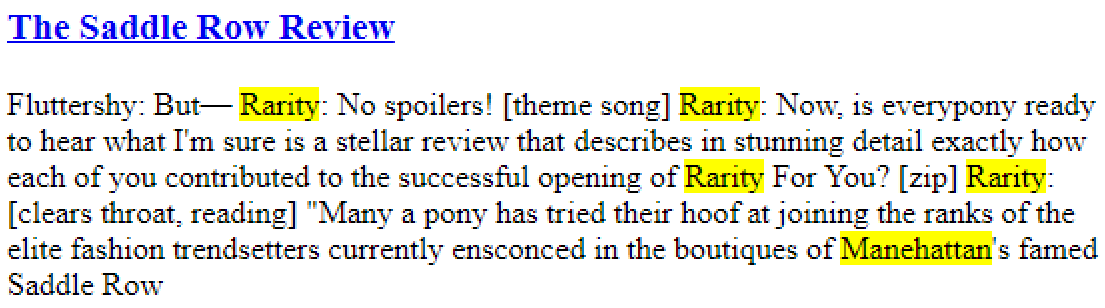
Overview

Search Engine Result Page (SERP) Design

For the search results page, we chose to create static HTML pages. The code for this is in the create_thml_file.py. It takes the result data in a json format and adds in the necessary HTML tags to it in order for it to be displayed accordingly. While creating the Search Result Page, we took the a few important concept criteria from Croft Chapter 5 and implemented them for our search Page. A few of them are mentioned below:

In order to meet the basic requirements of matching the search results from the query entered, we followed the idea of KWIC, or Query-Oriented Summaries.

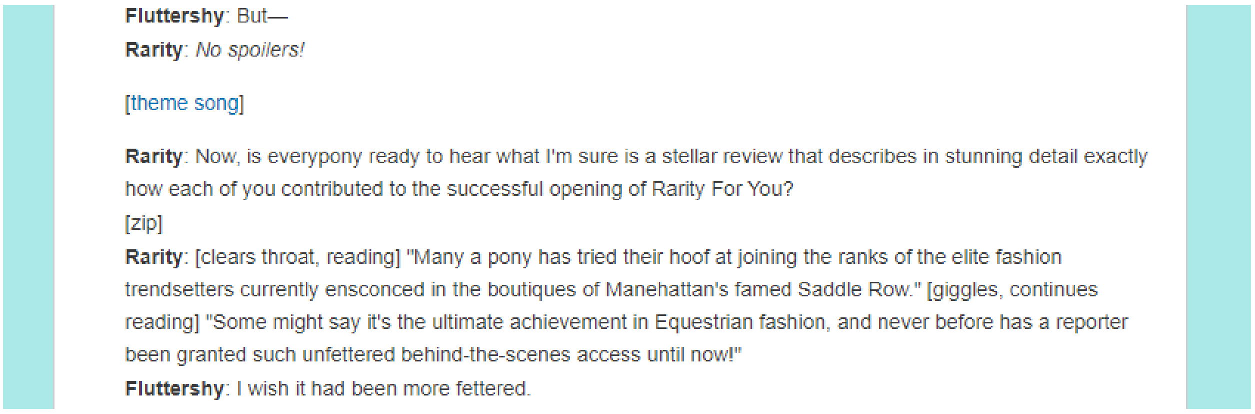


The Saddle Row Review

Fluttershy: But— Rarity: No spoilers! [theme song] Rarity: Now, is everypony ready to hear what I'm sure is a stellar review that describes in stunning detail exactly how each of you contributed to the successful opening of Rarity For You? [zip] Rarity: [clears throat, reading] "Many a pony has tried their hoof at joining the ranks of the elite fashion trendsetters currently ensconced in the boutiques of Manehattan's famed Saddle Row

Image 1.0

We ensured to display small snippets or blocks of information below the URL links for the search results. As shown in Image 1.0, an important property of modern webpage is to have some summary or a small snippet of data indicating what and where has been matched in the following URL link. This matching of terms is called keyword-in-context extraction for use in display of retrieval results. (Hearst, 2009) Here, we have ensured that we summarize the document, being displayed, enough in order to not only show the match query terms but also to maintain a sufficient amount of context for the user to see and thus evaluate whether the grabbed search result is of any relevance.



Fluttershy: But—
Rarity: *No spoilers!*

[theme song]

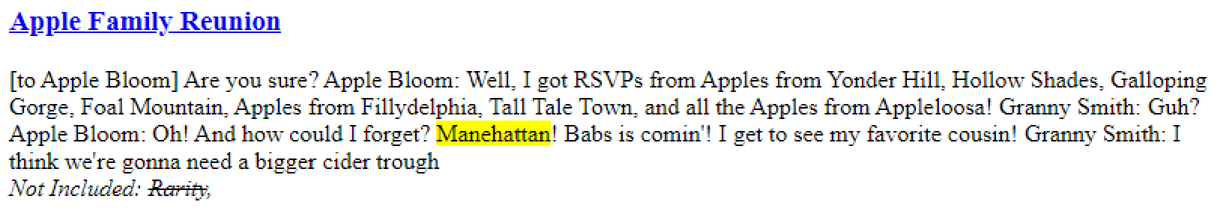
Rarity: Now, is everypony ready to hear what I'm sure is a stellar review that describes in stunning detail exactly how each of you contributed to the successful opening of Rarity For You?
[zip]
Rarity: [clears throat, reading] "Many a pony has tried their hoof at joining the ranks of the elite fashion trendsetters currently ensconced in the boutiques of Manehattan's famed Saddle Row." [giggles, continues reading] "Some might say it's the ultimate achievement in Equestrian fashion, and never before has a reporter been granted such unfettered behind-the-scenes access until now!"
Fluttershy: I wish it had been more fettered.

Image 1.1

The URL links on each search result will navigate the user to the position in the webpage where the script matches the query that has been entered. This webpage result can be seen in Image 1.1.

From Image 1.0, we also made sure to implement the concept of Highlighting the Query Terms. We agree with the usefulness of this feature when it comes to searching through relevant information on the web. In order to make this ‘eye-catching’ we highlighted the words in yellow background so that it distinctly stands out from the text. Our idea was that by highlighting the query terms, the user will be prompted to read a few words on either side of the snippet. Thereby, giving some insight of the context and again, giving assurance to the user about the relevance of the resulted document.

Another interesting feature we added to our search results was indicating the query results that weren’t present in a resulting document. This can be seen in Image 1.2.

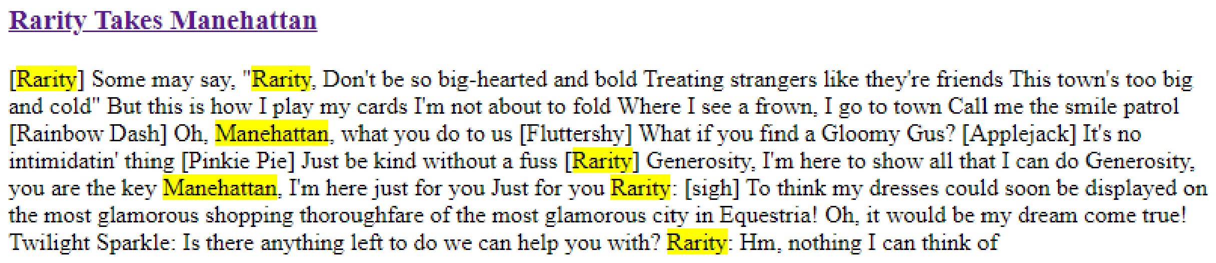


Apple Family Reunion

[to Apple Bloom] Are you sure? Apple Bloom: Well, I got RSVPs from Apples from Yonder Hill, Hollow Shades, Galloping Gorge, Foal Mountain, Apples from Fillydelphia, Tall Tale Town, and all the Apples from Appleloosa! Granny Smith: Guh? Apple Bloom: Oh! And how could I forget? Manehattan! Babs is comin'! I get to see my favorite cousin! Granny Smith: I think we're gonna need a bigger cider trough
Not Included: ~~Rarity~~,

Image 1.2

By incorporating this feature, we are indicating the search diversity of the resulting documents. If there are multiple search queries entered by the user, the program removes the unnecessary or, stooping words and searches the documents by highlighting the unique words. This in turn returns a set of documents where either all words are being matched or only a few terms are being matched. The on in Image 1.2 shows that only one is being matched. Image 1.3 shows both the relevant searched terms are being matched. This method can further allow the user to narrow its search by ensuring whether the user requires the documents where all the terms are being matched or not.



Rarity Takes Manehattan

[Rarity] Some may say, "Rarity, Don't be so big-hearted and bold Treating strangers like they're friends This town's too big and cold" But this is how I play my cards I'm not about to fold Where I see a frown, I go to town Call me the smile patrol [Rainbow Dash] Oh, Manehattan, what you do to us [Fluttershy] What if you find a Gloomy Gus? [Applejack] It's no intimidatin' thing [Pinkie Pie] Just be kind without a fuss [Rarity] Generosity, I'm here to show all that I can do Generosity, you are the key Manehattan, I'm here just for you Just for you Rarity: [sigh] To think my dresses could soon be displayed on the most glamorous shopping thoroughfare of the most glamorous city in Equestria! Oh, it would be my dream come true! Twilight Sparkle: Is there anything left to do we can help you with? Rarity: Hm, nothing I can think of

Image 1.3

The hit summary is calculated on the top 10 search results that are displayed to the user. We find the number of lines in each document. Then we compute the ‘limit’ above which all the words ae considered to be significant words. The formula that we use to get this done is shown in Image 1.4. After this, we loop through each sentence converting in the document, tokenize them and stem these tokens. Then these tokens are compared with the query terms.

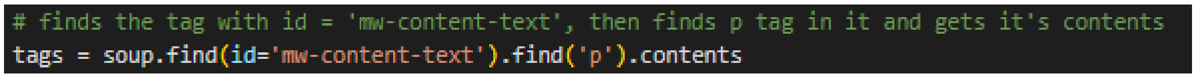
$$f_{d,w} \geq \begin{cases} 7 - 0.1 \times (25 - s_d), & \text{if } s_d < 25 \\ 7, & \text{if } 25 \leq s_d \leq 40 \\ 7 + 0.1 \times (s_d - 40), & \text{otherwise} \end{cases}$$

Image 1.4

Finally, we loop through the tokens and find all the significant words from the previously calculated ‘limit’. The one with the highest significant count is sent back.

Indexing and Retrieval

We implemented the library called Beautiful Soup that allowed us to successfully work on the HTML Tags and their content. By identifying the ‘mw-content-text’ ID, we decided to use the find method of Beautiful Soup and find all the ‘p’ tags within the document and grabbed all the comments in it. By doing this, we were able to parse through the entire HTML document and grab all content under these ‘p’ tags. The line of code that allowed us to complete this task is present in the preprocessing.py file and is shown in Image 2.0



```
# finds the tag with id = 'mw-content-text', then finds p tag in it and gets it's contents
tags = soup.find(id='mw-content-text').find('p').contents
```

Image 2.0

From this content that we just processed, we next looped through each tag. We checked for ‘H2’ tags, which was the title of the documents that we were processing. Upon getting the ‘H2’tag, we added or concatenated all the data till we reached another ‘H2’tag. By doing this, we theoretically got an entire document. This allowed us to get the content of each document. We also grabbed the text inside the ‘H2’ tags so that we were able to get the title of the documents. Lastly, to find the end of the document, we found the ‘table’ tag whose class attribute was ‘Navbox’. By doing this, we were officially done with loading the entire document.

Next, we started with indexing. The program looped through all the documents then looped through the lines in the document. We split each line while looping. The splitting is done on spaces and punctuations. In order to be certain to take care of all ‘punctuations’, we imported punctuations from the String package. The tokens were stripped of white spaces, which were tabs and new line characters. Then, we looped through the list of tokens and converted the tokens to lower case. Finally stemmed each token using nltk by importing the PorterStemmer method. Naturally, we ignored all the empty strings. We created a dictionary to count the number of occurrences of a word within each document. We also calculated the inverted index by adding the document ID of the current document to the inverted list of the current token. We also kept track of the document lengths to calculate the average doc length. By doing so, we were able to incorporate the BM25 model.

It is one of the most successful retrieval models. This model considers the length of the documents and uses it to normalize the term frequency thereby removing an ’unfair advantage’ trend between big and small documents. Without the BM25 approach, the larger document would get a higher score as they would have more area to include more occurrences of a term. which would not ‘be fair’ and correct with the accuracy of the search results being shown.

Compared to tf.idf, the BM25 model is a refinement to the

Results

Test Queries

Effectiveness

Efficiency

Discussion

References

Hearst, M. (2009). Presentation Of Search Results (Ch 5) | Search User Interfaces | Marti Hearst | Cambridge University Press 2009. *Search User Interfaces*. Retrieved November 9, 2020, from http://searchuserinterfaces.com/book/sui_ch5_retrieval_results.html