Relevant XKCD - Organization and Retrieval of the XKCD Webcomic

Tim Burke and Anuj Ramakrishnan

It is a common meme that there is a “relevant XKCD” for every situation because of the huge range of topics covered in the XKCD webcomic. However, finding that comic is difficult because XKCD is not organized for queries by topic. While the XKCD fan community has made an attempt at organizing the comic through the website Explain XKCD,[1] the categorization of comics is not intuitive. Our project offers a remedy for this problem, creating improved topic categories and introducing a search function for general queries.

Our resource of interest is individual XKCD comics, which are short comics authored by Randall Munroe. The comics often focus on science, technology and math related topics, but cover a wide array of subjects such as politics and relationships as well. An individual comic contains a title and an image with roughly one to four panels that usually contain text. Additionally, it contains “title text” that is revealed by hovering the mouse over the image. Comics are identified by a title, a permanent URL to that comic on the XKCD website and an image URL for hyperlinking. There are more than 1900 unique XKCD comics which are organized sequentially by the order they were created - the first is labeled “1” and the latest is, for example, “1908.” The official website does not offer any general search functions, and comics can only be browsed one at a time by going to a random comic, the next comic in order of publishing date, or the last comic in order of date.

While the official XKCD website only organizes the comics by their order of creation, there does exist a fan-made wiki named “Explain XKCD” which tracks additional information about each comic and attempts to better organize them. To this end, Explain XKCD includes the image of each comic, transcripts of the main text and rollover text from the comic and an explanation of the what is going on in it. The explanations can be surprisingly in-depth, sometimes well over 1,000 words with charts and links to external resources. The website also organizes the comics by date, by the characters that appear in them and by user-defined topics including “computers,” “science,” and so on. The range of topics is broad and eclectic, including smaller categories such as “crossbows,” “airships,” and “electric skateboards.” There are no strict rules about choosing the topic(s) of a comic - users reach a consensus about which subjects are covered and organize them accordingly.

[1] http://www.explainxkcd.com/wiki/index.php/Main_Page

The structure between comics in Explain XKCD is not entirely intuitive. The topic categories themselves include both parent categories and “subcategories” that are more specific subsets of the parent. Examples of this are “computers” as a parent category, and then “programming” and “cyber security” as subcategories. These categories are defined by the Explain XKCD user base. Comics are thereby structured according to user consensus of the topic(s) covered within, with “related” comics being members of the same topic categories. However, the way the Explain XKCD community uses “subcategories” might be better described as “related categories,” which can cause some confusion. For example, programming is a subcategory of computers but also contains comics that are not included in the computer parent category. This does not make much sense, because any comics that are about programming are necessarily related to computers as well and should be in the same category.

We re-structure the Explain XKCD topic categories so that subcategories are actually a subset of their parents, and parent categories contain all comics in their subcategories. We believe it is more intuitive for comics that share topics to have membership in the same categories - all comics with computer-related topics should be found in the computer category, including those that are also found in subcategories of computers. To constrain the scope of the project, we only organize the Computers topic category, one of the largest in Explain XKCD with 142 unique comics and nine immediate subcategories.

To collect the comics, we scraped the Explain XKCD website using the BeautifulSoup package in Python. Specifically, we scraped a list of comic titles and URLs from the Explain XKCD website, and further parsed each URL to acquire the title text, explanation and transcript of each comic. We use the title of comics to reorganize the categories, merging the set of comics in each parent category with the set of comics in each subcategory. We repeat this process for the subcategories of each subcategory, so that all parent categories contain the full set of comics within their subcategories. Doing this for the computers parent category adds an additional 256 unique comics to take it to a total of 398 unique comics.

We should note that the descriptions of each comic were changed in the process of our reorganization. We chose to exclude the actual image from the comic because we believe any information from the image will already be captured in the transcript and explanation text. Furthermore, we chose to exclude the discussion section because discussions could stray off topic and confound efforts to query comics. The process of scraping transcripts and explanations also results in some loss of information - because we captured text as strings stripped of formatting, information such as tables and graphs included in the explanations are

generally not captured and code snippets are caught as ordinary text without proper formatting. While our comics are therefore a simplified version of the resource on Explain XKCD, we believe that our representation is sufficient for the task of getting accurate queries.

Once we completed the reorganization of the topic categories, we created a retrieval process to allow users to query across comics. The search is meant to evaluate the similarity between the text of a user query and the text contained in comics themselves. Specifically, we evaluate similarity across the comic title, title text, explanation, transcript and topic categories. Our approach is twofold - we represent queries and comics as both TF-IDF vectors and as bag of words (BoW), then find the similarity between the user query and comic using zone-weighted cosine similarity and Jaccard distance. These measures were chosen because both measure the similarity of the words contained in the query and comic, but with two slightly different approaches. A TF-IDF vector contains one element corresponding to each word in the vocabulary across all our documents, where the value in each element is a TF-IDF score. The TF-IDF score is meant to represent the importance of a term in a document and across a whole collection - term frequency increases as a word appears more in a document, but inverse document frequency will decrease the impact if the word is also common across all documents (for example, “the” appears in almost all documents and thus has a low TF-IDF score). Once the title, explanation, transcript, category and title text of the comics are represented as TF-IDF vectors, we can use cosine similarity to compare the comic text vectors to the query text vector. Cosine similarity is a measure of the dot product of two vectors normalized by the product of the magnitude of both vectors. More intuitively, this measures the cosine of the angle between the two, and ranges from -1 (opposite directions) to one (exactly the same direction). If our query vector contains more similar elements to a comic vector, then the two will be pointing in a more similar direction. By finding the highest cosine similarity scores between a query and several comics, we hope to return those that are most relevant.

Jaccard distance is a somewhat more straightforward approach to similarity. We first represent the query and text portions of each comic as a bag of words, meaning a set of the unique words contained within them. The Jaccard distance between two sets is simply the intersection of the sets divided by the union of the sets - in other words, the count of elements the two sets share divided by the number of unique elements across both sets. This similarity measure does not capture as much information about the text as TF-IDF - we do not know anything about the frequency of each word in the text or how frequently the words appear

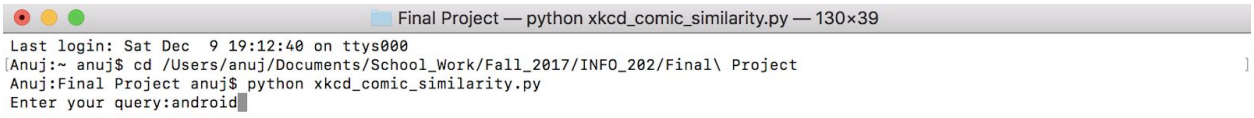
across all comics. Nevertheless, we wanted to see if Jaccard distance would be able to get results that were similarly relevant to TF-IDF and cosine similarity.

We implement TF-IDF and cosine similarity using the Scikit Learn library for python, specifically the TfIdfVectorizer function and the cosine_similarity function. To ensure uniformity across all the TF-IDF vectors, we first fit the TF-IDF vectorizer on all the text of all comics before we transform the comic titles, title text, explanation, categories and transcript to TF-IDF vectors. All text is tokenized using the NLTK word_tokenize function. We then transform the query into a TF-IDF vector and compute the cosine similarity between the query vector and all comic text. For our 398 comics, this results in a 398 by 5 matrix of cosine similarity scores (the 5 columns are the similarity scores for the title, title text, category, explanation and transcript of each comic). Jaccard distance is constructed similarly - we first tokenize the query and comic text and transform them into sets. Using a simple function we wrote for Jaccard distance, we then calculate another 398 by 5 matrix of Jaccard distance scores.

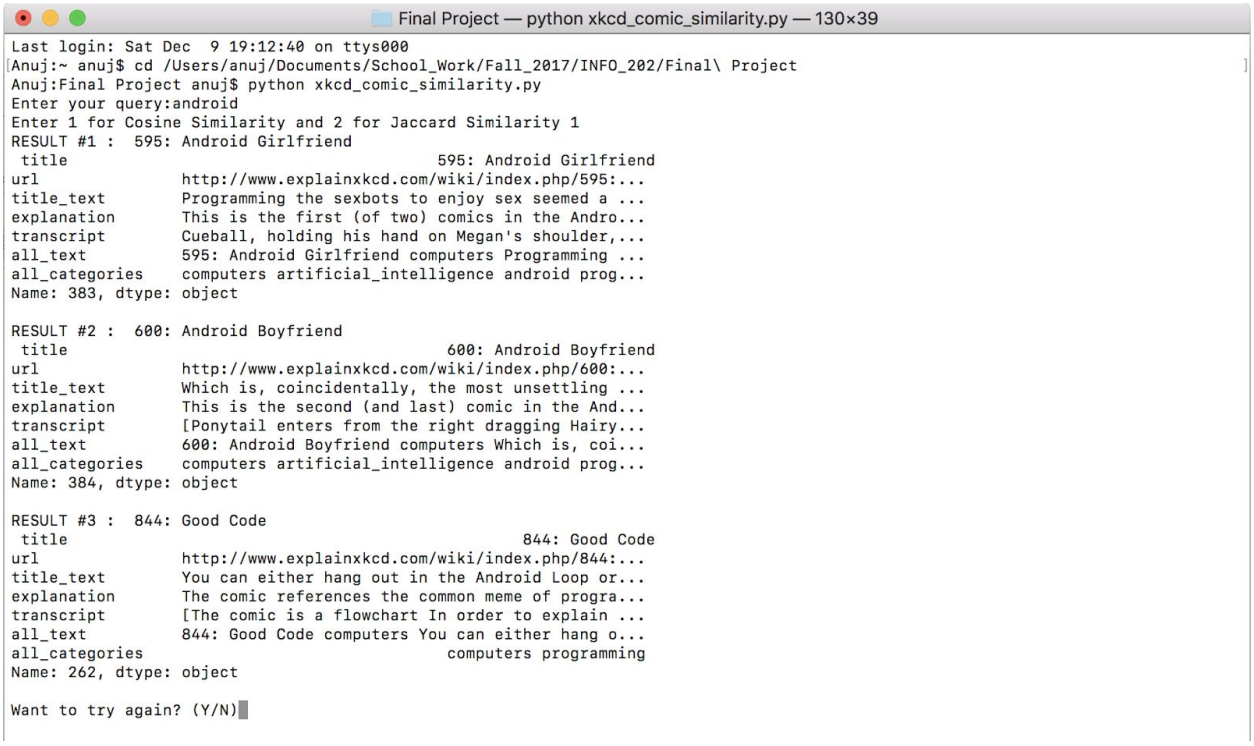
Finally, we implement zoned weighting so that we can differently weight the importance of similarity between the query and each portion of the comic text. The weights are a 5 by 1 vector of positive scalar values. We initialized these weights to be higher for the title and categories, but lower for the explanation (which is longer and may pursue tangents that aren't closely related to the comic's subject and also since the explanation is not inherent to the XKCD comic but is an addition made by the Explain XKCD community). The ultimate similarity score for each comic is determined by taking the dot product of the comic similarity scores with the zone weights (raising the similarity score of comics with more similarity in the highest-weighted zones). This is implemented as a matrix multiplication between the similarity matrix and zone weights to create a 398 by 1 vector of similarity scores. We return the comics which correspond to the top 3 similarity scores.

Program Execution:

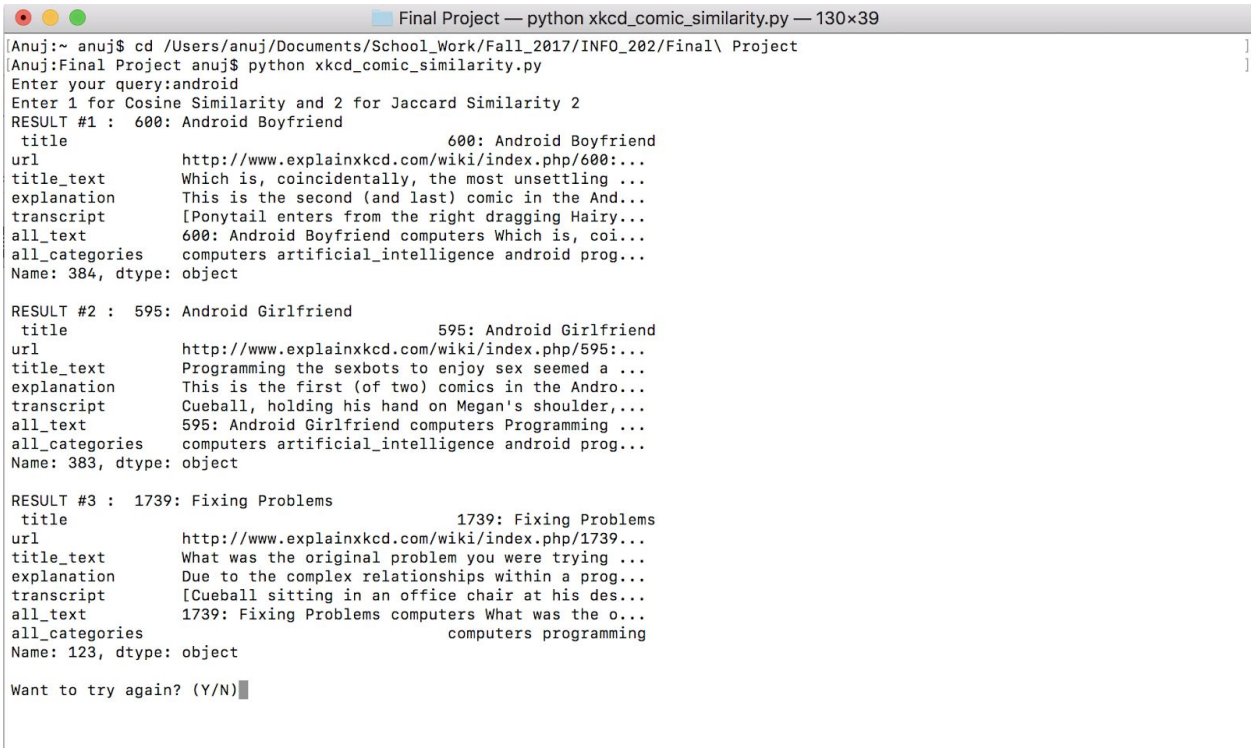
We have created a simple command line interface for the similar comic retrieval program. The program asks the user to input the query to be searched. The query can be a single word or a sentence. As an example, we have used "android" as a query.



Once, the user enters a query, they are presented with the option of choosing the kind of similarity metric they want to use to retrieve the list of similar comics. As discussed above, the two options for similarity metrics are 1) Cosine Similarity and 2) Jaccard Similarity. The snapshot below shows the output when Cosine Similarity is chosen as the similarity metric. The output shows all the relevant resource descriptions of the three most relevant comics.



The next snapshot shows the output for the same query (“android”) but with the Jaccard Similarity chosen as the similarity metric.



Comparing the output using the two similarity metrics, we see that the third most relevant comic is different in the two cases. This difference is because of the way the two similarity metrics are calculated. On inspecting the two results manually, we find that the results returned by the cosine similarity metric are typically more relevant than those returned by using the jaccard similarity metric. This comparison was done for single word queries as well as for queries which spanned a few words.

Furthermore, note that the zone weights were set manually and evaluated on the outcome of several searches, rather than learning optimal weights from a labeled training set. Operationalizing a measure of “relevance” is difficult, so we instead tried searching words relating to comics we were familiar with (such as searching “Git” to find the comics “Git” and “Git commit.” The outcomes were not highly sensitive to the zone weights and we only were able to search over a fairly limited set of queries, so the weights likely could be improved. Ideally, this would be by measuring how relevant users find the results of a search (e.g. which comics they clicked on) and using those measurements as labels to train the weights.