Codify: Code Search Engine

Dimitriy Zavelevich (zavelev2) Kirill Varhavskiy (varshav2)

Abstract:

Codify is a vertical search engine focusing on searching code and coding problems due to it's ability to search with special characters. It allows a user to put in any snippet of code and we will find the closest match with the intent of finding the problem. Users also have the ability to search a certain question as they would on stack overflow and Codify would give them any immediate results it finds. The combination of code and question would be accepted, as our program is able to separate the two. Due to the limited scope of our crawler's documents, our results are much more focused and geared towards the people who need them most - programmers.

Introduction:

Going through the CS curriculum at UIUC and interning at various companies, we have come to realize that although Google is a good resource, it does not always provide the best search results for code. The queries are stripped of symbols and the leftover syntax is hardly unique to our query, yielding irrelevant results. What we set out to do with Codify is create a search engine to aid programmers by giving them quick results with minimal effort. This framework allows for searching by code rather than just its presumable context.

With Codify, we wanted to tackle a big issue: speed and efficiency of coding help. Most programmers' go-to resource when either a problem exists in their code or they are unsure how to do something in a language, is to use Google or Stack Overflow. Although sometimes there are immediate results, it is also common that one might need to search through multiple pages to find the desired solution, or may not find an answer at all. The problem is that Stack overflow takes time in between a question being asked and when it is answered and Google is hard to deal with when using special characters along with a very small search box that is inconvenient for code; furthermore, Google also has the tendency to return many unwanted results, some of which aren't even related to code. There still does not exist a widely used code search engine and so we set out to make one.

The main problem with the most commonly used search engine right now (Google) is that it indexes a very large variety of pages and documents. This means that a simple search such as "inherit child" can return information regarding java code and inheritance, but it is more likely to return information on the inheritance of a human child. In Codify though, we would not only interpret that the question is regarding Java, but also the first result returned will deal with inheritance in Java. Also, as we mentioned before, a one line search box for queries is simply not enough to paste a couple lines of code. Lastly Google lacks the ability to search with special characters which is one of the largest deficiencies when searching code. Codify is more concentrated with a more narrow focus like most vertical search engines. We ensure that the search results are of the upmost relevance to any coder looking for a quick tip.

Our search engine also has various features catered towards better understanding user queries and tailoring results to be more accurate. Additionally, we index the most popular questions in every language we support (provided from stack overflow) and when the user

searches that they want, we assume that it will be one of the more popular or common searches. This way we are likely to meet the needs of a majority of our users since it is likely that a given person will have a more “common” question. We can also learn the language a person uses simply based on their search via a semantic meaning voting system. Furthermore, a given query does not have to match a substring of a document exactly, rather it only looks for the most “similar” documents.

The novelty of our system as opposed to others lies in the fact that unlike many other search engines, ours is built specifically with the idea that most users will be searching specific code or code-related problems. In doing this, we make sure that our system is able to read all normal characters along with special characters. Also, we ensure that it is fast and efficient so that the user can receive immediate results. This should be useful considering they come from the most popular Stack Overflow questions. It is also important to note that we do not require the user to separate code from questions as we are able to find the language they search for without them having to specify it; combining code with questions in the same search box does not modify the system’s ability to find the most correct solution.

We took a careful approach to the project and started planning out the high level ideas before coding it up. What we needed was a robust platform for searching code, and we’re lucky enough to have a website that can provide us with the relevant documents we want to index. Stack Overflow is a website that is often used to search for programming questions and explanations, so our plan was to index its questions as separate documents. We crawled three languages: Java, Javascript, and Python, gathering thousands of Stack Overflow questions regarding the language. What we took from the Stack Overflow page was the link of the page, the question being asked, the most relevant tag, and the best answer given. Once we collected about 3,500 documents for every language, we set to index the documents for rapid searching. Our inverted index is implemented in a sqlite database with a python indexing library called Whoosh. This database allows us to quickly reference our data when searching. We built our website on top of the Django framework which enabled us to use our python codes to their full extent.

Related Work:

Some related works are Google (which searches anything as we all know), Stack overflow (which will post any user’s question to a public forum), Symbolhound (a search engine like Google, while not a robust, that also understands special characters), and Ohloh Code (a code search engine). When comparing Codify to Google, we see that we have a more precise range of questions and a more code-friendly search environment while also having the ability to read all special characters. When compared to Stack Overflow, we are able to get immediate results without the need of waiting on another person to answer it. Furthermore, we provide a more robust search. A simple search query such as “for(var a=1;) {a++;}” will return no results in stack overflow, but will return multiple results in Codify, the first of which dealing with termination of for loops in Javascript. Putting Codify against Symbolhound, we notice the same advantages as we do against Google except that symbolhound does understand special characters. Finally, when compared to Ohloh Code we focus more on the exact search rather than function names or language specific syntax; We also look at real answered results rather than a large quantity of open-source code.

Problem Definition:

Our first problem was using all new technologies. We have both used python in the past and being fans of it's possibilities and simplicity, we wanted to implement everything we can in the language. Therefore, we used many new platforms and having to learn and adjust to each one took a while as we ran into many different problems. Also, once we finished the basic implementation, we wanted to use a more efficient search function. We started with just using BM-25 but we wanted something more efficient than that and more tailored towards our search. Since we store the title, question, most relevant tag, and answer of the top results from stack overflow per language, we changed the weighting of each of those. There were also ideas we wanted to implement but realized would be very inefficient. The first ideas that came to mind was to remove stop words and do stemming before comparing, but that wouldn't work as the user might enter code which is very specific and needs stem words regarding syntax like "for" or "if." We also wanted to try to implement relevance feedback but found there was no directly logical way for it to be done without having to index extremely large amounts of data. Since we can't find most similar queries (due to the fact that two similar pieces of code may not look similar at all), we thought to look at the group of results that is returned and compare it to another similar group of results so that we can adjust weights based on clickthroughs. Unfortunately though, this would take a lot of extra room on the system and did not improve our results after testing so it was later removed.

Although the more detailed weighting scheme is provided below in Methods, the computation problem is as follows. We use BM25 to rank the results, however, we weigh different aspects of the document in order to yield better results. We get results from querying just the title of the document, just the question, just the tag, and just the answer, and we combine them with various weights to yield the final rankings. The search for the ideal weights though was a tricky one. We ran the code using different algorithms to display different results in parallel in order to choose the most optimal options.

The output to any query (preferably code) will consist of our personalized layout of the stackoverflow questions which you can click on to show the best answer to that question along with a button to take you to the actual StackOverflow page. For your given search, you will see the most similar questions which will hopefully clear up whatever problems you might have. We use bootstrap on the front end as we had a goal of simplicity and efficiency when it came to displaying results. We will also return the language or languages that we interpreted from the user's search queries (i.e. Python, Java, JavaScript) as determined by the voting algorithm and based off of most relevant tags.

In order to access Codify, users need nothing more than a simple web browser once our code is uploaded and hosted online. On our end we need Whoosh along with it's dependencies, Django, Scrapy, and Python. Whoosh is an indexing library, Django is a web framework, and Scrapy is a widely used web scraper tool.

Methods:

We narrowed down the scope of our search such that we mine only the relevant sources and show the best results. We accomplish this task with different ranking algorithms and our ability to identify the language which you are searching for. Our system is intelligent enough to

detect the programming language based on its top search results via a voting algorithm, and adjust the results accordingly. This voting algorithm looks at your results in order and lets the top results “vote” on their language and eventually it sees which language is dominant. Since each crawled page stores the most relevant tag from Stack Overflow (the one pertaining to the language), each result will vote for their tag. To implement a vote, we create a hash table which continually increments the value for the tag key. Highest results have more voting value and it decreases as the rank of a result decreases. We do not eliminate answers relating to other coding languages, however, we do give them less precedence after rearranging results.

Furthermore, in order to give only the best and most relevant results for a search query when putting it in BM-25 we created our own weighting scheme. Since for any Stack Overflow page we store a title, question, tag, and answer, we realized the following method would yield the best results. We looked at the title like the anchor text and gave it the largest weighting as it says most about the question, that weight being 3. Then we look at the question which has the second highest weighting, since a similar question implies a desired answer and give it a weight of 2. Lastly, the answer and the tag have the lowest weighting of the 1 since tags are very common throughout results and good answers may often not share many words with the question. After running pages through each of these weights, we combine results and recompute to see the final weighted results. Also out of all the results we have returned, we discard of any possible results after the first 60 as we assume the user will not want to be going through pages of results like we learned in class. Codify allows the user to combine code with english questions within the same textbox because we do not check for exact matches but rather look at BM-25 results after running queries. So this means if code had a strong match, but not the question itself, we will not ignore that result.

Ranking was an interesting problem to solve because of the abundance of information we could have gotten from a webpage. Luckily, we figured out what the most helpful information on the page is, and that took us to our next dilemma, how to sort our results. Adjusting the sorting algorithm has given us a way of focusing on the most relevant queries while leveraging all the meta data that we have gathered about the site.

However, our project hit several bumps on the way. We had a lot of challenges with the various frameworks that we used, in particular, the Django web framework. This prevented us from making the results more cohesive by pulling data from StackOverflow when the searches would be loaded. Due to some of security constraints of Django, we were not able to fully implement it. Eventually, we overcame this problem by storing the answers in the database and displaying them on demand.

Evaluation/Sample Results:

In terms of speed, our solution is lighting fast, but what’s even more impressive is that it not only provides relevant results, but it also retrieves multiple answers for your question. This reduces the time you have to attribute to searching for your answer by providing all the information you need at a moment’s notice.

The experiments we have done to quantitatively evaluate our methods have been fairly direct as our index is constrained in order to save room on the system. We viewed this project as a small scale application that can be easily expanded at any time. The small scale was achievable as our code was written in a way that our index would be limited by how large of a

size it was allowed to store. In order to test, we only allowed the index to store a few megabytes of data. Due to this small scale it was logical to search queries and compare the given results to the results that we expected. Along with these tests, we also would compare our search versus stack overflow and google search to see how relevant our results are versus theirs. This was a good measure considering we create our index via stack overflow questions/answers. As given in multiple examples above, we saw many queries that returned more relevant results that both Google and Stack Overflow.

Most users of our system have praised its simplicity and conciseness but mentioned that its small scope of supported languages and broadness of questions hindered a problem. This is good news because the way we have written our code makes it fairly easy to expand it. More languages can be added by just changing a single line of code (i.e. dealing with what languages to crawl from the most popular results) and more questions can be added by increasing the storage capacity of the index. Being that this is all hosted locally on a single computer, we didn't want to clutter it with too much data. The search simplicity and conciseness is what we are going for with this project and we feel that it was achieved.

At the same time, the small index we intentionally set up also hurt us. Unfortunately, to have a better index, we need more pages to crawl, and we are currently limited by our 10,000 documents because it does not cover nearly as much coding questions as it should. Additionally,as we mentioned before, we would like to add more languages to the mix so that this search can be universal and not only limited to Java, Javascript, and Python. This are both possibilities that can be expanded easily on a larger system.

Conclusions & Future Work:

Working on this project, we learned a variety of things, from crawling websites, to indexing documents. The class has given us glimpses into how a search engine works, but this has given us insight into how we can make one ourselves. There are so many helpful resources and libraries out there that aid in this difficult task, and luckily we had python libraries to assist in our journey. The most interesting part of this experience was fiddling with how we crawl websites and how we display the results. During our crawling, StackOverflow occasionally throttled our signals giving us error codes when we mined their sites too often, so implementing and using autothrottling was an interesting approach to a previously unknown problem to us.

If we continue this project, it can redefine how we search for our code. Instead of hoping that Google or Bing doesn't cut off all the syntactic sugar from your code query, you are free to input your entire code into the search box. This project can grow into something that improves workflow and avoids vague search results. Additionally, this system would benefit from a feedback algorithm that can improve the search results overtime.

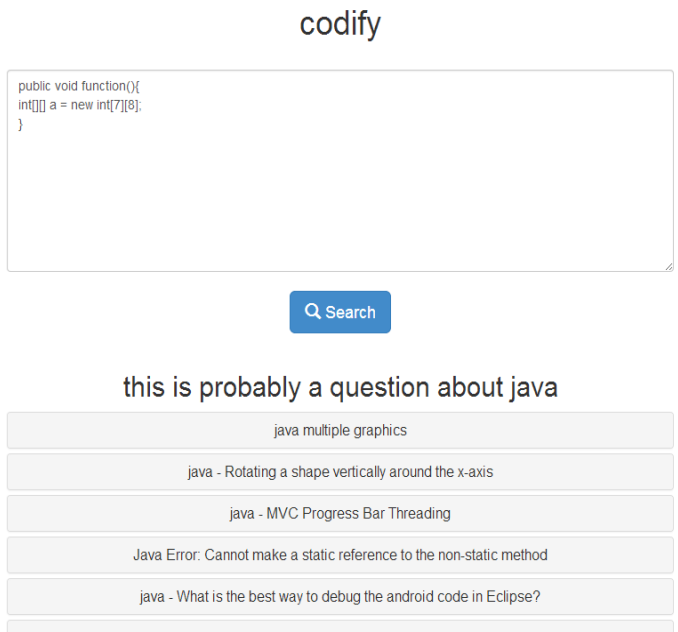
Appendix:

Kirill: researched frameworks (mainly scrapy and whoosh), crawled sites, did indexer, worked on improving frontend, worked on the main search functionality, worked on essay

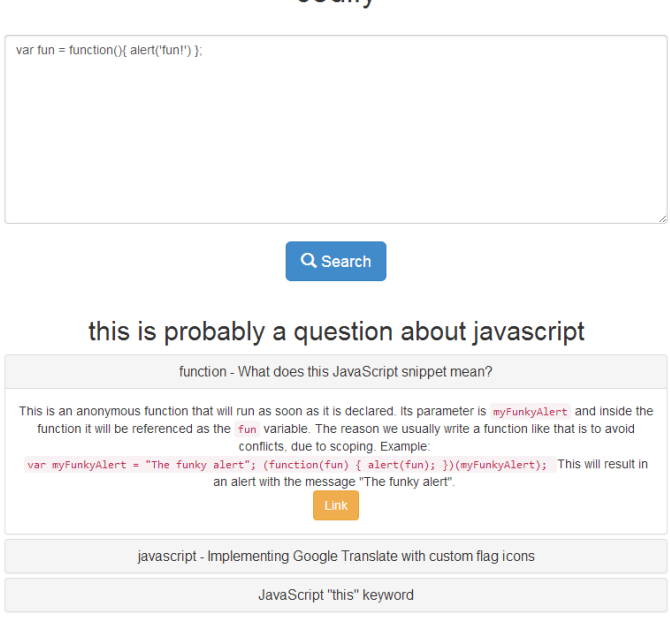
Dimitriy: researched django framework, did general frontend and code skeleton, worked on main search functionality, worked on essay

References:
Google: http://google.com
Stack Overflow: http://stackoverflow.com/
Symbolhoud: http://symbolhound.com/
Ohloh Code: https://code.ohloh.net/

Screenshots:



Searching for a Java function will yield search results pertaining to Java. The query has to do with graphics, so the top results are graphics-oriented.



Searching for a JavaScript query will lead the system to believe you are looking up JavaScript function declarations.