CS 6200 Information Retrieval
Final Project Report

Rajesh Sakhamuru
12/16/2020

Huffington Post News Article Search

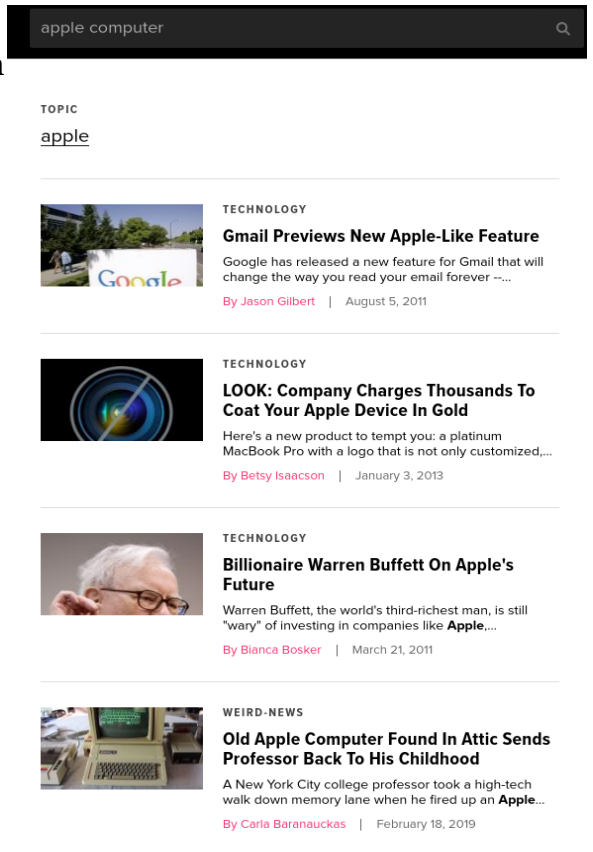
Github repository is located at: https://github.com/rajes95/HuffPost_NewsArticleSearch.git

Introduction

HuffPost, formerly *The Huffington Post*, is an American news organization founded in 2005 offering news in politics, business, entertainment, environment, technology, popular media and many other news categories. *HuffPost* also offers satire, blogs and original content. It is currently owned by Verizon Media and was valued at $1 billion dollars in 2015. According to a recent *MarketWatch* article, *HuffPost* brings in about $50 million dollars in revenue each year while annual expenses are around $65 million dollars creating massive losses in its most recent years.
https://www.marketwatch.com/story/verizon-said-to-be-looking-to-unload-money-losing-huffpost-11601422627

The early *Huffington Post* strategy was to craft search engine optimized stories and headlines based around trending keywords. It proved to be very effective and allowed the company to grow rapidly, and many of those strategies are still being used today by news organizations to have their articles picked up by search engines and drive traffic to their sites. However, while *HuffPost*'s articles are frequently recommended by search engines such as Google, their own website's search feature could use some work.

The search feature of the *HuffPost* website is sub-par compared to other news organizations of the same caliber. Users can search for articles on the *HuffPost* website using any query and it should return relevant, relatively recent news articles which satisfy the users' information need. The current implementation does not always meet this standard. The problems of *HuffPost*'s search engine as well as how it can be improved is discussed below.



Problems Identified

HuffPost search engine is lacking in quality for certain search terms and information needs. An example of a search query on the HuffPost website of 'apple computer' demonstrates some of the shortcomings of HuffPost's information retrieval system →

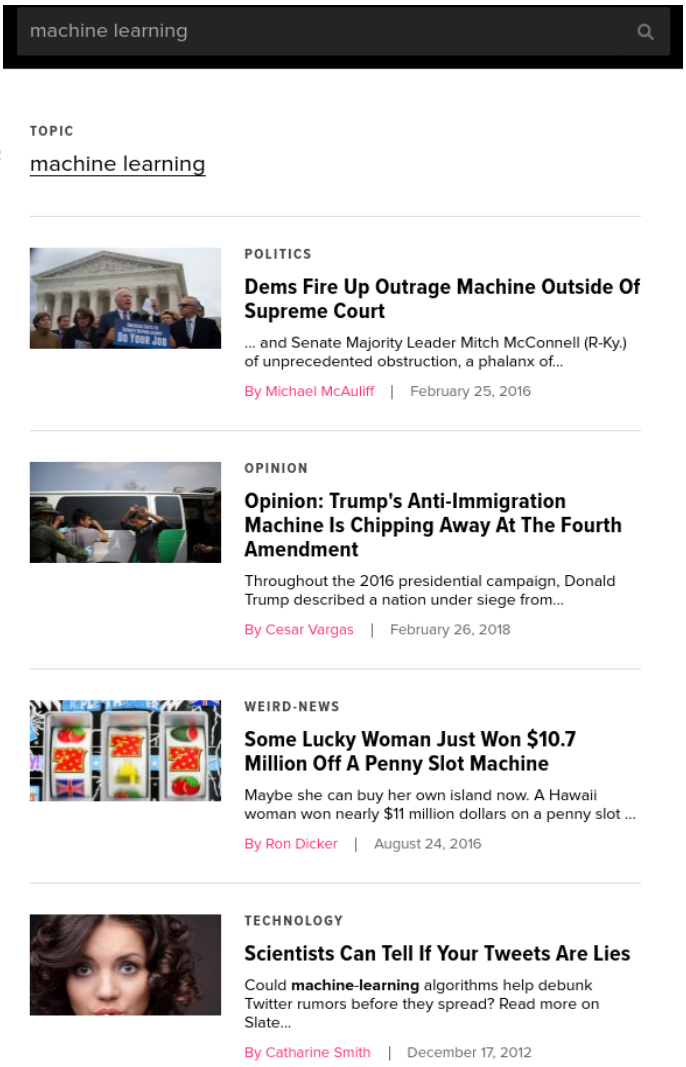
1. Query Term Relevancy vs Recency

In the search query screenshot for "apple computer" on the right, we can immediately see that the first 3 articles

are over 7 years old. It makes some sense that articles are prioritized for keyword relevancy in the search feature over their recency if the user is searching for archived content. However, that is not always the case and a news information need could often require searching for the latest articles on a particular topic. The problem that can be seen here is that there is no option to prioritize more recent relevant articles over older ones.

2. Full Phrases not Prioritized

Here we have an example of the search query “machine learning” on the HuffPost website:



As we can see here, along with some of the problems from the last slide, none of the first three search results contain the full phrase “Machine Learning” despite there being many such articles if I scrolled past the bottom of the page. Instead ‘Machine’ and ‘Learning’ are searched as separate terms, leading to one term being prioritized over the other (machine) because it is a less frequent word in their corpus.

What should be happening instead is that articles which contain the query terms in the correct order as a phrase will be prioritized over the individual word occurrences. Additionally, articles where the query terms appear in close proximity with each other or in the right order should be prioritized over articles where the terms are far apart or in the incorrect order.

3. Not enough results per Page

Additionally, the above image is the left side of a screenshot of their website search feature on a 1080p desktop monitor, and we are only getting 4 results before having to scroll down to see more. This may be due to preserving the website format for a mobile experience, but at least on desktop it is far too few links, and given some of the other problems mentioned

a user will likely have to scroll to find articles which pertain to their information need. The user may benefit from having more articles being shown initially so that if the first four articles are not valuable to their information need, they will not have to take an additional action to either search again or scroll down to see more results.

4. Summaries Need Improvement

In the "machine learning" search example, notice that the summaries don't all contain the excerpt where the search term is present, but rather just highlights the search term if it appears at the start of the article. By not having summaries which contain the query term, we are depriving the user of context regarding the reason that article was returned as a result of their search query. This kind of context about the articles being returned through summaries which contain the query terms could assist the user in assessing how well their query represents their information need. They could understand how their query terms are appearing in the articles returned and subsequently how they could modify their query to potentially improve their article search results.

Proposed Solutions:

There are several avenues which could be taken to solve the previously described problems. By providing the user the option to prioritize recency over just query term relevance, we would provide them with a search service which can serve a greater amount of information needs.

Full phrases from the query could be prioritized over individual words in the articles through the use of a positional inverted index. This index could also be used at search time to weigh the closeness of terms in the articles higher than for articles where the terms are further apart.

Improved summaries which are shown to preview returned articles could be improved by dynamically generating summaries containing the specific query terms. Coherent summaries could be generated which relay to the user immediate feedback as to why that article was returned as a response to the given query by dividing the article body text into sentences, selecting for sentences containing the specific keywords and then returning a random selection of these sentences in sequential order.

Before going into detail about the implementations of these improvements to the search engine, below I explain the details of the *Huffington Post* article dataset and how it was put together.

Dataset

I have found a dataset of over 200,000 links to HuffPost news articles to use as documents to serve as a foundation of the inverted index and provide better search queries for users: https://www.kaggle.com/rmisra/news-category-dataset. One unfortunate result of relying on this dataset is that the articles are only from 2014 to 2018 which means that the most recent articles which I will use will be two-years out of date.

The JSON file from this dataset contained 'Category', 'Headline', 'Authors', 'Link', 'Short Description', and 'Date' for 200,000 Huffington Post news articles from 12/27/2014 to 5/26/2018. I built a focused webpage crawler which traversed ~105,000 of the articles and extracted the article text from all of the ones where it could. The crawling process for these 83680 links took approximately 24 hours to run. After cleaning up the article texts and eliminating bad links and articles with missing dates or descriptions the dataset ended up being 83680 total news article documents saved as a CSV file.

Including the 'Body' of each article on top of the provided categories from the JSON file was expected to greatly improve search results by providing additional context to each article and each section of the documents could be weighed differently to provide the best search results. The inclusion of the 'Body' of the articles also allows easy parsing of the text in order create the dynamic summaries mentioned in the previous section to provide a better user experience.

All of these news articles would need to be loaded into an inverted index where their texts are tokenized, indexed and made available for search. In the next section I will go into more depth about the details of the implementation.

Code Implementation

There are three coded sections for this project. The project was implemented in Python 3 as the main language, with the following libraries used to create the search engine:

- Python JSON and CSV libraries (crawling)
- Python HTML parser (crawling)
- Python Pandas (crawling)
- Elasticsearch with Python (backend)
- Python Flask (frontend)

- **Crawler**

The dataset JSON provided by Kaggle gave a link for each article of the 200,000 from the Huffington Post. Some of the links were expired, invalid links, or redirected to a 404 error page. Each article listing in the JSON file appeared as in the following example:

```
{"category": "ENTERTAINMENT", "headline": "Will Smith Joins Diplo And Nicky Jam For The 2018 World Cup's Official Song", "authors": "Andy McDonald", "link": "https://www.huffingtonpost.com/entry/will-smith-joins-diplo-and-nicky-jam-for-the-official-2018-world-cup-song_us_5b09726fe4b0fdb2aa541201", "short_description": "Of course it has a song.", "date": "2018-05-26"}
```

At each article page the actual body of the text was between the only ' ' or paragraph tags. The implementation to access these 'p' tags was handled by the HTML parser which sequentially reads all of the tags in the HTML page and saved the paragraph text between those specified tags. One of the problems that this led to, was that a few articles put unrelated data between 'p' tags on the webpage such as external article descriptions or unrelated advertisement image links. This led to some articles' bodies being scraped with a lot of junk data attached to the end.

Each article was added to the CSV file one at a time, without keeping the article in RAM as to enable the program to run for a long time without running out of computing memory. An issue that was encountered occasionally was when due to the frequent calls being made to the *Huffington Post* web domain, after a few thousand articles were searched the program would sometimes stall and need to be restarted, likely caused by anti-scraper countermeasures taken by the *Huffington Post* website.

Once I scraped about ~105,000 articles and saved them to a CSV file, I reloaded the CSV file using Python Pandas and went through and found errors and junk which were also scraped by the HTML parser and cleaned them up using the string data libraries built into Pandas for working with large text datasets. Once the CSV was cleaned and crawled articles with any missing data or corrupt data were removed, I ended up with a complete dataset of 83,680 articles.

This dataset needed to be loaded into the Elasticsearch server via the Python API which I discuss in the next section regarding the program's backend.

- **Backend**

Once the data was scraped and cleaned and saved as a CSV file, it was reloaded into memory via Python Pandas and each row was inserted into the Elasticsearch as a document. The initial indexing of all 83,680 documents takes roughly 5 minutes, but after having been indexed once, the program does not need to re-index any of the data. Within the Elasticsearch index, the document sections were kept separately which allows the queries to be customized to weigh different sections differently when calculating document relevance.

Next, the Elasticsearch Python API was used to develop custom search queries which can either prioritize recency or query term relevance, and even custom queries which account for some level of spelling errors which may occur on the user end. Below are some screenshots with explanations on how these custom queries were constructed:

```
queryCross = {
    "from": 0,
    "size": 20,
    "query": {
        "function_score": {
            "score_mode": "sum",
            "boost_mode": "multiply",  # The documents relevance is multiplied with the sum
            "functions": [
                {
                    # The relevancy of old posts is multiplied by at least one.
                    "weight": 1
                },
                {
                    # Published in last 150 days get a big boost
                    "weight": 1.5,
                    "gauss": {
                        "date": {
                            "origin": "2018-05-25",
                            "scale": "150d",
                            "decay": 0.5
                        }
                    }
                },
                {
                    # Published in last 1200 days get a boost
                    "weight": 1.25,
                    "linear": {
                        "date": {
                            "origin": "2018-05-25",
                            "scale": "1200d",
                            "decay": 0.5
                        }
                    }
                }
            ],
            "query": {
                "multi_match": {
                    "query": queryText,
                    "type": "cross_fields",
                    "fields": ['headline', 'short_description^2', 'body^3', 'authors', 'category^2'],
                    "auto_generate_synonyms_phrase_query": True,
                    "minimum_should_match": '25%'
                }
            }
        }
    }
}
```

The custom query on the left is built to prioritize recency of articles to the 'current date' which we are pretending is 5/25/2018 for the sake of this project. There are two additional weights here to boost article recency, where articles published over the last 150 days get a boost which decays via a Gaussian curve the older the article is within that time frame. The second weight is for articles published over the last 1200 days which get a smaller boost which decays linearly. The second boost I found particularly useful to bring up very relevant articles published between 150 and 1200 days prior which would otherwise be suppressed by the heavily weighted more recent articles from the last 150 days.

At the bottom of the query, we can see how the query weighs different fields of the documents. The query type is 'cross_fields' which means it matches the terms from the query across all 5 fields which we are looking at, 'headline', 'short_description', 'body', 'authors' and 'category', and then sums them together to get get the final score which is used to rank the documents. The '^3' and '^2' which can be seen in the 'fields' section of the query indicates

that those specific sections are weighed more by a multiple of 3 or 2 respectively. These values were arrived at after much trial and error and making subjective changes based on rankings of returned articles from different queries. By weighing 'body' with '^3' it gets 3 times as much weight when the word appears in the 'body' and is prioritized over articles which only have query terms in the 'headline' or even 'short_description'.

Also of note is the 'auto_generate_synonyms_phrase_query' which is set to 'True'. This parameter automatically creates match phrase queries using multi-term synonyms to expand the queries and provide better search results which may contain terms not explicitly specified in the search query.

Another custom query which I've included is only ever used in the case of very few results being returned from the 'cross_fields' multi_match query type. This is because unless there is a spelling error or terms used which are not at all in the corpus, the results are typically best when retrieved by the custom query already shown above. However in the case of a spelling error in the search query, the following 'fuzziness' query is used →

While very similar to the above query with the recency scoring, this match query uses 'best_fields' which gives the article a ranking score based on how well the terms match in any single field and picks the best field. While the results through that are not great by themselves, this method allows us to use Elasticsearch's built in 'fuzziness' parameter which takes words within a short Levenshtein distance from the query terms and

```
queryFuzzy = {
    "from": 0,
    "size": 20,
    "query": {
        "function_score": {
            "score_mode": "sum",
            "boost_mode": "multiply",  # The documents relevance is multiplied with the sum
            "functions": [
                {
                    # The relevancy of old posts is multiplied by at least one.
                    "weight": 1
                },
                {
                    # Published in last 150 days get a big boost
                    "weight": 1.5,
                    "gauss": {
                        "date": {
                            "origin": "2018-05-25",
                            "scale": "150d",
                            "decay": 0.5
                        }
                    }
                },
                {
                    # Published in last 1200 days get a boost
                    "weight": 1.25,
                    "linear": {
                        "date": {
                            "origin": "2018-05-25",
                            "scale": "1200d",
                            "decay": 0.5
                        }
                    }
                }
            ],
            "query": {
                "multi_match": {
                    "query": queryText,
                    "type": "best_fields",
                    "fields": ['headline^2', 'short_description', 'body', 'authors', 'category'],
                    "auto_generate_synonyms_phrase_query": True,
                    "fuzziness": 'AUTO',
                    "max_expansions": '4'
                }
            }
        }
    }
}
```

searches for up to 4 of them. This is particularly useful and effective if very few results are returned due to a spelling error.

The above two shown custom query screenshots contain the custom queries in the case of prioritizing 'sort by recency' with recent articles first. However in my implementation, the user has the option to switch to search queries which do not take into account publish date at all and return the articles based only on query term relevancy. For those 'sort by relevancy' queries the 'multi-match' queries are the same but without the additional 'function_score' weighing recency. In all cases the top 20 ranked articles based on the search query are returned and shown to the user.

The Elasticsearch index with all of the custom queries uses a Positional Index, which allows the index to know which terms are next to each other in each article. This positional index when using match queries, without any modifications by me, weighs the closeness and order of the query terms in articles more heavily than in articles where the terms are further apart or out of order. This solved the major problem with the HuffPost search of having the initial results prioritize different query terms individually rather than as phrases.

Another key backend implementation is of generating summaries dynamically for the user based on their search query. Coherent summaries are generated for each returned article by dividing the article body text into sentences, selecting for sentences containing the specific keywords and then returning a random selection of these sentences in sequential order. This technique allows us to return summaries which can inform the user of some context as to why particular articles are being returned based on their search query.

- **Frontend**

The frontend of the user interface which allows the user to interact with the backend and make queries into the Huffington Post article index which return the best possible results was created via the Python Flask module. The interface consists of a single HTML webpage for the user to view, with the initial landing page just having an empty search bar and the option for the user to choose between prioritizing relevancy or recency for articles shown. Also of note, is that the search bar has been increased in length, subliminally indicating to users that a search query of any length is acceptable so they do not restrain themselves only to short queries.

Once a search query is submitted by the user, the top 20 ranked articles are displayed with URLs to access them and dynamically generated summaries:

Query the HuffingtonPost News Articles from 12/27/14 to 5/25/18

Prefer Recency to 5/25/18 | Sort by Query Relevancy

Submit

Searching For: 'machine learning'

Amazon Is Making It Easier For Companies To Track You
"Some of this work is highly visible: our autonomous Prime Air delivery drones; the Amazon Go convenience store that uses machine vision to eliminate checkout lines; and Alexa, our cloud-based AI assistant... " Here's where it starts to get more interesting: "But much of what we do with machine learning happens beneath the surface... programs without having any machine learning expertise themselves... "Though less visible, much of the impact of machine learning will be of this type," Bezos says in his letter, "quietly but meaningfully improving core operations...
The Atlantic, ContributorExploring the American idea since 1857
2017-04-24

AI Predicts Autism Based On Infant Brain Scans
[11 Facts Every Parent Should Know About Their Baby's Brain] With all of the data in hand, the researchers set out to first train their machine learning program , and then use it to run predictions... Machine learning is a kind of artificial intelligence system that gets smarter based on the data it processes... In the end, the machine learning program was correct in 82 percent of the cases in which the children did develop autism... In this case, the program was learning to spot differences between the functional connections imaged in the MRI data collected at 6 months old that correlate with cognition, memory and behavior and the details from the behavioral assessments collected at 24 months...
Tracy Staedter, Live Science
2017-06-12

This $3.8 Million Vehicle Is The World's Most Expensive SUV
It features a 40-inch television, Playstation, internet, refrigerator, bar, integrated laptops, satellite navigation system, coffee machine and a rainbow of mood lighting... And for nearly $4 million, we would sure hope to have a coffee machine included...
Andy McDonald and Damon Dahlen
2018-04-27

Joy Reid's Hacking Claims Look Increasingly Unlikely
To accept Reid's hacking claim at face value, you would also need to explain at least some of the following: The screenshots circulated this week were allegedly captured using Internet Archive's Wayback Machine, an expansive digital library that archives sites in real time so people can continue to access the pages... Since Wayback Machine captures sites in real time, this means someone would have had to hack Reid's blog numerous times between 2005 and 2009... If someone hacked her blog recently and attempted to change the timestamps to a previous year, Wayback Machine would still archive the posts according to the date they were actually hacked... " The Library of Congress, which uses a local installation of the Wayback Machine, contains the disputed posts, CNN reported Tuesday...
Hayley Miller
2018-04-26

Apple Is Hiring People To Predict What You're Thinking
Apple does not reveal the number of people working on its machine learning efforts... But one former Apple employee in the area, who asked not to be named to protect professional relationships, estimated the number of machine learning experts had tripled or quadrupled in the past few years... " But Google and others have an edge in spotting larger trends, meaning Apple's predictions may not be as good, said Gonzalez, echoing a commonly held view among machine learning experts... "They are gradually engaging a little more openly," said Michael Franklin, who directs UC Berkeley's Algorithms, Machines and People Lab, which Apple sponsors...
Julia Love, Reuters
2015-09-08

Pauley Perrette Reveals 'Multiple Physical Assaults' After 'NCIS' Exit
" Perrette, whose final episode aired on CBS last week after her announced departure in October , pointed to false tabloid articles and a "powerful publicity machine" attempting to silence her... ? There is a "machine' keeping me silent, and feeding FALSE stories about me... A very rich, very powerful publicity "machine"...
Ron Dicker
2018-05-15

What's For Dinner When Kim Jong Un Meets South Korean President Moon Jae-in
When North Korean leader Kim Jong Un and South Korean President Moon Jae-in meet for their summit on Friday, they will dine on cold noodles made by a special machine and a fried potato dish from Kim's youth, officials announced... He and his staff will use a noodle-making machine to create the dish...
Nick Visser
2018-04-25

Donald Trump Directs Justice Department To Take Steps To Ban Bump Stocks
" BREAKING: President Trump says he's directed AG Sessions to clarify if bump stocks are illegal and to propose regulations to "ban all devices that turn legal weapons into machine guns... " Bump stocks are devices that allow a semiautomatic rifle to fire as quickly as a machine gun ... In 2010 and again in 2012, the Bureau of Alcohol, Tobacco, Firearms and Explosives ruled that it did not have the authority to regulate bump stocks because they are not technically "machine guns," which federal law defines as firearms that shoot more than one shot per pull of the trigger... "I am directing the Department of Justice to dedicate all available resources to complete the review of the comments received, and, as expeditiously as possible, to propose for notice and comment a rule banning all devices that turn legal weapons into machineguns," Trump said Tuesday in a statement...
Hayley Miller and Nick Wing

For this example query we can see the dynamically generated summaries around the query phrase 'machine learning'. Articles with the complete phrase 'machine learning' in order are prioritized as well. We can also see that there are 8 immediately visible results with relatively recent articles so that the user does not need to scroll further down if their preferred result is not within the top 4 like on the *HuffPost* website.

When the search query has misspellings resulting in very few articles returned, the 'fuzzy' search is used instead and often produces very good results. However the summaries generated are not as good due to the ambiguous search term and so when 'fuzzy' search is used the summaries are just the beginning of the articles or the article 'short_description' depending on length.

Conclusion

The *Huffington Post* article search engine seeks to improve upon the search implementation present on the *HuffPost* website. After identifying problems with the way that their search engine returned results, this implementation was built to solve them. While by no means a perfect solution, it reliably returned search results relevant to the users query and enabled them to get results prioritized by recency or just relevance. The dynamic summaries also turned out to be quite effective, and the user interface was simple enough to be used by anyone. This is only a prototype model which would need to be expanded to a more complicated implementation involving distributed systems and possibly millions of articles if it were ever to be a solution used by *Huffington Post* themselves.

Future Improvements

I would like to implement images in the search results and overall make the user interface more 'beautiful'. Additionally there is some clean-up which could be done to the summary implementation per query also including the highlighting of keywords within the summaries to be more clear to the user.

I could also create a better implementation when dealing with spelling errors rather than just relying on Elasticsearch's 'fuzzy' implementation, because currently if only a single term in a multi-term query is misspelled, if that correctly spelled word has many matches, then the misspelled word is completely ignored in the ranking of results.

Additionally I could potentially implement a web-crawler and scraper which can get the most recent Huffington Post articles as well as potentially include articles from other news outlets. This could expand the possibilities of this application to be used as a general purpose news article search tool.