San Jose Times

Chungchen Ran (chungchen.ran@sjsu.edu)
Sakshi Tyagi (sakshi.tyagi@sjsu.edu)
Shabari Girish Ganapathy (shabarigirish.ganapathy@sjsu.edu)
Saumya Goyal (saumya.goyal@sjsu.edu)

San Jose State University
One Washington Square, San Jose, California -95192-0080, USA

Abstract

Redesign a blogging website to provide smart summaries of each blog in the title card,improve search results and implement autocomplete on search query to help enhance user experience. Using Natural Language Processing we try to enhance the look of an already existing blogging website or any news article website. The machine generated summarization feature leads to users spending more time on the site as the reader gets a taste of what the article is about before committing to read the whole thing. The auto-complete and auto-correction feature used to provide search results are tailor made for each website so that the users does not end up with random results.

***Keywords*— Natural Language Processing, Text Summarization, Autocomplete, Elasticsearch**

1. Introduction

Active news readers generally used to spend their time with physical newspaper during their everyday routines, say with morning coffee or on the way to work in a public transport. With the advancement in technology and the reduced cost of smart devices, physical news papers are being replaced with mobile applications or online news websites. Such digital content seem to have an issue though. Say you scroll through a bunch of articles, there would always be a few which have a catchy headline or a tag phrase that interests you. But when you actually read the entire content the article may not be that interesting and you have now wasted a couple of minutes of your read time. The more this happens the less likely you tend to use the same application or the website. This issue arises because content writers come up with the catchy phrases to get you hooked to an article. The issue you would notice is the search results seem quite bad as a result of poor spell-check and suggestions provided. News giants do not bother about such issues as they already have a group of loyal followers. In this project we took The New York Times as our case study.

Fig. 1 Example of no spell-check and suggestions

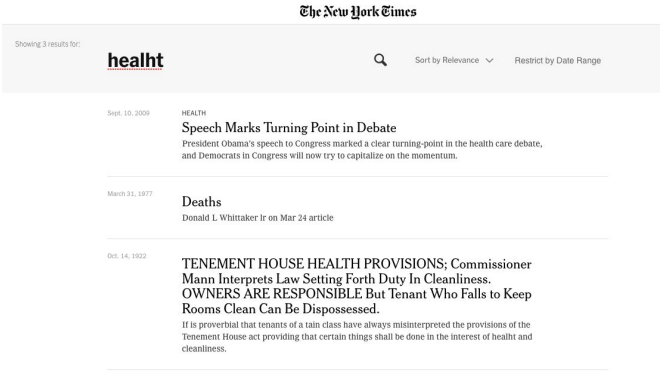

Fig. 2 Example of poor search result

Fig. 1 and Fig. 2 show the actual, current search outputs from New York times website. We can notice how due to lack of spell-check or suggestions the user ends up with a random search result. Though it may not seem that important and the websites might claim the content is all that matters, as a user we need improvements in such areas to actively get back to this site. Only when the user's frequency of visits increase can such blogs or news websites make money through advertisements. This project is an attempt to solve the above mentioned issues while maintaining the rendering time of the website almost the same.

2. Architecture

In this section we look at the two main backend architectures in place for the website to work efficiently.

2.1 Machine Learning Server

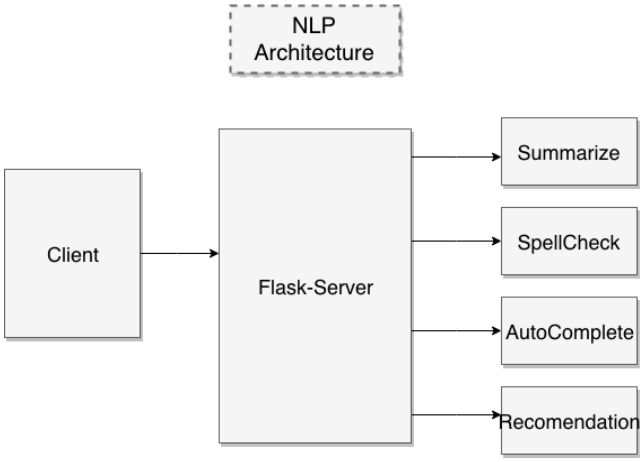

Fig. 3 Architecture for NLP functions and flask server

The four important NLP function are exposed using individual API's. Any one of them can be access by clients as and when required. The AutoComplete and SpellCheck functions talk internally to provide better suggestion results. The summarization of content is done before hand and hence we do not represent a the central database in this architecture. New articles added to system get their content passed though the Summarize function before getting stored in database.

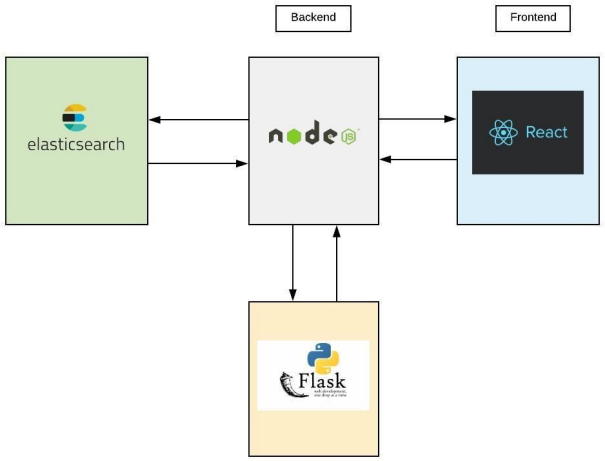

Fig. 4 Architecture for user interface

For user Interface, a web application is created with front end in ReactJS. The frontend communicates with a NodeJS based web server. This server inturn communicates with ElasticSearch and Flask Server for each search request. The database has been pushed to the Elasticsearch so that the retrieval time for each query is fast.

3. Development

3.1 Summarization

Python NLTK framework was used to generate the summaries for each article's content. The content is first split into individual sentences. These sentences are further split into words.

$$W_i = f_i / f_{max}$$

Equation. 1 Weight of each word

$$IW_i = 1/W_i$$

Equation.2 Inverse frequency weight

Equation. 1 represents the mathematical statement to define weight of each world in the article's content. Here f_i is the frequency of occurrence of word i and f_{max} is the maximum frequency of any given word in the entire content. We use the inverse frequency function represented in Equation. 2

so that words with recurrent occurrences get a lower weight than the more important ones.

$$S_j = \sum W_{ji}$$

Equation. 3 Sentence weight

Finally each of the split sentence is given a weight value depending on the sum of weights of words in that sentence. The sentence are now sorted in descending order on their weights and the summary is generated using the top five sentences obtained using the above method. It should be noted that a sentence can only be returned if it passes the minimum threshold value set by the client.

The inverse frequency weight were used so that the summarization function can work as an independent unit and it will not be based on the dataset provided. Any chunk of English text can be passed to the API and the user will get the summarization.

As it can be seen from Fig. 5 a wikipedia content has been summarized by the system even though it did not have any context regarding the dataset it was generated from.

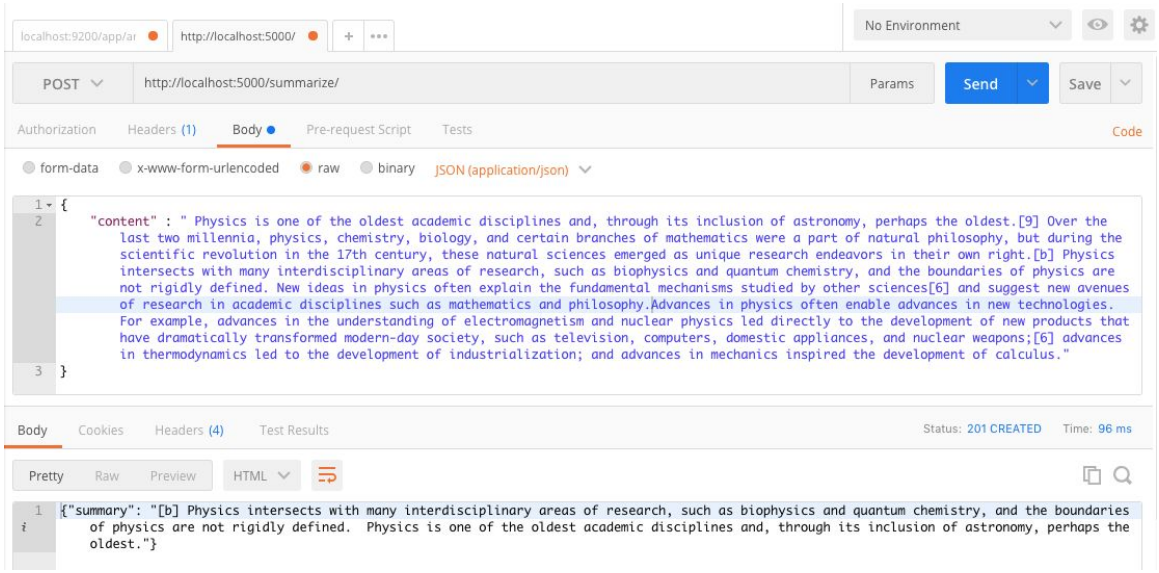

Fig. 5 Summarizing wikipedia content

3.2 Spellcheck

Important words from each of the article's title are extracted. This is accomplished by removing common stop words present in the English language (Example : is, on, the .. etc). All possible permutations are calculated for such words by breaking them into substrings. All such permutations are stored in a dictionary with the actual correct word as the key. When a new word is encoutered from the search bar, its permutations are matched with the existing database and a corrected word is returned.

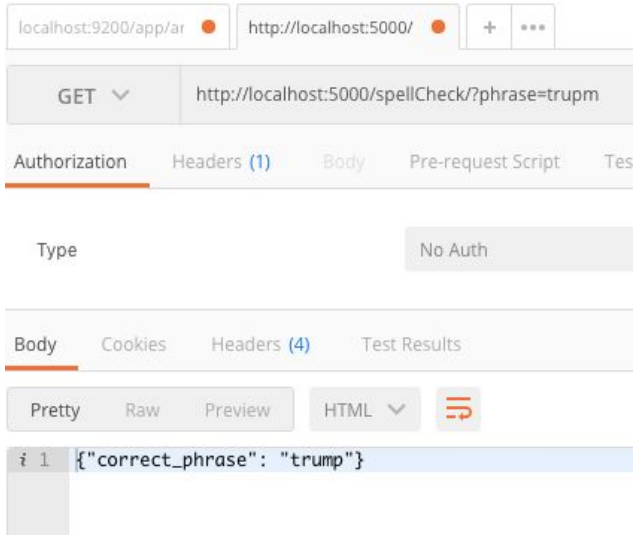

Fig. 6 Spellcheck functionality

Fig 6 shows how the word "Trump" when misspelled as "trupm" in the search bar returns the original word before the search is executed.

3.3 Autocomplete

The titles from the website's articles are used to generate trigram autocomplete suggestions. Each title is split into a set of tri-grams and stored along with the weights of each world as calculated by Equation. 1. When a user enters a phrase,word or part of a word the tri-grams with the maximum weight calculated by Equation 3 are returned to the suggestion bar.

Fig.7 shows the autocomplete suggestion for the word trump when typed into the search bar.

The spellcheck and autocorrect are tailor made for individual dataset so that the user will not be mislead to search a phrase that is not present in website's articles.

3.4 Recommendation

The recommendation function provides content based filtered articles that are similar to the one that the user is reading at the moment. Content based recommendation was used as the user get to see similar articles and at the same time will not be put into a filter bubble created by his/hers content likes and views.

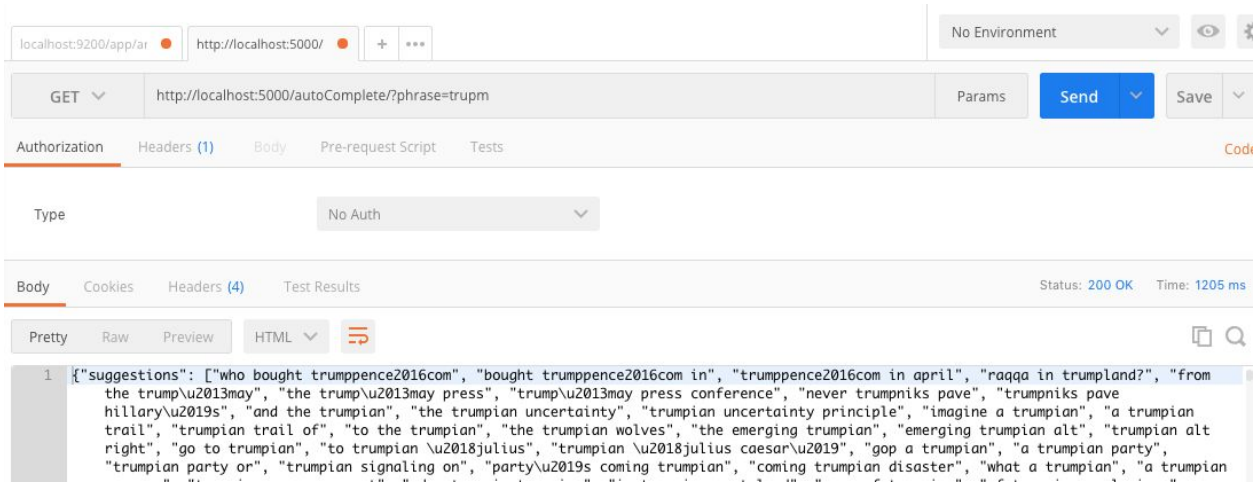

Fig. 7 Autocomplete functionality

The recommendation engine uses tf-idf of words and generates a cosine similarity matrix custom made for the websites articles. Though the engine can be deployed for any website just by passing a new database.

3.5 Frontend

User puts a search query as to what type of article he/she is looking for. The query hits our flask server and it gives suggestions based on our autocorrect algorithm. The user could select any one the suggestions and click on search. The query hits our NodeJs backend server which hits our elasticsearch to get the results. The headings of the related articles with their summaries returned from elasticsearch are displayed on the search page of the application. The summaries displayed are calculated by our summarization algorithm. The user can then click on any one of the article headlines to view the full article.

4. Future Enhancement

The current version of the system used statistical analysis to generate the summaries for each article. Currently work is on progress to generate a model that uses Deep Recurrent Neural Networks that generate summary using LSTM and the content they have already seen.

5. Conclusion

The new system generates faster summaries when compared to the ones generated by humans, though a content writer can construct a summary way more elegant, the current system generates summaries that are unbiased and does not include any content outside of the article.

6. Project Repository

6.1 Github Repo :

https://github.com/SJSU272LabF18/Project-Team-26

6.2 NLP server :

https://ec2-3-17-29-51.us-east-2.compute.amazonaws.com

6.3 Website Link :

http://ec2-34-211-101-36.us-west-2.compute.amazonaws.com:3000/

7. Reference

- **https://nodejs.org/en/docs/**
- **https://www.nltk.org/**
- **http://flask.pocoo.org/docs/1.0/**