CIS 530 Fall 2013 Final Project

Instructor: Ani Nenkova
TA: Kai Hong, Jessy Li

Released: November 28, 2013
Due: 11:59PM December 13, 2013
Latest Submission time: December 17, 2013

Overview

For the final project you will implement several generic multi-document summarization systems. You will implement three basic summarizers according to the specification we provide and you will come up with a summarizer of your own. Your task is to generate a 100-word summary for each input. Inputs will consist of about 10 news articles on the same topic.

The best 5 new summarizers among all groups could get 10 points extra credit for this project. For evaluation, we will use the ROUGE toolkit. It automatically compares the summaries with gold-standard summaries written by people for the same input, based on n-gram overlap. We will use ROUGE-2 recall for evaluation, based on bigram matches.

Scores for each task

- Three Basic Summarizers (50 Ponits):
  - TF*IDF Summarizer
  - LexRank Summarizer
  - KL divergence Summarizer
- Your Own Summarizer (30 Points):
  - Can be either extractive or abstractive, at least two NLP tools or resources must be used. If you prefer, you can also implement some techniques studied in class in place of one of the tools. A list of tools is given here:
  `http://www.cis.upenn.edu/~cis530/Project.html`
- Write-up (20 Points)
  - The write-up should be no more than 3 pages in PDF. The write-up should include motivation and description of your approach, evaluation results and interpretation. Clear and complete write-ups with strong method and analysis sections will receive up to 20 points extra credit.
- The top 5 summarizers among all groups will get 10 points extra credit for this project.

Submission CheckList

- Code for the three basic summarizers.
- Code for your summarizer. Also give your summarizer a Name!

- Summaries generated from the four summarizers, on both development and testing set.
- Write-up
- A readme file (readme.txt) including the files submitted and their contents

Data

For this project, we use the data from a standard evaluation challenge for summarization, run by NIST. The input documents you summarize here are newswire articles. All of the data is locally placed in `/home1/c/cis530/final_project`. This folder includes:

- dev_input: The development set.
- dev_models: Human summaries for the development data.
- test_input: The test set.
- baseline: Summaries generated from a baseline system for development set.
- rouge: The ROUGE automatic evaluation system.
- nyt_docs: A large corpus of New York Times articles, used to compute inverse document frequency (IDF).
- report: Template for the write-up.

ROUGE

ROUGE Score

We will be using the ROUGE [4] system for evaluation purpose. ROUGE is the most commonly used metric of content selection quality. It is cheap and fast, and the scores it produces are highly correlated with manual evaluation scores for generic multi-document summarization of text. ROUGE is based on the computation of n-gram overlap between a summary and one or more human gold-standard summaries (models). ROUGE also has numerous parameters, including word stemming, stopwords removal and n-gram size. We here use ROUGE-2 recall, with stemming and without stopwords removed as our main metric, due to the reason that it worked the best for comparing different machine summaries [3].

To score your summaries with ROUGE, you simply need to call the script `ROUGE-1.5.5.pl` with a configuration file. We suggest you download ROUGE to your local folder because you will need to run it often. You will need to write your own configuration files if you want to track the performance of your own summarizers. The setting we will be using for comparison is as follows:

```
./ROUGE-1.5.5.pl -c 95 -r 1000 -n 2 -m -a -l 100 -x config.xml
```

The configuration file, `config.xml`, tells ROUGE what systems that you are evaluating (called peers in the configuration file) and human summaries (called models) to use each time.

Below is an example of a call to `./ROUGE-1.5.5.pl` to evaluate the baseline summarizer. The score `0.07926`, is the ROUGE-2 Recall we are looking at.

```
./ROUGE-1.5.5.pl -c 95 -r 1000 -n 2 -m -a -l 100 -x config.xml
---------------------------------------------
baseline ROUGE-1 Average_R: 0.34500 (95%-conf.int. 0.33340 - 0.35768)
baseline ROUGE-1 Average_P: 0.34063 (95%-conf.int. 0.32967 - 0.35289)
```

```
baseline ROUGE-1 Average_F: 0.34275 (95%-conf.int. 0.33164 - 0.35526)
---------------------------------------------
baseline ROUGE-2 Average_R: 0.07926 (95%-conf.int. 0.07182 - 0.08774)
baseline ROUGE-2 Average_P: 0.07828 (95%-conf.int. 0.07104 - 0.08665)
baseline ROUGE-2 Average_F: 0.07876 (95%-conf.int. 0.07148 - 0.08726)
```

The first three results are the average recall, precision and f-measure ROUGE-1 scores for all the test inputs, along with a 95% confidence interval which gives an indication of the range of the likely performance. In our analysis we will use only the ROUGE-2 (first line in the second group of three) average recall and will ignore any of the measures.

You could generate summaries of more than 100 words, because the toolkit will automatically truncate the summaries to 100 words with the command `-l 100`. If you want to test locally, you need to write your own configuration file. You may find the following link helpful, which tells you how to set up ROUGE: `http://kavita-ganesan.com/rouge-howto`. Below we also give an example of the configuration file.

Configuration File

In the configuration file, you need to specify the root directory and file names for both machine and human summaries.

```
<PEER-ROOT>
baseline
</PEER-ROOT>
<MODEL-ROOT>
models
</MODEL-ROOT>
<INPUT-FORMAT TYPE="SPL">
</INPUT-FORMAT>
<PEERS>
<P ID="baseline">summary00.txt</P>
</PEERS>
<MODELS>
<M ID="A">D30001.M.100.T.A</M>
<M ID="B">D30001.M.100.T.B</M>
<M ID="C">D30001.M.100.T.C</M>
<M ID="D">D30001.M.100.T.D</M>
</MODELS>
```

`PEER-ROOT` is the path to the folder where your system summaries are located, `MODEL-ROOT` is the folder where model summaries are located. In the example above, `baseline` is the root folder of the summaries; `models` is the root folder of the models; the summary `summary00.txt` is compared with its corresponding four human summaries, named `D30001.M.100.T.*`.

Baseline System

For this system, we simply select the first sentence from each document in the input to form a summary. Selecting the first sentence is a strong baseline for single document summarization. It also offers a competitive baseline in multi-document summarization, with a ROUGE-2 recall score of 0.07926. Randomly selecting sentences is a much weaker baseline, yielding ROUGE-2 recall of 0.045.

1 Three Basic Summarizers (50 points)

You will need to implement three of the summarization approaches described in class: **TF*IDF**, **LexRank** and **KL divergence**. Below we provide more descriptions and instructions. We have left unspecified some of the details which you need to make your own choices, including sentence scoring, redundancy removal, etc. Regardless of your choices in these particular details, you should make sure that your implementation of the core algorithm is correct. Name the main function of your summarizers with the form of `TFIDFSum(input_collection, output_folder)` (`LexRankSum, KLSum`) so that they can be easily called.

Generally, you need to lowercase the words. You can choose whether or not to include the stopwords. You can get the stopwords the same way you did for homework-4.

1.1 TF*IDF Summarizer (16 Points)

TF*IDF combines term frequency (TF) for words in the input and inverse document frequency (IDF) from a large background corpus. Words with high scores by TF*IDF are often more descriptive of an input document collection. For this summarizer, we evaluate the importance of sentences based on the TF*IDF scores of the words within one sentence. After generating weights of words, the score of the sentences could be generated by the formula below:

$$Scores(S) = \frac{1}{|S|} \sum_{w \in S} TF * IDF(w) \tag{1}$$

Here w can be either the unique words in the sentence, or the tokens of the sentence.

From Sentence to Summary

Once all sentences are assigned scores, we can use a greedy algorithm as a simple way to form the summary. We iteratively choose the sentence with the highest score, append the sentence to current summary if it is valid, until we've reached the word-limit. Valid sentences will confirm to some length limit (we will exclude too short or too long sentences) and do not repeat information conveyed in sentences that are already included in the summary (we can use cosine similarity to check for that). More formally, denote `Sens` as the set of all sentences from the input, `sum_length` as the maximum length of the summary. Then the summaries can be generated in a greedy fashion:

Algorithm 1 Greedy TF*IDF Summary Generation

procedure TFIDF-SUM($Sens$)
  for each sentence s_i in $Sens$ **do**
    $Score(s_i) \leftarrow \frac{1}{|s_i|} \sum_{w \in s_i} TF * IDF(w)$
  end for
  Sort all Sentences according to $Score(S_i)$ in descending order, then storing into Queue Q.
  $Sum \leftarrow []$.
  while $Len(Sum) \leq 100$ **do**
    $s_{cur} \leftarrow Q.pop()$
    if Valid(s_{cur}) **then**
      $Sum.append(s_{cur})$
    end if
  end while
  return Sum
end procedure

Specifically, two things need to be considered for a sentence to be **Valid**:

Sentence length Some work has suggested that the sentences included in summaries should have a minimum number of 9 words and maximum number of 45 words. Think about reasonable restrictions you may want to impose, **none** is generally considered as a bad option.

Divergence Some of the highly scoring sentences may have overlaps in content. One crude way to check for repetition is to compare the vector of the current sentence with the vectors of the sentences already in the summary. The weights for the components of vectors could be binary, frequency or TF*IDF, as we did in homework-1. We will choose not to include a sentence if the similarity score exceeds some pre-defined threshold (such as 0.2, 0.4, 0.6, etc). Obviously, more sophisticated methods of checking for shared content may result in better summaries.

In the writeup, please clarify the approach you use to score the sentences. Also please record clearly your selection of sentence length limit, redundancy removal approach and any other parameter settings.

1.2 LexPageRank Summarizer (17 points)

LexRank [1] is a graph-based method of computing sentence importance. It represents each sentence as a node, then computes cosine-similarity between sentences to generate edges. The final representation of the input is a graph (without weights on its edges): if the similarity between two sentences is above a predefined threshold, then there is an edge between the two nodes, otherwise there isn't. Again while computing sentence similarity, it is preferred to represent your vector using TF-IDF rather than term frequency alone, so that function words contribute less to the similarity. After the scores of the sentences have been assigned, you should follow the same framework as in previous method to generate summaries.

More details about LexRank could be found in this paper:

`http://acl.ldc.upenn.edu/acl2004/emnlp/pdf/Erkan.pdf`

For the writeup, please include your choice of threshold when selecting edges and your criteria for stopping the iterative process in the PageRank algorithm. Also record the approach you use to form the summaries, as well as details of the parameter settings within the algorithm.

1.3 Greedy-KL Summarizer (17 points)

Here we describe a summarization system based on KL divergence [2]. This approach incorporates knowledge about the overall unigram distribution of the words in the input. The idea is to select a set of sentences which closely match the unigram distribution of the input. We set Q as the unigram distribution of the input collection, P as the unigram distribution of the summary. Our objective is to minimize the following function:

$$KL(P \parallel Q) = \sum_{w} P(w) \cdot \ln \frac{P(w)}{Q(w)}$$

A few conditions need to be satisfied in order to compute $KL(P \parallel Q)$. Firstly, both P and Q should sum to 1, which is naturally satisfied. Secondly, $Q(w) = 0$ should lead to $P(w) = 0$, and this is satisfied since the sentences selected is a subset of the original input. For the case $P(w) = 0$, we simply assign $P(w) \cdot \ln \frac{P(w)}{Q(w)} = 0$. You are also free to do smoothing if you feel uncomfortable with the zero probability.

Note that the KL approach, unlike the two previous approaches, evaluates how good the entire summary is, rather than the importance of individual sentences. However, enumerating all subsets of sentences (all possible summaries) and computing $KL(P \parallel Q)$ for each possible subset will be prohibitively slow. Here we introduce greedy-KL, that picks the sentence which minimizes the value of $KL(P \parallel Q)$ iteratively. A brief illustration of the greedy algorithm is shown below: for each iteration, we select the sentence s_i which would minimize the KL-divergence, until a desired length is reached. Again you may want to check if one sentence

is valid or not by looking into sentence length and diversity. We omit this part within the psuedo-code here. Here D is the original input collection.

Algorithm 2 KL_{sum} Algorithm

procedure KLSUM(D)
  $Sum \leftarrow []$
  while $Len(Sum) \leq 100$ **do**
    j = $argmin_i$ $KL((set(Sum) \bigcup s_k) \parallel D)$
    $Sum.append(s_j)$
  end while
  return Sum
end procedure

For the writeup, please include your choice of smoothing if you decided to use any and the parameters used for checking sentence validity.

2 Your Summarization System (30 Points)

You are now ready to design your own summarizer. You summarization system can be either extractive or abstractive, and it can be designed in any way you want. A requirement is that you must use at least two tools or resources in your summarization systems, for example, MPQA, WordNet, or NLP processing modules from CoreNLP. If you prefer, you can also implement some techniques studied in class in place of one of the tools. You may find this project resource page helpful:

`http://www.cis.upenn.edu/~cis530/Project.html`.

3 Ranking According to ROUGE Score (For Extra Credit)

For competing, we will evaluate the performance of your own summarizer on the testing set. The only metric we are using here is ROUGE-2 recall. For the final submission, make sure that your code can successfully generate the summaries you submit. Don't forget to give your summarizer a name!

4 Write-up Requirement (20 points)

You need to use the templates for the write-up of the final project, which can be found at the project resource page or the folder **report**. Your write-up should include the following sections:

1. Three basic summarizers:
   (1) For each summarizer, explain how you set any of the parameters of your system and any design decisions you made for options that were left unspecified in the assignment. Explain the motivation of your choices.
   (2) Performance: ROUGE-2 recall of the three basic systems on the development set.

2. For your own system, you need to describe:
   (1) The general idea/method for your system.
   (2) What resources or tools you have used and how are they included in your implementations.
   (3) The result of your summarization system on development set.

3. Discussion and Analysis.
   You need to do analysis based on the output summaries of your system and the basic systems. Compare

the results of your system and the basic summarizers you implemented. Discuss any special aspects of your summarizer and include experiments that clarify the source of improvement.

The write-up should be no more than 3 pages in PDF. Clear and complete write-ups with strong method and analysis sections will receive up to 20 points extra credit.

References

[1] Gnes Erkan, Dragomir R. Radev, and Dragomir R. Radev. Lexpagerank: Prestige in multi-document text summarization. In *EMNLP*, pages 365–371, 2004.

[2] Aria Haghighi and Lucy Vanderwende. Exploring content models for multi-document summarization. In *Proceedings of HLT-NAACL*, pages 362–370, 2009.

[3] Karolina Owczarzak, John M. Conroy, Hoa Trang Dang, and Ani Nenkova. An assessment of the accuracy of automatic evaluation in summarization. In *NAACL-HLT 2012: Workshop on Evaluation Metrics and System Comparison for Automatic Summarization*, pages 1–9, 2012.

[4] Chin yew Lin. Rouge: a package for automatic evaluation of summaries. pages 25–26, 2004.