Introduction to Information Retrieval - Project One

Ryan Reed
University of Southern Maine
Portland, Maine, USA

Figure 1: Gaming Development Stack Exchange logo, 2022.

ABSTRACT

Stack Exchange is a forum which fosters the discussion of a multitude of field-specific methodologies, tools, techniques, resources, and general Q/A, divided into sub-websites. Two of the more common, and simplistic, models of information retrieval are Boolean retrieval and the simple inverted index. By taking one of the child sites of Stack Exchange, we receive a collection to analyze and perform the models upon. After which, an evaluation of those models upon the collection can be performed and discussed.

CCS CONCEPTS

• **Information systems** → *Top-k retrieval in databases*; *Similarity measures*; *Presentation of retrieval results*; *Content analysis and feature selection.*

KEYWORDS

boolean retrieval, inverted index, information retrieval, IR system evaluation

1 INTRODUCTION

The sub site of Stack Exchange chosen for this project, is the Game Development Stack Exchange. In this sub-forum, the specialization of discussion is on the field of video game development; ranging from a multitude of topics such as User interface design, game engines, programming languages, and audio design.

The sub-forum was specifically selected for the purposes of this project because of its topic diversity, as well as its generalized size. The topic diversity was able to aid in the creation of a diverse query set, and the size of the sub-forum was more than needed to create a large data collection. The sub-forum can be accessed from https://gamedev.stackexchange.com/.

2 TEST COLLECTION

The collection is composed of several different XML documents containing data related to multiple site components. Particularly relevant are the components of users, comments, and primarily posts. The Post data, for instance, contains a listing for each post on the site, reporting question/answer, unique post ID, poster ID, the presence of an accepted answer, among other characteristics.

2.1 Collection Analysis

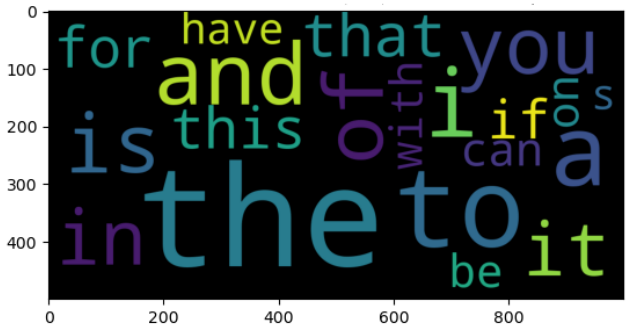

Figure 2: Word Cloud for the top-20 most common terms in the collection, stop word inclusive.

A first step in analyzing the collection, is to collect the general frequency of terms. Figures 1 and 2 show generated word clouds for the top-20 most frequent words within the post data, the later stopword exclusive. Discussion featured within the game development stack exchange is largely inclusive of programming, and so the term frequency results do coincide with what is expected. It is a reasonably large collection, and the diversity of topics means that the likelihood that specific terms will appear more frequently over

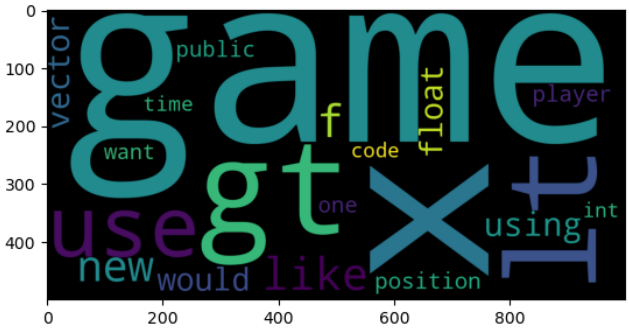

Figure 3: Word Cloud for the top-20 most common terms in the collection, stop word exclusive.

Table 1: Top 10 Most Frequently Occurring Post Tags

Tag	Count
unity	15367
c#	7939
opengl	4187
c++	4125
java	3397
2d	3361
xna	3072
collision-detection	2365
libgdx	2274
physics	2259

stopwords is extremely unlikely. Figure 3 shows the top-20 words

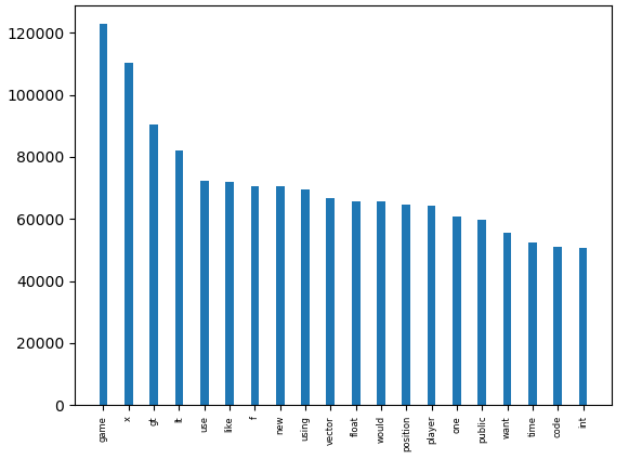

Figure 4: Frequency of top-20 words plotted, stopword exclusive.

plotted by frequency, and in combination with the aforementioned word clouds, reveal that the collection does follow Zipf's law, under the fact that stopwords make up a majority of the overall term frequency.

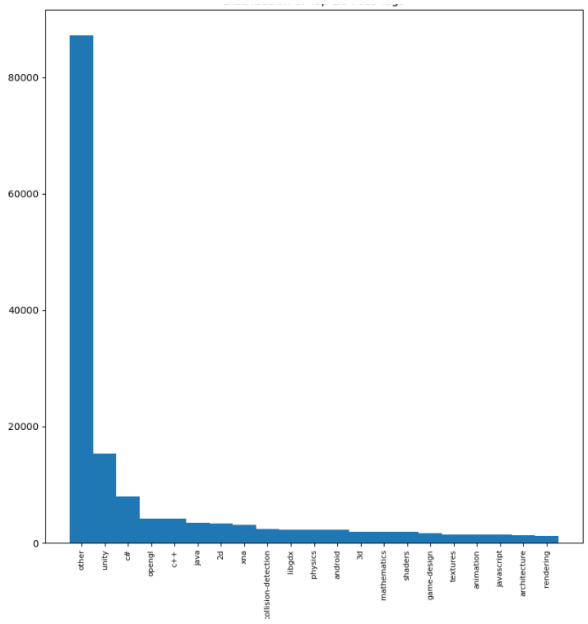

Figure 5: Distribution of top 20 tags and others.

Table 1 and figure 5 show the results of analyzing post tag counts, these are consistent with expected results on the sub-forum, and show a trend towards discussion of the unity game engine, and various programming languages typically used within the field of game development.

Computation over the collection revealed an average of 181.6 words per post, with an average of 17.8 sentences. The average number of answer posts, per question post, was 1.45 answers. Approximately 28,660 questions posted on the sub-forum had accepted answers, while 7,497 questions posted had no answers. Questions that had no answers appeared to either refer to obscure topics, or had relatively poor legibility.

Of the accepted answer posts, approximately 27,708 of them were the first answer posted in regards to a question. Calculation of the correlation coefficient between the accepted answers posted first, and the reputation score (a measure used by the sub-forum to indicate how the community measures a persons contributions), resulted in a correlation coefficient of 0.03637. This indicates that there was either a weak correlation, or no correlation. It is not always the case that the accepted answer of a question post has the highest score, an example: https://gamedev.stackexchange.com/questions/23.

Readability was measured for question posts that had no answers, and computation of the correlation of readability and the chance of receiving an answer resulted in 0.07, indicating weak to no correlation between the two values. The Flesch reading ease formula was used to measure the readability of question posts:

$$206.835 - 1.015\left(\frac{totalwords}{totalsentences}\right) - 84.6\left(\frac{totalsyllables}{totalwords}\right) \quad (1)$$

Due to differences in sub-forums on Stack Exchange, especially within the moderation of the community, there are no posts closed by moderators for being duplicates. A possible explanation is that due to the nature of the Game Development field, the moderators

might not close posts due to specific differences, even if questions are quite similar.

Comment sections beneath questions showed a pattern of Commenters recommending that the question Asker post to a more relevant Stack Exchange sub-forum, or Commenters placing their answers there instead of posting answers to the question. An example can be accessed at https://gamedev.stackexchange.com/questions/7.

Community involvement is an important value for a forum of any form, to analyze this aspect of the sub forum, personal user upvotes and downvotes, values which all user profiles have stemming from a rating system which users use, were utilized to measure the distribution of user activity. The logic being, the more a typical user posts or interacts, the more likely they are to receive these votes from other users. The Formulae used:

$$meanvotes = \frac{\sum(all\ uservotes)}{user\ count} \tag{2}$$

$$percentage\ of\ users\ below\ mean\ votes = \frac{\sum(users\ below\ mean)}{user\ count} \cdot 100\% \tag{3}$$

The computation resulted in 91.43% of users being below the mean number of upvotes and downvotes, this is an expected result. Manual analysis reveals a majority of the users on the sub-forum do not have any votes, nor do they have any history of interaction.

2.2 Query Selection

For this project, and for utilization of the information retrieval models, 20 diverse queries were selected. These queries consist of a question title from the post data of the collection.

Table 2: Query Categories

Category	Count
Game Development Resources	4
Programming Languages	3
Game Engine	3
Project Management	5
Algorithms/Debugging	3
Other	2

Table 3: Query Lengths

Word Length	Count
Short (≤7 Words)	5
Medium (7<x<9) Words)	9
Long (≥10 Words)	6

For a query set of size 20, this is a reasonable amount of topic and length diversity. An example query, "Why is it so expensive to develop an MMO", is the title to the question post accessed at https://gamedev.stackexchange.com/questions/90.

As an important note, when dual assessment of relevance judgements for a query result is mentioned, the Cohen's Kappa was used:

$$k = \frac{\sum P1 - \sum P2}{1 - \sum P2} \tag{4}$$

Where P1 is the sum of personal assessments of relevance judgements, and P2 is the sum of a peers assessment of relevance judgments.

3 BOOLEAN RETRIEVAL

The Boolean retrieval implementation is fairly simple, and divisible into three parts. The first portion of the code handles the creation of a postings file, this enables accessible postings, as well as ensures that the code runs more effectively. This code block is not useful if the postings file is already created, as it will simply recreate it. This is followed by two lines of code, loading the created tsv file into a DataFrame for manipulation purposes. Data written to the tsv file is processed before hand, the file is properly formatted for loading. The second section of code, a function named intersect, takes two terms and the associated postings lists, and intersects them appropriately based on two Boolean operations: AND, OR.The third section, a function named query_handler, handles queries by sequentially going through each term token within a given query string, and using the intersect function to compute the search. For querying, an operation must be specified, either AND or OR operations are enabled.

Before summarizing the evaluation of the system, the workstation used to create run the Boolean retrieval system was operating 64bit Windows, with 16.0 GB of RAM, and an i7-6400 Quad-core intel processor (3.40 GHz).

The first evaluation metric for the Boolean retrieval system is the time to create the postings file, which took approximately 2177s, with the average retrieval time being 58.5ms.

Evaluation was performed using the aforementioned query set of size 20. The average precision of the system at cuts 5 and 10 were 0.165 and 0.152, respectively. The average nDCG values at cuts 5 and 10 were 0.568 and 0.419, respectively. A peer-driven dual assessment was performed for the relevance judgements for one of the queries, and the kappa agreement measure result was 33.3%.

Strengths of the Boolean retrieval system included a quick retrieval time, and a near-half efficiency in the nDCG ranking measure at cut 5. The precision drop off rate between cuts 5 and 10, was also a smaller amount, implying precision consistency.

The greatest weakness of the the Boolean retrieval system stems from the time cost to construct the postings file for retrieval, with a total of 2177s, which is a significant time expense. Overall precision, while consistent at different cuts, was expressively low, reporting only a precision rate of 16% at cut 5, and 15% at cut 10.

The code is accessible at https://github.com/RyanReed12/COS470/blob/main/IR_Project1/IR_Project1_SourceCode_IRSystems.py.

4 SIMPLE INVERTED INDEX

Similar to the implementation of the Boolean retrieval system, the simple inverted index implementation creates a saved file of term indexes, with term frequency calculated for each document. The

search is performed with the term-at-a-time approach. The algorithm can be split up into three parts, the first creating the indexes file in tsv format, and performing the processing of post text, calculation of term frequency within each post, and saves this data to file. The second segment is a few lines of code to load the data from file. Finally, the third section of code performs the index searching, calculating the complete results, similar to the Boolean retrieval system, it takes a query string and performs the search.

The evaluation of the system utilized the same workstation used for the Boolean retrieval system and was operating 64bit Windows, with 16.0 GB of RAM, and an i7-6400 Quad-core intel processor (3.40 GHz).

The first evaluation metric for the simple inverted index system is the time to create the postings file, which took approximately 172s, with the average retrieval time being 33.5ms.

Evaluation was performed using the aforementioned query set of size 20. The average precision of the system at cuts 5 and 10 were 0.235 and 0.210, respectively. The average nDCG values at cuts 5 and 10 were 0.559 and 0.443, respectively. The dual assessment for the simple inverted index netted an agreement measure of 27.3%.

Strengths of the simple inverted index system included a quick index construction, a fast retrieval time, near half-rate efficiency in ranking at cuts 5 and 10.

The biggest weakness within the simple inverted index implementation was similar to the boolean retrieval system, and while higher, the inverted index implementation had low rates of precision, namely 0.235 and 0.210 at cuts 5 and 10.

The code is accessible at https://github.com/RyanReed12/COS470/blob/main/IR_Project1/IR_Project1_SourceCode_IRSystems.py.

5 RESULTS

After evaluating both systems, it is clear that the implementation of the simple inverted index was generally more efficient, and effective, in nearly all aspects then that of the Boolean retrieval system. The biggest outlier of efficiency is construction of the postings versus the construction of the index. The boolean retrieval system took approximately, 2177s, versus the time it took for the inverted index, 172s, meaning that the boolean retrieval system took an additional 2005s over the collection of data for posts. The precision rates showed an average of 0.06 improvement for the simple inverted index, over the boolean retrieval system. A seeming effectiveness trade off, between these two implementations, is that the precision drop off rate for boolean retrieval was lower over a range, while the nDCG drop off rate for the simple inverted index was lower over a range. Simple inverted index had the advantage of nearly halving the average retrieval time reported by the boolean retrieval system.

6 CONCLUSION

In conclusion, the collection analysis revealed significant details to the collection, and largely matched the expectations set prior to analyzing. Oddities existed within the results of analyzing duplicate questions, as well as the correlation of user reputation score and the chance of posting the accepted answer first. On the former, after the results, manual checking occurred and the expectations were revised to match the results upon discovery that the sub-forum moderation acts different than other Stack Exchange sub-forums, namely that duplicate posts are not closed for being duplicates. On the later, this was a relatively surprising result, but just as with the former issue, expectations had been revised following manual checking. While these oddities came at a surprise, they are correct.

As for evaluation of the information retrieval model implementations, from previous work, the simple inverted index seems to display better performance on larger collections than boolean retrieval. An interesting result of these evaluations included the boolean retrieval having similar nDCG results as the simple inverted index, considering that the simple inverted index had a ranking system while the boolean retrieval lacked this component.

My overall personal experience with this project was resoundingly positive, implementation of the systems was relatively easy, as well as debugging. The collection analysis was less easy, but the importance of this project to my learning became paramount as I had striven towards completion. I found the general evaluation of the systems to be quite enjoyable. The biggest aspect to reinforce, in terms of learning, were the diversity of topics. After completion of this project, I feel far more comfortable in explaining the models of boolean retrieval and inverted index systems. Within the collection analysis, I had to reverse engineer, as a turn of phrase, how to construct my own variants of the file parser. Building on that notion, learning how to interact with XML files and processing them correctly, was quite a new idea to me, and has given me a better outlook on dealing with web data. On a final note, writing this report familiarized me with the syntax of Latex and report-writing with the ACM format.

7 ACKNOWLEDGEMENTS

To Professor Behrooz Mansouri for two python files used to collect and gather post information, and for explaining the topics relevant to this project, and to Anthony Hollow for aiding in performing a dual assesment for relevance judgments of query results.

REFERENCES

[1] Internet Archive. 2020. *Stack Exchange Directory Listing.* Retrieved October 17, 2022 from https://archive.org/download/stackexchange
[2] W. Bruce Croft, Donald Metzler, and Trevor Strohman. 2009. *Search Engines: Information Retrieval in Practice.* Pearson Education Inc.
[3] David Harel. 1979. *First-Order Dynamic Logic.* Lecture Notes in Computer Science, Vol. 68. Springer-Verlag, New York, NY. https://doi.org/10.1007/3-540-09237-4
[4] Long T. Le, Chirag Shah, and Erik Choi. 2019. Assessing the quality of answers autonomously in community question–answering. *Springer-Verlag* 351, 367, Article 20 (Aug. 2019). https://doi.org/10.1007/s00799-019-00272-5
[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2009. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, England.
[6] Behrooz Mansouri. 2022. post_parser_record.py. Python code to interpret stack exchange post files..
[7] Behrooz Mansouri. 2022. Post.py. Python code to create a post object..

Received 16 October 2022