each sentence as a separate document (in the terminology of information retrieval). After the preprocessing has been done, we parse the text, using the same type of parser as in information retrieval. This way a term-document matrix is

prepared, which in this chapter we will refer to as a *term-sentence* matrix. Thus we have a matrix $A \in \mathbb{R}^{m\times n}$, where m denotes the number of different terms and n the number of sentences. The element a_{ij} is defined as the frequency[32] of term i in sentence j.

The column vector $(a_{1j}\, a_{2j}\, \ldots\, a_{mj})^T$ is nonzero in the positions corresponding to the terms occurring in sentence j. Similarly, the row vector $(a_{i1}\, a_{i2}\, \ldots\, a_{in})$ is nonzero in the positions corresponding to sentences containing term i.

The basis of the procedure in [114] is the simultaneous but separate *ranking* of the terms and the sentences. Thus, term i is given a nonnegative *saliency score*, denoted u_i. The higher the saliency score, the more important the term. The saliency score of sentence j is denoted v_j.

The assignment of saliency scores is made based on the *mutual reinforcement principle* [114]:

> A term should have a high saliency score if it appears in many sentences with high saliency scores. A sentence should have a high saliency score if it contains many words with high saliency scores.

More precisely, we assert that the saliency score of term i is proportional to the sum of the scores of the sentences where it appears; in addition, each term is weighted by the corresponding matrix element,

$$u_i \propto \sum_{j=1}^{n} a_{ij} v_j, \qquad i = 1, 2, \ldots, m.$$

Similarly, the saliency score of sentence j is defined to be proportional to the scores of its words, weighted by the corresponding a_{ij},

$$v_j = \propto \sum_{i=1}^{m} a_{ij} u_i, \qquad j = 1, 2, \ldots, n.$$

Collecting the saliency scores in two vectors $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$, these two equations can be written as

$$\sigma_u u = Av, \tag{13.1}$$
$$\sigma_v v = A^T u, \tag{13.2}$$

where σ_u and σ_v are proportionality constants. In fact, the constants must be equal. Inserting one equation into the other, we get

$$\sigma_u\, u = \frac{1}{\sigma_v} AA^T u,$$
$$\sigma_v\, v = \frac{1}{\sigma_u} A^T A v,$$

[32] Naturally, a term and document weighting scheme (see [12]) should be used.

which shows that u and v are eigenvectors of AA^T and A^TA, respectively, with the same eigenvalue. It follows that u and v are singular vectors corresponding to the same singular value.[33]

If we choose the largest singular value, then we are guaranteed that the components of u and v are nonnegative.[34]

In summary, the saliency scores of the terms are defined as the components of u_1, and the saliency scores of the sentences are the components of v_1.

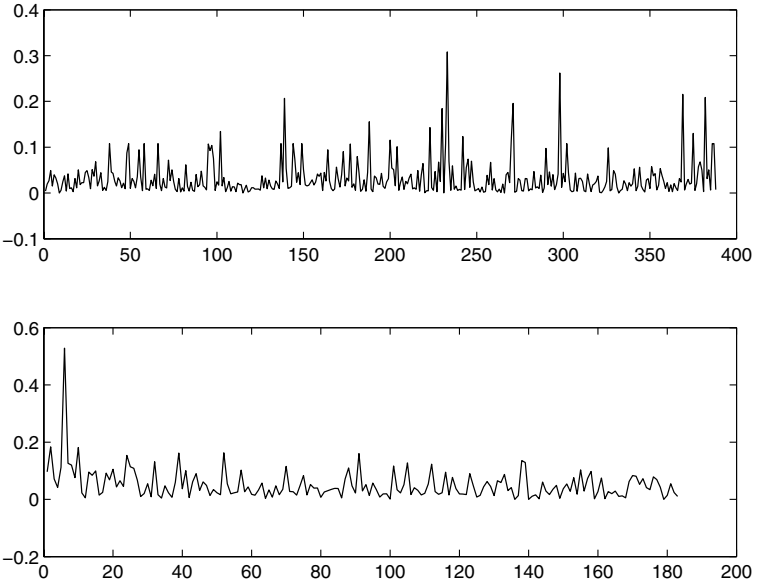

Figure 13.1. *Saliency scores for Chapter* 12: *term scores* (*top*) *and sentence scores* (*bottom*).

Example 13.1. We created a term-sentence matrix based on Chapter 12. Since the text is written using LaTeX, we first had to remove all LaTeX typesetting commands. This was done using a lexical scanner called detex.[35] Then the text was stemmed and stop words were removed. A term-sentence matrix A was constructed using the text parser TMG [113]: there turned out to be 388 terms in 183 sentences. The first singular vectors were computed in MATLAB, `[u,s,v]=svds(A,1)`. (The matrix is sparse, so we used the SVD function for sparse matrices.) The singular vectors are plotted in Figure 13.1.

By locating the 10 largest components of u_1 and using the dictionary produced by the text parser, we found that the following words, ordered by importance, are the most important in the chapter:

[33]This can be demonstrated easily using the SVD of A.

[34]The matrix A has nonnegative elements; therefore the first principal component u_1 (see Section 6.4) must have nonnegative components. The corresponding holds for v_1.

[35]http://www.cs.purdue.edu/homes/trinkle/detex/.

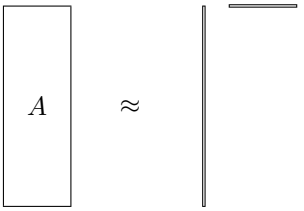

Figure 13.2. *Symbolic illustration of a rank-*1 *approximation:* $A \approx \sigma_1 u_1 v_1^T$.

page, *search*, *university*, *web*, *Google*, *rank*, *outlink*, *link*, *number*, *equal*

The following are the six most important sentences, in order:

1. A Google search conducted on September 29, 2005, using the search phrase *university*, gave as a result links to the following well-known universities: *Harvard*, *Stanford*, *Cambridge*, *Yale*, *Cornell*, *Oxford*.
2. When a search is made on the Internet using a search engine, there is first a traditional text processing part, where the aim is to find all the Web pages containing the words of the query.
3. Loosely speaking, Google assign a high rank to a Web page if it has inlinks from other pages that have a high rank.
4. Assume that a surfer visiting a Web page chooses the next page from among the outlinks with equal probability.
5. Similarly, column j has nonzero elements equal to N_j in those positions that correspond to the outlinks of j, and, provided that the page has outlinks, the sum of all the elements in column j is equal to one.
6. The random walk interpretation of the additional rank-1 term is that in each time step the surfer visiting a page will jump to a random page with probability $1 - \alpha$ (sometimes referred to as *teleportation*).

It is apparent that this method prefers long sentences. On the other hand, these sentences are undeniably key sentences for the text. ∎

The method described above can also be thought of as a rank-1 approximation of the term-sentence matrix A, illustrated symbolically in Figure 13.2.

In this interpretation, the vector u_1 is a basis vector for the subspace spanned by the columns of A, and the row vector $\sigma_1 v_1^T$ holds the coordinates of the columns of A in terms of this basis. Now we see that the method based on saliency scores has a drawback: if there are, say, two "top sentences" that contain the same high-saliency terms, then their coordinates will be approximately the same, and both sentences will be extracted as key sentences. This is unnecessary, since they are very similar. We will next see that this can be avoided if we base the key sentence extraction on a rank-k approximation.

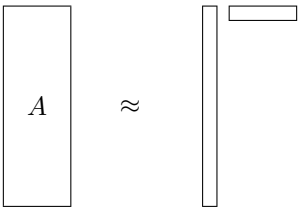

Figure 13.3. *Symbolic illustration of low-rank approximation:* $A \approx CD$.

13.2 Key Sentence Extraction from a Rank-k Approximation

Assume that we have computed a good rank-k approximation of the term-sentence matrix,

$$A \approx CD, \qquad C \in \mathbb{R}^{m\times k}, \quad D \in \mathbb{R}^{k\times n}, \tag{13.3}$$

illustrated in Figure 13.3. This approximation can be be based on the SVD, clustering [114], or nonnegative matrix factorization. The dimension k is chosen greater than or equal to the number of key sentences that we want to extract. C is a rank-k matrix of basis vectors, and each column of D holds the coordinates of the corresponding column in A in terms of the basis vectors.

Now recall that *the basis vectors in C represent the most important directions in the "sentence space,"* the column space. However, the low-rank approximation does not immediately give an indication of which are the most important sentences. Those sentences can be found if we first determine the column of A that is the "heaviest" in terms of the basis, i.e., the column in D with the largest 2-norm. This defines one new basis vector. Then we proceed by determining the column of D that is the heaviest in terms of the remaining $k-1$ basis vectors, and so on.

To derive the method, we note that in the approximate equality (13.3) we may introduce any nonsingular matrix T and its inverse between C and D, and we may multiply the relation with any permutation P from the right, without changing the relation:

$$AP \approx CDP = (CT)(T^{-1}DP),$$

where $T \in \mathbb{R}^{k\times k}$.

Starting from the approximation (13.3), we first find the column of largest norm in D and permute it by P_1 to the first column; at the same time we move the corresponding column of A to the first position. Then we determine a Householder transformation Q_1 that zeros the elements in the first column below the element in position $(1,1)$ and apply the transformation to both C and D:

$$AP_1 \approx (CQ_1)(Q_1^T DP_1).$$

In fact, this is the first step in the QR decomposition with column pivoting of D. We continue the discussion in terms of an example with $m = 6$, $n = 5$, and $k = 3$.

To illustrate that the procedure deals with the problem of having two or more very similar important sentences (cf. p. 164), we have also assumed that column 4 of D had almost the same coordinates as the column that was moved to the first position. After the first step the matrices have the structure

$$(CQ_1)(Q_1^T DP_1) = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix} \begin{pmatrix} \kappa_1 & \times & \times & \times & \times \\ 0 & \times & \times & \epsilon_1 & \times \\ 0 & \times & \times & \epsilon_2 & \times \end{pmatrix},$$

where κ is the Euclidean length of the first column of DP_1. Since column 4 was similar to the one that is now in position 1, it has small entries in rows 2 and 3. Then we introduce the diagonal matrix

$$T_1 = \begin{pmatrix} \kappa_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

between the factors:

$$C_1 D_1 := (CQ_1 T_1)(T_1^{-1} Q_1^T DP_1) = \begin{pmatrix} * & \times & \times \\ * & \times & \times \\ * & \times & \times \\ * & \times & \times \\ * & \times & \times \\ * & \times & \times \\ * & \times & \times \end{pmatrix} \begin{pmatrix} 1 & * & * & * & * \\ 0 & \times & \times & \epsilon_1 & \times \\ 0 & \times & \times & \epsilon_2 & \times \end{pmatrix}.$$

This changes only column 1 of the left factor and column 1 of the right factor (marked with $*$). From the relation

$$AP_1 \approx C_1 D_1,$$

we now see that the first column in AP_1 is approximately equal to the first column in C_1. Remembering that the columns of the original matrix C are the dominating directions in the matrix A, *we have now identified the "dominating column" of A.*

Before continuing, we make the following observation. If one column of D is similar to the first one (column 4 in the example), then it will now have small elements below the first row, and it will not play a role in the selection of the second most dominating document. Therefore, *if there are two or more important sentences with more or less the same key words, only one of then will be selected.*

Next we determine the second most dominating column of A. To this end we compute the norms of the columns of D_1, excluding the first row (because that row holds the coordinates in terms of the first column of C_1). The column with the

largest norm is moved to position 2 and reduced by a Householder transformation in a similar manner as above. After this step we have

$$C_2D_2 := (C_1Q_2T_2)(T_2^{-1}Q_2^TD_1P_2) = \begin{pmatrix} * & * & \times \\ * & * & \times \\ * & * & \times \\ * & * & \times \\ * & * & \times \\ * & * & \times \\ * & * & \times \end{pmatrix} \begin{pmatrix} 1 & * & * & * & * \\ 0 & 1 & * & \epsilon_1 & * \\ 0 & 0 & \times & \epsilon_2 & \times \end{pmatrix}.$$

Therefore the second column of

$$AP_1P_2 \approx C_2D_2$$

holds the second most dominating column.

Continuing the process, the final result is

$$AP \approx C_kD_k, \qquad D_k = \begin{pmatrix} R & S \end{pmatrix},$$

where R is upper triangular and P is a product of permutations. Now the first k columns of AP hold the dominating columns of the matrix, and the rank-k approximations of these columns are in C_k. This becomes even clearer if we write

$$AP \approx C_kRR^{-1}D_k = \widehat{C}\begin{pmatrix} I & \widehat{S} \end{pmatrix}, \tag{13.4}$$

where $\widehat{C} = C_kR$ and $\widehat{S} = R^{-1}S$. Since

$$\widehat{C} = CQ_1T_1Q_2T_2 \cdots Q_kT_kR$$

is a rotated and scaled version of the original C (i.e., the columns of C and $\widehat{C}$ span the same subspace in $\mathbb{R}^m$), it still holds the dominating directions of A. Assume that $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ are the first k columns of AP. Then (13.4) is equivalent to

$$a_{i_j} \approx \hat{c}_j, \qquad j = 1, 2, \ldots, k.$$

This means that *the dominating directions, which are given by the columns of $\widehat{C}$, have been directly associated with k columns in A.*

The algorithm described above is equivalent to computing the QR decomposition with column pivoting (see Section 6.9.1),

$$DP = Q\begin{pmatrix} R & S \end{pmatrix},$$

where Q is orthogonal and R is upper triangular. Note that if we are interested only in finding the top k sentences, we need not apply any transformations to the matrix of basis vectors, and the algorithm for finding the top k sentences can be implemented in MATLAB as follows:

```
% C * D is a rank k approximation of A
[Q,RS,P]=qr(D);
p=[1:n]*P;
pk=p(1:k);    % Indices of the first k columns of AP
```

Example 13.2. We computed a nonnegative matrix factorization of the term-sentence matrix of Example 13.1 using the multiplicative algorithm of Section 9.2. Then we determined the six top sentences using the method described above. The sentences 1, 2, 3, and 5 in Example 13.1 were selected, and in addition the following two:

1. Due to the sparsity and the dimension of A (of the order billions), it is out of the question to compute the eigenvector using any of the standard methods described in Chapter 15 for dense matrices, as those methods are based on applying orthogonal transformations to the matrix.

2. In [53] an adaptive method is described that checks the convergence of the components of the pagerank vector and avoids performing the power iteration for those components.

The same results were obtained when the low-rank approximation was computed using the SVD. ∎