University of Michigan Event Search Engine

Tianying Zhang (danniez)
Jiayang Ding (vcdjy)
Cindy Yu (xinyuy)
Haijian Wu (willwuhj)
Hezheng Fan (hzfan)

Abstract

In order to facilitate student involvement in campus activities, our team is constructing an event search engine for UM Ann Arbor campus that allows users to search for activities of interest. This report includes our problem description, followed by a description of our data collection and crawler designs. We then discussed our modeling approach, including semantic query expansion, usage of the vector space model with tf-idf, BART pretrained model, and query optimization. There is also a description of the evaluation metrics we used and the results. A list of related works, some data samples and output examples are also included.

1 Project Description

1.1 Motivation and Current Question

For freshmen and transfer students who are new to campus, one of the best ways to meet new people, make friends and get to know their surroundings is to participate in campus activities. Current students can also benefit from exploring new activities and discover more aspects of campus life. However, the event searching task for campus activities is currently difficult for several reasons:

First of all, although event listing websites such as events.umich.edu, events.engin.umich.edu, and others exist, we found that information from these resources are often not integrated. In other words, students need to go to different websites to find more department-focused information, which is neither desirable nor convenient. More importantly, the more integrated platforms, such as events.umich.edu, can be improved on some of its core features. For instance, events.umich.edu is an excellent source if users would like to see all events under a specific label such as 'career/jobs', or to perform further filtering such as (being held in) 'Michigan Union' and other criteria. However, if the user instead wants to keyword search for a topic like 'psychology speaker', events.umich.edu would return a list of related event names only, and the user needs to click on each to access the event details page.

With the goal of improving student experience to search for events, we are inspired to create an activity searcher that integrates event resources on campus and performs reliable and informative query search.

1.2 Solution

The primary aim of our project is to facilitate students' campus involvement through designing a UMich Ann Arbor campus-wide "activity searcher", which will capture all kinds of activities held by student organizations, major programs, or university departments. The students can keyword search and get a list of query responses of different activities ranked by relevance, along with important event information such as where and when the event will be held.

The search engine mainly utilizes the data from a web crawler to build a database based on existing event listing websites. The engine also determines whether the database stores duplicate events by checking the event name, date and time, location, and other important information. The main model of

the search engine is a vector space model that stores relevant data using an inverted index. When a query comes in, the model semantically expands the query and calculates its similarity scores with events in the database. The content of returned events are then summarized using sentence embedding and are outputted by ranked scores. Users then have the option to provide relevance feedback to help optimize the query and retain an updated result set improved by Standard Rocchio method.

For instance, users are able to search for events by keywords such as "psychology", "cookies", "game night", etc. Users can also add more keywords such as "Ross" or "Michigan League" to filter for more specific events. Only events that happen after the user's time of search will be shown. The activity searcher will then output top N most relevant results to users. Along with the event results, there would also be some useful information integrated into each search result, including event starting time, where the event will be held, and relevant event description links or RSVP links. Users will be given an option to "get a better result" and respond with yes or no. If answered yes, the user will be asked to enter relevant event IDs (or enter -1 if nothing is relevant). An improved result set will then be displayed. Users will then be able to search for another query or exit the program.

2 Example

Query input: "music concert violin"
Results (only part of event descriptions are displayed):

```
eventID:530
Name:Tzu Kuang Tan, piano
Place:Walgreen Drama Center
Time:2022-04-13 20:00
Link: https://events.umich.edu
/event/94344

Johannes Brahms Piano Trio Op.1
No.1 - Ludwig Van Beethoven
No Image Descripiton Avaliable.
Sonata No.3 for Violin and
Piano Op.3 - Johannes Brahms...

eventID:241
Name:Briggs Competition
Place:Walgreen Drama Center
Time:2022-04-09 13:00
Link:https://events.umich.edu
/event/94055
Chamber music groups of every
kind-from string quartets, wind
quintets, piano trios, sax ensembles,
improvising groups, mixed percussion
groups, and more...

eventID:903
Name:Martin Katz, piano
Place:Earl V. Moore Building
Time:2022-04-20 20:00
Link:https://events.umich.edu
/event/93823
Music by Benjamin Britten for piano,
piano and voice, and piano and horn.
Additional performers: mezzo-soprano
Rose Mannino, soprano, Nicholas Music,
tenor Aiden Alcocer, french horn..
...

'DO YOU WANT TO GET A BETTER
RESULT (yes/no)?'
```

3 Related Work

There are some text mining techniques that have been used prevalently for event extraction. The first approach is data-driven, which would require a large text corpora. Researchers would employ quantitative methods to approximate linguistic phenomena by learning from this text corpora. The second approach is knowledge-driven. This technique requires expertise in linguistics. It requires experts to use pre-defined or discovered linguistic patterns-lexico-syntactic and lexico-semantic patterns to extract event information efficiently. This approach would require less training on large data sets, which means it saves time at this step. It also generates more specific and interpretable results. A knowledge-driven method could be particularly useful for casual users who don't want to be bothered by mathematical details. However, this method does

need researchers to have sufficient prior knowledge. Maybe researchers would employ a hybrid method by combining these two approaches, thereby obtaining advantages of each method(Hogenboom et al., 2011).

Some activities are shared in the form of posters and images. It's necessary to process keywords in the image to know more information about the activity. To solve these images, we choose the Tesseract OCR Engine (Ray Smith, 2020). The engine works in two steps. First, it recognizes the words in the image. The Line Finding and Baseline Fitting methods are often applied to clear, aligned texts with common fonts, and the Fixed Pitch Detection and Chopping and Proportional Word Finding methods are more often used for chopped words and the text with multiple spacing (Ray Smith, 2020). The engine also utilizes a classifier to determine the words after chopping joined characters or associating broken characters (Smith, 2007).

Keyword searching and identification are important to search engines in this case. A web crawler is a primary unit of such search engines for collecting enough data for training, and their optimization could have been a major aspect of improving the efficiency of search(Rajiv and Navaneethan, 2021). The search engine employs the crawler to grab the data from the internet and extract the keywords by the image process tools and pdf process tools to derive important information from event posters and documents.

The search engine employs the vector space model and consequently requires a weight scheme to build weight vectors for documents and queries. The frequently used scheme was the TF-IDF weighting scheme that uses the terms of frequency for both queries or documents for computing their similarity (Rajiv and Navaneethan, 2021). Keyword optimization provides support to an information system to discover information from the dynamic web environment. Their results showed that the method can build keyword collections with better quality(Rajiv and Navaneethan, 2021). In this way, the search engine utilizes tf-idf weighting scheme as the main weighting scheme.

Since query usually does not contain a comprehensive keyword set to extract all related events, Word Embedding is used to expand query based on semantic similarity. Word Embedding in this project is developed based on Word2Vec model(Tomas Mikolov, 2013), it is pretrained on other large corpus and uses dense vector to represent words in a statistical way. Based consine similarities, the engine extract top-k similar words to each query words in vocabulary as an expansion.

The engine outputs event summarization using BART(Mike Lewis, 2019) pretrained model. During the summarization phase, the model will first use BART tokenizer to transform raw string to BART usable tokens, then use these tokens as input to the BART model to generate summary tokens, finally transform the summary tokens back to texts that will be represented to the user.

This paper talks about various evaluation measures for search engines and separately discusses their concepts, advantages, and disadvantages for explicit and implicit metric with detailed examples. From this paper, we obtained an overview of different evaluation techniques used specifically on search engines. Given the nature of our engine, we did not consider implicit metrics since click count or session duration do not help in our situation. From the explicit metrics, it is hard to evaluate with recall for relevant results since our original problem is to find all relevant results. Considering our ability to incorporate user feedback, we eventually chose Average Precision and Normalized Discounted Cumulative Gain. (Sirotkin, 2013)

4 Data Collection and Data Samples

4.1 Data Collection Method

For our dataset, we collected a variety of University of Michigan events data. In order to collect this data, we designed a crawler to crawl all events that could be found when using the seed pages events.umich.edu and events.engin.umich.edu.

For each seed page, we manually inspected the HTML structure of the webpage to identify the HTML tags that are most commonly associated with event information present on each page. Then, for each HTML webpage, we utilized the BeautifulSoup HTML parser in order to extract event information under our predefined HTML tags. As an example, event_info.find("div", "class": "event-title") extracts the event title from events.umich.edu. For

each event, we extract its title, start time, place to be held, and its event specific link. Then we again use the BeautifulSoup HTML parser to look at the event specific page to retrieve its description. For one of the seed pages, we check the event start time against the current time to ensure expired events don't proceed further into our database, which will be described in more detail in the following section.

In addition to the information that we could extract from the HTML of the webpages, we also attempted to extract information from the image/poster on the event specific page. For this process, we utilized the image_to_string function from Python library pytesseract to extract text from images, and implemented the edit distance algorithm to correct mis-read text and ensure their readability. This information is then appended to the corresponding event description. Only some images or posters contain text information, and not all events include an image. In the latter case, there would be no proper image-to-text output.

Finally, to aggregate this data into a "data store" that can be utilized by our model, we utilized the pandas dataframe to tabularize our data and remove duplicate events. When each data entry enters the pandas dataframe, it would also be given an unique event id for modeling purposes. Also for model training purposes, recurring events with different start times are removed to avoid crowding the model output with the same event. We then combine the event data from two source pages and save them as an excel document for modeling use.

4.2 Issues During Data Collection

The first issue we encountered during data collection was the difference in formatting for source pages. While our initial design was to have one main program that crawls all source pages, each event listing website has their own style of displaying events and their own naming conventions for classes. For instance, events.umich.edu default shows all events on its main page, and the crawler can simply obtain the next page's url to crawl the following pages. However, events.engin.umich.edu default shows all events in the current month; thus the crawler needs to keep track of both the urls for the next month and the page numbers for each month to loop through all available pages. To accommodate for this difference, we eventually constructed separate crawlers for each seed page: projectcrawler.py crawls events.umich.edu, while projectcrawler2.py crawls events.engin.umich.edu and combines their output.

Another issue we encountered was datetime formatting. As previously mentioned, different source page formatting leads to a variety of representations of datetime objects. Since events.umich.edu contains considerably more data, we made its datetime representation as default and used the python datetime library to reformat data entries from the other source. Moreover, since all events of the current month are shown together in events.engin.umich.edu, we need to manually filter by date so that events that took place before the current date (date of running the crawler) are not included. As a result, we also need to rebuild the database everyday to display timely events only.

The last issue has to deal with how we extract the text from images contained on the event page to add additional information to event descriptions. At the beginning, we observed that the image text extractor generates words that are off by a couple of alphabets and are thus not meaningful. Then we attempted to autocorrect the words before adding it in our data structure. By using the edit distance algorithm, we find the closest existing English word with each word we extract from images. Then we use the closest English word to autocorrect the words we get from images and ensure information readability.

4.3 Data Sample

We utilize the event from this link (https://events.umich.edu/event/89362) as a sample. With our crawler, we can get some important information in the following.


```
{
  "@context": "https://schema.org",
  "@type": "Event",
  "name": "CJS Thursday Lecture Series
  | Literary Toilet Papers of Japan
  and Beyond",
  "startDate": "2022-04-14T12:00:00-04:00",
  "endDate": "2022-04-14T13:30:00-04:00",
  "location": {
    "@type": "Place",
    "name": "Virtual",
```

```
            "address": {
          "@type": "PostalAddress",
          "streetAddress": null,
          "addressLocality": null,
          "postalCode": null,
          "addressRegion": null,
          "addressCountry": "US"
      }
          },
}
```


Crawler output is the following, with the majority of event description omitted by '...':


```
334
CJS Thursday Lecture Series
| Literary Toilet Papers of
Japan and Beyond
2022-04-14 12:00
Virtual
https://events.umich.edu/event/89362

"Please note: Due to updated
guidance from the university
in regards to the COVID policy,
this lecture will...
No Image Description Available"
```


Each portion corresponds to event id (for modeling purposes, not retrieved from original data), title, start time, place, link, and description. As mentioned before, information extracted from event images are appended to the event description. In this case, there is no retrievable text from the event image, thus "No Image Description Available" is attached.

Some interesting data samples we found involve languages other than English. There is one sample about a theater performance that has its event description entirely in Chinese, and other attributes in English. Another sample has its description entirely in Korean. There is also one event that contains emojis in its description. Due to the nature of the vector space model, we did not remove these samples and our model is able to treat these events the same way as others, but they will not show up in the output unless the user directly searches with the correct language.

5 Approach

5.1 Method Description

The search engine first semantically expands query based on word embeddings(Tomas Mikolov, 2013) and then uses VSM with tf-idf to calculate cosine similarity scores for a query with all events stored in the database. The content of returned events are summarized using BART pretrained model(Mike Lewis, 2019), and then the engine reranks the retrieved events based on the summarizations using sentence embedding[3]. Finally, the engine outputs event IDs, names, places, time, links and summaries based on the optimized ranking. Users can provide relevance feedback to optimize the query and gain an updated result improved by Standard Rocchio Method.

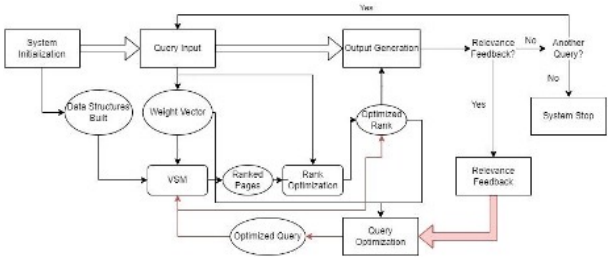

Figure 1: The logic of model

5.2 Vector Space Model

The main model used for this search engine is the vector space model with the weighting scheme tf-idf based on the literature review stated above in the related works part. From the data in the document database from the crawler, the engine obtains the page content for every document and converts it to a weight vector with the rule of tf-idf. Then, the user inputs the keywords of the event they would like to search for as the query text in the search engine, and the query text is converted into the query weight vector with the same rule. Then, the search engine utilizes the similarity score between every document weight vector and the query weight vector to determine the rank of each page.

The search engine utilizes the dictionary to store the weight vector for every document for future use and also implement an inverted index matrix to improve the efficiency of the engine by searching for the documents that contain at least one to-

ken from the query text. The length of all document weight vectors are also stored in the dictionary for normalization. The main weight scheme used by the search engine is tf-idf (tfc-tfx in TermWeightingSalton). The codes also include the options to use the weighting scheme tfc-nfx or nxx-bpx, but these two weighting schemes are not recommended for the design of search engines.

5.3 Query Semantic Expansion

During query processing, the query will be expanded based on the semantic meaning of words in the query. To calculate semantic similarity between a query word and words in database, we use the pretrained word embeddings from "word2vec-google-news-300" contributed by genism based on word2vec model(Tomas Mikolov, 2013), then use cosine similarity as similarity score. The engine uses top 5 similar words(including query word itself) to each query word as semantic query expansion. It is likely the expansion contains stopwords that we want to remove, thus after semantic query expansion the engine will filter stopwords again.

5.4 Content Summary

In order to provide users an understandable summary within a reasonable length, the engine summarizes content on the event webpage. We use pretarined BART(Mike Lewis, 2019) as backbone to perform summarization using conditional generation. To be more specific, we use "bart-large-cnn" model on Hugging Face contributed by Facebook. We set the maximum length of generated summarization to be 50.

5.5 Rank Optimization

Based on our empirical observation, the Vector Space Model that uses tf-idf and other similar weighting scheme retrieves a set of documents with high recall, however it has a poor similarity score in terms of ranking, since we want to present the most relevant event at the beginning, and so on so forth.

To solve this problem, we must design a model to perform rank optimization by rerank all documents retrieved by the previous Vector Space Model. Here, we again utilize pretrained genism word embeddings "word2vec-google-news-300" and use word embeddings to calculate cosine similarity between query and summary (generated by BART conditional generation) embeddings of an event, then rank them from high to low. The reason why we design rank optimization based on summary rather than original context is: first, most content contains a lot of stopwords, by summarization we can extract only important information; second is that normally the generated summary is significantly shorter than the original content, thus we can improve the runtime speed of our engine.

We construct query and summary embedding by the following equation: $\sum_{i=1}^{N} V_i$ where N is the total number of words in a query or a summary. V_i is embedding vector of word i.

The rank optimization performs well. Before rank optimization, the top-10 average precision is around 0.2, however after rerank all events the top-10 average precision boost to around 0.9.

5.6 Query Optimization

Users may not input all keywords accurately for the event they want to search the first time. For example, a user may want to find a research opportunity in school, but they forget to add more restrictions such as "summer" and "on campus". In this way, the results they gain may include many off-campus research opportunities and lab positions in winter and fall semesters. In this way, the search engine gives them a chance.

Another scenario is that our search engine does not return accurate results because of the model design. Many general disadvantages and limitations of the vector space model, such as losing the order of words, results of word substrings, and poor representation of the long documents, may provide users with a poor result.

As a result, in addition to inputting more information about the event and searching it again, we provide users with a choice that users can annotate the relevant pages from the top 10 results (this can be modified from the source code), and the search engine will update the query weight vector. The search engine employs the Standard Rocchio Method and sets all tunable weights to be 1 generally. The weight vectors of the annotated relevant documents are added averagely, and the weight vectors of the irrelevant documents are subtracted from the query averagely. After that, the search engine will calcu-

late the similarity scores for all documents and return the updated results to users.

The search engine does not store users' feedback in the database for future optimization because of subjectivity from different users. A good instance is that some users may believe the pages about Directed Reading Program and introductory courses about research for new undergraduates are also research opportunities, but some users may dispute this decision. Users can absolutely choose not to receive an updated result by entering "no" in the program.

6 Evaluation

To evaluate the search engine, our group utilizes some sample queries to gain the feedback and calculate the precision. The event pages in the database are not authoritatively assigned tags and categories, so there is no standard solution to the relevancy and cumulative gains for pages. As a result, the relevancy of returned pages is determined by users. To experiment with the accuracy of the engine, we invite some users to evaluate the relevance of the same query and then take the average. Admittedly, this method is strongly affected by users' subjectivity. But the search engine in the real world also depends on the people's feedback, so the extent of subjectivity is acceptable.

The sample query text is "research summer 2022". This query text is used because it includes a clear topic and a frequent time. After the search engine works it, the returned pages are sent to volunteering users, and all returned pages are annotated clearly. Every returned page is annotated by (2, 1, 0). 2 means that this page is very relevant to the query, and 1 means that this page is relevant but includes some unnecessary information, and 0 means that the page is totally irrelevant. Then, the users input relevant page ids into the search engine, and the search engine utilizes Standard Rocchino Method to update the rank of returned pages and return new pages if possible. Then the users repeat annotating each page as stated above. Then, the whole process ends. Then, we calculate the precision and other metrics for the system.

Main metrics used for this search engine are Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG). These two metrics are useful to evaluate the performance of the search engine: AP values can show us whether the returned pages are relevant enough, and NDCG values can stress the importance of ranking and especially the top pages. Because the system cannot derive the recall by lacking the total number of relevant pages, PR-curve cannot be derived. But AP can still help to evaluate whether the system returns enough relevant pages with a given number of pages.

	AP	**NDCG**
original query	0.86995	0.96647
author names	0.92398	0.89635

Table 1: Sample "research summer 2022" evaluation matrix within 20 returned pages.

Table 1 shows the evaluation matrix consisting of AP and NDCG with the sample query "research summer 2022". From the table, it's easy to notice that average precision has increased from 86.995% to 92.398%. This increase helps the users to find more relevant pages. In general, if the search engine returns 20 pages in total for users, the users can find one more relevant page after giving relevance feedback. However, NDCG decreased about 7%. The main reason is that the Standard Rocchino Method cannot trace back the exact tokens of the optimized query after the users provide relevance feedback. In this way, even if AP increases, lacking rank optimization according to the summary of the event content may result in decreasing NDCG, especially for top 5 pages, which strongly affects NDCG scores.

Search engine efficiency is also important. Without GPU running, a normal query search always costs about 6 minutes to get the output. The main time cost is for summarizing event content. With GPU running, it takes about 80 seconds to get the output because the time for summarization is reduced by the help of GPU. Time for database setting depends on the size of the database.

7 Conclusion

Overall, our search engine performed well on generating relevant events output for a user input query, from a pre-generated database of existing events. Its performance can also be improved upon receiving proper user relevance feedback. Our crawler

successfully gathers and combines information from different source pages and incorporates graphic information for available use. Building upon a traditional vector space model, our model is able to semantically expand the user query and generate a content summary, which is then used through word embedding to further optimize rankings.

The search engine successfully returns the ranked event pages including necessary information to users based on their query. In addition to employing the vector space model, the search engine semantically expands the query to include the semantically related events and optimizes ranks by comparing the original text query and the returned pages in a larger pool. The search engine also considers relevant feedback from users to improve the performance by employing the Standard Rocchino Method.

The extraction of image information from event images/posters using pytesseract was not "successful" on our first attempt. Some of the extracted words were misread, and some of them were purely unreadable. Therefore, we implemented an edit distance algorithm to correct these mis-read and unreadable words, which helped us to obtain logical results from our image text extraction effort.

Because the Standard Rocchino Method cannot reverse the weight vector of the optimized query weight vector to a string, the search engine cannot optimize the rank of the returned pages with users' relevant feedback. As a result, NDCG often decreases though Average Precision increases.

7.1 Future Work

The engine can utilize another method or algorithm to use relevance feedback, which can reverse the weight vector to the string, or the engine can develop another rank optimization method. In this way, the engine can optimize ranks of the returned pages with relevant feedback and optimized query.

In order to facilitate user experience, future work can also include a front-end platform such as a website to better present results. A more complex platform will also allow users to filter events by more criteria, for instance, a calendar can be implemented into the website in order to filter for events in a given time period.

Further improvements also include expanding the database to include more umich.edu event listing websites that were not included this time because they utilize json packages. Another way to expand the database is to consider activities that are happening in Ann Arbor but not necessarily on campus.

8 Contributions

Tianying Zhang: At the beginning of the project, along with my other team members, I worked on the conceptualization of the project, brainstorming of ideas, and helping to define basic project structure, components, and technologies. During the implementation phase, I mainly worked on the crawler and data extraction/cleaning. For the crawler, I worked together with my group to investigate University of Michigan events web pages in order to ensure that our crawler is able to extract data as accurately as possible from web pages with different HTML structures. I worked with my group on the general code implementation of the crawler and data extraction from images using pytesseract. In addition, I worked on investigating and implementing the most effective method for our team to store, clean/annotate, and transfer data extracted by the crawler. Finally, I worked with my group members on the poster, documentation, and report.

Jiayang Ding: I worked together with the team members to explore new ideas at the beginning. I participated in the team work to figure out the main project structure, and specific methods to use. I mainly took my part in working with the crawler. We defined the code structure together after exploring the potential seed pages that we were going to use. We explored the methods to extract the most and accurate information from the pages that we crawl. We implemented the basic crawler structure first. During the implementation, I also made efforts to improve the basic crawler model. I participated in working on extracting texts from images using pytesseract. To further improve on the accuracy of information we gain from the images, I worked on the algorithm to autocorrect the words we obtain from the images. I experimented with multiple distance algorithms including jaccard distance algorithm, and ended up choosing edit distance at the end for better performance. I also wrote documentation, mostly on the crawler, with my group members, like README.md, and the reports.

Cindy Yu: Besides generating ideas and deciding on project structure at the beginning, I mostly contributed to the crawler implementation. I helped with determining the structure and implementation of the first crawler. For the second crawler, I developed most of the source code for the second seed page, looked into accommodating for formatting differences between the two seed pages, and utilized python libraries to maintain datetime formats. I also explored other potential seed pages that use json objects until we made the design decision not to include them. I then worked with other team members to construct the documentation and the report.

Haijian Wu: Working with Hezheng Fan, I mainly contributed to the vector space model of the engine and the Standard Rocchino Method for query optimization. The user interaction section, including the initialization of the software, entering the query text, and providing the relevant feedback, is also designed and implemented with my efforts. In addition, Hezheng Fan and I have worked to optimize the search results by preprocessing the query text by semantically expanding the query and removing the strong relative but negatively affective words, such as time-relating tokens, from the query. We also work together for the evaluation of the model, including average precision and NDCG calculation, and improve the performance of the model.

Hezheng Fan: I worked with Haijian Wu on the retrieval, ranking, and event summarization part of the project. I mainly contributed to the design and development of semantic query expansion, event summarization, and rank optimization. For semantic query expansion, I developed a Word2Vec-based word embedding model to find semantically similar words to the query; For event summarization, I developed a conditional generation model based on BART pretrained model and utilized Facebook pretrained parameters to generate summarization; For rank optimization, based on our initial ranking results from the Vector Space Model, I proposed a reranking layer to rerank events based on event summarizations which led to a 65% mean precision boost on top 10 results. I also engineered the model to be in a GPU friendly way that led to a 20x fast inference time. Haijian and I worked on system debugging and performance analysis.

References

FP Hogenboom, Flavius Frasincar, Uzay Kaymak, and Jong FMG. 2011. An overview of event extraction from text. *CEUR Workshop Proceedings*, 779, 01.

Naman Goyal Marjan Ghazvininejad Abdelrahman Mohamed Omer Levy Ves Stoyanov Luke Zettlemoyer Mike Lewis, Yinhan Liu. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*.

S Rajiv and C Navaneethan. 2021. Keyword weight optimization using gradient strategies in event focused web crawling. *Pattern Recognition Letters*, 142(1):3–10.

Pavel Sirotkin. 2013. On search evaluation metrics. 02.

R. Smith. 2007. An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633.

Kai Chen Greg S. Corrado Jeff Dean Tomas Mikolov, Ilya Sutskever. 2013. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems 26 (NIPS 2013)*.