Text Mining in Matlab using K-Means Clustering & LGK Bidiagonalization

Adrian Andersson
Daniel Sahlin
Joel Paulsson

1 Introduction

This report presents a project in the course TNA009, *Computational methods for science and technology*, at Linköpings University, during the autumn semester of 2019.

The concept of text mining refers to different methods used to extract interesting aspects of large and often unstructured data sets, containing information from written resources like books, articles or websites, to mention a few [1]. This report describes the mathematical background of two different methods for text mining and the process of implementing these.

2 Aim

The aim of this report is to present a comparison of two different methods of data mining for a specific dataset, namely the *Medline* dataset. The two methods used to retrieve relevant documents is K-Means clustering and Lanczos-Golub-Kahan-bidiagonalization, further referred to as LGK. The experiment is based on the content in chapters 12.4 and 12.6 in [1] and are implemented in MATLAB. The results are evaluated using a precision-recall graph from which conclusions are drawn in addition to comparing the computational time of both algorithms to retrieve documents.

3 Method

The *Medline* dataset is commonly used when validating different text mining methods and was used in this project. *Medline* consists of 1033 documents of medical abstracts, 30 search phrases and a dictionary with 4163 words. The dataset also contains the relevant documents for each search query so a comparison can be made. The focus in this project is to retrieve the abstracts that is most relevant to a specific query. A typical query from the dataset can look like this:

Q9: "the use of induced hypothermia in heart surgery, neurosurgery, head injuries, and infectious diseases."

This is the ninth query in the query data, referred to as Q9, and was used in this project to test the text mining methods.

To be able to find relevant abstracts from a query some preprocessing must be done to the data. Stop words need to be eliminated and words needs to be stemmed. Stop words are words that is found in any document and can thus be removed because they do not make one document different from another, the only words that remain are keywords that describe a document. Stemming is a process that reduce each word to its stem, for example comput is the stem of computable, computation, computing, computed, computational [1]. To be able to use the data in MATLAB the data exists in a *vector space*. The vector space is represented in a term-document matrix where each column vector represent a document and each row vector represent a word. Elements that are nonzero in the columns corresponds to keywords in the document. Note that the term-document matrix A (1) contains elements that often are zero, and can then be stored as a sparse matrix.

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \\ 1 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 1 & 0 & 0 & \\ 1 & 0 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 1 & 0 & .. \\ 0 & 0 & 1 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 1 & \\ & & . & & & \\ & & . & & & \end{pmatrix} \in \mathbb{R}^{m \times n} \qquad (1)$$

To determine how important a keyword is in a document a *term weightning scheme* is applied. An element in a term-

- *Adrian Andersson*
  E-mail: adran117@student.liu.se
- *Daniel Sahlin*
  E-mail: dansa828@student.liu.se
- *Joel Paulsson*
  E-mail: joepa811@student.liu.se

document matrix A can be described as (2).

$$a_{ij} = f_{ij} log(n/n_i) \tag{2}$$

Where f_{ij} is the keyword frequency, the number of times that the keyword i appeared in the document j, n is the number of documents and n_i is the number of documents that contains keyword i. If a keyword is found many times and is only found in a few documents then the weight a will be large.

3.1 Cosine Similarity

There are many different ways to determine how similar a document and a search query are, one is the cosine similarity, which measure the angle between two vectors. This method is good when the dimension gets high since it does not consider the length of the vectors [2]. If two vectors are close to each other the value converges to one, see figure 1. In the implementation the n closest documents are picked as relevant.

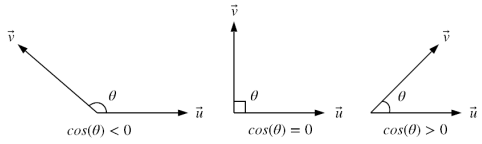

Fig. 1. *Distance measurement with cosine similarity between two vectors $\vec{u}$ & $\vec{v}$.*

3.2 K-means Clustering

In a collection of documents it is likely for some documents to be more similar than others regarding their content. If one makes a search for interesting documents in the collection, the search can either be executed over all documents directly or by first grouping similar documents into clusters and search for a group. To do the latter, the K-means clustering can be used. K-means is an unsupervised learning algorithm that creates clusters of similar data.

For our case, working with documents (abstracts) represented as vectors in the term-document-matrix A, we can think of these vectors as points in a space $\mathbb{R}^m$. By doing so, it makes sense to do a low-rank approximation of the dataset by grouping similar documents into k clusters resulting in a rank-k approximation of the term-document-matrix. With this approach, similar documents end up under the same cluster C_i and one can measure the distance for a search query q to the centroid of each cluster in the new basis by solving a least squares problem (3).

$$\min_{\hat{G}_k} \|A - C_k \hat{G}_k\|_F \tag{3}$$

To make this process more efficient it is preferred to first compute the QR-decomposition of C_k (4), i.e. making the columns of C_k orthogonal before solving the least squares problem (5).

$$C_k = P_k R, \quad P_k \in \mathbb{R}^{m \times k}, \quad R \in \mathbb{R}^{k \times k} \tag{4}$$

$$\min_{G_k} \|A - P_k G_k\|_F, \quad G_k = P_k^T A \tag{5}$$

With this approach we need to transform the search query q to the same basis (6), (7) and finally calculate the cosine similarity between the transformed search query and the columns of G_k (8). Pseudo code for the implementation can be seen in Listings 1.

$$q^T A \approx q^T P_k G_k = (P_k^T q)^T G_k = q_k^T G_k \tag{6}$$

$$q_k = P_k^T q \tag{7}$$

$$cos(\theta) = \frac{q_k^T g_j}{||q_k||_2 ||g_j||_2} \tag{8}$$

Listing 1. Pseudo code for clustering of a document matrix A.

```
% Set the amount of clusters
k = 50;

% Clustering; idx contains labels with
    cluster index, C contains centroids
[idx, C]= kmeans(A_norm', k);
C = normalize(C'); idx = idx';

% Compute the thin-QR
[P_k, R] = qr(C,0);

%Create G_k
G_k = P_k'*A_norm;

%Transform the query
q_k = P_k'*q;

%Compute distances between the query and
    the documents
Match = CosineDist(q_k, G_k);

%Find the 3 closest documents in the new
    vector space
[~, Docs] = maxk(Match, numberOfDocs);
```

3.3 LGK Bidiagonalization

The LGK algorithm is a recursive method to counter ill-conditioned problems by creating a rank-k approximation of the vector space spanned by a matrix. In a text mining application this means a low-order approximation, a *Krylov* subspace, of the term-document matrix A. Equation (9) demonstrates the mathematical problem. The LGK algorithm includes the solution (in this case a query) to let it influence the approximation in order to have faster decaying residuals [1].

$$\min_{y} ||AZ_k y - q||_2 \tag{9}$$

In the equation above Z_k denotes the new low-rank approximate basis for our solution and q denotes a query. As before A is our term-document matrix. Furthermore the following matrices can be defined as shown in equation (10).

$$AZ_k = P_{k+1}B_{k+1} \tag{10}$$

The matrix B_{k+1} is a bidiagonal of the first k steps of the recursive method and P_{k+1} is orthogonal as presented in equation (11).

$$B_{k+1} = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \\ & & & \beta_{k+1} \end{pmatrix}, \tag{11}$$
$$Z_k = (z_1 \quad \dots \quad z_k),$$
$$P_{k+1} = (p_1 \quad \dots \quad p_{k+1})$$

To compute these matrices of rank-k the algorithm 1 can be implemented. This gives us the matrices shown in equation (11).

Algorithm 1 *Pseudo code*

1: $\beta_1 p_1 = q, z_0 = 0$
2: **for** $i = 1$ to k **do**
3: $\quad \alpha_i z_i = A^T p_i - \beta_i z_{i-1}$
4: $\quad \beta_{i+1} p_{i+1} = A z_i - \alpha_i p_i$
5: **end for**

For the implementation it is noteworthy to mention that the matrices gradually lose their orthogonality as the rank increases, thus a fewer number of steps are preferred. There are methods to re-orthogonalize [1] the matrices but they will not be considered in this implementation. Once computed, the matrix B_{k+1} is decomposed using the QR method, see equation (12).

$$B_{k+1} = Q_{k+1} \begin{pmatrix} \hat{B}_k \\ 0 \end{pmatrix} \tag{12}$$

Using the decomposition of B_{k+1} the following matrices is constructed, as presented in equation (13).

$$W_k = P_{k+1} Q_{k+1} \begin{pmatrix} I_k \\ 0 \end{pmatrix}, \qquad Y_k = Z_{k+1} \hat{B}_k \tag{13}$$

The matrices W_k and Y_k is an approximation of our original term-document matrix A, $A \approx W_k Y_k^T$. This approximation however is biased towards the current query and is thus not certain to be a good approximation for any other query. The query vector is now transformed according to equation (14) below.

$$\tilde{q} = W_k W_K^T q \tag{14}$$

As a last step the transformed query vector is used to measure the cosine distance between all documents and a set amount of least distances are chosen to be relevant.

4 Result

In figure 2 the average precision and recall of both methods are presented as well as for the much simpler method of only computing the closest cosine distances between documents and the query. The LGK uses a rank-5 approximation and the K-means method uses 50 clusters. Due to the random nature of the initial placement of the centroids in the K-means clustering method the result slightly differs between runs. The amount of documents retrieved varies from 1 to 18.

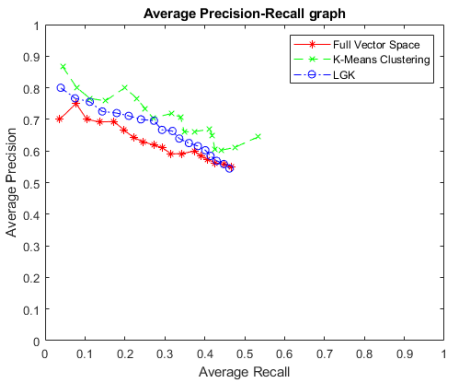

Fig. 2. *Precision-recall graph for a the search query Q9.*

Table 1. Average execution times for both algorithms.

Algorithm	Average execution time (s)
K-Means	0.0060
LGK	0.6324

5 Discussion

For our implementation we have not done any experimenting regarding the choice of the number of clusters in K-means, or the number of recursions in the LGK algorithm. Therefore a single conclusion can not be made toward which algorithm produces the best results.

Regarding K-means clustering, the algorithm might return different result each time the program is executed depending of the random initial placement of the centroids.

The LGK method needs to recalculate the approximate subspace for each new query and is thus somewhat slower than K-means, as presented in table 1, this however means that adding or removing documents and updating the low rank approximation is cheap compared to updating the clusters of a K-means algorithm. It is important to consider the number of recursions when computing LGK since a high rank approach will result in a loss of orthogonality, hence a low rank approach is preferred.

The results are also affected by the number of documents retrieved, another approach could be to set a threshold value and only retrieve documents with a lower cosine distance than this threshold.

6 Conclusion

With current parameter settings K-means clustering gives a better accuracy and performance over LGK with the downside that it is costly to update the term-document matrix. Both algorithms have a significantly better accuracy than using the full vector space to compute the cosine distances.

References

[1] Lars Eldén. *Matrix Methods in Data Mining and Pattern Recognition, Second Edition*. 2019.

[2] Gavin Patinent. *The #$%! of Dimensionality*. 2017. URL: https://towardsdatascience.com/that-cursing-dimensionality-ac317fb0fdcc.