Text Summarization
CS335 (AI & ML)

Yash Jain, Mohit Gupta, Ritik Roongta, and Manas Shukla

IIT Bombay, India

Abstract. Automatic text summarization is one of those applications of Natural Language Processing (NLP) that takes a source text and presents the most important content in a condensed form in a manner sensitive to the user or task needs. Summarization requires semantic analysis and grouping of the content using world knowledge. Our main focus is to implement a Text summarisation extractive and abstractive model. Its primary aim would be to develop summaries for articles which would help us to categorise them and drive reasonable meaning from them. The datasets used are various articles from BBC and Amazon fine food reviews. The summarization systems are ranked according to the similarity of the main topics of their summaries and their reference documents. We evaluate out text summariser using LSA (Latent Semantic Analysis) and ROGUE measure scores used to compare extractive/abstractive summaries with expected ones. The implementation is solely based on python and different python libraries are being used.

Keywords: Text summarization, automatic extract, summary evaluation, latent semantic analysis, ROGUE score

1 Introduction

The importance of having a text summarization system has been growing with the rapid expansion of information available on-line. Summarization is a tough problem because the system has to understand the point of a text. The extractive algorithm involves TextRank and PageRank approach. In this, we find vector representation (word embeddings) for each and every sentence. Similarities between sentence vectors are then calculated and stored in a matrix. The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation. Finally, a certain number of top-ranked sentences form the final summary.
Summarization evaluation methods can be broadly classified into two categories. In extrinsic evaluation, the summary quality is judged on the basis of how helpful summaries are for a given task, and in intrinsic evaluation, it is directly based on analysis of the summary. The latter can involve a comparison with the source document, measuring how many main ideas of the source document are covered by the summary or a content comparison with an abstract written by a human. Our major focus in on intrinsic analysis. Latent semantic analysis(LSA) is a

technique for extracting the hidden dimensions of the semantic representation of terms, sentences, or documents, on the basis of their contextual use. ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in natural language processing. ROUGE-N, i.e. Overlap of N-grams between the system and reference summaries.

2 Problem Statement

With the amount of information available online, it is literally impossible to process everything. But to excel in this competitive world, one needs to have knowledge of almost everything. To solve this problem of information VS time trade-off, we decided to design a system that could prepare a bullet-point summary for us by scanning through multiple articles.

3 Algorithm Used

3.1 Extractive Text Summarization

- Split the text into individual sentences. We use tokenize function of ntlk library to implement this.
- Find vector representation (word embeddings) for each and every sentence. GloVe word embedding dataset is used for this which contains approximately 400000 words.
- Similarities between sentence vectors are then calculated and stored in a matrix after cleansing of them from noises i.e. stopwords.
- The similarity matrix is converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation.
- Finally, a certain number of top-ranked sentences form the final summary.

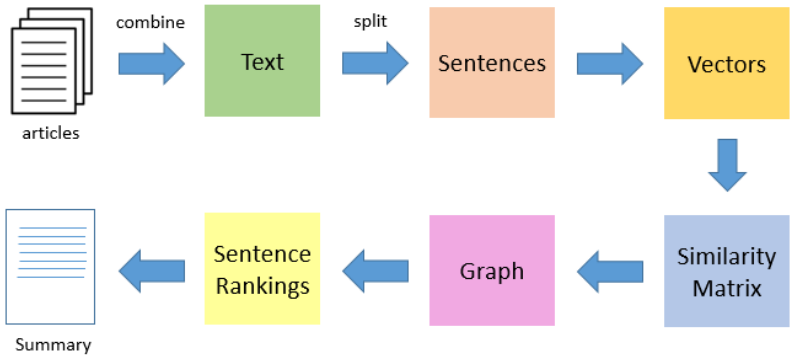

Fig. 1. PageRank Algorithm

3.2 Abstractive Text Summarization

The Algorithm is divided into two phases:

- Training Phase
  An Encoder BiLayer LSTM reads the entire input sequence wherein, at each timestamp, one word is fed into the encoder. It then processes the information at every timestamp and captures the contextual information present in the input sequence. The decoder is also an LSTM network which reads the entire target sequence word-by-word and predicts the same sequence offset by one timestamp. The decoder is trained to predict the next word in the sequence given the previous word.
  - w_i - input tokens of source article
  - h_i - Encoder hidden states
  - P_{vocab} = softmax(Vh_i + b) is the distribution over vocabulary from which we sample out_i

- Inference Phase
  Encode the entire input sequence and initialize the decoder with internal states of the encoder. Pass $\langle start \rangle$ token as an input to the decoder. Run the decoder for one timestamp with the internal states. The output will be the probability for the next word. The word with the maximum probability will be selected. Pass the sampled word as an input to the decoder in the next timestamp and update the internal states with the current time step. Repeat steps 3-5 until we generate $\langle end \rangle$ token or hit the maximum length of the target sequence.

- Attention Mechanism
  The basic encoder-decoder model performs okay on very short sentences but it fails to scale up.
  - The main bottleneck is the fixed sized encoding of the input string, which is the LSTM output of the last time-step. Due to its fixed size it is not able to capture all the relevant information of the input sequence as the model sizes up
  - At each generation step, only a part of the input is relevant
  - At each step, the decoder outputs hidden state hi, from which we generate the output.

  Attention calculates the importance of each input encoding for the current step by doing a similarity check between decoder output at this step and input encodings. Doing this for all of the input encodings and normalizing, we get an importance Vector. We then convert it to probabilities by passing through softmax. Then we form a context vector by multiplying with the encodings.
  - importanceit = V tanh(eiW1 + htW2 + battn).
  - Attention Distribution a_t = softmax($importance_{it}$)
  - ContextVector h_t^* = Σ e_i*a_t

Context Vector is then fed into two layers to generate distribution over the vocabulary from which we sample.

- P_{vocab} = softmax(V_0(V[h_t,h_t^*] + b) + b_0)
- For the loss at time step t, $loss_t$ = log P(w_t), where w_t is the target summary word
- LOSS =1/T Σ $loss_t$ is the total loss across all timesteps

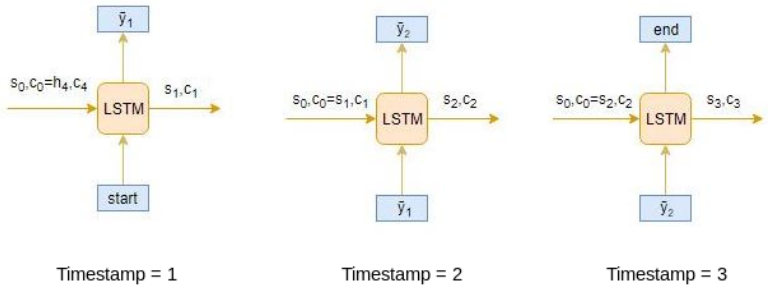

Fig. 2. LSTM Architecture

4 Datasets used

4.1 BBC dataset

Various articles along with their summary on business, lifestyle, etc, extracted from BBC.
https://www.kaggle.com/pariza/bbc-news-summary

4.2 Amazon fine food reviews

Scraped data of review and its human summary collected from various food review websites.
https://www.kaggle.com/snap/amazon-fine-food-reviews

5 Evaluation Metric

5.1 Latent Semantic Analysis (LSA)

LSA (Latent Semantic Analysis) uses bag of words (BoW) model, which results in a term-document matrix (occurrence of terms in a document). Rows represent terms and columns represent documents. LSA learns latent topics by performing a matrix decomposition on the document-term matrix using Singular value decomposition.

We use gensim library of python to implement LSA. The data is loaded and pre-processed. We generate LSA model using gensim and the most important words are calculated from the reference as well as extracted library.

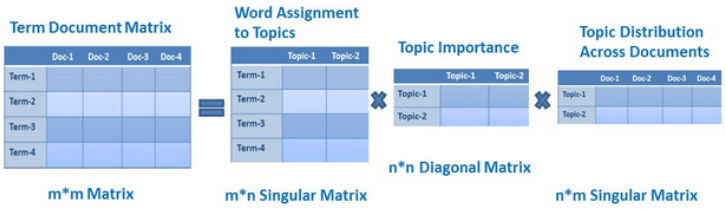

Fig. 3. Latent semantic Analysis

5.2 ROGUE Score

ROUGE tries to assess the adequacy, by simply counting how many n-grams in your generated summary matches the n-grams in your reference summary. There are 3 ROGUE score indicators:

- ROUGE-n recall (r) = 40% means that 40% of the n-grams in the reference summary are also present in the generated summary.
- ROUGE-n precision (p) = 40% means that 40% of the n-grams in the generated summary are also present in the reference summary.
- ROUGE-n F1-score (f) = 40% is more difficult to interpret, like any F1-score.

5.3 Human Perception

This evaluation metric aims at how well the generated summary aligns with human perception. We asked people to rank different compression versions of a sentence in terms of grammaticality. The people are asked to rate the generated summary out of 10 based on their evaluation and their score is finally averaged out.

6 Results

The generated summary matched with the referenced summary upto a good extent. Quoting an exact number would not be justified as referenced summary would be a variable feature for different people and therefore accuracies would change accordingly.
But, following metrics provide a good judgement criteria.

6.1 LSA

Around **79%** of the most important words generated via LSA model turn out to be present in the generated summary.

6.2 ROUGE Score

Extractive
We generated the r and the f scores for the generated summary.
R score = 0.48081
F score = 0.32478
This signifies that **48%** of the generated summary matches the referenced summary.
Abstractive
We generated the r and the f scores for the generated summary.
R score = 0.12323
F score = 0.13357
This signifies that **12%** of the generated summary matches the referenced summary. This is because Abstractive generates new words based on its understanding of the text.

6.3 Human Perception

Around **54%** of the people found the generated summary to be apt as compared to the referenced summary.

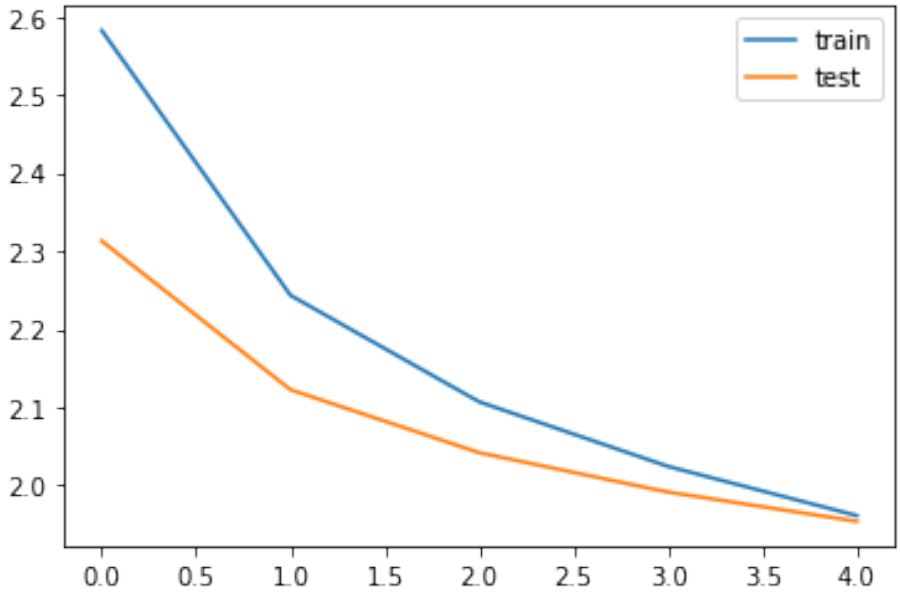

Fig. 4. Train/Test loss for abstractive summarization

References

https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python
https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/
https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/
https://stats.stackexchange.com/questions/301626/interpreting-rouge-scores
http://home.iitk.ac.in/ soumye/cs498a/report.pdf