Stack Overflow Search

Shruthi Raghuraman

Github Source Code: https://github.ccs.neu.edu/sraghuraman1/6200-StackExchangeSearch

Introduction:

StackOverflow is an extremely valuable tool in knowledge sharing between programmers. From their website, it is an "open community for anyone who codes" that focuses on answering tech questions and much more. It mainly acts as a question and answer site for programmers and other technology enthusiasts.

As we learned this semester, community-based question answering systems provide a lot of advantages. One benefit is that users can find answers to complex or obscure questions. Since the answers are from other people and not pieced together by algorithms, the search can take advantage of an archive of previous questions and answers. However, this comes with a few complications. One of the most evident problems with these systems is that questions are not guaranteed an answer and some of those answers may be wrong.

It is interesting to note that a lot of these complications are mitigated by measures taken by the StackOverflow ecosystem. With community monitoring, StackOverflow ensures content quality by making sure that all questions on there are tightly focused on a specific issue. Any questions that are vague/non-technical are rejected and closed. Duplicate questions are linked to other posts. Lastly, answers are also monitored with comments, votes, and other discussion. These functionalities make this site quite useful and relevant to the needs of its target audience.

Even though this ecosystem helps with curating what questions and answers get published on the site, the search functionality on StackOverflow is less than ideal and ultimately holds the site back from fully utilizing the formidable aspects of its community driven ecosystem. As a user, I have experienced quite a few issues with StackOverflow's search functions. These problems make it incredibly difficult to find the right information in a timely manner. Although the dataset on the site is comprehensive, users have noticed StackOverflow's search does not retrieve relevant results on queries asked. By going through community postings on this topic, I discovered that the current ranking system of the results places extreme importance on defining relevance based on text within the title and not enough on showing answered/active questions. The following proposal improves StackOverflow's search by incorporating a new ranking and retrieval system that measures relevance based on a comprehensive and processed text snippet from a post's metadata rather than just the title.

Problem Identified:

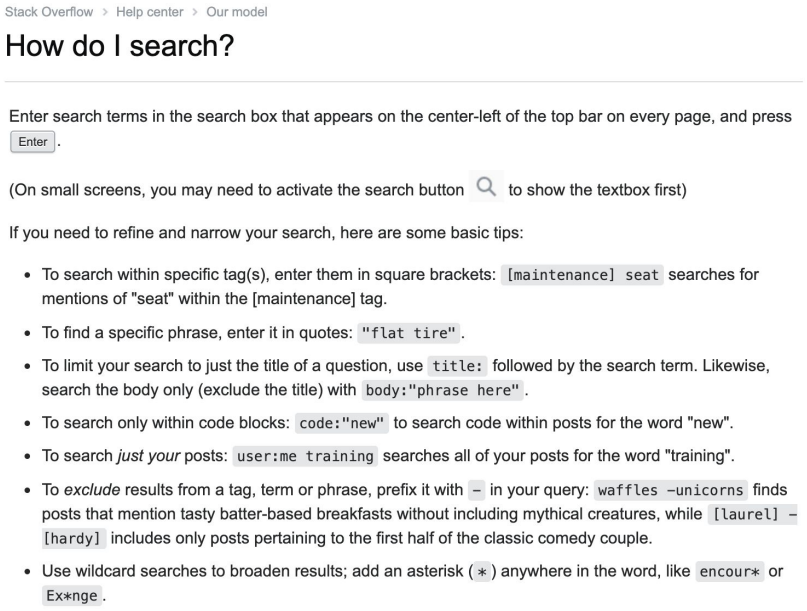
When using a search engine, users expect an interface that is easy to navigate as well as for the top ranked results of the search to give an appropriate solution to the question asked. StackOverflow has a plethora of viable solutions to many programmers' common and obscure questions. Even though the information a

user needs is typically available on the site, the website tends to fall short by giving a different and less viable solution to the question posed by the user.

Problems with the current search can be classified into two components: the user interface on StackOverflow's web page and the retrieval from the system.

1. **User Experience with Search Bar:**

   StackOverflow's basic search engine has quite a few features that are both useful and cumbersome for new users. The instructions from the website's Help Center for the basic search (not including the more verbose Advanced Search Options) are shown below.



Stack Overflow › Help center › Our model

How do I search?

Enter search terms in the search box that appears on the center-left of the top bar on every page, and press Enter.

(On small screens, you may need to activate the search button to show the textbox first)

If you need to refine and narrow your search, here are some basic tips:

- To search within specific tag(s), enter them in square brackets: `[maintenance] seat` searches for mentions of "seat" within the [maintenance] tag.
- To find a specific phrase, enter it in quotes: `"flat tire"`.
- To limit your search to just the title of a question, use `title:` followed by the search term. Likewise, search the body only (exclude the title) with `body:"phrase here"`.
- To search only within code blocks: `code:"new"` to search code within posts for the word "new".
- To search *just your* posts: `user:me training` searches all of your posts for the word "training".
- To *exclude* results from a tag, term or phrase, prefix it with `-` in your query: `waffles -unicorns` finds posts that mention tasty batter-based breakfasts without including mythical creatures, while `[laurel] -[hardy]` includes only posts pertaining to the first half of the classic comedy couple.
- Use wildcard searches to broaden results; add an asterisk (`*`) anywhere in the word, like `encour*` or `Ex*nge`.

   Although this is a well documented constraint-based search, the amount of very specific instructions acts as a very high barrier of entry when using the search engine. This has resulted in a general consensus among users to not utilize the website's search bar and, instead, use Google's search bar knowing that Stack Overflow will be a top ranked choice when asking code specific questions using Google. A question on StackExchange asking "What are some good tips for searching Stack Overflow?" from a beginner user garnered the following responses:

   - "Use Google to search Stackoverflow."
   - "You can always use Google, which is really Stack Overflow's home page."
   - "In all honesty, your best bet is to use google instead when searching."
   - "Search now *HEAVILY* weights title in the results, since people seemed to really like that approach (used on the /ask page, which searched exclusively on title alone)."

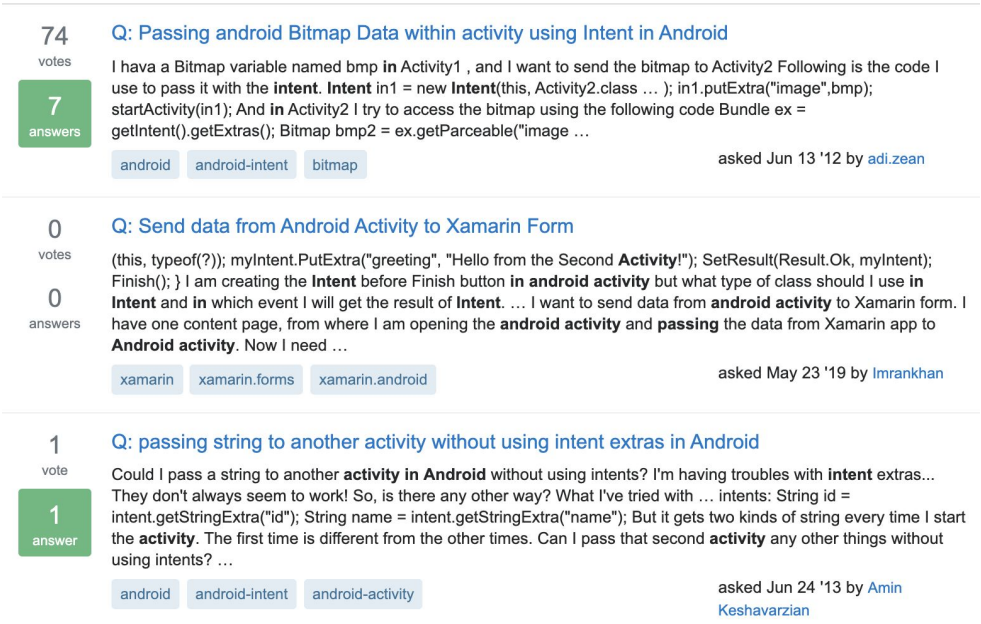
   Some users suggested a good (but roundabout) way to have the information need met would be to post a question on the site and see if it is marked as duplicate based on the title.

Although the suggestion to use Google to query StackOverflow is a solution, I believe that improving the StackOverflow search engine is beneficial to the community. From the user interface point of view, the engine could benefit from this proposal of implementing a less constraint-based search.

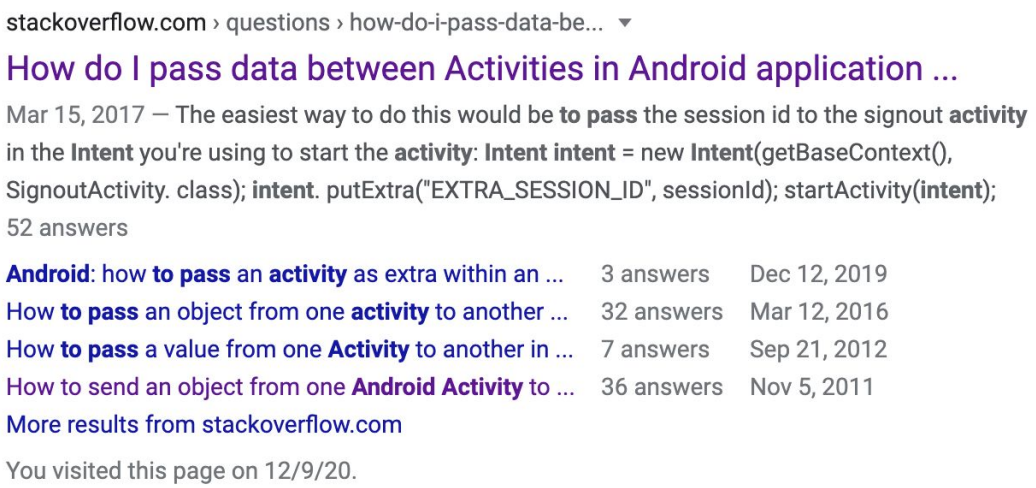
2. **Retrieval from a User Query:**

Another major problem with the current search engine is how it retrieves relevant information. For example, if I want to start learning about Android Development, and I would like to do a basic search such as "passing in android activity intent" into the StackOverflow search engine, the results are only loosely related to the topic. Although some of the words searched appear in the titles of the search results, the content of these posts is complicated and greatly deviates from the user's information need. It is frustrating that the majority of the words searched appear in the title, while the corresponding posts and answers do not satisfy the question asked.

The example for the above query is shown below:



The same query done through Google gives much more relevant results:



The retrieved information directly matches what I am looking for not only based on title but also based on phrase content within the posts. A complication brought on by a majority of the users not being able to correctly utilize the StackOverflow search is the idea of duplicates. If people are not able to find the information they need on the website, then they post a question. However, the google results show that the same questions often get posted repeatedly. Though in the above example, comments in several of the posts like them together and identify the duplication, it seems as if the community monitoring has to make up for the issues from the search.

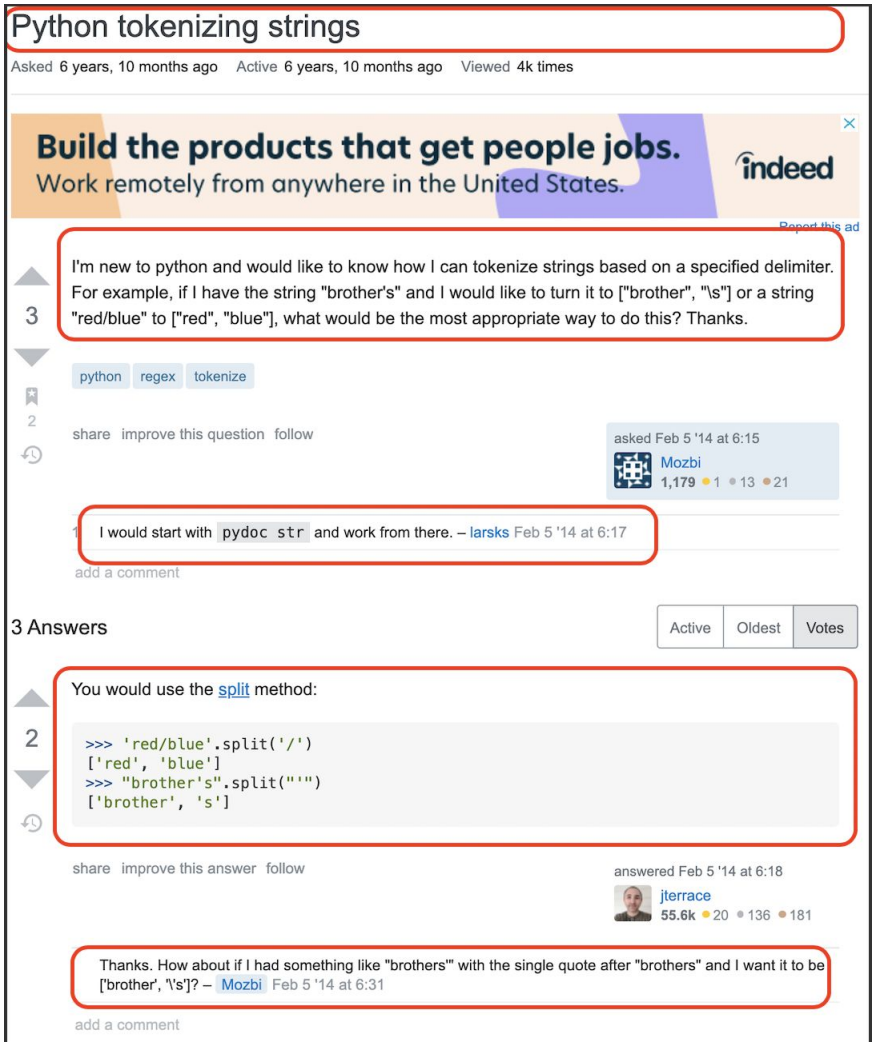
Solution Proposed:

Based on the issues identified, I believe that the relevance of the search engine output can be significantly improved if the ranking relevance decreases importance on the title and incorporates text from other metadata. In the improved ranking, the main difference is putting an emphasis on the community activity on the post rather than the post's title. The query will be processed to retrieve results that match based on phrases in the "Body", "Title", "Answer", and "Comment".

A first iteration of the proposal suggested that additional attributes be taken into account such as "CreationDate" and the "LastActivityDate". This way the ranking prioritizes the retrieval of posts created recently with the most up to date answers. However, the corresponding retrieval system prioritized recent and sparsely active posts. This occurs for the use case of StackOverflow because a recent post is often not answered and marked duplicate. Due to the technical and comprehensive question and answering system on this website, it is better to base relevance on community involvement in a post rather than it's recency.

Based on a detailed look at the dataset, this proposal will implement a search where relevance is based on the similarity between query text and title, post content, comments on posts, answers on posts, and comments on answers.

In the image below, parts of a StackOverflow post are circled with the content that was indexed for ranking:



Dataset Collected:

The data ecosystem of StackOverflow is quite large and cumbersome with over 40GB of data. The same archives (maintained by StackExchange, which oversees StackOverflow as well as a network of other webpages) however have smaller collections of data on other topics that are structured in the same way. For the scaling purposes of this project, the data utilized for the search engine came from codeReview StackExchange. The size of the data queried and indexed is approximately 1GB altogether.

The second complication in collecting the data is that it is separated into several different xml files, each of which represent a component of a post on StackExchange. I have outlined the overview of the relevant data with their corresponding files below:

- Comments.xml
    - Id

        - Text, e.g.: "@Stu Thompson: Seems possible to me - why not try it?"
- Posts.xml
    - Id
    - PostTypeId
        - 1: Question
        - 2: Answer
    - ParentID (only present if PostTypeId is 2)
    - AcceptedAnswerId (only present if PostTypeId is 1)
    - Title=
    - AnswerCount
    - CommentCount
    - Body

Other aspects of the data can be seen on the archive documentation; however, since they span over two pages in structure, for the purposes of brevity I have only included the portions of the dataset the corpus for the search index was built on.

After obtaining the data from the XML files, a basic tokenization was implemented on the “Body” of the posts. This involved removing stop-words that did not contribute heavily to the information retrieval. Punctuation nor html tags were removed due to the nature of the posts. The Code Review StackExchange supports several different languages as well as several different writing conventions to express questions. Both punctuation and non-english words play a significant role in such a technical forum.

Comments on posts and answers were also processed by removing the @ annotation on the text field in the comment. The @ with a username links to someone else in the conversation is a major writing tag on the website. However, there is already a constraint currently in the StackOverflow website that can search user content solely based on the user @ handle. Since this engine is based on the content rather than user constraints, notations with @ handles in the comments were removed.

Lastly, the data was filtered to exclude all posts that do not have a title. Not all posts contain a title. StackOverflow currently has a system that allows for users to look at duplicate posts with similar titles at time of post creation. Therefore, it is less likely that a user duplicates a post if they make sure to include a title since they will have alternate suggestions to ensure that their questions have not already been answered. The alternate suggestions also allow the user to look at other similar questions and phrase their post appropriately for the topic. As a result, for the purposes of increasing the quality of the data set used for the indexing, posts without titles were filtered out.

Implementation:

This project is a python application. The requirements.txt specifies all the libraries utilized in the application:

- Flask==1.1.2
- elasticsearch
- bs4

- bleach

All services that make up the application in order to start the ElasticSearch cluster are specified in the docker-compose.yml. The dataset is stored as xml files and then parsed into dictionaries. ElasticSearch API python client is utilized to implement the search query. Lastly a Flask server is utilized for the backend.

Backend:

Data restructuring and Index Creation:

After data collection, restructuring is done to map all the contents of the post to its corresponding post id. In the first round of indexing, each comment is processed and mapped in a dictionary to its "PostId". In the second round of indexing, the Posts.xml data is split into posts that are questions and posts that are answers (the original dataset combined both). The boolean distinction between the two types of posts is indicated by the "PostTypeId" field. If the post is a question, it is mapped to its "Id". If the post is an answer, it is mapped to its "ParentId". This way both questions and answers can be mapped to post content. The third indexing is done to "Title" If the post is a question and, if a title tag is included, then the "Title" is mapped to the "Id".

The result of the above restructuring allowed for four different indexes: comments, title, post and answers. Utilizing the id attribute of each post, a combined dictionary is created of the processed comments, title, post, and answers to allow for a nested mapping of the Id of a post to its content as defined for this proposal. A simple combined index of {'id': 'processed text of corresponding post'} really helped to simplify the data for the search query. It matched all post content and nested the titles with questions and answers and comments. This restructuring was a key component in including not just the title but also the valuable community engagement within a post relating to the title.

Search Query:

Utilizing the combined dictionary, the ElasticSearch API is utilized to store the index. The indexing here is simple due to the significant restructure and index creation of the data. The resulting index matched the id as the unique "Id" of each post and the "text" as the combined text of each post. For the query, the result is determined by directly matching the query (not necessarily as a phrase) to the processed index of each post. From the API library, the match query is able to search for analyzed tokens rather than exact terms so parts of phrases can also be matched.

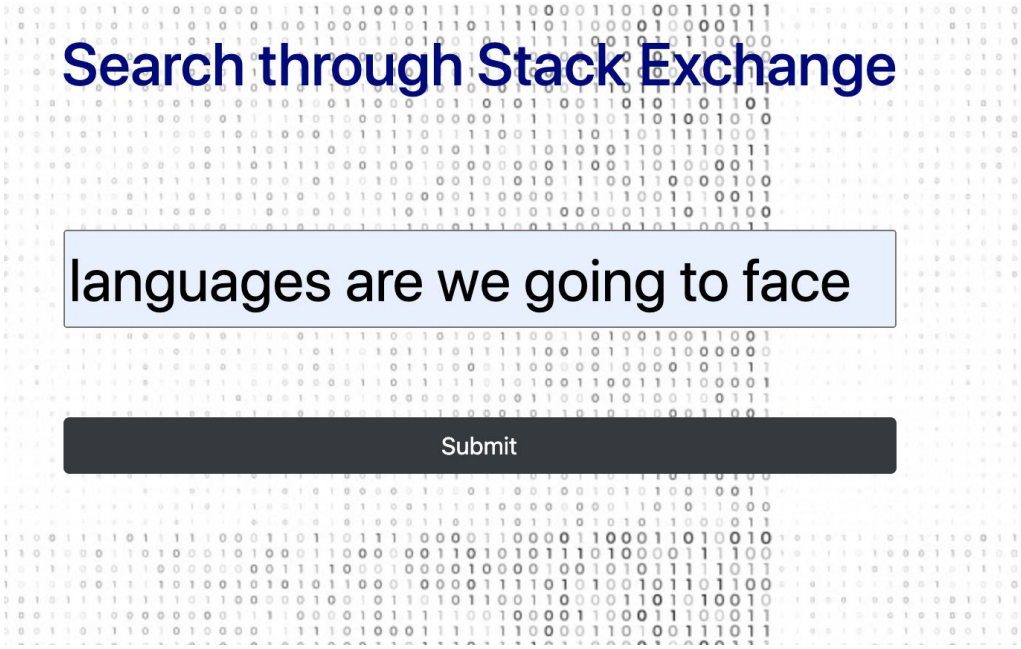
The result (result['hits']['hits']) of the match query returns the number of documents that match the given ask. The scoring for the document is based on the field match between the query and the text indexed. This is derived from Lucene's Practical Scoring function, which ElasticSearch utilizes under the hood. This scoring is comparable to the TF-IDF scoring in vector space. A future work in this component of the backend would be to redefine the scoring function to be less computationally expensive if the dataset (in the case of the original StackOverflow dataset) is too large or expensive.

The fields retrieved from the ElasticSearch query were result["hits"]["hits"]["_source"]["text"] (with the list of documents that were matched) and the result["hits"]["hits"]["_score"] (with the scoring of each document). A third field is retrieved with result["hits"]["hits"]["highlight"]["text"]. This is returned as a

customization of the query to include html tags around the phrases that the elastic search query matched. This is done after receiving feedback from the professor on the UI during the in class demo.

From the results of the search, a heap structure is utilized to rearrange the results in descending order based on the ElasticSearch score of each ranking.

User Interface:
The primary UI page is very simple and mimics the base engine format. Feedback from the demo to include a large search bar has been incorporated. This encourages users to use larger queries and not limit themselves to smaller phrases.



Once all documents are returned, additional processing is done to check if data returned contains html. At point of rendering, the UI originally had been cluttered with post content. It is very difficult to understand why something is matched if the UI displayed the whole post as text. The html tags help identify which part of the result contains the matches. However, rendering with dependency on the highlight html tags meant that other parts of the post can break the UI.

As a result, further checking is done to see if ElasticSearch matched with english phrases within the post or if it matched within the html metadata inside a post. The complicated nature of the wide variety of posts in StackExchange makes it difficult to dismiss data if matched within the html. This is especially important in terms of “tags” within posts. Since this solution allows for matching all post content, tags within posts could also be a reason for relevance. Checking if the highlighted match result is a HTML or a phrase allows for better rendering on the UI. The following images are results from searches performed during the in-class demo. The ranking function has not changed but the UI has been improved to allow for easier user understanding of why a result is matched.

1. **Knuth’s Web**

1. **Title: How should I fix a problem in my code (which is not addressed in existing answers)?**

   Go to post!

   Post Context: The reason is that I did not translate *Knuth's* algorithm correctly into Swift code, and some

2. **Title: These mysterious $A framework questions**

   Go to post!

   Post Context: to manage local and session storage for login purposes ** - Note - underscore each() will error on *web*

3. **Title: SiteSmashers - Reviewing Website designs?**

   Go to post!

   Post Context: few times pals in the JS chat over at Stack Overflow have provided some really insightful comments on *web*

4. **Title: Can I remake other people's websites using HTML/CSS (only for education purposes) and post in Code Review?**

   Go to post!

   Post Context: *Web* design is subjective.

2. elasticsearch java api

Top 10 StackExchange Results

1. **Title: Do we have some rebel tags that shouldn't exist on Code Review?**

   Go to post!

   Post Context: *elasticsearch java api* not available in preview. However, post is relevant through links and tags so proceed to site.

2. **Title: What's the [api] tag for?**

   Go to post!

   Post Context: a REST *API*.

3. **Title: Are implemented API reviews without code on-topic?**

   Go to post!

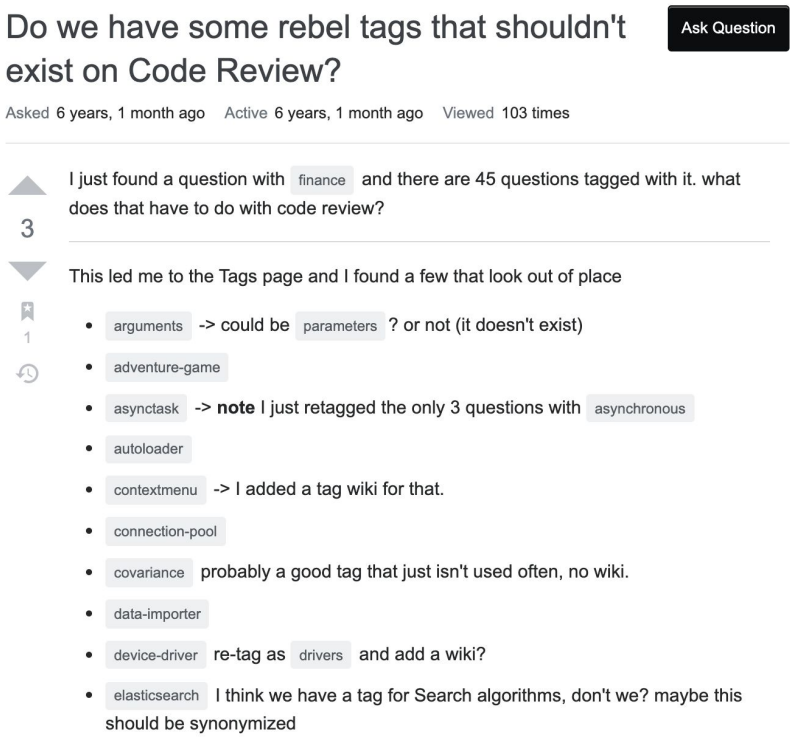
   Post Context: To be clear, I'm not looking for a comparative "is this *API* better than this other *API*" or help

4. **Title: Should we burn [Google]?**

   Go to post!

   Post Context: " class="post-tag" title="show questions tagged 'google-*api*'" rel="tag">google-*api* tag.

*Note that elasticsearch is present as a tag in this dataset. The first result leads to this post: https://codereview.meta.stackexchange.com/questions/3673/do-we-have-some-rebel-tags-that-shouldnt-exist-on-code-review



Do we have some rebel tags that shouldn't exist on Code Review?

Ask Question

Asked 6 years, 1 month ago Active 6 years, 1 month ago Viewed 103 times

3

I just found a question with `finance` and there are 45 questions tagged with it. what does that have to do with code review?

This led me to the Tags page and I found a few that look out of place

- `arguments` -> could be `parameters` ? or not (it doesn't exist)
- `adventure-game`
- `asynctask` -> **note** I just retagged the only 3 questions with `asynchronous`
- `autoloader`
- `contextmenu` -> I added a tag wiki for that.
- `connection-pool`
- `covariance` probably a good tag that just isn't used often, no wiki.
- `data-importer`
- `device-driver` re-tag as `drivers` and add a wiki?
- `elasticsearch` I think we have a tag for Search algorithms, don't we? maybe this should be synonymized

From here, clicking on the tag elasticsearch leads to all the (controversially) tagged elasticsearch posts in the site.

3. Off-topic

1. **Title: Should we have just one Meta tag for topicality?**

   Go to post!

   Post Context: a href="/questions/tagged/*off-topic*" class="post-tag" title="show questions tagged '*off-topic*&#39

2. **Title: Should answers to clearly off-topic questions be encouraged or discouraged?**

   Go to post!

   Post Context: This question is clearly *off-topic*

3. **Title: Move to SO or Programers**

   Go to post!

   Post Context: it with a quick search; But: But there must already be a question here on meta about moving *off-topic*

4. **Title: Should the FAQ on off-topic questions link to other SE sites?**

   Go to post!

   Post Context: there is a *off-topic* not available in preview. However, post is relevant through links and tags so proceed to site.

5. **Title: Gold tag-badge users closing questions as pre-written off-topic duplicates**

   Go to post!

   Post Context: Gold tag-badge users closing questions as pre-written *off-topic* duplicates

It is important to note that phrases within the metadata including post body, tags, comments, and answers are being highlighted in the match phrase. This demonstrates the original goal of making community engagement a key identifier in ranking rather than just the title.

The last feature that is key in the UI is the “Go to post!” link. This link allows for the user to easily navigate to the corresponding post. The link is created through url normalization based on the same id utilized as the key in the combined index. For example the post with an id of 10 is created by concatenating "https://codereview.meta.stackexchange.com/questions/" + 10. This is very important because it is difficult to find the exact post on the current StackOverflow Engine.

Conclusion:

The main purpose of this proposal is to improve on the current StackOverflow search engine by decreasing query matching with emphasis on the title and, instead, making community engagement the primary determinant of relevance. Community engagement here has been determined by a combined index of post body, title, answers, comments on the answers, and comments on the posts. The resulting ranking system now promotes retrieving less sparse, higher quality matches that can really benefit the user. Future work still needs to be done by exploring a larger data set and improving the relevance to better process html components and code within a post.