Text Summarization in Hindi Corpus

Prajneya Kumar - 2019114011
Shivansh S. - 2019114003
Tejasvi Chebrolu - 2019114005

Human Evaluators: Eshika Khandelwal, Shubhankar Kamthankar, Trisha Kaore, Yash Agrawal, Vidush Bhartari, Abhinav Menon, Shashwat Singh

Introduction

Text summarization, in general, refers to the process of shortening long pieces of text without losing its essence. The intention is to create a coherent, logical, and fluent summary containing the main points in a document. Efficient text summarization is required in today's world as it helps people access large chunks of data in a short period of time, especially in particular languages.

Majority of the research work carried out so far has been emphasized more on widely used English and other European languages. Indian Languages have been explored little because of the amount of information available in non-English language is relatively less.

We approach this very problem of text summarization, focusing on Hindi as a language for documents. We explore and analyze different techniques for the same, and finally compare two of our own summarization systems with multiple summary evaluation techniques.

The summarizers we built currently produce summaries for single text documents in Hindi, our corpus being Wikipedia Articles. The system is built on rule-based and extractive techniques, which are discussed in the following topics.

Literature Review

There are a lot of literature which exists for Text Summarisation, but a general pattern observed is that they are not transferable to Indian Languages. This is either because they are restricted to patterns observed in the language or due to lack of resources in IL. Some techniques we reviewed which could be helpful in some way or the other include:

1. Text Summarization using Clustering Technique and SVM Technique [4]: Clustering technique mentioned here takes a group of documents and clusters them together by how semantically close they are. This distance is determined by cosine distance.

2. Multi-Document Summarization using Distributed Bag-of-Words Model [3]: Uses Distributed Bag of Words Model to represent documents and distances.

3. An Improvised Extractive Approach to Hindi Text Summarization [6]: This uses thematic roles based approach which in itself is based on frequent words approach.

4. Text Summarization of Hindi Documents using Rule-Based Approach [1]: The paper suggests using a set of 9 handcrafted rules to determine the importance of each word. The sum of the importance of each word is equal to the importance of each sentence.

5. Rouge [5]: A Package for Automatic Evaluation of Summaries: The paper talks about the different kinds of Rouge methods and why they are a good indication on the accuracy of a summary.

Summarizer

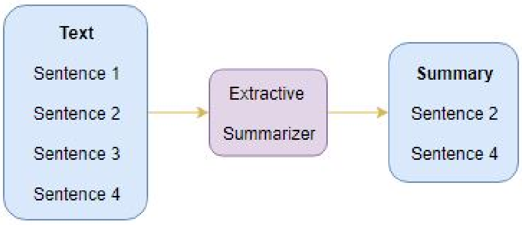
A summarizer is any piece of software that converts a long piece of text into a shorter, more concise one. This helps users save time, and prevents them from going through long articles to find what they are looking for. A good summarizer is one that manages to capture the essence of the entire article with it being as short as possible.

There are two types of summarizers based on the process/algorithm followed by the summarizer. They are:

Extractive Summarizer

Extractive approach involves of extracting the most important phrases and sentences from the documents. It then merges them to create a concise summary. Here, each and every part of the summary is a part of the original text.

Flow Chart

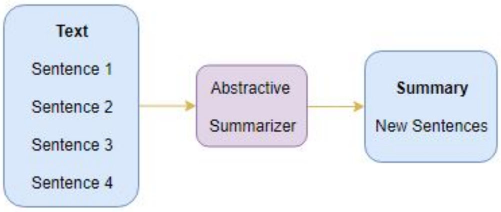


This is the method that we are following for our summarizer.

Abstractive Summarizer

The abstractive approach involves summarization based on deep learning. It generates new phrases and terms, based on points from the original document. It keeps the essence of the article the same, just like how humans summarize. Therefore, it is much harder than the extractive approach.

Flow Chart



Methods

Proposed Summarization System - I

The following procedure was followed to build the summarization system:

Step 1: Elimination of Stopwords

A collection of 264 stopwords of the Hindi language was fetched, and the article was cleaned off of these words. Stopwords are semantically null words, which happen to be commonly used words such as ’मैं’, ’यह’, ’की’.

Step 2: Hindi Morphological Analysis

The text document after the removal of stopwords is stemmed. Stemming is a procedure where each of the words in the document is stripped of their inflections. We could not find a proper Stemmer for Hindi, so a naive algorithm that strips suffixes based on different lengths was employed.

Suffix Strip Algorithm

The Suffix Strip algorithm is implemented as follows:

Step 1: Suffixes of five different lengths are listed in an array as can be seen in the code file attached.

Step 2: For the word, each suffix is checked. If a suffix is found, it is removed, and the stem word is obtained.

Step 3: All stem words, including the original word is stored in an array as potential radix terms.

This algorithm is naive and can produce wrong results at times. However, wrong outputs are not of our concern, as they get automatically discarded in the DTM (Document Term Matrix) that we build in our next step.

For example, the word ’मराठा’ contains the suffix ’ा’, which is present in our array of suffixes. Removal of this suffix will lead to the stem word being ’मराठ’, which is wrong. However, our potential radix terms arrays store both ’मराठा’ and ’मराठ’, and when the DTM is built, ’मराठ’ automatically gets discarded since there is no occurrence of the same in the entire document.

Step 3: Document Term Matrix

Now that the document contains only morpheme terms of the semantically significant words of the text document, these words are sorted based on their frequency of occurrence. A DTM (Document Term Matrix) is built for this procedure, iterating over the entire text.

A DTM is basically a matrix, with documents designated by rows and words by columns, that the elements are the counts or the weights, in our case, the weights being the frequency of occurrences.

In this proposed model, we consider each sentence in the article to be a document to proceed forward with. Therefore we have our DTM as a two-dimensional matrix whose rows are the terms and columns are the documents, so each entry (i, j) represents the frequency of term i in document j.

Step 4: Sentence Scoring

Each of the sentences in the original document is now scored according to the following formula:

$$Score = \sum \frac{DTM[i,j]}{|Terms|}$$

where DTM[i, j] represents the value of i^{th} word in the j^{th} sentence, according to the Document Term Frequency Matrix, and |Terms| represents the number of terms in the sentence.

Once the sentences are scored, we take the top 0.3*N of them based on their score, where N is the Number of sentences in the original article. Then we concatenate them according to the order of occurrence in original article.

Proposed Summarization System - II

The algorithm used here is similar to System I, but has lots of improvements added.

Step 1: Elimination of Stop Words

This is done by using Spacy. We import a predefined set of Stop words from the Hindi model so as to cater to our needs. These were then removed from each and every article, since they are semantically void words, which would give us problems during next steps.

Step 2: Data Cleaning

Basic data cleaning was done on top of stop words removal. This involved usage of regex. Here we removed any extra white spaces, escape sequences, and all English alphabets which might have crept in. This is done to ensure that the data we use our model on is clean and will not give biased result due to format of the article.

Step 3: Calculating TF

We first download a set of Hindi Wikipedia articles[2]. It gives us 55k Hindi articles to work on, to calculate TF.

To calculate TF, we use a simple formula

$$\text{TF(Word, Document)} = \text{Freq. of Word in that Document}$$

Later, we improved this calculation by involving Lemmatized words instead of actual word, to give a more accurate result when calculating TF. Since we could not find a pre-existing Lemmatizer for Hindi, we ended up training our model on hin-dep-parser-treebank. We used trained a Spacy model on it so as to use lemmatizer function. This resulted in a new formula which was:

$$\text{TF(Word.lemma, Document)} = \text{Freq. of Word.lemma in that Document}$$

This came at a huge cost of time complexity. Since lemmatizer took a lot of time to produced the desired result, there was a trade off to be made. Either stick to the old formula which had 55k articles, to increase the range of calculation of IDF (which had to be done later) at the cost of decreased TF or to use the lemmatizer to get a more accurate representation of TF and IDF with around 1k articles.

Finally, the decision was made to use the Lemmatizer, since the accurate description of words in articles was deemed more important. This was done with a small condition. We took only the articles with >100 lines, so as to ensure we have a lot of words to get proper TF. This ended up making our generated summary more accurate.

Step 4: Calculate IDF

First, we ended up calculating the DF for all the words only in the article. Firstly, we sent the article of choice by the user to the TF calculator, so it is included as well. Then we calculated DF, initially based on a simple formula

$$\text{DF(Word.lemma)} = \text{No. of Documents having that word.}$$

We calculate the RHS based on Pre calculated TF. If TF(word, doc) exist, within all the documents, then DF is incremented by 1 for each doc it exists in.

From DF, we calculate IDF using the formula

$$\text{IDF(Word.lemma)} = \log \frac{len(document)}{df(word.lemma)+1}$$

To improve upon this model, some amount of customisation was done. We observe that DF is a good representation of whether a word is common or not among documents, but it does not represent how common it is in all documents. To solve this problem, so as to create a better representation based on our needs, we change the formula a little bit.

Since we had calculated TF for each word in each document, this makes our life easy in the current step.

$$\text{DF(Word.lemma)} = \sum_{d \in D} \text{TF(Word.lemma, d)}$$

Here, D represents the set of all documents. So what this does is instead of calculating just how many documents it is in, it calculates how frequently it occurs in each document. Essentially, we end up calculating number of times a word has occurred in all the occurrences. Inverse of this is calculated using the same formula, and hence this is a normalised value of TF.

Step 5: Calculate TF-IDF

It is important to note that we have calculated IDF for only the words present in the specified article by the user. This is done because we only need to calculate TF-IDF for the given article. This makes the formula into

$$\text{TF-IDF(Word.lemma)} = \text{TF(Word.lemma, article)} \times \text{IDF(Word.lemma)}$$

Where article refers to the article inputted by the user. This then gives us the TF-IDF of each and every word in the sentence

Step 6: Ranking of Sentences

Once we find the TF-IDF of each word in the article, now we have to rank them according to the importance. Here as well, two approaches were thought off, and we applied the second since that was more robust and accurate.

The first approach was a naive one, and involved sorting the words according to their TF-IDF values. Once those values were obtained, we would find the sentences which had the first occurrence of that word, and append it to a list. If that sentence already existed in the list, we would look at the next occurrence of that word. This resulted in a model which extracted sentences based on words, and thus resulted in a less-than-decent summary in general.

The final approach applied was a three step process. The first step was to calculate the normalised sum of TF-IDF for a sentence. The basic formula used here was:

$$TF - IDF_i = \frac{\sum_{x=0}^{n} TF-IDF(Word_i)}{n}$$

Where n represents number of words in the i^{th} sentence of the article. What this ends up giving is the average amount of important words each sentence has.

The second step involves identifying whether a given word occurs in the article heading or not. Since the document heading is not specified in our datasets, a heuristic is used. This heuristic assumes that the first line of a wiki article will be the most informative, and will consider as the Header line. From this the stop words are removed. A weight $W_1 = 15$ is assigned to each word in a non heading sentence which occurs in the header line. This modifies the above formula as

$$TF - IDF_i = \frac{\sum_{x=0}^{n} TF-IDF(Word_i)}{n} + W_1 \times \text{Number of Header words}$$

Where $W_2 = 7$ and $W_3 = 5$.

It is to be noted that these values of the weight were experimentally determined, and if in future it is observed that a more suitable value of W_i exists, it is easily changeable.

Step 7: Forming the Summary and Wordcloud

We use a heuristic which determines that summary of an article should be only 30% of its length. We have slightly modified it to be 30% of number of sentences in the original article, in lieu of how we have designed our model. Firstly, we sort all the sentences according to the calculated $TF - IDF_i$. Once we have done that, we then extract the top 30% of the sentences in another list. From that list, we start forming our summary by appending each sentence in the order in which it occurred in the original article.

Once we have our final summary, we create a wordcloud so that the user can easily determined the most frequently used words. We ensure that we have removed all stop words while doing this, lest they may dominate and give an inaccurate representation of our summarizer.

Observations and Insights

We have done much Experimentation in Proposed Summarisation System 2. Comparing the Second System to the First, we need to observe what are the differences in the processes and what do they translate into.

We observe that the first system only looks at the given article. It is unfavourable to our frequency-based model since it determines the importance of words, and hence sentences only based on a single document. On the other hand, by taking IDF (or Modified IDF) in the Second System, we incorporate the data about frequency presented by a collection of documents. This difference would change the ranking system, and thus the sentences being chosen.

The next point of observation revolves around the usage of 'sentence' to sort importance instead of 'words'. It was observed, based on multiple Human Evaluation, that the former was preferred to the latter. This observation can be attributed to how a sentence can contain a single meaningful word, but nothing else. In such cases, it would end up generating a less to-the-point summary than the former would have. This approach also results in a higher number of irrelevant words occurring in summary.

Another observation was that instead of applying standard TF-IDF if we were to apply modified TF-IDF, which is adjusted according to Linguistically useful heuristics, we arrive at a better summary. This is seen when we apply Lemmatizer instead of a Stemmer and when we modify the TF-IDF formula to be more logically appropriate.

Here, all heuristic we have used are language independent. As long as there exists a Lemmatizer, Set of Stop Words and Dataset of Articles (for the second system), we will be able to reproduce the results effortlessly. This approach leads us to the insight that if there were to be any language-dependent heuristic, it would only affect the second system. Any such heuristic, in the second system, would be easily encoded as weights assigned, if they were to exist. Therefore it can be said that our method is linguistically independent of languages, but instead focusses on how information is coded in language.

Evaluation Methodology

Human Evaluation

Once we have generated a whole bunch of summaries, it is important for us to actually evaluate how accurate our models are. For this it is essential that the Gold Standard we define is as close to the assumption that we have taken, and is unbiased.

To remove biases, we have involved external human evaluators. We had 7 of them, and each one of them was given a single article. They were given the instruction that if there article had N sentences, they had to extract the 0.3N most important sentences according to them which would form the best summary possible. It was made sure that no information about how information value of a sentence being determined was given to them, and no feedback was given regarding quality of summary until the whole process was finished.

Except this, human evaluators were involved in Proposed Summarization System - II as well, where human evaluators were given an article and two different summaries, to evaluate which one was more informative and coherent. This helped us to get the best steps possible to improve the accuracy of our model.

ROUGE

ROUGE stands for *Recall-Oriented Understudy for Gisting Evaluation*. The quality of the summary is determined by comparing summaries made by humans with the machine-generated summary. There are several references created by humans and the generated candidate summary by machine to evaluate the model. The intuition behind this is if a model creates a good summary, then it must have common overlapping portions with the human references. Chin-Yew Lin, University of California, proposed it.

The standard versions of Rogue are:

ROUGE-n

This method finds the number of common n-grams between the two summaries and calculates a ratio equal to the number of common n-grams divided by the number of n-grams in the created summary.

An n-gram is defined as a continuous list of n items from a text or a speech.

ROUGE-L

This method generates the longest length of matching words between the two summaries. This is an important method that is very commonly used in abstractive methods of summarization.

We shall be using the *ROUGE-1* method for our summary evaluation. This choice is made because we are using a rule-based and an extractive method. Therefore having just a unigram would be sufficient to understand the accuracy.

EXAMPLE

Consider two sentences from the respective gold and hypothesis summaries, respectively.

"The cat fell down the ditch."
"The cat fell down the ditch."

The computer-generated sentence(summary) has a length of 10 words while the gold summary has 6 words. There are 6 common words - "The, Cat, Fell, Down, The, Ditch"

The accuracy given by the formula is:

$$\text{ROUGE-1} = \frac{p}{q}$$

p = Number of common words = 6
q = Number of words = 10

$$\text{Accuracy} = \frac{p}{q} = \frac{6}{10} = 0.6$$

Therefore the accuracy is 0.6 for this case.

Results and Analysis

We have tested out our product based on the processes mentioned above. Seven articles were chosen and were run on both the systems in order to evaluate these results. These were then compared to the Gold Standard given by the Human Evaluators, and results were evaluated using the ROUGE model.

Firstly, we compare the final models. In this, Proposed System I scores 74.1% whereas Proposed System II scores 83.4%. The barebone version of Proposed System II scored 74.7%, which was hardly any improvement compared to System I.

It is seen that the difference in Stemmer and Lemmatizer was not huge, since in the barebone version, even on using Lemmatizer instead of Stemmer, no significant difference was observed (0.6%). This insignificant increase can be attributed to how poor the Hindi Lemmatizer provided by Spacy was, since it did not even contain a pre-trained model.

Another point of analysis includes the modified TF-IDF formula in the final Proposed System II. It was seen that it could represent how words in different documents affect each other. This observation can be attributed to a simple logical reason. The concept of DF is used to determine the importance of a word in the current article, with respect to its occurrence in the dataset. DF is then inverted so to convey that the more number of documents it occurs in, the less favourable it is. The modified TF-IDF conveys the same concept more robustly. There is a significant difference between a particular word occurring once in multiple articles (possibly as a reference), compared to it occurring multiple times in an article (reduces importance). This reduced importance would logically be conveyed via the modified TF-IDF model, and this is statistically backed by the results produced.

Drawbacks and Challenges

During the process, we faced a few drawbacks which were mainly attributed to the problem we wanted to solve, lack of resources in Indian Languages.

One of the major hits we took consisted of a lack of a proper Lemmatizer. This problem is reflected in the Results column. The difference in accuracy on a Stemmer vs Lemmatizer was only of a 0.6%. It is seen that the lemmatizer provided by SPACY, the model for which we had to train, was not up to the task.

A drawback of Proposed System II is due to the time complexities associated. Ideally speaking, we had a dataset of 55k Hindi Wikipedia Articles. If we could have used all of them, we would have had a highly accurate representation of TF-IDF of all the words in our given article. It could not be done due to the time complexity of Lemmatizer. By experimental approximation, we could roughly determine the complexity of Lemmatizer per sentence to be of $O(n^2)$ to $O(n^3)$. Even if we were to take the lesser one, we observe that we will need to make a huge tradeoff. An ideal point was determined to be to take 1000 articles, each containing at least 100 sentences. That in itself gives as 100,000 sentences to be lemmatised. As we can see, the complexity of this process is enormous. It is observed that it takes roughly around 10-15 minutes for the code to execute.

To solve this challenge, we have used Jupyter Notebook instead of regular python file. It was done so that we would only have to calculate TF of 1000 articles ones, instead of doing it again and again if that were not to be the case.

Conclusion

To conclude, our paper describes the different types of Summarisation techniques: Extractive and Abstractive. It then proceeds to define two different systems, with two similar algorithms to gauge the differences introduced by computational linguistics heuristic. It was realised that a barebone TF-IDF system was hardly an improvement over a Document Term Matrix-based approach. Using the modified TF-IDF formula, we were able to get a wider margin of improvement, thus solidifying our reasoning of using it.

The accuracy obtained, 74.1% and 83.4%, for System 1 and System 2 respectively, represent a good margin of correctness, and we can claim that the system works soundly.

Future Work

1. **Post Processing:** This approach can be used to increase the accuracy in both systems mentioned. Firstly, we would determine the 0.6N number of sentences according to our models. Then we give them individual context weights according to linguistically based heuristics. Finally, from the 0.6N sentences, we extract the 0.3*N sentences.

2. **Thematic Weights:** This approach would require a thematic role assigner for Hindi (Karak roles). This method is currently complicated due to the lack of resources, but assuming it were to be possible to get the Karak roles, we could determine importance based on them. Similar to how we have used POS tags, we could use Thematic Weights instead and could conjoin it with Post Processing.

3. **Graph-Based Approach:** Since we have a massive dataset of documents, we could use Topic Modelling to connect similar topics. Based on the similarity of topics, then we could assign weights to DF of these documents; thus it is becoming a more accurate representation of how TF-IDF should ideally work since it will give less consideration to non-related documents.

References

[1] Manisha Gupta and Naresh Garg. "Text Summarization of Hindi Documents Using Rule Based Approach". In: Sept. 2016, pp. 366–370. DOI: 10.1109/ICMETE.2016.104.

[2] *Kaggle Hindi Wikipedia Articles.* URL: https://www.kaggle.com/disisbig/hindi-wikipedia-articles-55k.

[3] et all Kaustubh Mani. "Multi-Document Summarization using Dis-tributed Bag-of-Words Mode". In: (2018). arXiv: 1710.02745 [cs.CL].

[4] Shivakumar Km and R. Soumya. "Text summarization using clustering technique and SVM technique". In: 10 (Jan. 2015), pp. 25511–25519.

[5] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of summaries". In: Jan. 2004, p. 10.

[6] Divakar Yadav and Vimal Kumar K. "An Improvised Extractive Approach to Hindi Text Summarization". In: vol. 339. Sept. 2015. ISBN: 978-81-322-2249-1. DOI: 10.1007/978-81-322-2250-7_28.

*****