Text Summarization for Wikipedia Articles

Prasoon Goyal
Courant Institute of
Mathematical Sciences
New York University
pg1338@nyu.edu

Sahil Goel
Courant Institute of
Mathematical Sciences
New York University
sg4187@nyu.edu

Kshitiz Sethia
Courant Institute of
Mathematical Sciences
New York University
ks3839@nyu.edu

Abstract

With the increasing amount of textual data, automatic summarization has become an important task in natural language processing. In this work, we consider the summarization of Wikipedia articles, using graph-based and machine learning approaches.

1 Introduction

Automatic summarization is a key task in natural language processing. The problem of automatic summarization can be formulated as follows (taken from Wikipedia): Automatic summarization is the process of reducing a text document with a computer program in order to create a summary that retains the most important points of the original document.

Automatic summarization tasks can be divided into categories based on several criteria. If the summary is generated by selecting sentences from the given document, it is referred to as *extractive summarization*. On the other hand, if the summary is constructed by paraphrasing the most important content of the given document, it is referred to as *abstractive summarization*. If the summarization task involves summarizing the content of a single document, it is called *single document summarization*, while summarizing the content of multiple documents into a single summarized text is called *multi-document summarization*. In this work, we consider the problem of **single document extractive summarization**.

2 Problem definition

We consider the following problem of automatic summarization of Wikipedia articles. Given a Wikipedia article, generate an extractive summary of the article. To evaluate our system- generated summary, we compare it against the introduction section of the original Wikipedia article, which is roughly a summarization of the most important content of the entire article for a large number of Wikipedia articles. Therefore, our system takes a Wikipedia article *without the introduction section* and generates a summary for the same.

3 Overview of the pipeline

Our pipeline for the entire task can be summarized as follows:

1. **Procuring data**: We worked with `enwik9` which is the first 10^9 bytes of the dump of English Wikipedia on March 3, 2006, which contains 243,426 article titles, of which 85,560 are #REDIRECT to fix broken links, and the rest are regular articles. [1]

2. **Cleaning up**: The data in the dump are present as a single file that has all the articles in XML format concatenated together. We convert this to plaintext format, removing all XML tags, and other redundant information.

3. **Filtering summarizable articles**: A large fraction of the articles are such that the introduction section is far from an approximate summary of the remainder of the article. We use a heuristic to filter out such articles. The details are described in section 4.

4. **Summarization**: We use two kinds of models to generate summaries. The first one involves sequentially selecting sentences that are maximally similar to the rest of the article. The second one involves generating a sentence score for each sentence that estimates the likelihood of the sentence being in the summary, using a regression model. The details of these models are given in sections 5 and 6.

[1] https://cs.fit.edu/ mmahoney/compression/textdata.html

5. **Evaluation**: We use ROUGE metric for evaluating our performance, by comparing our system generated summary with the introduction section of the article. For baseline, we use a summary that is generated by selecting random sentences from the given article.

4 Preprocessing

4.1 Data cleaning

The plaintext sources of the Wikipedia articles require significant preprocessing, including the following tasks:

- Removing XML tags
- Removing metadata such as sections "See also" and "References".
- Removing infoboxes, tables, lists
- Modifying links to retain only the text of the link
- Removing formatting markups

For preprocessing, we primarily used a Perl script [ref] modified to suit our task. [2]

4.2 Filtering summarizable articles

We found that for several articles, the introduction section could not be considered a summary of the remainder of the article for one or more of the following reasons:

- The introduction section of the article is empty or contains very few sentences.
- The remainder of the article is empty or contains very few sentences.
- The size of the introduction section is not comparable to the expected size of the summary of the article.

We used the following approach to filter out the articles of the above types. First, any article with less than 200 words in the introduction section or the rest of the article is discarded. Next, if the ratio of the number of words in the introduction section to the number of words in the remainder of the article is not between 7% and 37%, we discard the article.

After this filtering operation, we end up with 3789 articles.

4.3 Sentence chunking

To obtain sentences from the processed text, we use the following regular expression:

$$regex$$

We found the performance of this to be reasonably good.

5 Optimal extractive summary

Evaluating our generated summary with the first section of the articles using ROUGE measure did not give promising results. We hypothesize that many articles, even after our filtering process, have introduction paragraphs that cover significantly different topics than the rest of the article. Therefore, we attempted to find the best possible ROUGE score between an extractive summary from the given article and the original summary. For a summary of size k for an article with n sentences, this would require computing $\binom{n}{k}$ ROUGE values, which is clearly intractible. Thus, we came up with the following heuristic to approximate the best ROUGE score possible.

For each sentence in the original summary, we compute its ROUGE score with each sentence in the remainder of the article, and pick the sentence with the highest score. All these sentences are put together to create an approximately-optimal summary.

6 Graph-based approaches

We represent the article as a fully connected graph in which each sentence is represented as a node, and the weight of the edge between two sentences is the similarity between the two sentences. To compute the similarity between the sentences, we used the following approach:

1. For each word in sentence 1, compute its similarity to all words in sentence 2, to get mn similarity values, where m and n are the number of words in sentences 1 and 2 respectively.
2. From these mn similarity values compute the average of the maximum k values.

[2] We started by collecting the plaintext sources of 5000 most popular Wikipedia pages from [ref], implemented data cleaning from scratch, but could not get a perfectly working system. As such, we found the Perl script along with the enwik9 dataset, and switched to the same.

For similarity between words, we used the following two metrics

- Cosine similarity between word embeddings of the words.
- Negative Euclidean distance between word embeddings of the words

where we experimented with pretrained Google Word2Vec [ref] and GloVe [ref] to get word embeddings for words.

After constructing the graph, we incrementally construct the summary by picking the node with the highest sum of weights of outgoing edges, and modifying the graph at each step to remove the selected node and edges incident on it.

7 Machine learning-based approaches

We modelled the summarization task as a regression task in which, given a sentence from an article, the model seeks to predict the likelihood of the sentence being in the summary.

For these approaches, we performed a random split of our 3789 articles into 3000 training articles and 789 test articles.

We used the following features for a sentence in an article:

- **Sentence level features** : This included the word embedding representations of the top k words in the sentence ranked by their tf-idf scores. We took $k = 5$, and $50-$dimensional word embedding vectors to get a $250-$dimensional vector for the sentence level features.
- **Context level features** : This included the word embeddings of the neighboring sentences. We considered the previous three and the next three sentences. To get an overall sense of each of these neighboring sentences, we added the word embedding vectors of the top 5 words in each sentence to get a $50-$dimensional vector for each sentence. This resulted in a $300-$dimensional vector for context level features.
- **Paragraph level features** : To capture the overall topic of the paragraph to which the current sentence belongs, we used Latent Dirichlet Allocation (LDA), which is a well-known technique for topic modelling. We trained our LDA model on all paragraphs in the training corpus.
- **Article level features** : We used LDA to generate article level features as well, training a model on all articles in the training corpus.

A crucial parameter in LDA is the number of latent topics. We used the values (35, 15) for paragraph and sentence level features respectively, decided by some manual inspection. The values were chosen keeping in mind that the paragraph level topics should be much more fine-grained than the article level topics and that the number of LDA-based features in the feature vector of a sentence should be a significant fraction of the entire feature vector for the To generate the target value of a sentence, which according to our model corresponds to the likelihood of the sentence being included in the summary, we computed the ROUGE measure of the sentence with all the sentences in the actual summary and taking an average of the top k values for robustness. We chose $k = 3$ in our experiments.

We learn a support vector regression model with radial basis function kernel on the dataset thus generated.

8 Experiments

In this section, we briefly describe the approaches we tried and the problems encountered.

8.1 Preprocessing

The first step was to preprocess the articles removing any unwanted characters, and metadata.

We then implemented a method to filter summarizable articles from the rest. For this, we considered keeping articles that had a high similarity between the introduction section and the remainder of the article. To compute the similarity, we converted the introduction section and the remainder of the article to their respective bag of words models, scaling each word by its tf-idf score. We implemented the code to compute tf-idf scores. We tested this filtering on the 5000 most popular Wikipedia articles [ref], but ended up with a set of articles that were still not all summarizable. In particular, many of these articles had very few sentences in the introduction section, or very few sentences in the remainder of the article or the

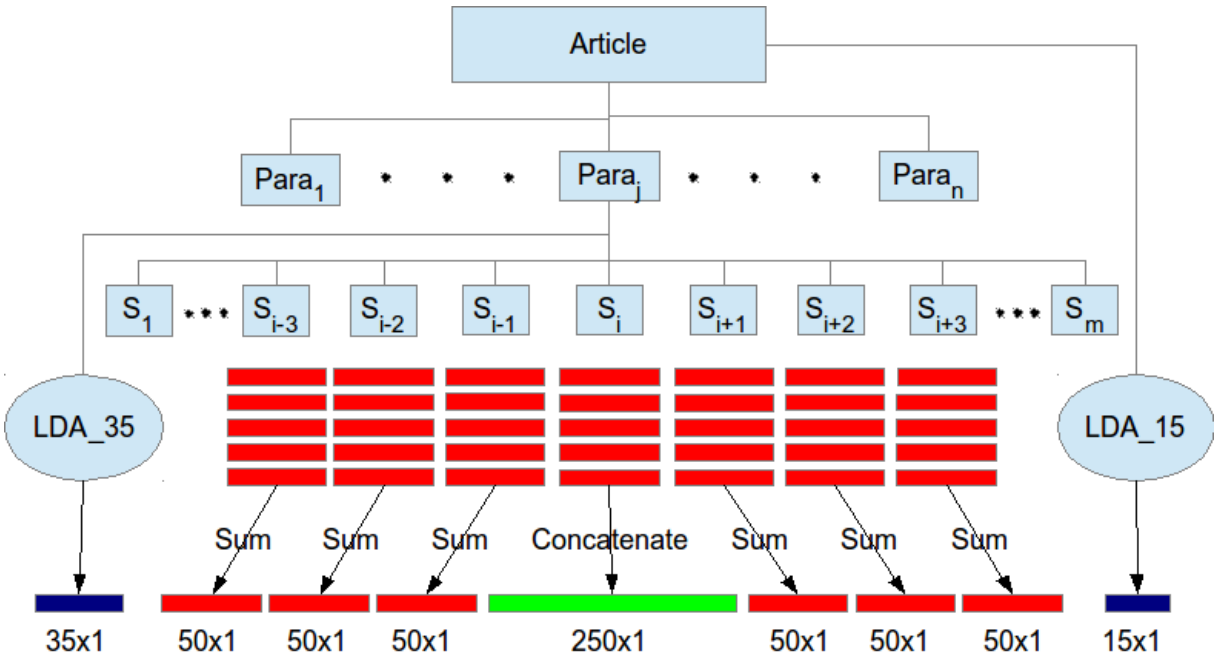

Figure 1: Schematic diagram of feature vector for machine learning based approach

size of the introduction section was too small or too large compared to the remainder of the article. Hence, in addition to the cosine similarity, we imposed a penalty term based on the lengths of these quantities as follows. We computed a summary by picking the first sentence of each paragraph from the remainder of the article, to get a summary of length l_s. Denote by l_d the number of words in the remainder of the article, and l_{os} the number of words in the original summary. Then, our penalty function for an article was given by

$$\sqrt{\frac{l_s - l_{os}}{l_d - l_{os}}}$$

which was subtracted from the cosine similarity computed earlier. Filtering out articles based on a reasonable threshold left us with only a few hundred articles to work with.

We looked for a larger collection of articles, and came across `enwik9`, which has about $130,000$ regular articles. Along with it, we also found a preprocessing script written in Perl that we found to be more robust than our implementation. We ran our filtering code on this larger dataset, but the tf-idf computation on such a large dataset was taking extremely long. We also tried using Gensim's implementation of tf-idf, but that was also found to be very slow for such a large dataset. Therefore, we modified our filtering scheme to consider only the penalty function. This was also found to work reasonably well, because the cosine similarity values that were used earlier were quite noisy in many cases.

We also experimented with the following penalty function

$$\max\left\{1 - \frac{10x}{9}, \frac{10x - 1}{9}\right\}$$

where $x = \dfrac{l_{os}}{l_d}$. This function takes the value zero when $x = \frac{1}{10}$, and increases linearly as x deviates from this value in either direction. This penalty function was found to be better than the previous one, and therefore was used as the final filtering function.

8.2 Graph-based approach

As described in section [ref], to compute the similarity between sentences, we computed the similarity between the word embeddings of all pairs of words. For this, we used Google Word2Vec [ref] that is pretrained on Google News dataset.

8.3 Machine learning-based approach

This step required using the following libraries:

- Gensim's tf-idf: To compute the top k words in a sentence, we used Gensim's implementation of tf-idf trained on our corpus.

- GloVe: To compute the word embeddings of the top k words, we intended to use Google Word2Vec for consistency with our graph-based approach, but since the pre-trained vectors were $300-$dimensional, it would have resulted in very large feature vectors. Therefore, we used GloVe word embeddings, which are $50-$dimensional vectors.

- Gensim's LDA: We used Gensim's LDA implementation to train our LDA models on paragraph and document levels.

- Scikit-learn's SVR: Finally, we used `sklearn` implementation for support vector regression using 5-fold cross-validation to train our model

8.4 Other comments

It is also worth mentioning that because our entire pipeline consists of several steps, and each step takes a quite a while to execute, a the need for a small change in the pipeline based on results from a later stage require running the entire pipeline over again, making the overall experimentation fairly time-consuming.

9 Results

10 Conclusion

Acknowledgments

ROUGE, Word2Vec, Gensim, NLTK

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual*. American Psychological Association, Washington, DC.

Association for Computing Machinery. 1983. *Computing Reviews*, 24(11):503–512.

Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.