

Project Capstone Report

Extractive Summarizer Using BERT/Transformer Model

Date of Submission: 03-04-2020

Submitted By:

|  |
| --- |
| Eugin Lee Yu Jun |
| Koay Seng Tian |

# Abstract

The goal of this project is to provide an annotation tool to accentuate key points in a corpus of text.

We utilised an extractive summariser to find representative sentences in a corpus and annotate it. The summarisation technique is powered by a Bidirectional Encoder Representations from Transformers (BERT) model.

In this report, we describe our thought process from backend code to the user interface, in addition to discussions on model selection and results evaluation.

# Acknowledgement

We would like to thank Timothy Liu and Zhang Sheng from Nvidia Singapore Development Pte Ltd for their technical assistance, advice and guidance throughout the project. Our project supervisor/mentor, Poh Keam has also been extremely supportive in providing the necessary resources and scoping for this project.

**Thank you guys! ☺**

**Table of Contents**

[Abstract 3](#_Toc36632811)

[Acknowledgement 4](#_Toc36632812)

[1. Background 6](#_Toc36632813)

[1.1 Goal 6](#_Toc36632814)

[2. Methodology and Design 7](#_Toc36632815)

[2.1 Software Data Flow 7](#_Toc36632816)

[2.2 Collecting User Feedback 8](#_Toc36632817)

[2.3 Scoring and evaluation system 8](#_Toc36632818)

[3. Findings 9](#_Toc36632819)

[3.1 BERT Models Are Not Equal 9](#_Toc36632820)

[3.2 ROUGE and not BLEU 10](#_Toc36632821)

[4. Evaluation and Analysis 11](#_Toc36632822)

[4.1 ROUGE scoring model 11](#_Toc36632823)

[5. Results 13](#_Toc36632824)

[6. Conclusion 14](#_Toc36632825)

[7. Recommendations 15](#_Toc36632826)

[8. Appendices 16](#_Toc36632827)

[8.1 GitHub 16](#_Toc36632828)

[8.2 Development Environment 16](#_Toc36632829)

[8.3 Extractive Summarizer 17](#_Toc36632830)

[8.4 Running the Summarizer 18](#_Toc36632831)

[8.5 ROUGE scoring model 19](#_Toc36632832)

[8.6 Library Versions 21](#_Toc36632833)

[9. References 22](#_Toc36632834)

[9.1 Main modules/Libraries 22](#_Toc36632835)

[9.2 Reading and Research list 22](#_Toc36632836)

# 1. Background

This project is a collaboration between Nvidia and Republic Polytechnic (RP) as a capstone project for the Tech Immersion and Placement Programme (TIPP) programme by Infocomm Media Development Authority (IMDA). The project aims to provide a real-world work environment for the students to apply artificial intelligence techniques taught in the course. Supervisors in return receive the intellectual property of the final product.

## 1.1 Goal

The goal of this project is to provide an annotation tool, to differentiate key takeaways in a corpus of text. The summarisation technique will be powered by BERT models.

# 2. Methodology and Design

Project: Extractive Summarizer using BERT transformer model.

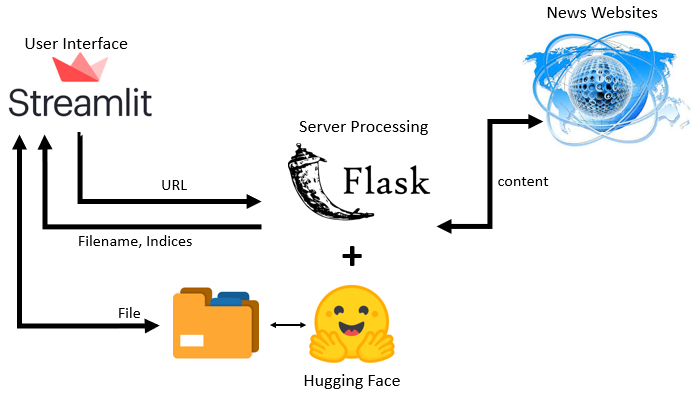
i) The main entry to the application is via a webpage where user enters a URL as an input. This BERT model summarizer aims to extract key feature sentences of the main corpus.

ii) These extracted features will then be presented as an annotated form, together with the main corpus as an output document.

iii) Using a client-server model, the web application provides seamless transition between server (Flask), user interface (Streamlit) and the underlying Python code. The software is developed using open-source software, libraries and/or modules.

iv) For evaluation, we will be using the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scoring system which is optimised to calculate distances/similarities between summarised articles. By using ROUGE and comparing between BERT generated summaries against human generated summaries, we calculate the similarity between the text and recommend ways to further improve the results.

## 2.1 Software Data Flow

  
Figure . Software Data Flow

The software is developed using available open-source application frameworks (Streamlit, Flask, Newspaper3k etc.) and Hugging Face BERT/Transformer model. Python scripting language is mostly used.

Streamlit architecture is based on mirroring a web application the same way a plain Python script is written and displayed. Streamlit applications have a unique data flow: every time a change is made on the user interface (UI), it triggers an automatic call to the server and trigger an update to the screen (for example, when the application needs to response to a button press), Streamlit will attempt to rerun the entire Python script from top to bottom.

This will pose a challenge for the application/UI developer because it is not implemented as a 'call back', like most web applications. As a remedy, the streamlit community generally recommended modifying the underlying code using Streamlit's cache decorator (i.e. streamlit@cache) which allows developers to skip certain costly computations when the application reruns. However, such technique as we have observed, may create stability issues.

## 2.2 Collecting User Feedback

User can feedback or suggest user-defined summary by checking or unchecking the returned check boxes. Check boxes are used to reflect machine or user selected summary. The original and enhanced summaries can be saved as CSV (Comma Separated Values) files for future model fine tuning and improvements.

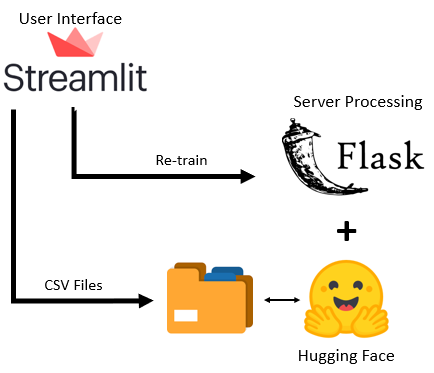


Figure 2. Capturing User Feedbacks

## 2.3 Scoring and evaluation system

User can use a ROUGE scoring system to evaluate if the computer-generated annotation is appropriate for the corpus. Scoring results is generated through a Python application, with the CSV file inputs from user feedback and BERT annotations. In terms of output, a recall score, precision score and F1 score will be generated together with a short user description.

# 3. Findings

## 3.1 BERT Models Are Not Equal

When we first started the project, we utilised the “stock” BERT model (bert-base-uncased) that was embedded in the summarizer library to run our models. While testing, we quickly realised that we may face some usability issues as the waiting time for the extractive summary to complete took around seventeen (17) seconds to complete.

According to recent research on attention spans, we discovered that our current human attention spans range between eight (8) and twelve (12) seconds. This means that at seventeen (17) seconds, users might find our program “too slow”, and may not even use the product even if it was a perfect product. As a matter of fact, seventeen (17) seconds might be long enough for them to read the article themselves! We need to find a compromise between speed and performance.

To achieve this, we started to explore different BERT models available in the market. We eliminated the “large” BERT models at first cut, because they contain more parameters which will intuitively mean longer processing time. This narrows our focus to the base BERT model. To quickly evaluate the performance of different BERT models, we run a loop test, controlling all parameters except for a model swap. In this experiment, we recorded the performance in the form of processing time and text length.

At this point, we are using just quantitative measures, such as (1) processing speed and (2) text length as our selection criteria.

**Within the BERT models, we chose DistillBERT as our default model** as its processing speed falls within the sweet spot between eight (8) and twelve (12) seconds while the summarized text length seems reasonable.

To artificially improve on the run time, we also created a time illusion (to make time seem to pass faster), by inserting a “loading animation” to stimulate an active program during the processing runtime.

|  |  |
| --- | --- |
|  |  |
| Figure 3.1. Human Attention Span Infographics (digital information world) | Figure 3.2. BERT summarizer |

## 

## 3.2 ROUGE and not BLEU

**We selected ROUGE as our main technique to measure model accuracy.** The main reason is because ROUGE provides more granularity insights into the scoring mechanism than BLEU (Bilingual Evaluation Understudy). During our project, one of the difficulties is to find a way to evaluate whether a summary is good or bad, and also a quick way to locate areas to improve. We find that ROUGE fits our requirements.

In ROUGE, the precision and recall scores are also generated. Using a low precision or recall score, we can find out whether the machine generated text, is too short or too long, or is it not relevant. With this knowledge, we could either tweak the parameters correctly, or if the summary is bad, we can fine-tune it by training the model on a longer relevant corpus. As compared to BLEU, another popular scoring technique, only the final BLEU score (between 0 and 1) is generated; however, this gives us no clear direction on how to improve our model.

ROUGE and BLEU are both popular methods in summaries evaluation. They use similar calculation methods surrounding n-grams and variations of recall and precision. When implementing, we discovered that BLEU does not seem to work very well when there are differences in corpus length (between BERT and human generated). This could be due to the brevity penalty effect or an input restriction. This was acknowledged in the BLEU documentation, which recommended a smoothing technique to overcome this issue. However, even with the adjustments, we did not find the value to be as intuitively accurate as ROUGE. Therefore, rating the techniques in terms of usability, we decided to choose ROUGE.

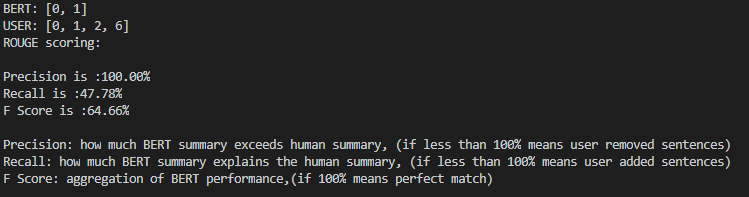


Figure 4. Rouge Score (Python)

# 4. Evaluation and Analysis

## 4.1 ROUGE scoring model

To evaluate our summarizer model, we use a ROUGE scoring model.

Using the annotated corpus derived from previous steps, we will be able to fit it into our ROUGE scoring model for scoring.

3 scenarios were considered: (1) When the translation is perfect, (2) when the translation is not perfect, and user highlighted more sentences, (3) translation is not perfect and user deleted sentence.

**i) *SUMMARY IS PERFECT: Summary is equal to user annotation***



Figure 11. SUMMARY IS PERFECT: Summary is equal to user annotation

In this case, no adjustment is needed.

**ii) SUMMARY IS NOT PERFECT: BERT has less sentences** 

Figure 12. SUMMARY IS NOT PERFECT: BERT has less sentences

In this case, we can adjust the model to allow for more summarised results.

**iii) SUMMARY IS NOT PERFECT: BERT has more sentences**



Figure 13. SUMMARY IS NOT PERFECT: BERT has more sentences

In this case, we may consider reducing the amount of summarizer results, or more fine-tuning.

# 5. Results

We successfully created and deployed a web application that allows a user to (1) enter a Uniform Resource Locator (URL) and received a full text with annotated key points, (2) user can provide feedback, which can be used to provide a quantitative scoring and (3) we could improve the BERT model based on the scoring.

Our average runtime for the model is between eight (8) and ten (10) seconds.

Project managers from Nvidia are extremely satisfied with the progress we made.

# 6. Conclusion

We concluded that the DistillBERT model is the optimal BERT model for a general news extractive summarizer. A ROUGE scoring model is the most appropriate scoring system as it provides more feedback. Using a modular coding style, a user could swop out and replace individual parts of the summarizer with minimum effort.

# 7. Recommendations

Given that the code is designed to be modular based and robust, the engineer will be able to switch out part of the code and further customised it to his project. For example, as our parser is optimised for newspaper articles, it may not work as well for PDF documents. In which case, the engineer could switch in a PDF parser instead. This instruction is similar should the engineer wants to switch a different BERT model.

The Streamlit application works perfectly for a proof-of-concept user interface; to quickly unlock the power of the underlying model and test out concept feasibility. However, the UI content has some limitations such as lack of feedback functionality (like a HTML application), and limited fine-tuning options, which limits our ability to improve the product. As an improvement, the user can migrate to a full-fletch web development environment, probably using Javascript, HTML5, Bootstrap, jQuery, for example.

In Natural Language Processing (NLP) literature, the computer-generated summary is always compared against a “gold standard” summary to determine its accuracy. However, in our case, the BERT summary is compared against to a random user feedback. This user may have its own biasness and may have an inconsistent proficiency of the subject. This could affect our evaluation score of the model. To avoid this issue, it might be useful to collect an aggregation of user responses, before tweaking the model.

# 8. Appendices

## 8.1 GitHub

The source codes, documentation and the test data for the project are hosted in Github.

Repository URL: <https://github.com/koayst/rp_capstone>

If you have issue accessing the GitHub, please contact any one of us.

Github uses Git as an open-source version control system. The purpose is to keep the revisions straight, storing modification in a central repository. This allows us, as a developer, to easily collaborate, as we can download a new version of the software, make changes and upload the newest version, after we make the modifications.

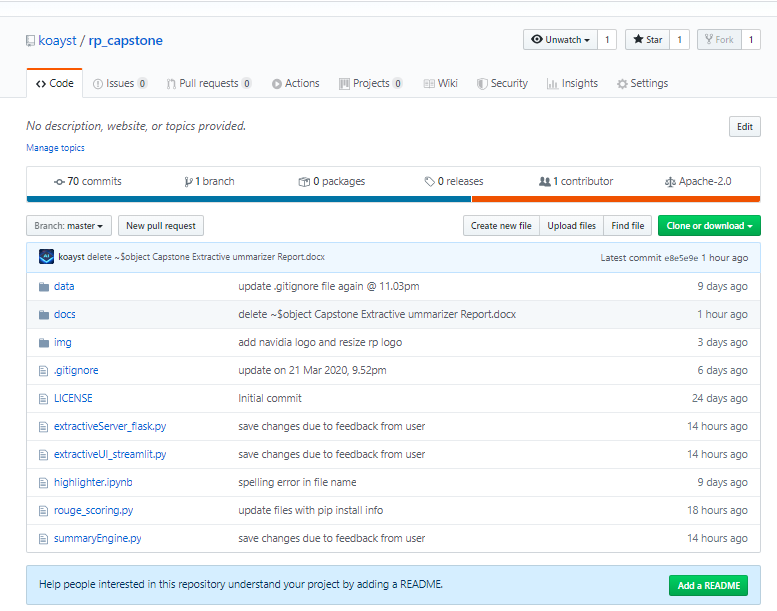


Figure 5. GitHub Repository Screen Shot

## 8.2 Development Environment

We are using Microsoft Windows Operating System (Windows 10 and Windows 8) as our development environment.

Anaconda Individual Edition (2019-10) was installed and then relevant modules are installed/updated as required by the project.

## 8.3 Extractive Summarizer

To run the system, it is advised to create a separate virtual environment. Use Git to download the source codes by cloning it from GitHub.

Install the required, Python, Python libraries and modules.

Open two terminals on a local machine (command prompt for Windows OS) i.e. one to run the Flask server and the other one to run the Streamlit webapp server.

* + Flask server: *python extractiveServer\_flask.py*
    - Running on http://127.0.0.1:5500/ (Press CTRL+C to quit)
  + Streamlit: *streamlit run extractiveUI\_streamlit.py*
    - Local URL: <http://localhost:8501>

To run the ROUGE scoring application, open another terminal to run it.

* + Rouge Scoring: python rouge\_scording.py data\testbert.csv data\testuser.csv
    - testbert.csv is the output from the summarizer engine
    - testuser.csv is the output after user feedback

Once Streamlit is running, you will notice the webapp is running in your browser. If the Streamlit web app is not running in your browser, you can bring up the Streamlit web app by issuing this URL: ‘http://localhost:8501’.

We are using Chrome browser for development and testing.

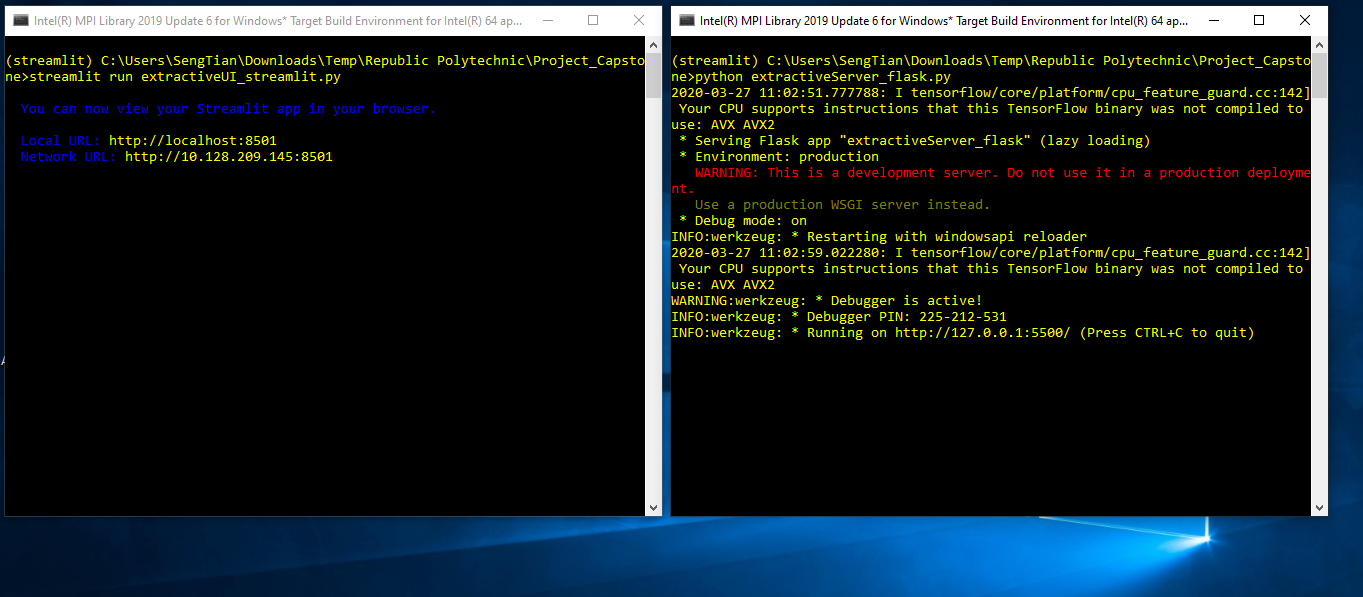


Figure 6. Screen shots for running the summarizer

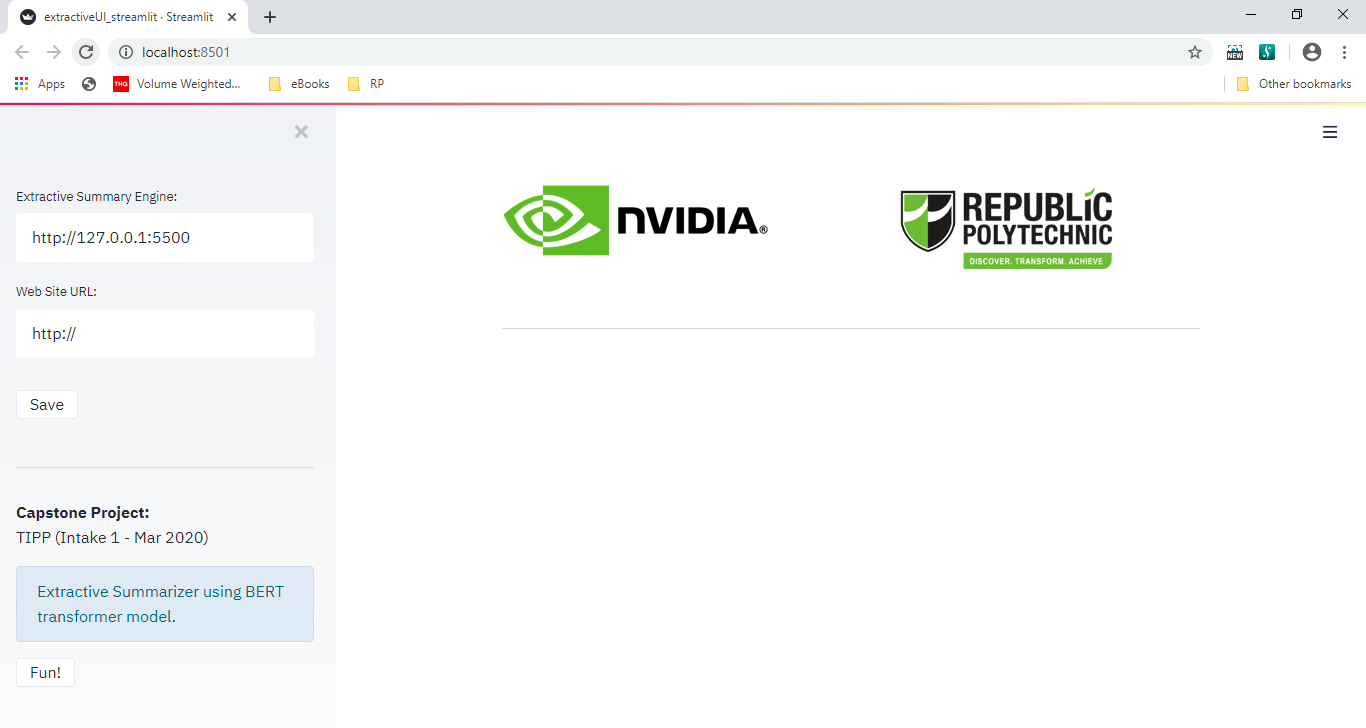


Figure 7. Streamlit webapp screen shot

## 8.4 Running the Summarizer

To test, go to a sample webpage, Channel News Asia Site (<https://www.channelnewsasia.com/>) and select a news article you want to summarize. Copy the URL and paste it to the Streamlit webapp. Press ‘Enter’ to start the summarizer engine.

The output will show as a list of checkboxes. User can enhance the modelling by checking and unchecking the checkboxes.

User’s feedback can be saved by clicking the ‘Save’ button.

Two CSV (Comma-Separated Values) files are stored in the data directory. It is highlighted as shown below

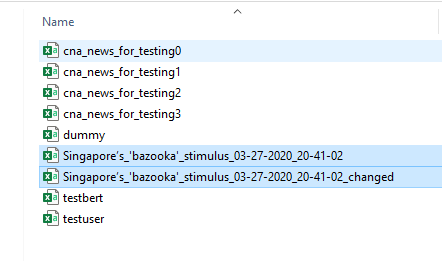


Figure 8. The CSV files needed (2 files)

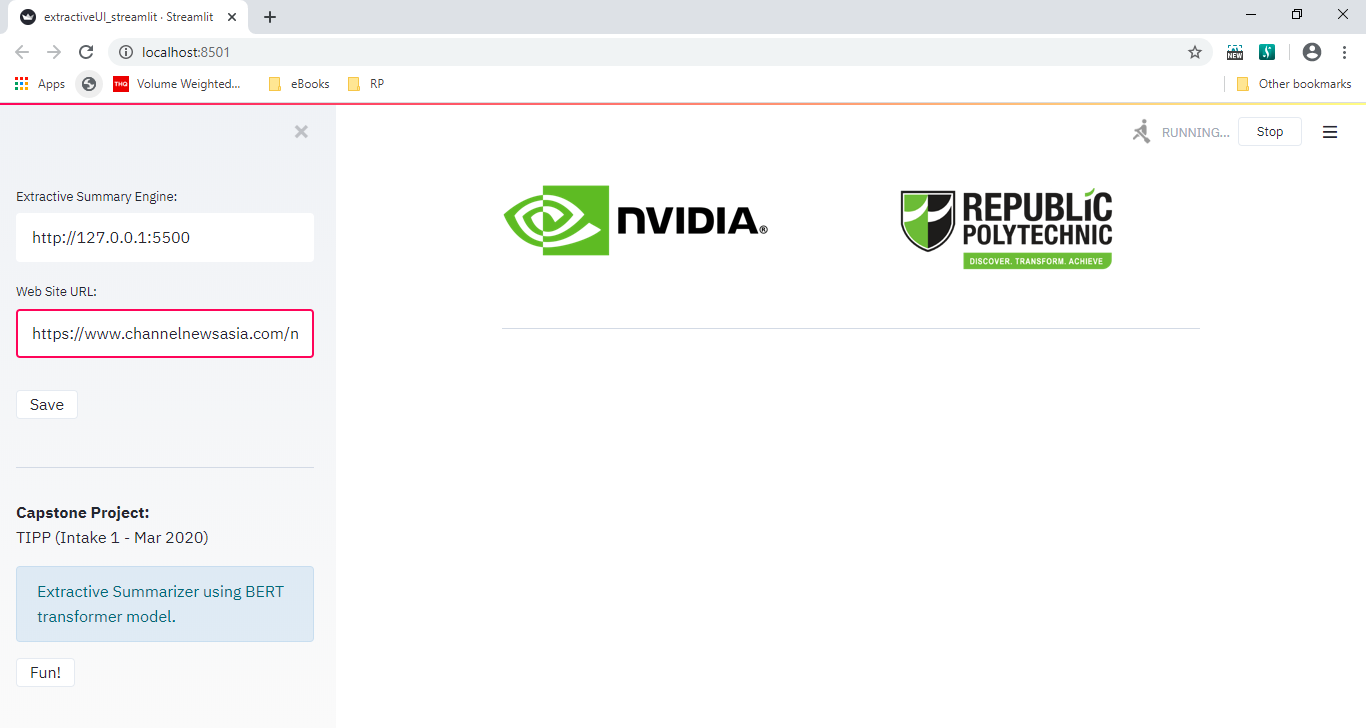


Figure 9 Input URL and Press ‘Enter’

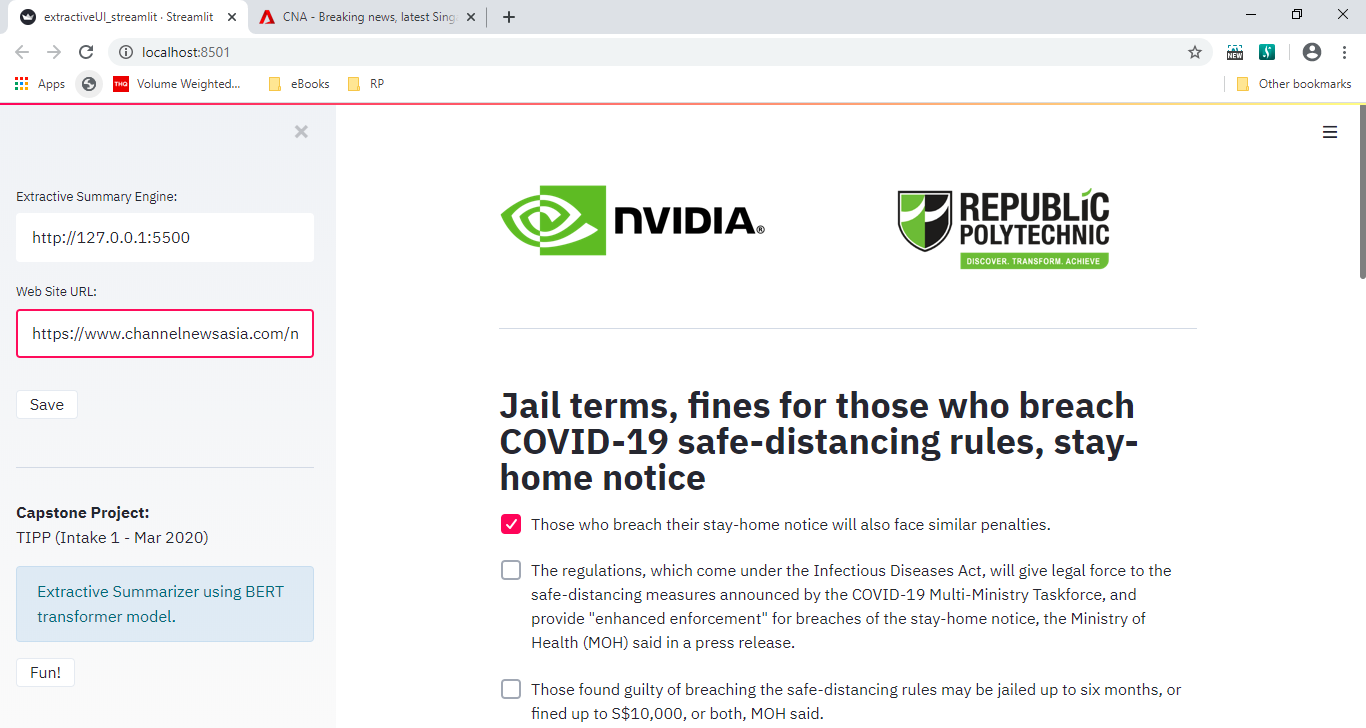


Figure 10. Output of Summarizer

## 8.5 ROUGE scoring model

From the files generated above, we will be able to fit it into our ROUGE scoring model for scoring. We display 3 scenarios: (1) When the translation is perfect, (2) when the translation is not perfect, and user highlighted more sentence, (3) translation is not perfect and user deleted sentence.



Figure 11. SUMMARY IS PERFECT: Summary is equal to user annotation



Figure 12. SUMMARY IS NOT PERFECT: BERT has less sentences



Figure 13. SUMMARY IS NOT PERFECT: BERT has more sentences

## 8.6 Library Versions

|  |  |
| --- | --- |
| **Name** | **Version** |
| Python | 3.7.6 |
| Flask | 1.1.1 |
| Keras-applications | 1.0.8 |
| Keras-preprocessing | 1.1.0 |
| Matplotlib | 3.1.3 |
| Newspaper3k | 0.2.8 |
| Numpy | 1.18.1 |
| Pandas | 1.0.1 |
| Pytorch | 1.4.0 |
| Regex | 2020.2.20 |
| Request | 2.23.0 |
| Rouge | 1.0.0 |
| Streamlit | 0.56.0 |
| Tensorflow | 2.0.0 |
| Tokenizers | 0.5.2 |
| Torchvision | 0.5.0 |
| Transformers | 2.2.0 |
| Urllib3 | 1.2.58 |

# 9. References

## 9.1 Main modules/Libraries

This project was implemented using Flask Python. Flask is a popular Python web framework for developing web application.

* Flask - <https://flask.palletsprojects.com>

Streamlit is a recent new tool that allows engineers to quickly build interactive web application around the data.

* Streamlit – <https://www.streamlit.io>

## 9.2 Reading and Research list

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Articles** | **Summary** | **Done By** | **Type** | **URL** |
| 1 | Unsupervised Text Summarization using Sentence Embeddings | Flow of a text summarisation process | EL | Web link | <https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f> |
| 2 | Awesome NLP | A linkful of NLP code examples | EL | github | <https://github.com/keon/awesome-nlp> |
| 3 | Pytorch Tutorial | Get a quick basic tutorial on Pytorch. Not sure whether to learn Pytorch in deep. | ST | Web link | <https://pytorch.org/tutorials/> |
| 4 | Pytorch Guide | A Beginner-Friendly Guide to PyTorch | ST | Web link | <https://www.analyticsvidhya.com/blog/2019/09/introduction-to-pytorch-from-scratch/> |
| 5 | Scrape and Summarize News Articles in 5 Lines of Python Code | Module: newspaper3k scraping | EL | Web link | <https://towardsdatascience.com/scrape-and-summarize-news-articles-in-5-lines-of-python-code-175f0e5c7dfc> |
| 6 | Mark.js | Highlighting JavaScript | ST | Web link | <https://markjs.io/> |
| 7 | Stools | JavaScript to highlight web text | ST | Web link | <https://www.nsftools.com/misc/SearchAndHighlight.htm> |
| 8 | Newspaper article scraping | How did I scrape news article using Python ? | ST | Web link | <https://medium.com/@ankurjain_79625/how-did-i-scrape-news-article-using-python-6eff936b3c8c> |
| 9 | Bert based summarizer notes | Extractive (we need) and abstractive summarization | EL | Web link | <https://medium.com/lsc-psd/a-bert-based-summarization-model-bertsum-88b1fc1b3177> |
| 10 | BERT based extractive summarizer | Extractive summarizer | EL | Web link | <https://pypi.org/project/bert-extractive-summarizer/> |
| 11 | List soup | How to compare lists | EL | Web link | <https://www.techbeamers.com/program-python-list-contains-elements/> |
| 12 | incipits | A Python based HTML to text conversion library | ST | Library | <https://pypi.org/project/inscriptis/> |
| 13 | distillBERT | a light BERT, released recently | EL | Web link | <https://github.com/huggingface/transformers/tree/master/examples/distillation> |
| 14 | Transformer library | Shows all the BERT model, use this to optimise | EL | github | <https://github.com/huggingface/transformers> |
| 15 | NLP - The Age of Transformers | 1-Building a machine reading comprehension system using the latest advances in deep learning for NLP. | ST | Web link | <https://blog.scaleway.com/2019/building-a-machine-reading-comprehension-system-using-the-latest-advances-in-deep-learning-for-nlp/> |
| 16 | Understanding text with BERT | 2-Building a machine reading comprehension system using the latest advances in deep learning for NLP | ST | Web link | <https://blog.scaleway.com/2019/understanding-text-with-bert/> |
| 17 | BERT Research | BERT Concept - #1 | ST | YouTube | <https://www.youtube.com/watch?v=FKlPCK1uFrc> |
| 18 | Wordpiece model and Tokenizer | BERT Concept - #2 | ST | YouTube | <https://www.youtube.com/watch?v=zJW57aCBCTk&pbjreload=10> |
| 19 | Fine Tuning | BERT Concept - #3 | ST | YouTube | <https://www.youtube.com/watch?v=Hnvb9b7a_Ps> |
| 20 | Blog site for the above |  | ST | Web link | <http://mccormickml.com/2019/11/11/bert-research-ep-1-key-concepts-and-sources/> |
| 21 | Summarizer | Documentation | EL | github | <https://github.com/icoxfog417/awesome-text-summarization> |
| 22 | Fast Bert | Bert models with potential retraining | EL | Web link | <https://pypi.org/project/fast-bert/> |
| 23 | Fast Bert | Overview | EL | Web link | <https://medium.com/huggingface/introducing-fastbert-a-simple-deep-learning-library-for-bert-models-89ff763ad384> |
| 24 | ROUGE scoring | Evaluation method | EL | weblink | <https://en.wikipedia.org/wiki/ROUGE_(metric)> |
| 25 | ROUGE explanation | Interpreting rouge scoring | EL | weblink | <https://stats.stackexchange.com/questions/301626/interpreting-rouge-scores> |