Hamburg University of Technology
School of Electrical Engineering, Computer Science and Mathematics

Institute of Mathematics

Seminar paper

Matirx Bidiagonalizations and its Applications in Text Mining

Seminar - Matrix Methods in Data Mining and Pattern Recognition

by

Fin Bießler
Matriculation Nr.: 52906

Submission Date: December 8, 2020
Supervisor: Dr. Haibo Ruan

Contents

Listings

1 Introduction

Text mining (TM) is the part of computer science that deals with the discovery of new information by automatically extracting the information from written documents. It has applications in various sectors like publishing and media, where TM is aimed at filing unstructured textual material into categories and structured fields. In the pharmacy, research and healthcare niche TM is used to classify and extract information from articles, scientific abstracts and patents. Market analysis can also make use of text mining, here TM is used to identify new potential customers, monitor and analyse competitors or determine a company's image through the exploration of press and media data [1]. Apart from these there are many more fields where TM is applicable, almost any business can benefit from using data to analyse, optimise, adjust or verify their processes.

As well as all these business related applications there are services nearly everyone on earth uses, for example search engines, which use text mining to optimise the process of information retrieval based on a search queries or rank the immense number of results retrieved by relevance.

In the course of this seminar paper matrix bidiagonalization will be illuminated in the context of text mining. Throughout that the focus is put on bidiagonalization using the householder transformation method paired with and without the LGK algorithm. At first a theoretical introduction on how to bidiagonalize a matrix using householder transformation is given, followed by a closer look into performing the bidiagonalization procedure using the LGK algorithm. Afterwards the process of implementing the named algorithms in Python is elicited. The paper is closed with a short summary about the topic including an outlook how to apply the portrayed concepts. Unless otherwise stated results and information were taken from the [2], which served as a theoretical basis for this seminar.

2 Bidiagonalization - Householder Trasformations

Before going into the theory of householder transformations the necessary definitions are given.

2.1 Definitions

Definition 2.1 (Orthogonal matrices)

A matrix $A \in R^{n\times n}$ with the property $A^T A = 1$ is called orthogonal.

Definition 2.2 (Symmetric matrices)

Let a matrix $A \in R^{n\times n}$ have the property that $A^T = A$ then A is called symmetric.

Y*-3º1

2.2 Constructing Householder Transformations

Reflection matirces are also known by the name householder transformations. In the following a theoretical foundation on the concepts is given.

Take an arbitrary vector $v \neq \vec{0}$ and put

$$P = I - \frac{2}{v^T v} v v^T \tag{2.1}$$

then P is symmetric and orthogonal, this can be verified by a simple computation which is left to the reader. Suppose the two vectors x and y are given with the property that $\|x\|_2 = \|y\|_2$ can we find a refelction matrix P such that

$$Px = y \wedge y_i = \begin{cases} \alpha & i = 1 \\ 0 & i \neq 1 \end{cases} \tag{2.2}$$

Using equation (2.1) we can rewrite the equation in the following way

$$x - \frac{2v^T x}{v^T v} v = y \tag{2.3}$$

this can be represented by $\beta v = x - y$ where $\beta = \frac{2v^T x}{v^T v}$. Now adding the constraint that $\beta = 1$, $v = x - y$ is obtained. Taking $v = x - y$ and multiplying it from the left with v^T, note that $v^T = (x-y)^T = x^T - y^T$, we get

$$v^T v = x^T x + y^T y - 2x^T = 2(x^T x - x^T y), \tag{2.4}$$

because of $\|x\|_2 = \|y\|_2 \implies x^T x = y^T y$ (See Definitions 2.3 and 2.4). By equation (2.4) we have that

$$v^T x = x^T x - y^T x = \frac{1}{2} v^T v$$

Following from that, we obtain

$$Px = x - \frac{2v^T x}{v^T v} v = x - v = y$$

like we wanted. The construction of P in the way that

$$y = \begin{pmatrix} \alpha \\ 0_{n-1} \end{pmatrix} = x - v \implies y_i = x_i - v_i = \begin{cases} \alpha = \|x\|_2 & i = 1 \\ 0 & i \neq 1 \end{cases} \tag{2.5}$$

implies that we have that $y = \alpha e_1$ where $e_1 = \begin{pmatrix} 1 & 0 & \dots & 0 \end{pmatrix}^T$. By equation (2.5) we know that $\alpha = \pm \|x\|_2$ Thus v is be put equal to

$$v = x - \alpha e_1$$

Since we have computed v we can simplify equation (2.1) and get

$$P = I - \frac{2}{v^T v} v v^T = I - 2uu^T, \quad u = \frac{1}{\|v\|_2} v \tag{2.6}$$

We call u by the name householder vector and it's length is 1.

2.3 Biagonalization of Matrices using Householder Transformations

With regard to it's application in text mining, it is inconvenient to store the whole Householder matrix P in most cases because in applications, matrices which are bidiagonalized are in the order of 1000×1000 and even greater depending on the use case. Thus storing the whole transformation matrices makes the algorithm very inefficient and demanding on large storage volume.

But then one could ask how to apply the householder transformation P to a matrix or vector without knowing it explicitly. For multiplying a householder matrix with a vector x it comes in handy knowing that $P = I - 2uu^T$ from equation (2.6), thus we have

$$Px = x - (2u^T x)u. \tag{2.7}$$

Analogously for the application of a householder transformation P to a matrix X we obtain

$$PX = X - 2u(u^T X). \tag{2.8}$$

In the following an illustration on how to use householder transformations to bidiagonalize a sample matrix filled with arbitrary values is given. Assume a matrix $A \in R^{5\times 4}$, starting the bidiagonalization in the columns of A from left to right, we want to zero all elements in the first column that are below the main diagonal. Namely all elements of the first column but the very first. Thus we want to apply a householder transformation P_1 such that

$$P_1 A = P_1 \begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{pmatrix} = \begin{pmatrix} \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \end{pmatrix} =: A^{(1)} \tag{2.9}$$

($\times$ indicate numbers that are in their initial state or remained in it after applying the transformation, $\star$ indicate numbers that have changed due to the transformation)

We continue the bidiagonalization procedure by zeroing all elements of the first row but the elements in position one and two. This is achieved by multiplying $A^{(1)}$ by a different householder transformation Z_1. Since we don't want to change the numbers we just introduced (especially the zeros) we embed Z_1 in a larger unit matrix in the following way

$$R^{4\times 4} \ni W_1^T = \begin{pmatrix} 1 & 0 \\ 0 & Z_1^T \end{pmatrix}, \tag{2.10}$$

now multiplying $A^{(1)}$ with W_1^T from the right we obtain

$$A^{(1)}W_1^T = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{pmatrix} W_1^T = \begin{pmatrix} \times & \star & 0 & 0 \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \end{pmatrix} =: A^{(2)} \tag{2.11}$$

Continuing, we zero all elements below the main-diagonal in the second column of $A^{(2)}$ by a transformation from the left again we have to ensure that the zeros we introduced earlier are not destroyed thus the new transformation is of the following form

$$R^{5\times 5} \ni P_2 = \begin{pmatrix} 1 & 0 \\ 0 & \hat{P}_2 \end{pmatrix}, \; with \; \hat{P}_2 \in R^{4\times 4} \tag{2.12}$$

applying P_2 to $A^{(2)}$ we get

$$P_2 A^{(2)} = \begin{pmatrix} \times & \times & 0 & 0 \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & \star & \star \end{pmatrix}. \tag{2.13}$$

This procedure is continued in an analogous way until you have

$$PAW^T = \begin{pmatrix} \star & \star & 0 & 0 \\ 0 & \star & \star & 0 \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \hat{B} \\ 0 \end{pmatrix} = B. \tag{2.14}$$

With

$$P = P_1 P_2 \cdots P_4, \quad W^T = W_1^T W_2^T$$

which are products of householder transformations, and

$$\hat{B} = \begin{pmatrix} \beta_1 & \alpha_1 & 0 & 0 \\ 0 & \beta_2 & \alpha_2 & 0 \\ 0 & 0 & \beta_3 & \alpha_3 \\ 0 & 0 & 0 & \beta_4 \end{pmatrix}$$

is upper bidiagonal.

In conclusion it remains to be said that embedding a householder transformation into a larger identity matrix is inefficient and should not be done. Instead one should apply the householder vector manually to the column or row of the matrix that are effected by the transformation and then put the column/row back into the matrix that is bidiagonalized. In this way the overall performance (storage consumption and calculation speed) is increased. As a remark, if one is interested in the transformations themselves it is impossible to apply this optimization.

2.4 Solving Least Squares Problems using our Results

Suppose we want to solve a least squares problem of the form $min_x \|b - Ax\|_2$, where $A \in R^{m \times n}$ is some arbitrary matrix. Now chose $C = (b \quad A)$ (where $(b \quad A)$ means to put b as a column in front of A) we can apply the results from preceding sections,

namely equation (2.14) to optimize our solution in the following way

$$PCW^T = P(b \quad A)\begin{pmatrix}1 & 0\\ 0 & Z^T\end{pmatrix} = (P^Tb \quad P^TAZ) = \begin{pmatrix}\beta_1 e_1 & \tilde{B}\\ 0 & 0\end{pmatrix} \tag{2.15}$$

with $\tilde{B}$ being

$$\tilde{B} = \begin{pmatrix}\alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_n & \alpha_n \\ & & & \beta_{n+1}\end{pmatrix}$$

Note that

$$\|b - Ax\|_2 = \left\|(b \quad A)\begin{pmatrix}1\\ -x\end{pmatrix}\right\|_2$$

defining $y = Z^Tx$ we can simplify in the following way

$$\begin{aligned}\left\|(b \quad A)\begin{pmatrix}1\\ -x\end{pmatrix}\right\|_2 &= \left\|P(b \quad A)\begin{pmatrix}1 & 0\\ 0 & Z_1^T\end{pmatrix}\begin{pmatrix}1\\ -y\end{pmatrix}\right\|_2 \\ &= \left\|(P^Tb \quad P^TAZ)\begin{pmatrix}1\\ -y\end{pmatrix}\right\|_2 = \left\|\beta_1 e_1 - \tilde{B}y\right\|_2\end{aligned} \tag{2.16}$$

this least squares problem which is now reduced to bidiagonal form can be solved in $O(n)$ flops. For this one would have to reduce to upper bidiagonal form beforehand using plane rotations.

3 Bidiagonalization - LGK Algorithm

In the upcoming section an alternative approach how to compute the bidiagonalization of a matrix C is presented using the so called LGK-Algorithm (Golub-Kahan-

Lanczos). Note that the algorithm computes the bidiagonalization in a recursive manner.

Algorithm 1: LGK Bidiagonalization

Result: Bidiagonalized matrix; matrices P and Z

$\beta_1 p_1 = b$;

$z_0 = 0$;

for $i = 1 : n$ **do**

$\alpha_i z_i = A^T p_i - \beta_i z_{i-1}$;

$\beta_{i+1} p_{i+1} = A z_i - \alpha_i p_i$

end

The coefficients α and β are determinded such that $\|p\|_2 = \|z\|_2 = 1$

4 Implementation of the presented Algorithms

4.1 Implementing the Bidiagonalization using Householder Transformations

In this subsection an overview on how to implement the bidiagonalization with householder transformations (as discussed in Section 2) is given. Due to it's simplicity and strong third party libraries like Numpy, Python is the recommended programming language to implement the presented algorithms. Starting off, a brief description on the input and outputs of the algorithm is given, following that a description on how to implement the procedure in the chosen language is given.

4.1.1 Input and Outputs of the Algorithm

Input:

As an input any matrix $C \in R^{m \times n}$ can be given to the algorithm.

Outputs:

As outputs the algorithm not only produces the bidiagonalized matrix B but also the left and right householder transformations P and W^T.

4.1.2 Description of the Implementation

Here a brief description on how to implement the discussed algorithm is given using pseudo-code. First off we have the main method of the householder bidiagonalization `houseBidiag(A)` which basically does all things like producing the householder vectors and applying them to the matrix A, which is to be bidiagonalized.

Note that importing and using the Numpy module can simplify the implementation tremendously. To make the code more readable and adaptable to new requirements one can define the methods `computeHouseVec(x)` and `blockMatrixEmbedHouseVec(n, colAndRow, v)`. `computeHouseVec` computes the householder vector v for a given vector x. `blockMatrixEmbedHouseVec` embeds this householder vector into a larger block matrix.

```
houseBidiag(A):
    create matrix P of size m x m
    create matric W of size n x n

    for each column of A:
        compute householder vector for column
        apply householder vector to column, save result in A
        //when interested in actual householder transformations
        put householder transformation in P
        //for this use function blockMatrixEmbedHouseVec

        if column is not above n-2:
            compute householder vector for row
            apply householder vector to row, save result in A
            //when interested in actual householder transformations
            put householder transformation in W
            // for this use function blockMatrixEmbedHouseVec
    proceed to next column
return P, A, W
```

Listing 1: Householder Bidiagonalization

Furthermore more insights on how to implement the householder vector calculation is given in the following pseudo-code snippet, where x is a column vector of A.

```
computeHouseVec(x):
    //dot_wo1
    dot-product of x (except 1. element) with itself, save result
    save copy of x in v
    set first element of v to 1.0
    compute norm of x, save result
    if first element of x <= 0:
        set v[0] to x[0] - norm of x
    else:
        v[0] = -dot_wo1 ( x[0] - norm x)
    v = v / v[0]
    return v
```

Listing 2: Computing the Householder Vector

Last but not least a snippet on how to embed a Householder vector v in a identity matrix.

```
blockMatrixEmbedHouseVec(n, colAndRow, v):
    //n is the size of the full matrix -> n x n
    //colAndRow indicates from which column and row on
    //the householder vector should be embedded
    //v is the householder vector to embed
    create identity matrix of size n x n
    compute outer product of v with itself, save result
    return subtract prod from identity  //column=colAndRow to end;
   row=colAndRow to end
```

Listing 3: Embedding a Householder Vector in an Identity Matrix

4.2 Implementing the Bidiagoanlization using the LGK Algorithm

For this adjusted bidiagonalization algorithm the in- and output are nearly the same. Just the input is treated slightly different. The first column of the input matrix is assumed to be the right hand side of the equation $Ax = b$.

Input:

As an input any matrix $C = (b \quad A) \in R^{m \times n}$ can be given to the algorithm.

Outputs:

As outputs the algorithm not only produces the bidiagonalized matrix B but also the vectors p_i and v_i.

5 Summary and Outlook

The concepts described in this paper have applications in text mining or to be more precise in information retrieval. Now a short outlook on how to apply them explicitly is given. Besides representing the term-document matrix A by a low-rank approximation based on Sigular Value Decomposition (SVD), clustering, and non-negative matrix factorization one can also use the LGK-Bidiagonalization to compute the

low-rank approximation. This is particularly expedient when one wants to add or delete documents from the term-document matrix, since with SVD, clustering or non-negative matrix factorization the process of updating the approximation is very costly. For the method using the LGK-Algorithm one uses the right-hand side of the equation $Ax = b$ as a starting vector. In the application of text mining the matrix A would represent the term-document matrix and the vector b would be the query q. Note that the amount of work for each query matching becomes higher since one has to compute the low-rank approximation for each query. For a given query vector q we use the Algorithm 1 (Section 3) with $b = q$.

Following that, one uses the cosine measure to compare the angle between the query and the original documents. To determine how many steps of the recursion should be performed one can monitor the norm of the residual vector of the least squares problem, when the norm stops decreasing substantially one should stop the recursion, since we than can assume that the query is represented as well as is possible by a linear combination of documents. This would lead to a performance that is about the same as when using the full vector space model. Thus to obtain a better performance one should stop the recursion well before the residual vector starts to level off. Note that this can lead to more imprecise results of the query matching procedure.

References

[1] V. Gupta and G. Lehal, "A Survey of Text Mining Techniques and Applications," *Journal of Emerging Technologies in Web Intelligence*, vol. 1, no. 1, p. 68, August 2009.

[2] L. Eldén, "Matrix methods in data mining and pattern recognition," [Siam version], 2007. [Online]. Available: https://epubs.siam.org/doi/book/10.1137/1.9780898718867