2021

Fast and Memory-Efficient TFIDF Calculation for Text Analysis of Large Datasets

Samah Senbel
Sacred Heart University

Recommended Citation

Senbel, S. (2021). Fast and memory-efficient TFIDF calculation for text analysis of large datasets. In H. Fujita, A. Selamat, J. CW. Lin, & M. Ali (Eds.), *Advances and trends in artificial intelligence: Artificial intelligence practices* (pp. 557-563). Springer. Doi: 10.1007/978-3-030-79457-6_47

Fast and Memory-efficient TFIDF Calculation for Text Analysis of Large Datasets

Abstract. Term frequency – Inverse Document Frequency (TFIDF) is a vital first step in text analytics for information retrieval and machine learning applications. It is a memory-intensive and complex task due to the need to create and process a large sparse matrix of term frequencies, with the documents as rows and the term as columns and populate it with the term frequency of each word in each document.
The standard method of storing the sparse array is the "Compressed Sparse Row" (CSR), which stores the sparse array as three one-dimensional arrays for the row id, column id, and term frequencies. We propose an alternate representation to the CSR: a list of lists (LIL) where each document is represented as its own list of tuples and each tuple storing the column id and the term frequency value. We implemented both techniques to compare their memory efficiency and speed. The new LIL representation increase the memory capacity by 52% and is only 12% slower in processing time. This enables researchers with limited processing power to be able to work on bigger text analysis datasets.

Keywords: Text Analysis, Information Retrieval, TFIDF, Term Frequency, Data Structures, Memory Allocation.

1 Introduction

Information retrieval (IR) is an important part of computing, it is the process of obtaining information that are relevant to an information need from a collection of resources. IR searches can be based on text or other content-based indexing, and the search could be for information in a document, searching for documents themselves, and also searching for the metadata that describes data for data mining applications.

Text Analysis is one of the major application fields for machine learning algorithms used in IR. Natural language data is generated by humans through books, articles, reviews, and social media posts among other sources. However the raw natural language data cannot be fed directly to the machine learning or other text analysis algorithms because most are based on numerical feature vectors rather than the raw text with variable length. In order to solve this problem, researchers designed techniques to extract numerical features from text content. And then, the corpus of text content can be represented by a matrix.

In information retrieval, the "term frequency – inverse document frequency" (also called TFIDF), is a well know method to evaluate how important is a word in a document. TFIDF comes up a lot in research work because it's both a corpus exploration method and a pre-processing step for many other text-mining measures and models.

Research on the TFIDF is mostly concentrated on improving its performance and tailoring it to specific applications. Yamout et al. [8] devised and compared the performance of three new weighting techniques to improve the TFIDF weighting technique.

Zhao et al. [10] used a modified TFIDF method to do topic modeling on healthcare related publications. TFIDF is also used in several text mining applications such as research paper classification [2], hate speech on social media detection [3], grading and review systems [5], Spam email detection [7] and sentiment analysis [9].

In this paper, we present a new method for representing and calculating the TFIDF and compare its performance with the traditional method. This paper is organized as follows: In section 2, we describe how to calculate the TFIDF and the review the different methods to represent the data structure needed. We also describe our new algorithm for calculating the TFIDF. Section 3 presents our implementation results, and section 4 concludes our work.

2 TFIDF Calculation

Given a corpus of documents with N documents, our goal is to calculate the TFIDF of all words in the corpus. There are several variations on how to calculate it [6]. The term frequency (TF) used in this paper is defined as

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

where, $n_{i,j}$ represents the number of occurrences of word t_i in row d_j and $\sum_k n_{k,j}$ represents the total number of occurrences of words in row d_j. K and D are the number of keywords and documents (rows), respectively. The document frequency (DF) is how many times each keyword appears in the collection of documents. It is calculated by dividing the number of documents that contain a specific keyword by the total number of documents (eqn 2).

$$DF_{i,j} = \frac{\left|d_j \in D : t_j \in d_j\right|}{|D|} \tag{2}$$

Keywords with a high DF value cannot have high importance because they commonly appear in the most documents. Accordingly, the IDF that is an inverse of the DF is used to measure an importance of keywords in the collection of documents. The IDF is defined as:

$$IDF_{i,j} = \log \frac{|D|}{\left|d_j \in D : t_j \in d_j\right|} \tag{3}$$

Using Eqs. (1) and (3), the TF-IDF is defined as

$$TFIDF = TF \times IDF \tag{4}$$

In order to compute the TFIDF, we start by separating the words in each document into a list of individual words. Calculating the TF is straightforward as it only requires one document at a time: number of repetitions of each word is calculated and divided by the number of words in the document. Calculating the IDF is much harder: To get the IDF, we need four data structures:

- A list of all keywords in the corpus (K). This list is initially empty and grows as documents are read and processed.
- A one-dimensional list of all words in a certain document (W). This structure is used for storing the words of a document as it is being processed.
- The two-dimensional sparse array (TF) for storing the TF: the number of rows is the number of documents, and the columns represent the set of all words used in the corpus.
- A one dimensional list of document frequencies (DF), which is a parallel list of K and has the same size, it has the count of how many document contain that word.

The TF array is typically extremely large and sparse. It is extremely memory intensive and makes calculating the TFIDF impossible for a large dataset. It also slows down the calculation of the DF, as all cells have to added up, those with values and those with zeros. Therefore, it is typical to store it as a sparse matrix and do the calculation directly on it, and that also speeds up the calculations as well. There are several well-known techniques to compress a sparse matrix, figure 1 shows an example of a sparse matrix generated from a set of similar documents (tweets in this case) and three possible sparse matrix representations:

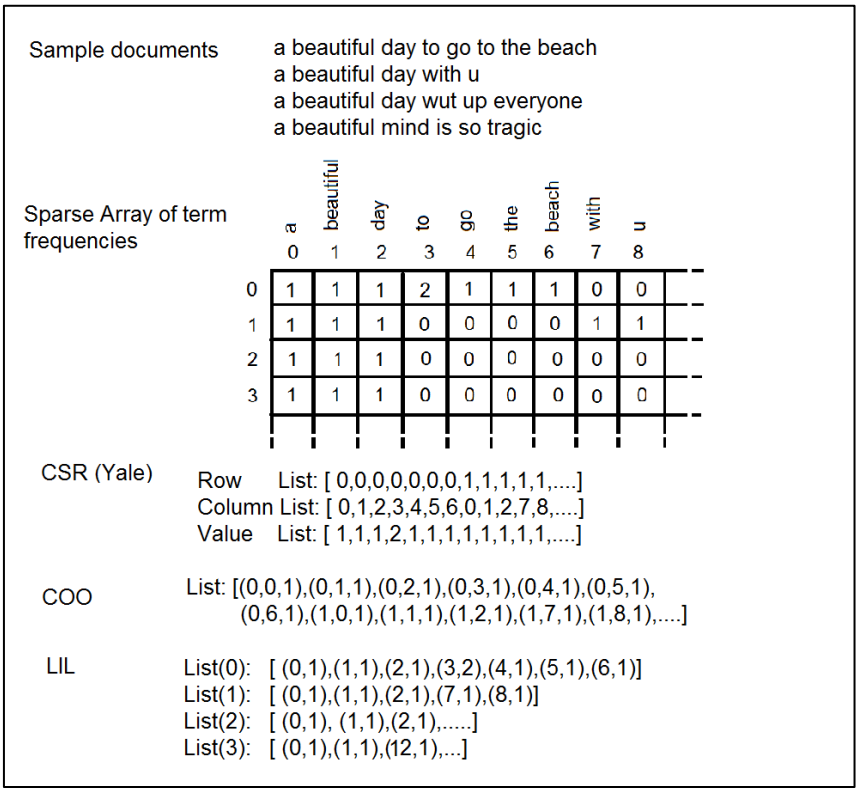

Figure 1. Sparse Matrix storage techniques

a) Compressed sparse row (CSR or Yale format)

The compressed sparse row (CSR) or Yale format represents a sparse matrix by three one-dimensional arrays that respectively contain nonzero values of the count of occurrences of words in a document, the document index, and the word id index in K [1]. This is the default representation used to generate the TFIDF in the R and the Python programming languages.

b) Coordinate list (COO)

COO stores a list of (row, column, value) tuples representing the document number, word index in K and number of occurrences in that document, respectively. The entries are sorted first by row index and then by column index, to improve access times.

c) List of lists (LIL)

LIL stores the data as group of lists arranged as an array: one list per document, with each entry containing the column index which references the word id in the word set array (K), and the number of times that word is found in this document. Typically, these entries are kept sorted by column index for faster lookup. This is the most compact representation, as the document number is not stored for each word.

The CSR is the typical representation used in TFIDF calculations. In this work, we implement and test the three representations for the calculation of the TFIDF to compare the performance in time and space. The LIL representation takes the least amount of space as there is no need to add the row number to each data point, providing a saving of about 30% over the CSR & COO and enabling the processing of larger datasets. Algorithm 1 shows our proposed algorithm to get the TFIDF based on the LIL data structure. The code opens the text file containing the documents, one per line, loads them into the data structures, and then calculates the TFIDF for all words in the corpus.

Algorithm 1: Proposed TFIDF calculation using the LIL representation

```
1       Initiate  lists:  K, W, TF and DF
2       // First: read data from input file and  get the term frequencies
3       r = 0   // data row counter
4       while fileinput != EOL
5           Initiate list W
6           Read one line of data and save it in W
7           for ( i=0; i<W.size();i++)
8                 Read one word w from row r
9                 if  w not in K
10                    Add w to list K
11                if w not in W
12                    Add w to list W
12                    Create tuple (K.get[w], 1) and insert into row r of TF
14                else
15                    Go back in TR until you find the column with that keyword
16                    Add one to its value
17            r++
```

```
18      // Next: get DF and calculate TFIDF for all terms
19      for (int i=0 ; i<r ;  i++)
20          for (int j= 0; j < TR[i].size() ; j++)
21              DF[TR[i].get(j).value]++;
22      for (int i=0; i<r ; i++)
23          for (int j= 0; j < TR[i].size() ; j++)
24              TFIDF = (TF[ii].get(j).count )* log( DF.size()/ (DF[TR[i].get(j).value]);
25              // Print it out or save it in TR
```

3 Results

In order to demonstrate the validation and applicability of the proposed technique, we evaluate the performance of the system based on actual social media data. As the experimental data of performance evaluation, we use a large dataset of tweets used for sentiment analysis [4] as our test set. We used the Java programming language to implement four TFIDF calculation techniques: the original two-dimensional representation and the three sparse matrices representation. All were coded and tested using the same i5 laptop with 8G of memory to compare their memory capacity and the time-efficiency. Table 1 shows the dimensions of the data matrix, and the running time for each technique.

Table 1. Performance results for the different data structures

Number of tweets (rows)	Number of unique words (Columns)	Number of Cells	Processing Time in seconds			
			2D array	CSR	COO	LIL
10,000	14,677	131,760	45	3.15	3.18	3.22
20,000	21,156	249,344	249	6.59	8.34	8.67
100,000	53,551	1,236,186	Fail	38.55	41.16	41.71
500,000	130,015	6,241,033	Fail	242	249	249
1,000,000	202,282	12,431,720	Fail	581	609	685
2,000,000	243,486	26,224,253	Fail	2559	2749	2823
3,000,000	273,866	41,482,271	Fail	Fail	4439	4559
3,200,000	277,921	43,947,980	Fail	Fail	Fail	5036
3,400,000	283,126	46,419,738	Fail	Fail	Fail	5211

The results show two findings: All three sparse matrix representations have a significant decrease in processing time compared to using the original two-dimensional matrix. This is due to the much smaller number of values to be processed. The CSR representation has a slight advantage on speed due to its simple structure of three lists of integers, with no complex tuple structures used.

The second finding is that the LIL data structure can accommodate a bigger dataset in memory for processing. This is due to the fact that there is a significant memory reduction due to the fact that there is no need to store the row number for each word. We were able to process 3,399,946 tweets compared to 2,236,948 using the popular CSR representation or COO.

4 Conclusion

TFIDF calculation is a starting point in performing text analytics for artificial intelligence and machine learning applications. It provides a challenge in its calculation due to the need to maintain a very large sparse array of the term frequencies. Traditional software libraries use a sparse matrix representation (CSR) that is optimized to reduce time complexity, but has a limit on how many documents can be in the corpus. This is particularly a problem in social media analysis where users tend to use vastly different words and have lots of misspellings that results in a large corpus.

We present a solution to the problem by saving the data into a sparse matrix representation (LIL) that stores the document as a list of lists, where each sub-list represents one document. This resulted in a much larger capacity (52%) more than the traditional representation used (CSR) but the cost is a slight increase in processing time (12%).

References

[1] A. Buluç, J. Fineman, M. Frigo, J. Gilbert, and C. Leiserson. 2009. *Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks*. In Proceedings of the 21st annual symposium on Parallelism in algorithms and architectures (SPAA '09). ACM, New York, NY, USA, 233–244. DOI:https://doi.org/10.1145/1583991.1584053

[2] Kim, SW., Gil, JM. *Research paper classification systems based on TF-IDF and LDA schemes.* Hum. Cent. Comput. Inf. Sci. 9, 30 (2019).

[3] G. Koushik, K. Rajeswari and S. Muthusamy, *Automated Hate Speech Detection on Twitter*, 2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA), Pune, India, 2019, pp. 1-4, doi: 10.1109/ICCUBEA47591.2019.9128428.

[4] M. Michailidis, *Sentiment140 dataset with 1.6 million tweets*, http://kaggle.com/kazanova/sentiment140 , last accessed January 19th, 2021.

[5] A. Romadon, K. Lhaksmana, I. Kurniawan and D. Richasdy, *Analyzing TF-IDF and Word Embedding for Implementing Automation in Job Interview Grading*, 2020 8th International Conference on Information and Communication Technology (ICoICT), Yogyakarta, Indonesia, 2020, pp. 1-4, doi: 10.1109/ICoICT49345.2020.9166364.

[6] J. Stoer, and R. Bulirsch, (2002). *Introduction to Numerical Analysis (3rd ed.).* Berlin, New York: Springer-Verlag. ISBN 978-0-387-95452-3.

[7] C. Varol and H. Abdulhadi, *Comparision of String Matching Algorithms on Spam Email Detection*, 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), ANKARA, Turkey, 2018, pp. 6-11, doi: 10.1109/IBIGDELFT.2018.8625317.

[8] F. Yamout and R. Lakkis, *Improved TFIDF weighting techniques in document Retrieval*, 2018 Thirteenth International Conference on Digital Information Management (ICDIM), Berlin, Germany, 2018, pp. 69-73, doi: 10.1109/ICDIM.2018.8847156.

[9] H. Yuan, Y. Wang, X. Feng, and S. Sun. 2018. *Sentiment Analysis Based on Weighted Word2vec and Att-LSTM*. In Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence (CSAI '18). ACM, New York, NY, USA, 420–424. DOI:https://doi.org/10.1145/3297156.3297228

[10] G. Zhao, Y. Liu, W. Zhang, and Y. Wang. 2018. *TFIDF based Feature Words Extraction and Topic Modeling for Short Text*. In Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS 2018). ACM, New York, NY, USA, 188–191. DOI:https://doi.org/10.1145/3180374.3181354