Automating Software Tools in Consulting: Enhancing the SitScape Platform with OpenSearch Advanced Text Search

CS4991 Capstone Report, 2024

Ethan C. Buckner
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ecb8pw@virginia.edu

ABSTRACT
SitScape offers custom solutions for businesses and organizations that integrate and automate many software tools. During my internship, I worked on a team to add advanced text search of document databases to the SitScape STP (straight through process) automation platform. We decided to use Amazon's OpenSearch tool to add this feature and worked in agile cycles throughout development. To offer semantic search, we tested many word embedding models on speed and search performance, finding that the MiniLM-L6-v2 model performed the best for our use case overall. By the end of my internship, our team successfully implemented widgets into the SitScape platform for document uploading and deleting, encoding documents with word embeddings, and searching of documents on regexes or semantic meaning. We also implemented basic security features like end-to-end encryption and user accounts with passwords. Immediate future work should focus on improving the security features of our widgets by adding features like document security levels, user permissions, and documents with masked fields.

1. INTRODUCTION

SitScape is a software consulting corporation that develops client software through a two-tiered development approach. At the high level, the SitScape platform is used by application developers to produce business automation flows called Straight Through Processes (STP). SitScape STPs are built like other low code automation tools like Microsoft Power Automate, using a flow chart presentation composed of software widgets that perform discrete tasks in sequence. At the lowest level, platform developers containerize software tools into widgets and integrate them into the SitScape automation platform. This involves writing backend PHP to call the software libraries and JavaScript for the widget options frontend. This two-tiered approach is motivated by the efficiency of separating concerns between application and platform developers and code reuse for common tools. Our project was for the platform side of development.

Our task was to develop a new widget for text searching into the SitScape platform. The widget was required to support conventional text queries and queries on semantic meaning. Text searching a large amount of data is a classic software engineering problem and is applicable to many business processes. Text searching is difficult to perform efficiently since search methods must either process the entire body of text linearly for each query or preprocess the text to speed up searches. This represents a tradeoff between prioritizing uploading speed and search speed.

2. RELATED WORKS

Zobel and Moffat (2006), provide an overview of the text search problem as it applied to search engines of the early 2000s. These techniques are still foundational to conventional (non-semantic) text search implementations like Apache Lucene. Zobel and Moffat explain that text searching for relevant documents differs from database records search because there are no unique keys to return or distinct boolean conditions to apply to fields. Text search must produce a list result ordered by relevance, a heuristic. This list can be of any length up to the size of the document collection since all documents have some level of relevance to the query. Adding to this complexity are issues unique to text data like very common words occurring in almost all documents, word morphology, and phrases that are sensitive to word order.

To determine relevance of a document to a query, Zobel and Moffat (2006) describe the inverted index approach that is core to efficient text search. In this approach, all documents are preprocessed on upload to add to an index file. This index file has an entry for each document, which includes a list of all words in the document and their number of occurrences. This index file can be very large, occupying a significant (20-60%) portion of the document collection's storage cost, but it contains all the data needed to determine a document's relevance to a bag-of-words query. In the relevance calculation described, documents will score highly if they contain many occurrences of terms in the query when compared to the average document in the collection. This indexing approach is much faster than a naïve linear scan to produce the same data, but trades off storage cost and processing time on upload.

Although the concept of encoding words as vectors of floating point numbers to capture their semantic meaning had been present since the 1950's, the approach received renewed attention after Mikolov's research at Microsoft and Google and the release of the word2vec embedding model. His research group at Microsoft published a conference paper that demonstrated word vectors or *embeddings*, produced via large language model training, successfully encoded grammatical concepts like number, tense, and gender (Mikolov et al., 2013b). This conclusion is substantiated by vector arithmetic results like the embedding for *queen* being very similar to the embedding for *king* plus the embedding for *woman* and minus the embedding for *man*. The paper also used similar vector distances between two word pairs with a common transformation like *man* and *woman* to *aunt* and *uncle* or *king* and *kings* to *queen* and *queens* to support their claim.

A later research team led by Mikolov at Google published a journal article that discusses the integration of the *skip-gram* approach into the embedding model and its improved results (Mikolov et al., 2013a). The end of the article details the team's tests on creating embeddings for phrases in addition to single words with promising results, although the problem of phrase identification is left unsolved. For this paper, phrase lists were produced using a naïve approach that leads to the total number of ngrams to embed growing quadratically with document length in words for an arbitrary maximum phrase length. Semantic search is the synthesis of conventional text search and word embeddings. A similar search operation is performed, but relevance is calculated with embedding vector distances rather than an inverted index.

3. PROJECT DESIGN

My team for the text search widget project was composed of four interns and a senior

engineer providing guidance. Over the duration of our summer internship, we worked in weeklong agile cycles. At the start of each week, we would meet with the SitScape CEO and CTO to determine our goal and on Friday, we would present to the entire SitScape dev team an update on our progress. Our team was able to divide our work however we wished, allowing us to specialize on tasks we were most effective at.

The first step towards adding the desired text search functionality to the SitScape platform was translating the non-technical requirements from leadership into technical specifications for a minimum viable product. This involved iteratively building a mockup user interface with feedback meetings until a final design was agreed upon. Required features for a minimum viable product included starting and stopping the document server, performing CRUD operations on documents, and issuing text search queries. It also must be scalable, supporting bulk upload natively. The final design for the text search project would be two widgets, one for conventional text searches and one for semantic searches. This is required since although user facing actions are very similar, the semantic search widget must embed documents on upload and store embeddings for each ngram. Both distinctions are too expensive if the user only requires conventional text search.

The next phase of development was focused on researching the state of the art in text search and comparing the various tools available. To efficiently use our time, we divided our team into a conventional search subteam and a semantic search subteam. Within these subteams, both interns would research different tools, building a list of notes. We then compared research notes to decide on a tool to use for the project. Among tools researched were Apache's Lucene and Solr and Amazon's OpenSearch. Lucene is most simple of the options, built on the inverted index technique discussed above. It is well suited to simpler applications or when computing power or storage capacity is limited, but more advanced tools like Solr and OpenSearch benefit from faster search times and data redundancy composing their document collections out of parallel Lucene instances. These instances, or *shards* can also be run on different computers communicating over a network. In deciding between Solr and OpenSearch, our team decided to move forward with OpenSearch because of its popularity relative to Solr, high quality documentation, and built in support for embedding models and semantic search. Machine learning models can be uploaded to and stored by OpenSearch like any other data and semantic search can easily be performed by adding a flag to the search query.

OpenSearch allows for defined workflows on document uploads, including creating word embeddings, but does not supply its own embedding model. This meant our second research phase was focused on selecting a suitable embedding model. The model must not be too large or computationally expensive to run on a local OpenSearch server, and must have acceptable embedding times at scale, and permissive usage terms. We were unable to quantify an error metric for our generated word embeddings, so testing was focused on speed, with user acceptance testing for results.

After we selected our document server and embedding model, we moved into the build phase of development. The most significant challenge of this phase was determining how to integrate the functions we implemented and tested with scripts into the large and unfamiliar SitScape codebase. Our team set up meetings with other senior developers to understand what portion of the codebase we would need to modify, focusing only on understanding what was related to our

project. We built our widgets using insecure OpenSearch over http first, then enabled encryption and https once basic features were functional.

4. RESULTS

By the end of our internship, my team finished the minimum viable product features for a conventional text search widget and a semantic search widget. These widgets hide all the implementation details of OpenSearch and abstract all document operations so they can easily be integrated with other SitScape platform widgets. The widgets support bulk insert operations, with acceptably low upload, embedding, and search times per page even for document collections with hundreds of thousands of pages. The final widgets use a secure OpenSearch instance and all operations are encrypted over https. They also support a basic username/password profile system and user whitelists.

5. CONCLUSION

My internship has prepared me for real world software engineering by giving me experience working on a large scale, production product that has users. Learning a new codebase was a unique experience that required me to improve at reading other's code and cross-referencing documentation. I also learned how to ask effective questions of experienced developers when answers couldn't be found online.

This experience was valuable because of how it differed to completing my undergraduate computer science degree. In school, your client is your professor who has technical knowledge and knows how the final product will be assembled. In real world engineering, your client does not have technical knowledge and does not know what is possible or how it can be done. This means much of the work of software engineering is in researching and determining what to do, rather than just writing code. It is also critical to be able to communicate results to an unfamiliar audience and provide an explanation for progress being slower than expected. I also learned how to divide work among a small team and maximize our individual strengths.

6. FUTURE WORK

If I was hired to continue working at SitScape to improve our project, I would first focus on improving security features like permission groups, where a user can be a member of any number of groups, and each document can restrict read access to only users in a set of allowed groups. OpenSearch also provides support for field level security which would allow only certain fields to be visible to some users. For example, this could allow low level users to see that a confidential document exists in the collection, but its contents cannot be read.

REFERENCES

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013a). *Distributed Representations of Words and Phrases and their Compositionality*. https://doi.org/10.48550/ARXIV.1310.4546

Mikolov, T., Yih, W., & Zweig, G. (2013b). Linguistic Regularities in Continuous Space Word Representations. In L. Vanderwende, H. Daumé III, & K. Kirchhoff (Eds.), *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 746–751). Association for Computational Linguistics. https://aclanthology.org/N13-1090

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys*, *38*(2), 6. https://doi.org/10.1145/1132956.1132959