Kyle Tapang ktapang2@illinois.edu (Team Captain)
Chunjuan Li cli111@illinois.edu
William Kiger wkiger2@illinois.edu

CS 410 Final Project Documentation- Team Beans

Introduction/Overview

Our team created a browser to rank articles based on both term frequency/inverse document frequency, using BM25, and the sentiment of the published articles in Hacker News (https://news.ycombinator.com/). Hacker News uses a unique ranking function that lists 30 articles per page based on the number of upvotes from readers [1]. Our team added functionality of this site by designing an application to scrape Hacker News articles, go in and grab the articles text, and implement our own browser/query tool for users to have the ability to query to find articles from the site. Our application runs very similar to a Google search, but with our own custom ranker.

IMPORTANT

The current configuration pulls article information from a "data.csv" file derived from our custom scraper so that testers of this project do not have to scrape Hacker News. The test dataset contains data from 150 articles.

Project Demo Link

https://drive.google.com/file/d/1dVXFDGihnh0_OH4CLhNIScn4FqNUUGuy/view?usp=share_link

Quick Startup Guide

1. (Skip to step 2 if you already have Docker). Download and install Docker.
    a. Apple: https://docs.docker.com/desktop/install/mac-install/
    b. Windows: https://docs.docker.com/desktop/install/windows-install/
2. From Docker run the command: docker pull ktapang2/cs410-project:latest
3. Run the command: docker run --publish 5000:5000 ktapang2/cs410-project

4. Type in your browser: 127.0.0.1:5000
5. The browser should be running. Type in your query and test out our Hacker News browser.

User Interface

In our project, we use Flask (implemented on Werkzeug and Jinja2) as the web framework for developing our web application using python, and HTML/CSS for describing the structures and styles of our web pages.

- ➢ home.html: The user landing page - “Welcome to Hacker News Searcher” includes 2 main elements (under the “form” section and method is POST) : the search data input box and submit button, the search data input box is for the user to provide the search data that the user wants to explore, and submit button is for enabling the backend for proceeding the search request. An error message is also defined as a checking condition for error handling on this page.
- ➢ searchResults.html: The result page “Hacker News” returns our top 5 ranked hacker news records with the related hyperlinks based on the user provided query. The “Go back” button is defined (under the “form” section and method is “GET”) to navigate the user back to the home page for exploring other queries.
- ➢ main.css: This file contains all the style definitions for our UI pages including headers, input box, and button, etc.
- ➢ app.py: The main source code contains how we use Flask to interact with our web pages and backend ranking source. We use the Flask “route” feature to define the URLs for our home and results pages, and use “redirect” and “url_for” methods to forward parameter value and navigate to different URLs, also use “render_template” to target the page (html file) to display for our web UI.
    - The 1st route is defined for the home page url, when the user enters a valid query and clicks the submit button, it will be redirected to the “results” route (2nd route) by Flask “redirect” and “url_for” methods. If the submit button is clicked without any query data, the home.html with the error message “Please enter your search data” will be the target page template

to display using Flask's "render_template" method. Otherwise, the target page template is home.html, for example, after the user clicks "Go back" on the result page, the home page will be displayed.

- The 2nd route is for the search results page, when a valid user search data is passed from the home page to the search results route, our backend "ranker" method will be called, and based on the returned results from the "ranker" method, the "title" and "link" values will be passed to results.html template using Flask "render_template", and the content of a proper result will be displayed to the user on the results page.

Hacker New Scraping Algorithm

The Hacker News scraping algorithm is designed to visit news.ycombinator.com, collect information about the listed articles that are sorted by Hacker News custom ranking algorithm that ranks based on the number of times a story was submitted to the website and scores them by points [1]. Our team noticed that Hacker News does not currently have the ability for a user to enter a query and have a ranked list of articles returned. This is where we came up with the idea to add this capability with a new search engine based on a BM25 and a BERT based sentiment analysis.

The custom Hacker News scraping algorithm we designed has 2 main capabilities. The first is to get every article's metadata including, the title, URL, and number of points assigned to that article. Second, the algorithm can accept a parameter to issue a get request to the article and attach all the text data from that website into a Python dictionary data structure. Obtaining all this data allows for BM25 and our BERT sentiment model to make inferences based on the collected text after being extracted with Beautiful Soup [3].

The current configuration of our search engine relies on a csv file derived from this Hacker News scraper. We learned that the amount of data this scraper collects is significant in size and that only scraping 3 pages deep, 150 articles, is a good number of articles to show our proof of concept. Further, we learned about the complex nature of creating a ranking algorithm when you might visit sites that offer no text. Hacker

News links against GitHub pages with only source code, websites with no text such as videos, and occasionally sites will block GET requests that our program relies upon. We handle the possibility of lack of page information by only ranking the title in the case of no article text.

BM25 Implementation

For our implementation of BM25, we decided to factor in the sentiment of a user's query to find documents with matching sentiment. To obtain our BM25 scores we used the python library rank-bm25. Prior to passing in our data through the functions provided by the library, each document was tokenized, filtered of stopwords, and all characters were lowercase. The sentiment of each document and query are obtained through William's implementation of BERT. Let's break down how we decided to formulate our ranking function around BM25.

Let S, denote a set of sentiment scores for each document

$$S = [s_1, s_2, s_3 \dots s_i]$$

Let B, denote a set of BM25 scores for each document

$$B = [b_1, b_2, b_3 \dots b_i]$$

We decided to subtract the sentiment scores for each document by the sentiment score of the user's query. This will allow us to see how close the sentiment of the query and the document match. The closer to zero the a is the better the match.

β is the query's sentiments score

$$a_i = |s_i - \beta|$$

So, how do we actually use a? Since each $1 > a > 0$, we had the idea to take the inverse of it and multiply it by the BM25 scores. This will reward documents with closer sentiments to the query.

$$score(d, q) = b_i \frac{1}{\alpha_i}$$

Additionally, we decided that there isn't much significance between a a_1 = 0.001 and a_2 = 0.0001, but using this method will mean that there is a 10 times difference between the two. To account for this, we used a log to diminish these effects. This gives us the final formula that we used for scoring our documents.

$$score(d, q) = log\ (b_i \frac{1}{\alpha_i})$$

Sentiment Analysis - BERT

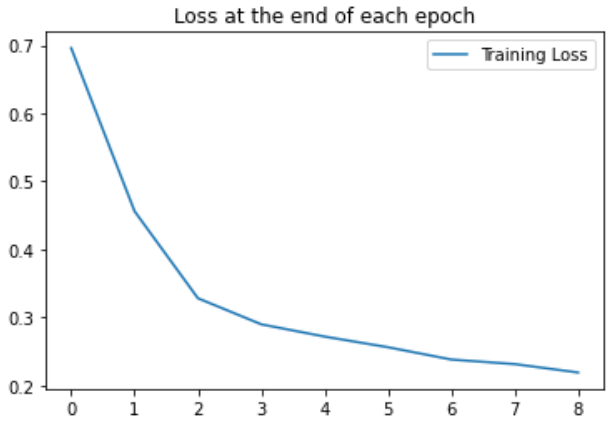
Bidirectional Encoder Representations from Transformers (BERT) is a Google algorithm that has the ability to take unlabeled text data and join left and right context in all layers and can be used to pre-train a deep bidirectional model that has many use cases [2]. The significance of applying BERT is that the pre-trained model can be modified with a single output layer to make a "state-of-the-art" model that can be used for multitude of tasks such as question answering and language inference without requiring much in the way of architecture modifications. Among the natural language processing tasks that BERT achieves state-of-the-art results are increasing the GLUE score by 7.7%, MultiNLI by 4.6%, a 1.5-point absolute improvement in SQuAD v1.1, and 5.1-point absolute improvement in v2.0 [2].

For our application, we used a BERT pre-trained model from HuggingFace which is fed into a Gated Recurrent Unit network to predict the sentiment of the input text. We trained our model built on BERT with a PyTorch dataset from IMDB movie review dataset with 50,000 reviews trimmed and divided up into training (17,500 reviews) and test datasets. We chose this dataset because it contains long reviews for the movies,

which will be similar to drawing an inference from articles from Hacker News. Both the Bert Tokenizer and the Bert Model (based uncased) were used from HuggingFace.

To train the model, we ran batch sizes of 16 and at 8 epochs while saving the model at each improvement in accuracy. The final model used by our program was saved at the 8th epoch. As shown below, we saw a steady decrease in loss through the 8th epoch.

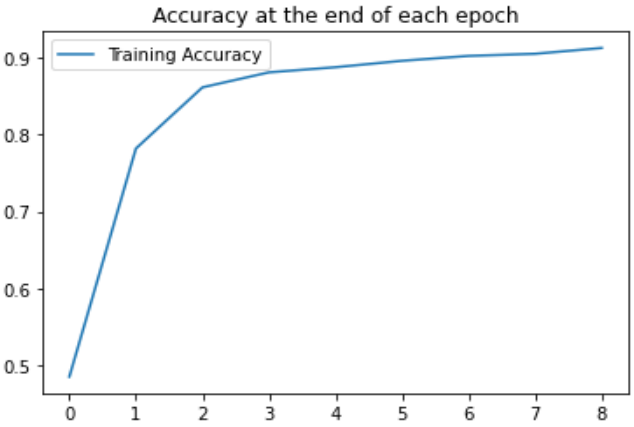
```
In [16]: plt.plot(train_losses_list, label='Training Loss')
         plt.title('Loss at the end of each epoch')
         plt.legend();
```



Y-Axis shows the loss. X-Axis is the Epoch

Further, we saw an increase of accuracy at each epoch.

```
In [17]: plt.plot(train_accuracy_list, label='Training Accuracy')
         plt.title('Accuracy at the end of each epoch')
         plt.legend();
```



Y-Axis shows the loss. X-Axis is the Epoch

Finally, on our testing set, our algorithm was testing at a 91% accuracy on determining the correct sentiment (positive or negative) of the IMDB movie review dataset.

In [13]:
```
# Checking the Testing set loss and accuracy.  Achieveing around 91% accuracy on the test set.
model.load_state_dict(torch.load('sentiment_model.pt'))
test_loss, test_accuracy = evaluate(model, test_iter, criterion)
print(f'Test Loss: {test_loss:.5f}')
print(f'Test Accuracy: {test_accuracy*100:.5f}%')
```

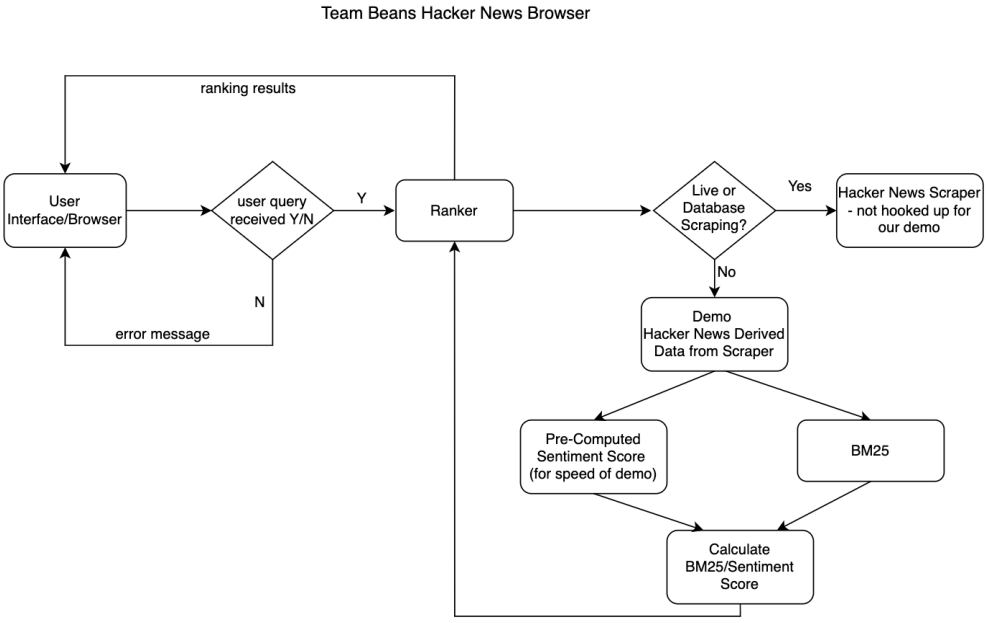
```
Test Loss: 0.22077
Test Accuracy: 91.49872%
```

The compiled sentiment model was to large to push to our GitHub repository, but can be downloaded here: https://drive.google.com/drive/folders/10erdKMVCZISb-4YPsuTo0eRpk6kNaZBT?usp=sharing and the code to generate this model can be found in our project at CourseProject/model/SentimentAnalysis.ipynb.

Team Member's Contributions

Task	ECT (hours)	Actual (hours)	Assignee
User Interface (UI) - Analysis, Design and Implementation	15	18	Chunjuan
Web Scraping	5	8	William
BM25	10	10	Kyle
Sentiment Analysis	10	18	William
Integration Testing	5	2	Chunjuan
Project Proposal	5	3	William
Progress Report	2	2	Kyle
Presentation	3	3	Kyle
Documentation	5	2	Kyle
Deployment	?	10	Kyle

Code Structure



Conclusion

Overall, we have made a unique search engine which utilizes an interesting combination of BM25 and BERT (sentiment analysis). Users of Hacker News now have an additional method of browsing their favorite website through our intuitive Google-esqe UI.

Related Work/Libraries/Code References

- Hacker News Scraper
  - https://pypi.org/project/beautifulsoup4/
- BERT Sentiment Analysis
  - https://huggingface.co/docs/transformers/model_doc/bert
  - https://bentrevett.com/
- BM25
  - https://pypi.org/project/rank-bm25/
- User Interface
  - https://flask.palletsprojects.com/en/2.2.x/quickstart/

References

1. Y Combinator, Hacker News FAQ. https://news.ycombinator.com/newsfaq.html
2. Devlin, J., Chang, M., Lee, K., Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding https://doi.org/10.48550/arXiv.1810.04805
3. Beautiful Soup Documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/