Exploradar: User-Centric Intelligent Document Retrieval System.

Samip Kalyani^{1*}, Sanjay Nayak², and Kriti Srivastava³

^{1, 2}Dwarkadas J. Sanghvi College of Engineering,

³Dwarkadas J. Sanghvi College of Engineering

ABSTRACT

Finding a document is one of the most time consuming and laborious task for a user and could affect the overall productivity of an individual in a workforce. Even if the user finds a document by himself/herself, it is still cumbersome to remember the file location of every document present in the computer system considering their volatile nature. For instance, files may be moved from one location to another, deleted, renamed, etc. Over time, it is also probable that the user forgets the actual name of the document and the content related to the document. The abstractive text summarizer implemented in Exploradar provides a summary of user documents. Using the user query, Exploradar provides the best results of the documents present in the user system that includes the file system and databases. Further, to simplify the problem of remembering the contents in a document, Exploradar classifies documents into different domains/tags that help the user in faster document retrieval.

KEY WORDS: QUERY, DOCUMENT RETRIEVAL, DATABASE, CLASSIFICATION, ABSTRACTIVE SUMMARY, SEARCH, NLP.

INTRODUCTION

Despite the advancements in cloud technologies and internet storage-based services, users still function on documents, which are stored on his/her local machine. And while operating on these documents, the user may forget the exact location and the contained information of every document in the computer system. Further, a survey conducted by International Data Corporation, White Paper in 2012 on 1200 IT professionals and workers, concluded that individuals spent approximately five hours searching/looking for their document [Melissa

Webster, 2012]. Individuals working in a particular field deal with documents that are related to certain domains and could be clustered together for accessing quickly. Moreover, from the advent of databases, users that operate on documents regularly prefer to store documents in large databases on their machines.

In this paper, we propose a web application to provide users with a common platform that fulfills the above requirements. The user would input a search query that could be based on the content of the document or what he/she assumes is related to the document, and Exploradar would provide the relevant documents to the user. For example, the search query of the user could be “presentation on thermonuclear rate estimation.” It could be possible that the same user, instead of querying the system for documents, selects the domains in which he/she thinks the document might be related. In either scenario, the user expects retrieval of relevant documents that could be present in the file system or the database.

ARTICLE INFORMATION

*Corresponding Author: samipkalyani@gmail.com

Received 12th Oct 2020 Accepted after revision 27th Dec 2020

Print ISSN: 0974-6455 Online ISSN: 2321-4007 CODEN: BBRCBA

Thomson Reuters ISI Web of Science Clarivate Analytics USA and Crossref Indexed Journal

NAAS Journal Score 2020 (4.31)

A Society of Science and Nature Publication,
Bhopal India 2020. All rights reserved.

Online Contents Available at: <http://www.bbrc.in/>

Doi: <http://dx.doi.org/10.21786/bbrc/13.14/49>

There are five primary goals of the Exploradar. They are

1. Fetching the documents based on the query.
2. Classify the documents into several types and use these classes for querying.
3. Faster preprocessing and querying to retrieve documents quickly.
4. Enabling search on files present in databases as well as in the file system.
5. Providing an abstractive summary of each document.

In addition to the above main features, Exploradar is also implemented to be independent of the operating system on which it is installed, cross-browser compatible i.e. having the same functionality irrespective of the web browser being used. Exploradar opens the document in the native application used by the user for that document. Exploradar also takes into consideration the “human-factor” and thus provides a feature wherein the user can add a class for a document that could be used for querying the document.

Related Works: In the domain of searching a particular document from many documents, the system must rank the documents in order of their relevance to the user query. The ranking implemented by the system can be of two types: query dependent and query independent. As the name suggests, in query dependent ranking, ranking depends on the user given query whereas, in query independent ranking, ranking is based on the anchor text present in the document discounting the user given query. One of the widely used implementations in the realm of query independent ranking systems is the PageRank algorithm [Lawrence Page, 1999]. This algorithm considers a document to be important if it has many numbers of document links pointing to it. The algorithm considers PageRank of the linking documents as well as the number of outgoing links from the document. Link-based ranking approaches are the basis of many web-based search engines today but may not be relevant for user documents because users do not have documents with links pointing to other documents. Support vector machines with modified hinge loss function are also used for information retrieval [Ramesh Nallapati, 2004]. This approach views the information retrieval as a binary classification wherein a document could either be relevant or not relevant. Hence, we consider query dependent approaches for document retrieval.

A semantic search engine is also implemented based on inverse document frequency [TS Jayalakshmi, 2016]. In this approach, an index consists of pointers to each word in the documents. However, this approach would not consider the scenarios in which words, when combined, have an interpretable meaning. For instance, in The White House, individually, “white” and “house” would have a different meaning. But when combined, they form an entity of single interpretation. An SVM-based one-class document classifier is implemented on the standard Reuters data set [Larry Manevitz, 2010]. Many machine

learning methodologies have been dependent on SVMs for multi-class classification. However, the performance of SVMs decreases as the number of labels and documents increases. A naïve Bayes classifier for document classification and achieves quality results among the other contemporary classifiers like neural networks, SVMs, and decision trees [S.L. Ting, 2011]. However, this implementation demands to identify features from the important document. A hierarchical attention network-based model that imitates the hierarchical structure of the document, as well as an attention mechanism that is applied on word and sentence level, is also used for document classification [Zichao Yang, 2016].

Various methods have been implemented for extracting text from the document to provide an overall summary. Supervised as well as unsupervised approaches have been used for extractive summarization. In unsupervised approaches, a graph-based method that understands the structure of the text in the document and then ranks important sentences selected from the document is used for document summarization [Jon Kleinberg, 1999]. Next, a fuzzy logic-based method wherein features like sentence length, sentence position, sentence similarity is considered, but the expressions which depend on other expressions are not considered is implemented for document summarization [L Suanmali, 2009].

Further, in unsupervised approaches, we have concept-based summarization wherein important concepts are extracted from an external knowledge base. A concept model that uses Wikipedia as an external knowledge base is used for summarization [Krishnan Ramanathan, 2009]. Lastly, in unsupervised approaches, we have Latent Semantic Analysis, which extracts the hidden semantic structure of sentences. In supervised learning approaches, a Bayesian methodology for automatic text summarization instead of using the conventional heuristic function [JL Neto, 2002]. RankNet algorithm proposed trains labeled data on a two-layer neural network and then extracts features from the test set as well as the training set [Krysta Svore, 2007]. Conditional random fields are used for feature extraction of each sentence and text summarization as a sequence labeling task [Dou Shen, 2007]. In the above two approaches under extractive text summarization, sentences are extracted from the text in the document rather than being generated.

Various structured based and semantic-based approaches are implemented for abstractive text summarization. A multi-document summary is generated using extraction templates [Sanda Harabagiu, 2002]. A syntax-based summarization approach is also proposed wherein for a sentence, common phrases are identified, which are then substituted and inserted by different phrases in such a way that the syntactic meaning of the sentence does not change [H Tanaka, 2009]. Again, such an approach uses a predefined corpus of words for substitution. A novel method of abstractive summarization is also proposed wherein, instead of generating a summary from sentences, it converts the document to an abstract

representation that is based on the concept of information items (INIT) [Pierre-Etienne Genest, 2011]. However, most INIT systems do not generate full sentences. Thus, an NLP based sequence to sequence encoder-decoder network is implemented to provide a qualitative summary.

MATERIAL AND METHODS

The process of searching starts with exploring every file present in the directories chosen by the user. An overview of the flow of Exploradar is shown in figure 1. First, when the user opens Exploradar, a prompt is shown wherein the user can enter the directories where the Exploradar should look for documents. Next, Exploradar begins preprocessing every file in the database and generates tags, summaries, indexes that are stored in the data frame. This is a one-time process after which Exploradar looks only for the updated files and directories. Then, the user is prompted with the main search page where he/she can enter his/her query. Exploradar takes this user query and processes it's searching algorithm (discussed later). The documents are then shown to the user on the screen where he/she opens the document in his/her native application.

A. Data collection and preprocessing: The contents of all the files across the selected directories/subdirectories as well as databases are read and then cleaned using multiple text-processing paradigms which include but are not limited to stemming of complex and non-singular forms of words to their basic (root) form, lemmatization of rarely used words to their simple synonyms. Punctuations and all less-meaningful words, also known as stopwords, are removed to enrich the quality cum meaning of the collected content.

A mix of indexing and phrasing is used to structurally store the collected data and enhance the process of searching in the later stages. For every file that has been processed, a set of all unique words and their frequencies are noted. The words are used as keys and their values are maintained as a list of all the files having their presence and their corresponding frequencies. The idea is to calculate the term frequency of each word and its corresponding inverse document frequency. Term frequency is the number of times a word occurs in a document. Inverse Document Frequency is the mathematical inverse of the number of documents that a

word is present in. The value obtained is then normalized to reduce the calculation time and complexity. To improve the results on phrase-based searches, the concept of n-grams is used, which is a continuous chain of 'n' words from a given piece of text. Mending all these calculations, a file vector is created, which is the vector space representation for that file.

B. Automatic Document Tagging: An unsupervised approach is used to assign relevant tags to every document based on its content. The first step is to parse and assign a POS tag (noun, verb, adjective, etc.) to every word by using a statistical model. Words having strictly 'nouns' as their POS tag are taken forward and if the condition (number of word cross threshold) arises, then all general and abstract nouns are discarded. The process of tagging now becomes a competitive process wherein, after every round of processing, words get eliminated. At a higher level of abstraction, this competitive process is simply a counter of their occurrences along with their uniqueness in the processed chunk of text. The terms that can cross these barriers i.e. the last few remaining words become the actual tags for that file.

C. Abstractive Summarization: A summary (basically a one-liner) of the contents from the file is shown to the user to help him choose the correct file without opening the file. Summarizing of the text-content is abstractive, unlike generic summarizers (extractive), which just pick high weighted sentences from the file. One of the concepts from deep learning is used to build the model required for summary creation. A Seq2Seq (Recurrent Neural Network) model is used to preserve the order in which the contents are stored in the file (sequential information). It is to be noted that gated recurrent neural networks provided commendable results on files of shorter lengths, but when large files were fed as input, Gated RNNs underperformed as they were unable to memorize long term dependencies.

Hence, the Long-Short Term Memory model, which takes into consideration the long-term dependencies overcoming the generic problem of vanishing gradient, is used for the seq2seq model; this involves the use of LSTMs in both the encoder and decoder of the model. The input to this model is the cleaned content of the file and the output is a short and meaningful summary for that file. The abstractive text summarization model has been trained on the amazon fine food reviews dataset that consisted of 568,454 mixed reviews given by 256,059 users. The technical architecture of the abstractive summarization model is shown in figure 2. The series of words from each document is given as input to this seq2seq model. Every word from the document is given as input to the encoder. The output of the LSTM unit is a hidden state, which is then given as input to the next LSTM unit. This individual LSTM unit at each time stamp processes the text and understands the conceptual information. The final state of the encoder is given as input to the decoder, which then predicts the next word in the summary based on the previous words.

Figure 2: Seq2Seq Model

Algorithm retriever_and_summarizer
(input: query, output: ranked files)

1. Read the contents of the file and apply cleaning algorithms
2. Use lemmatization, stemming, etc., to extract meaningful words and store them in a data frame.
3. Compute the TF-IDF score for every document that is present locally.
4. Form N-grams with the threshold 'n' set as 3.
5. Create file/document vectors and store them in the data frame, which is used later.
6. Parse and identify POS tags for every word in the document.
7. Except for proper and general nouns, remove all other words, & calculate their frequency.
8. Compete amongst available tags and finalize the top words as tags.
9. Store these tags in the data frame.
10. Feed the content of every file to a sequence to sequence model (with LSTMs as encoders and decoders).
11. Store the obtained summary in the data frame.
12. Accept user input as free text and create query vectors as done for the creation of file vectors.
13. Rank the documents by calculating the cosine similarity between the query vector and the document vectors.
14. Display the results to the user.

The above algorithm gives the flow of the processing done by Exploradar. Beginning with preprocessing, to building vectors, generating tags, summarizing the file content, identifying the most pertinent tags, and finally, servicing the user query with the most relevant documents.

Table 1. Comparison of processing time by document type and size

Exploradar	Primary size (Time in sec)	Twice primary size (Time in sec)
PDF	21.66	40.33
WORD	30.33	59.24
TXT	15.23	29.99
PPTX	22.31	36.77

RESULTS AND DISCUSSION

A windows-based computer was used to test Exploradar across files of various types and sizes. This process was repeated on Ubuntu OS and similar results were obtained on both the systems. The algorithm took around a minute to read the sample set of files, following which testing was done with smaller sized files and the time was

D. Text-based Querying: A text input (generally a free text query) is taken from the user and this query text vectorized on similar grounds is used for creating file vectors in the preprocessing step. For ranking the documents, a similarity measure called cosine similarity is used, which is predominantly seen in text-based and NLP-related operations. This involves matching the query vector with each file vector and ranking them based on their metric-values in non-increasing order.

calculated for a few sets e.g. PDFs of size 12.6 MB took around 21 seconds whereas PDFs of double the same size took around 40 seconds, a similar result was seen when it came to processing WORD files wherein 3.2 MB of files took 30 seconds for preprocessing whereas 7 MB of files took around a minute. Text files were the fastest to get processed; 10 MB of files took around 15 seconds and 20 MB of files took half a minute.

Figure 4: Loss vs epoch for Seq2Seq Model

Similar tests were carried on with ppt files and odt files, following which it can be deduced that the time for processing the files increased linearly with their size.

These results are summarized in figure 3 and table 1. It should be noted that the mentioned processing time is only valid for the first appearance of any file, successive running of the server wouldn't need the processing time as the content is already processed and stored in a pickle file. With an early stopping criterion of 10 epochs, good results were obtained from the summarization model, the training curve of which is shown in figure 4. The mentioned early stopping criteria were finalized after observing the curve crossover on datasets of various sizes. The summarizer produced commendable results for pdfs, word files, and text documents.

Figure 4: Loss vs epoch for Seq2Seq Model

CONCLUSION

Exploradar can be used by anyone and everyone who uses a computer to store his/her documents. Exploradar allows flexible search based on keywords and/or tags along with a summary of the contents present inside the file. It can be used as an offline search engine for local documents stored in directories and/or databases. Thus, Exploradar plans to improve and enhance the user experience when it comes to finding the document. In the future, the scope of Exploradar can be expanded to perform a search on the documents stored on the cloud and display results on a client software (web-based or android/ios or standalone). A personalized touch can be added to Exploradar by tracking the user queries and suggesting related files (content-based recommendation). Support for image files can be provided; this is possible with the help of a computer vision model for recognizing objects in an image that can be labeled to be used as tags for a particular image, thus expanding the reach of Exploradar.

REFERENCES

- Jayalakshmi, T.S. and Chethana, C., 2016. A semantic search engine for indexing and retrieval of relevant text documents.
- Kleinberg, J.M., 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5), pp.604-632.
- Manevitz, L.M. and Yousef, M., 2001. One-class SVMs

- for document classification. *Journal of machine Learning research*, 2(Dec), pp.139-154.
- Page, L., Brin, S., Motwani, R. and Winograd, T., 1999. The PageRank citation ranking: Bringing order to the web. *Stanford InfoLab*.
- Ramanathan, K., Sankarasubramaniam, Y., Mathur, N. and Gupta, A., 2009. Document summarization using Wikipedia. In *Proceedings of the first international conference on intelligent human computer interaction* (pp. 254-260). Springer, New Delhi.
- Suanmali, L., Salim, N. and Binwahlan, M.S., 2009. Fuzzy logic based method for improving text summarization. *arXiv preprint arXiv:0906.4690*.
- Tanaka, H., Kinoshita, A., Kobayakawa, T., Kumano, T. and Kato, N., 2009, August. Syntax-driven sentence revision for broadcast news summarization. In *Proceedings of the 2009 Workshop on Language Generation and Summarisation (UCNLG+ Sum 2009)* (pp. 39-47).
- Ting SL, Ip WH, Tsang AH. Is Naive Bayes a good classifier for document classification. *International Journal of Software Engineering and Its Applications*. 2011 Jul 3;5(3):37-46.
- Webster, M., 2012. Bridging the information worker productivity gap: New challenges and opportunities for IT. *IDC Information and Data*.
- Yang, Z., Yang, D., Dyer, C., He, X., Smola, A. and Hovy, E., 2016, June. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies* (pp. 1480-1489).
- Harabagiu, S.M. and Lacatusu, F., 2002, July. Generating single and multi-document summaries with gistexter. In *Document Understanding Conferences* (pp. 11-12).
- Neto, J.L., Freitas, A.A. and Kaestner, C.A., 2002, November. Automatic text summarization using a machine learning approach. In *Brazilian symposium on artificial intelligence* (pp. 205-215). Springer, Berlin, Heidelberg.
- Nallapati, R., 2004, July. Discriminative models for information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 64-71).
- Shen, D., Sun, J.T., Li, H., Yang, Q. and Chen, Z., 2007, January. Document summarization using conditional random fields. In *IJCAI (Vol. 7, pp. 2862-2867)*.
- Svore, K., Vanderwende, L. and Burges, C., 2007, June. Enhancing single-document summarization by combining RankNet and third-party sources. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)* (pp. 448-457).
- Genest, P.E. and Lapalme, G., 2011, June. Framework for abstractive summarization using text-to-text generation. In *Proceedings of the workshop on monolingual text-to-text generation* (pp. 64-73).