Automatic Summarization of Scientific Journals

Robin Cosbey
cosbeyr@wwu.edu

Josh Loehr
loehrj@wwu.edu

Abstract

In the realm of scientific research, the vast quantity of articles published every day makes it difficult for researchers to keep up with the latest developments in their fields. Having the ability to automatically represent large bodies of work in less text with salient and representative summaries would be invaluable. We propose the summarization of scientific journals by way of machine learning methods in which a trained model extracts the most relevant sentences from a document.

1 Introduction

Text summarization is a useful process that involves creating a shorter, more concise representation of a larger text or texts. As the number of textual documents available for any number of subject matters continues to grow, having access to summaries of each document can be helpful when searching for relevant information to a task. As the number of relevant scientific papers continues to grow, researchers are required to complete a large amount of literature review and the automatic text summarization becomes even more relevant. Having a model that can produce a summary of a papers contributions would make the literature review process more efficient and increase a researchers ability to understand the scope of their project early on.

Automatic summarization of text consists of two main types: abstractive summarization, in which a model produces newly generated sentences summarizing the text provided, and extractive summarization, in which a model extracts sentences that it has deemed most representative of the document, or documents, being analyzed. Text summarization can be further subdivided into single- and multiple-document summarization, where the difference lies in the input given to the model and the scope over which the summaries produced should cover. Given the time constraints of this project, and the data available for us to work with, we have opted to pursue extractive summarization over single documents, as this fits within the scope of the spefic problem we are attempting to solve. Our goal is to produce a summary containing the most important sentences of the body of a scientific paper that is of comparable quality to the corresponding document abstract.

Summarization by way of sentence extraction is typically considered a classification task, whereby each sentence of a document is classified as either being included or excluded from the summary. We propose a recurrent neural network architecture trained to produce rankings for each sentence of an article body, the order of which corresponds with the likelihood of each sentence being included in the document summary. The final summary is compiled by taking the top N sentences from this ordering, where N is variable with the desired length of the produced summaries, and listing them in the order in which they appear in the original text.

2 Related Work

(Das and Martins, 2007) compile a survey of a broad range of techniques and methods for automatic text summarization, in which they describe the seminal work of (Luhn, 1958), wherein it is proposed to use word frequency as a measure of a sentence's significance towards the overall document. After stemming and removing stop words, sentences are processed to produce a *significance factor*. The ranking of all sentences via this value is used to form the auto-abstract out of the top ranking sentences. Although the method by which the significance factor was derived differs from our own, we borrow heavily from this overall process to fill in for the lack of training labels in our data, as described in the next section.

Neural networks have been applied to problem of text summarization as well.(Svore et al., 2007) train a pair-based neural network algorithm called RankNet, which is used to rank a set of input sentences in a document. Their implementation, NetSum, is tasked with producing a three sentence extractive summary of news articles which resemble human generated highlights of equal length. NetSum was able to significantly outperform the baseline on over 70% of their document set.

Evaluation of automatic summarizations poses its own unique challenges. Traditionally, evaluation was left to human judgements, an effective yet often prohibitively costly method of ensuring the salience, readability, and overall quality of machine generated summaries. As an alternative, (Lin, 2004) proposes Recall-Oriented Understudy for Gisting Evaluation (ROUGE) as a package of methods for evaluation automatic summaries. ROUGE measures overlap of n-grams, word sequences, and word pairs between system and reference summaries. (Ganesan, 2015) extends this work with ROUGE 2.0, which adds synonym and topic overlap, among others, to improve the quality of evaluation for summaries which may not provide a verbatim recreation of the reference version.

(Lin et al., 2006) views the task of summarization within an information theoretic viewpoint. Specifically, by assuming the existence of some reference probability θ_r from which, given a text to summarize, a human generated reference summary is sampled, the goal is to develop a parameterized model which produces summaries from its own distribution, θ_a. Using *Jensen-Shannon divergence*, one can measure the "distance" between the two distributions, with better models producing distributions which minimize their divergence with θ_r. Interestingly, the authors find that because of the special properties of the Jensen-Shannon divergence, the metric performs well even when no smoothing is performed to account for out-of-vocabulary words.

3 Data

One of the largest challenges we faced was finding a dataset suitable to our task. For the task of extractive text summarization, the ideal training dataset would consist of documents paired with either human created summaries, or, alternatively, with labels for each sentence of the document indicating the suitability of that sentence as a summary component. Unfortunately, while automatic summarization has matured within the domains of newspaper articles, product reviews, and more, application towards scientific articles is less explored, and we were unable to find any such ideal dataset which was freely and readily available. Instead, we opted to create our own training labels, which would allow us to use any dataset of scientific literature containing both article bodies and abstracts. This process will be outlined in full detail in the next section.

To facilitate the creation of these artificially constructed training labels, we required a dataset composed of text which is easily parsed, and which provides a sufficient quantity of documents to train on. We settled on two such datasets, each of which provides published journal papers in XML format for easy document traversal.

The primary dataset we have been working with is Computation and Language (cmp-lg) corpus. Part of the TIPSTER SUMMAC effort, this dataset contains 183 documents published at Association for Computational Linguistics sponsored conferences, and is intended for use in information retrieval, extraction, and summarization. The documents have been automatically converted from LaTeX to XML, and much of the more complicated formatting has been reduced to simplified XML tags.

The second dataset we used was a collection of about 1.8 million MEDLINE/PubMed biomedical literature. This corpora spans a variety of medical and scientific journals compiled by the United States National Library of Medicine, and is accessible via the online PubMed system. Journal topics cover many disciplines, from health care to biology, chemistry, bioengineering, and many more. Although also provided in XML format, this dataset is not designed with automatic parsing in mind, and poses a greater challenge for successful preprocessing and text normalization.

4 Approach

4.1 Preprocessing

A series of Python and BASH scripts were used to facilitate the preprocessing of our datasets, in multiple steps. The pipeline was segmented so as to be as general as possible, to allow for maximum code reuse between datasets. The overall process is as follows:

1. BASH scripts download dataset documents, either in bulk or in batches.

2. Use XML structure to extract document abstracts and article bodies.

3. Remove extraneous subsections from article bodies (e.g. references, footnotes, acknowledgements)

4. Convert certain XML tags (e.g. REF, EQN) to special tokens.

5. Parse sentences using custom built regular expressions to detect sentence boundaries. Save raw sentences to file for later reference.

6. Process each sentence, and save tokens to file:

    - Remove stop words.
    - Replace symbols and numbers with special tokens.
    - Perform stemming.

This process is repeated for each document, both for the abstract and the article body. Once normalized tokens files have been created for each document in the corpus, TF-IDF vectors are created for each sentence. Each article body sentence vector is compared against all abstract sentence vectors of the same document, and a set of cosine similarity scores are produced. The maximum of these scores is kept, indicating the suitability of that particular article body sentence for inclusion in a summary. Once all article body sentences have been scored, those scores are ranked from higheset (most similar) to lowest (least similar). This ordering is mapped back onto the sentence indices, giving a list of sentence indices sorted by summary suitability. These are then used as the training labels for each document, to be fed into the neural network training algorithm.

4.2 Feature Creation

To build feature vectors for each article body, we use a combination of hand-selected features and sentence contents. Since every document in our datasets included headings and subheadings within the article body, we decided to index these and use their unique identifiers as a feature. For each sentence of the article body, a vector is built which consists of the ID number of the header under which that sentence falls within the original document, followed by the original sentence length, and lastly followed by a list of integers representing unique IDs for each token appearing in the corpus. Tokens which appeared only once in the corpus are treated as a single token with a shared ID, which greatly reduces the vocabulary size and helps hide lingering tokenization and parsing errors from the model. The resulting feature vector is a list of integers, padded with zeros to the maximum number of features in all of the documents. Each feature file, consisting of one feature vector per sentence in the document, is also padded with zeros to the maximum number of sentences in all of the documents. The features, labels, and other meta information is saved as NumPy binary files for efficient loading.

4.3 Model Architecture

We have approached this task by way of supervised machine learning methods. The model, in our case a recurrent neural network (RNN), is supplied with extracted document features as well as the true labels corresponding to the score each sentence received. We chose to work with a recurrent neural network because of the sequential nature of sentences within a document and we feed in one document at a time. The TensorFlow machine learning framework is utilized to build the RNN architecture and the model is trained to map from the features to the labels. Two types of RNN cells have been explored: basic RNN cells and Gated Recurrent Unit (GRU) cells. RNNs are excellent at modeling temporal dependencies in data. In basic terms, at each timestep, the RNN gives the current sentence a predicted score based on the sentence's features as well as information from previous timesteps. GRUs are a simple variant of the basic RNN that supply longer time dependent information; with this task, it is likely that we will want our model to remember specific information from the Introduction section. We provide a hidden layer size and our hidden layer weights and bias vectors are the product of the glorot uniform initializer. At each timestep, we compute an output matrix by matrix multiplying our feature matrix by the hidden layer weights and adding the bias vector.

4.4 Summary Generation

The output matrix produced by the model contains the predicted scores that the model has assigned to the sentences in a document. We take the maximum model generated scores contained in this output matrix for each sentence. The n-highest scores and their corresonding line numbers are extracted. In our case, n is set to equal the number of sentences in the abstract. Then, the line numbers are sorted from least to greatest; we chose to preserve order from the original document. The sentences at these line numbers are stored in a summary file.

4.5 Experiment Setup

We chose to use a standard 80% train, 10% development, and 10% test split of the cmp-lg data. The document length (number of sentences) as well as the sentence length (number of words) varied widely so we removed the outliers for a more uniform dataset to train on. Of the 183 documents within this set, we used about 170 documents. The RNN learns the mappings with the train set features and labels. After each epoch of training, the development set features and labels are supplied. After all training has been completed, we feed the test set features to the model and extract the model-generated scores for the sentences in each document to produce the summaries containing the highest scoring sentences from the original document. We chose to display them in the order that they appear in the document, so that the resulting summary would more likely be comprehensible. Hyperparameter tuning of the model has consisted of variably setting the number of epochs the model trains for as well as the size of the hidden dimension.

4.5.1 Baselines and Evaluation Metrics

We employ two main evaluation metrics: human analysis and the Recall-Oriented Understudy for Gisting Evaluation (Rouge) (Ganesan, 2015). Our baseline summaries for each document contain the first sentence of each paragraph present in the document text. After the model has generated summaries, we visually inspect the summaries to determine if pertinent keywords are present and how similar the structure is to the provided human-generated abstract. We also compare the summaries with the abstracts mathematically by way of the ROUGE metric. ROUGE is designed for automatic summarization tasks and calculates the overlap between the model-generated summary and the corresponding abstract. We invoke three ROUGE metrics: ROUGE-1 (unigram representation), ROUGE-2 (bigram representation), and ROUGE-L (longest common subsequence representation). We report precision, recall and F-score. Our datasets do not have true summary files to compare with but the abstracts are a decent substitute.

5 Results

5.1 Results

Results table here.

5.2 Analysis and Discussion

When comparing the indices of the model's highest scoring sentences, the GRU cells more often choose the first few sentences present in the documents than the basic RNN cells.

6 Conclusion

Overall, our model performed poorly. The main cause of this likely stems from the lack of an ideal dataset to work with. Although abstracts do often provide reasonable summaries of the papers they are attached to, often they are substantially different from what an extractive summary might look like. Likewise, we put a lot of faith into our method of constructing artificial "ground truth" labels for our dataset. TF-IDF vectors and cosine similarity are an effective yet rudimentary technique for comparing text, and do not take into account the actual substance of the topics discussed. The combination of these factors leads us to believe that the labels we fed into our model are suspect, and the model, which can only be as good as the data it is trained on, is thus suspect as well.

References

Dipanjan Das and Andr Martins. 2007. A survey on automatic text summarization. 12.

Kavita Ganesan. 2015. Rouge 2.0: Updated and improved measures for evaluation of summarization tasks.

C.-Y. Lin, G. Cao, J. Gao, and J.-Y. Nie. 2006. An information-theoretic approach to automatic evaluation of summaries.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries.

H. P. Luhn. 1958. The automatic creation of literature abstracts. *IBM Journal of Research Development*.

K. Svore, L. Vanderwende, and C. Burges. 2007. Enhancing single-document summarization by combining ranknet and third-party sources.