Key Word and Key Sentence Extraction

Kenton Larsson & Lucas Hägg

TNA010, Matrix Methods for AI
Examiner: Berkant Savas
October 8, 2024

Chapter 1

Introduction

This chapter is an introduction to the project as a whole, giving a detailed background as well as a description of the data set used in the project.

1.1 Background

Over the past decades there has been a rapid increase in volume of digital textual information available in diverse domains, the sheer amount of data is posing significant challenges for those who seek to efficiently extract relevant information. Manual methods for extracting key words and sentences may be prone to subjectivity, as well as being labor intensive, and therefore do not qualify as a viable option. Because of this there is a clear need for methods which perform these tasks automatically, providing users with an effortless way to ascertain significant information contained in a text document or corpus.

The aim of this project was to implement an extraction method to extract key words and key sentences from a data set in the form of a text document. The methods are based on the Salience score and the rank-k matrix approximation.

This paper presents the underlying theories and mathematical models behind the implementation of the methods, as well as giving a description regarding algorithmic and computational aspects of the implementation itself.

1.2 Data set

The data set used was a text file containing one of the chapters in Lars Eldén book *'Matrix Methods in Data Mining and Pattern Recognition'*, namely chapter 13[1]. The chapter is about page ranking for a web search engine, and how one can filter out pages that are assumed to be less interesting. The chapter goes through how this can be implemented with mathematical matrix methods.

The raw .txt file consisted of a total of 3176 English terms, contained in 192 sentences separated by a blank row. The file did not contain any formatted words, such as *LaTeX*, but did include both upper and lower case letters, and stop words, all of which need to be processed or otherwise removed in order to facilitate the extraction of information from the text.

Chapter 2

Theory

This theory chapter explores the theoretical foundations behind the practical implementation strategies which forms the basis for key word and key sentence extraction methods.

2.1 Data preprocessing

It is briefly mentioned in the previous section, that there is a need to clean up the data set in order to prime it for use in various methods, this is commonly referred to as data preprocessing. The preprocessing consists of several steps, including stemming, stop word removal, and making all characters lower case.

Stemming is a form of text normalization with the goal of mapping different forms of a word, such as plurals or verb conjugations, to a common root. Stemming methods operate by removing suffixes or prefixes of words, which may result in invalid words. This is however not an issue, as the objective of the method is only to group related words. The stemming method helps in simplifying the analysis of text data and improves the efficiency of text processing, as well as contributing to the consistency of the text representation, as words that convey similar meanings but have different grammatical forms will be mapped to the same stem.

Another important preprocessing step is the removal of stop words, i.e., words that occur frequently across different documents but carry little to no significant meaning on their own. Some examples of stop words include 'the', 'it', 'is', etc. Lists of stop words may vary depending on the application.

2.2 Saliency Score

To extract key words and key sentences from a document there is a for a way to quantify the measure of the significance of a given word or sentence, for this the concept of Saliency score is used. The primary objective is to identify and prioritize words and sentences in the document in such a way that promotes those with the most overall importance with regards to the information in that document.

To assign saliency scores the text of the document must first be represented as a term-sentence matrix. This is a matrix $A \in R^{mxn}$, where m denotes the number of different terms and n the number of sentences. The element a_{ij} is defined as the frequency of term i in sentence j. A common approach to compute saliency scores involves utilizing *the term frequency-inverse document frequency (TF-IDF)* weighting scheme to construct a term-document matrix. TF-IDF is a statistical measure that evaluates the importance of a term within a document relative to its frequency across a collection of documents, or in this case sentences. The calculation involves two main components:

Term frequency (TF) is the measure of a terms importance based on the frequency of occurrences in a given document, and is calculated as shown in (2.1),

$$TF(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2.1}$$

where $f_{t,d}$ is the number of times term t appears in document d, and the denominator is the total number of terms in the document d.

Inverse document frequency (IDF) is an assessment of the rarity of a term across an entire corpus, which means that terms that are unique and do not occur in many separate documents will receive a higher IDF. The IDF is calculated as shown in (2.2),

$$IDF(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2.2}$$

where N is the total number of documents in the corpus D, and the denominator is the number of documents d in corpus D in which the term t occurs.

With both the TF and IDF computed the TF-IDF can be computed as (2.3).

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D) \tag{2.3}$$

The term-sentence matrix encapsulates the importance of each term in relation to each document, and can be used to calculate saliency score. The assignment of saliency scores is made based on the mutual reinforcement principle, described in *Eldén*[1]:

"A term should have a high salicency score if it appears in many sentences with high saliency scores. A sentence should have a high saliency score if it contains many terms with high saliency scores".

The term i is proportional to the sum of scores of the sentences where it appears. Similarly, the saliency score of sentence j is defined to be proportional to the scores of its terms. The mathematical representation of this can be seen below in equation 2.4 and equation 2.5

$$u_i \propto \sum_{j=1}^{n} a_{ij} v_j, \quad i = 1, 2, \ldots, m \tag{2.4}$$

$$v_j = \propto \sum_{i=1}^{m} a_{ij} u_i, \quad j = 1, 2, \ldots, n \tag{2.5}$$

These two equations can be written as in 2.6 by collecting the scores in vectors.

$$\sigma_u u = Av, \qquad \sigma_v v = A^T u \tag{2.6}$$

Where σ_u and σ_v are proportionality constants. Inserting one equation into the other result in:

$$\sigma_u u = \frac{1}{\sigma_v} A A^T u, \qquad \sigma_v v = \frac{1}{\sigma_u} A^T A v \tag{2.7}$$

which shows that u and v are eigenvectors of AA^T and A^TA. It follows that u and v are singular vectors to the same singular value.

The saliency scores are then extracted from the largest singular vectors u_1 and v_1, for terms and sentences respectively. One way to to calculate the singular vectors is by *singular value decomposition (SVD)*.

2.3 Singular Value Decomposition

Singular Value Decomposition (SVD) is used in information retrieval to analyze and extract important features from matrices. When applied to a matrix A, it helps identify the most important features, and in the case of A being a term-sentence matrix, these are the terms and sentences. The SVD decomposes the matrix into three constituent parts, see 2.8.

$$A = U\ \Sigma\ V^T \tag{2.8}$$

U, contains the left singular vectors which relate to information about the column space of A, V^T, contains the right singular vectors which relate to information about the column space of A, and the diagonal matrix Σ, consisting of singular values which describe the importance of the columns in U and V.

The top singular vectors represent the most relevant features, allowing for the extraction of key terms and sentences by considering the largest components of u_1 and v_1 respectively.

2.4 Rank-k Matrix Approximation

A rank-k approximation of a matrix A involves constructing a new matrix A_k, with a rank no greater than k, that closely approximates the original matrix. See equation 2.9 below for the mathematical representation

$$A = CD, \qquad C \in R^{mxk}, \quad D \in R^{kxn} \tag{2.9}$$

The concept can be applied using various techniques such as *clustering, non-negative matrix factorization* or SVD[1]. The goal of the approximation

is to represent the essential information in the original matrix with a reduced set of dimensions, which can be useful for various tasks like noise reduction, compression, and capturing the most significant patterns data, such as in key word and sentence extraction.

The SVD of a matrix A, see 2.8, may be truncated at rank-k to yield an approximation of that matrix, [2], as shown in SVD-k.

$$A_k \approx U_k \ \Sigma_k \ V_k^T \tag{2.10}$$

where $U_k = C$ and $\Sigma_k V_k^T = D$.

2.5 QR-Decomposition with Column Pivoting

Consider a rank-k approximation like equation 2.9. We get the most important sentences by determining the columns of A that are the "heaviest" in terms of the basis vectors in C. This is achieved by ordering the columns in D based on largest 2-norm.
After the first largest 2-norm in D is found, a permutation matrix P_1 is applied to shift the column to the first position. At the same time, the corresponding column in A is moved to the first position. Then a Householder transformation Q_1 is determined so that the elements in the first column below the element in position (1,1) is zeroed. Then it is applied to both C and D.

$$AP_1 \approx (CQ_1)(Q_1^T DP_1) \tag{2.11}$$

The process is then repeated with new 2-norm calculations, permutation matrices $P_{2,3,\ldots k}$ as well as Householder matrices $Q_{2,3,\ldots k}$. For each iteration, the 2-norm is reducing the amount of rows included in the calculation by 1. The reason for this is to eliminate similar entries in the "direction space" of C. When a Householder matrix is applied, similar columns in D to the highest ranked column will be left with small unique values ε in the remaining rows. The 2-norm of these columns will be small and therefore ranked low.

Chapter 3

Implementation

The following chapter will detail implementation of the two methods used for key word and sentence extraction. The implementations were made in *Python*, and made use of several libraries such as *NumPy* and *SciPy*. Algorithmic implementation is presented as pseudo code, and is analysed in the aspect of computational complexity.

3.1 Method 1 - Saliency Score

The first step was to preprocess the text file. The algorithm involved removing non-alphabetic characters, tokenizing, removing stop words, and stemming.

$content = read('data.txt')$
$sentences = split(content)$
$corpus = []$
for $sentence \in sentences$ **do**
    $preprocess(sentence)$
    $appendtocorpus$
end for

Generating the TF-IDF matrix involved using the TfidfVectorizer from sklearn. There is no time complexity notation from the official documentation [3] but the assumed time complexity is linear with respect to the number of sentences multiplied with the amount of terms in each sentence (the same goes for the prepossessing step).

$vectorizer = TfidfVectorizer()$
$tfidf = vectorizer.fit_transform(corpus)$

SVD is used to decompose the TF-IDF matrix into the three matrices (U, S, V), representing the term-document relationships. SVD is a computationally expensive operation. If m denotes the average number of terms, and n the number of sentences, the time complexity is $O(m * n + n^3)$[4].

$u, s, v = svd(tfidf)$

Extracting the most important sentences and terms involves sorting and indexing operations, which are generally efficient. We used the function argsort from the numpy library, which again has no time complexity documentation [5]. We are assuming that the time complexity is $O(m * log(m))$ for sorting, and $O(1)$ for extracting the top k values.

$usv = SVD(tfidf)$
$u1 = u[:, 0]$
$v1 = u[0, :]$
$termsIndices = argsort(u1)$
$sentencesIndices = argsort(v1)$
$impSentences = [sentences[i] for i in termsIndices]$
$impTerms = [terms[i] for i in termsIndices]$

3.2 Method 2 - Rank-k Matrix Approximation

The first steps of this method included the same preprocessing as described in section 3.1, as well as the creation of a the term-sentence matrix using TF-IDF.

The next step, once again required computing the SVD of the TF-IDF matrix, but here the goal was to compute a rank-k approximation the matrix. With the resulting matrices U, S and V, these were then truncated by selecting their k first columns and rows respectively. With the truncated matrices S_k and V_k the matrix D could be computed as the product of the two matrices.

$v = v[k, :]$
$s = s[: k]$
$D = s * v$

As described in section 2.5, each column of D now held the coordinates of the corresponding column of the TF-IDF matrix in terms of the basis vectors. Following the principles outlined in 2.5, we perform QR-decomposition using column pivoting on the matrix D. We use the built in function from scipy[6]. The time complexity is assumed to be $O(k * n^2)$ but it depends on how the function is implemented. The resulting permutation matrix P, is then used to ascertain the indices of the most important sentences.

$Q, R, S = qr(D)$
$p = [1 : n] * P$
$indices = p[1 : k]$

Chapter 4

Results, Discussion & Conclusions

This final chapter presents the results of the implementation of the methods, as well as results showing the impact of varying some of the variables in the algorithms. The results are then discussed in regards to several aspects.

4.1 Experiments & Results

The results are presented as two separate experiments, one for each of the two methods. For the first method the experiment involved extending the list of stop words by including additional stop words, particularly the words 'matrix', 'eigenvalue', 'eigenvector' and 'vector', which as we see in the following section were among the top words in our first result. The experiment for the second method was using different values for k in the rank-k approximation.

4.1.1 Results of method 1 - Saliency score

The results of the saliency score method is shown below.

The top six most important terms, in order:
'matrix', 'eigenvalue', 'eigenvector', 'pagerank', 'vector', 'define'.

The top six most important sentences:
1. "We compute the eigenvalues and eigenvectors of the matrix with from and"
2. "The column-stochastic matrix defined in is irreducible (since A ¿ 0) and has the dominant eigenvalue.

3. "We now see that the pagerank vector for the matrix is well defined"
4. "Assume that the eigenvalues of the column-stochastic matrix are"
5. "The matrix in the previous example is modified to"
6. "In analogy to (13.3), we would like to define the pagerank vector as a unique eigenvector of P with eigenvalue 1"

The top six most important terms, in order, when using extended stop words:
'pagerank', 'method', 'power', 'iterate', 'define', 'compute'.

The top six most important sentences, using extended stop words:
1. "We computed the pagerank vector using the power method with = 0.85 and iterated 63 times until the 1-norm of the residual was smaller than 106"
2. "The Power Method for Pagerank Computation"
3. "We want to solve the eigenvalue problem where r is normalized"
4. "When computing the pagerank for a subset of the Internet, say, one particular domain, the matrix P may be of a dimension for which one can use methods other than the power method, e.g., the Arnoldi method; see [49] and Section 17.8.3"
5. "We now see that the pagerank vector for the matrix is well defined"
6. "The residual in the power iterations (top) and the pagerank vector (bottom) for the stanford.edu matrix"

4.1.2 Results of method 2 - Rank-k Matrix Approximation

The results of the rank-k method is shown below with $k = 10$ and $k = 120$. For these results extended stop words were used.

The top six most important sentences, using extended stop words, $k = 10$:
1. "Assume that an initial approximation r(0) is given"
2. "Theorem 13.10 implies that even if has a multiple eigenvalue equal to 1, which is actually the case for the Google matrix, the second largest eigenvalue in magnitude of is always equal to"
3. "Let be the computed approximation of the eigenvalue and the corresponding approximate eigenvector"
4. "The power method is given in the following algorithm"
5. "The only viable method so far is the power method"
6. "We compute the eigenvalues and eigenvectors of the matrix with from

and"

The top six most important sentences, using extended stop words, $k = 120$:
1. "A Markov chain is a random process in which the next state is determined completely from the present state; the process has no memory"
2. "To ensure irreducibility, i.e., to make it impossible for the random walker to get trapped in a subgraph, one adds, artificially, a link from every Web page to all the others"
3. "Let L be the adjacency matrix of the directed Web graph"
4. "It is of course impossible to define a generally valid measure of relevance that would be acceptable for all users of a search engine"
5. "The ranking procedure was based not on human judgment but on the link structure of the Web. Loosely speaking, Google assigned a high rank to a Web page if it had inlinks from other pages that have a high rank"
6. "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"

4.2 Discussion & Conclusions

The first thing one may notice about the result is that many of the resulting sentences seem to be cut off, and therefore do not make a lot of sense. This is not a result of the methods being applied but is the nature of the data set, which contains these sentences from the start.

The result of method 1 shows that commonly used terms in linear algebra are top rated. Therefore the sentences extracted have less to do with the overall meaning of the chapter. Since the chapter is about page ranking, we would want to see more terms and sentences about that topic. It it clear from the result of the first experiment that including additional stop words, which may be considered to be undesirable, results in more desirable sentences which actually describe the chapter.

We can see that when $k = 10$, some of the sentences that include the extended stop words are top ranked (notice the word "eigenvalue" is in sentence 2,3 and 6). This was first a surprise, but does seem logical when compared to the sentences extracted when $k = 120$. These sentences include words like "page" and "web" which are words more related to the chapter from the course book. It is also reasonable that a higher k-value would give a

better result as the QR algorithm described in section 2.5 is able to calculate 2-norms that includes more rows, that is, more information.

The winning algorithm in terms of performance and time complexity is QR-decomposition with column pivoting. The complexity will depend on your choice of k. If your k value is close to the amount of rows in the data set, then the algorithm performs in cubic time (n^3), which would be similar to the SVD algorithm's time complexity.

We can conclude that the use of different stop words has great impact on the resulting key words, and therefore sentences. That being the case additional stop words should be chosen carefully depending on the application. Another clear point is that higher values for k yields marginally better results and does not majorly impact the complexity of the rank-k method.

Bibliography

[1] Lars Eldén, *Matrix Methods in Data Mining and Pattern Recognition, Second Edition*, Society for Industrial and Applied Mathematics. (2019)

[2] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1. (1936),

[3] scikit-learn, *TfidfVectorizer*, 2023, read: 2023-12-07
Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature$_e$*xtraction.text*

[4] illinois education, *ref-16-svd*, 2020, read: 2023-12-07
Available: https://courses.engr.illinois.edu/cs357/sp2021/notes/ref-16-svd.html

[5] numpy argsort documentation, *numpy.argsort*, 2023, read: 2023-12-07
Available: https://numpy.org/doc/stable/reference/generated/numpy.argsort.html

[6] scipy qr documentation, *scipy.qr*, 2023, read: 2023-12-07
Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.qr.html