CIS 530 Fall 2014 Homework 4

Instructor: Ani Nenkova
Head TA: Jessy Li

Released: November 17, 2014
Due: 11:59PM December 1, 2014

Overview

The focus of this assignment is to implement a simple but strong baseline for multi-document summarization of news. You will also get some exposure to WordNet functionality. You will first use a tool to find words that characterize the topic of a document (according to a log-likelihood ratio test). Then you will use WordNet to relate the main topic words with other words in the document. Finally, you will write your own automatic multi-document summarizer with a greedy algorithm and compare it with a baseline.

To run your code, please use the Biglab machines. Large jobs will be automatically terminated on Eniac. Should there be any issues with the machines, email cets@cis.upenn.edu to let them know about the issue. For instruction on how to use Biglab, please refer to the link provided below:

`http://www.seas.upenn.edu/cets/answers/biglab.html`

Submitting your work

The assignment should be done individually, so each student should make one submission with their own code. The code for your assignment should be placed in a **single file** named `hw4_code_yourpennkey.py`, where `yourpennkey` is the first part of your Penn email address. For this assignment you will also submit: one summary for each summarizer, `results.txt` which records the performance of your summarizer, a list of the 20 most descriptive words in each input for summarization along with words related to the top descriptive words. You will also submit a single output file from the tool computing the log-likelihood ratio statistic.

All submissions will be electronic, done from your Eniac account. You need to copy your files to Eniac if you choose to work elsewhere. To copy files you can use a SFTP client, or secure copy `% scp local_file yourpennkey@eniac.seas.upenn.edu:PATH` where `local_file` is the path to your homework on the local machine and PATH is the path to the location on Eniac where you wish to copy it.

Once you have the file in place in your Eniac account, connect via ssh to `seas.upenn.edu` and use the `turnin` command to submit your files for grading:

```
% turnin -c cis530 -p hw4 hw4_yourpennkey.zip
```

You will get a confirmation message. You can run `turnin` multiple times before the deadline. Each time you run `turnin`, it overwrites your previous submission for that assignment. You can check that the homework was submitted successfully:

```
% turnin -c cis530 -v
```

This will show you the list of files you have submitted.

Code Guidelines

In the assignment below we specify a number of functions. *Your submission should include implementation for each of these required functions, with the specified number and types of arguments.* In addition, you may

write any extra functions that you find necessary. Avoid doing any preprocessing of the input text besides that explicitly described in the assignment. This will make grading easier.

Your code will be graded automatically. It is your responsibility to make sure your function can be called correctly and that they return exactly the type of values specified in the assignment. To make sure that your code will run correctly during grading, please include **only** function declarations in your source code or above the line `if __name__ == "__main__"`, and prefix **all** your global variables with `yourpennkey`.

1 Topic words and WordNet similarity

1.1 Finding topic words (15 points)

Topic words in a text are those words that occur with significantly greater probability in that text than in a large background corpus. Here we will introduce the likelihood ratio test which allows us to determine levels of statistical significance for topic words. The test compares the likelihood of the available data (input text + background) under the two hypotheses:

1. The word is not a topic word, so its probability would be the same in both the given text and the background collection.

2. The word is a topic word and its probability in the given text is greater than in the background.

In both cases, the occurrences of a word are modeled according to a Binomial distribution. This ratio λ is estimated assuming binomial distribution of words in a text and $-2\log(\lambda)$ has a χ^2 distribution. Topic words (also called topic signatures in the NLP literature) are a very useful feature for content selection in summarization. More information on how topic words are calculated can be found here: `http://research.microsoft.com/en-us/people/cyl/col2000-final.pdf`

You will not have to implement the calculation of the test statistic. Instead, we have provided a tool that will calculate the statistic for each word in a cluster of documents. The tool uses counts from a large collection of newspaper documents to estimate the general rate of occurrence of words.

For this tool, you need to set up a configuration file. A sample configuration file can be found at `/home1/c/cis530/hw4/TopicWords-v2/config.example`. Make a copy of the file in your own directories to edit. In particular, you will need to change the `inputDir` and `outputFile` fields each time you run the tool on each input cluster. Pointing the tool to a directory will instruct it to process all of the documents in that directory and create a list of topic words that characterize the cluster of documents.

To run the tool, navigate back to the directory containing the topic word tool:

```
cd /home1/c/cis530/hw4/TopicWords-v2/
```

Then use this command to run the tool, using the config file you modified in your home directory:

```
java -Xmx1000m TopicSignatures ~/config.example
```

This will create a `.ts` in the directory you specified in the `outputFile` field of the configuration file. The output file contains words and the respective χ^2 statistic for this word. The larger the value of the statistic is, the more evidence there is that the word is descriptive of the cluster and occurs there at rates higher than the rate of occurrence in a general collection. A χ^2 statistic of 10 for example indicates that there is 1% chance to obtain this test statistic under the hypothesis that the rate of occurrence in the cluster of documents and the background is the same. This would normally be considered a statistically significant difference. In the assignment, all words with statistic of 10 or higher will be deemed topic words, characteristic of the cluster of documents.

Please do not download the original tool. This homework uses a specially modified version of this tool. Use only the version on Eniac.

1. The folder `/home1/c/cis530/hw4/dev_input` consists of 40 clusters of documents on the same topic. Each cluster (subdirectory) contains 10 news articles covering the same event or topic. Your task is to find the 20 most descriptive topic words for each cluster. You may find it easier to write a script or function to batch-generate the config files. *As part of your submission, submit a single .ts file that you have created for one of the document clusters in the input directory.*

2. Write a function `load_topic_words(topic_file, n)` that, given a a `.ts` file, returns a `tuple` of two `lists`. The first list in the tuple consists of the top `n` topic words from that file, according to their χ^2 statistics. The second list contains all other topic words. Sort the two lists in descending order of χ^2.

1.2 Expanding keywords (20 points)

Now, we will expand our list of top 20 topic words by finding topic words related to them in WordNet, using Resnik similarity. WordNet is a built-in part of the NLTK package. You can find documentations regarding how to use WordNet with NLTK in `http://www.nltk.org/howto/wordnet.html`

Resnik similarity is equal to the information content of the lowest common ancestor of two words. For this implementation, the Brown corpus is used to calculate information content. In the NLTK package, you can directly get these probabilities (i.e., *information content*) from the Brown corpus by calling:

```
>>> from nltk.corpus import wordnet_ic
>>> brown_ic = wordnet_ic.ic('ic-brown.dat')
```

The Resnik similarity between two WordNet synsets is calculated as:

```
>>> from nltk.corpus import wordnet as wn
>>> cat = wn.synset('cat.n.01')
>>> dog = wn.synset('dog.n.01')
>>> cat.res_similarity(dog, ic)
```

Write a function `expand_keywords(keylist, candidatelist, ic, outputfile)` which takes as input a `list` of top 20 topic words (`keylist`), a `list` of candidate words (`candidatelist`) and *information content* (`ic`) values computed as shown above. It finds all words in the candidate list that have a non-zero Resnik similarity with at least one word of the topic words. Write to `outputfile` the 20 topic words and all words from the candidate list that have none-zero similarity with each of the topic words. The format of `outputfile` should be [topic word: all similar words from the candidate list sorted by similarity from high to low, separated by space], with one line for each of the most significant topic words.

You may notice that WordNet synset lookup requires a part-of-speech tag of the word. Normally you can run a part of speech tagger but we have already done this in past assignments and will use a heuristic instead. During your expansion process, you need only consider words in `keylist` and `candidatelist` that have at least one *noun* synset in WordNet, and will simply treat any such word as noun and take its first synset.

Run this function on all the subdirectories under `/home1/c/cis530/hw4/dev_input` and generate 40 expanded topic word files. Here `candidatelist` should be topic words that are not among the top 20 (but has a statistic of 10 or higher). In your submission, include a single sample, for one input cluster, of the 20 most descriptive words and the related other words.

2 Generate summaries automatically

In extractive multi-document summarization, a system will pick the best sentences from a set of documents from the same topic to form a summary. Here we limit our length of summaries to be at most 100 words. In this section, you will build two summarizers – a baseline and and a greedy summarizer. Then, you will run a standard tool to evaluate how good your summaries are, comparing to human written summaries.

Please do not do any special tokenization or preprocessing of the data. These are important, but simply not the focus of this assignment. The inputs already have sentences split, one sentence per line.

2.1 The baseline (20 points)

Before attempting anything fancy, one should first use an extremely simple approach to be the benchmark of a new algorithm. One simple approach is to take the first sentence of each document, until the length limit (100 words) is reached. Write a function `summarize_baseline(directory, outputfile)`, where `directory` is one of the subdirectories under `/home1/c/cis530/hw4/dev_input`. Your summary should be written into `outputfile`, one line per sentence.

It is ok to go slightly above exactly 100 words, as long as your summarizer stops adding sentences when your summary is already above the length limit. To name your summary output files, follow the naming convention in the configuration file of the ROUGE evaluation system specified in section 2.3. Please iterate through the files in alphabetical order of their names (i.e., simply call `sort()` before iterating the files). This will give us a deterministic order of sentences and will facilitate grading. *As part of your submission, submit a single summary file that you have created for one of the document clusters.*

2.2 Greedy KL summarizer (30 points)

Here we describe a summarization system based on KL divergence[1]. The approach incorporates knowledge about the overall content unigram distribution of the words in the input.

The idea is to compute a unigram language model (content words only) in the input and the summary respectively, then select a set of sentences which closely match the distribution of content words in the input. Let Q be the distribution of unigrams in the input cluster and P the distribution of unigrams in the summary. Our objective is to minimize the following function:

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

Note that the KL approach 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 pick sentences greedily to iteratively minimizes the value of $KL(P \parallel Q)$.

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.

Algorithm 1 KL_{sum} Algorithm

procedure KLSUM(Q)
  $Sum_i \leftarrow \emptyset$. ▷ Summary set
  while $Len(Sum_i) \leq 100$ **do**
    j = $argmin_i$ $KL((Sum_i \bigcup s_i) \parallel Q)$
    $Sum_i \leftarrow Sum_i \bigcup \{s_j\}$
  end while
  return Sum_i
end procedure

Implement a function `summarize_kl(inputdir, outputfile)`, where `inputdir` is the input cluster (i.e., a subdirectory under `/home1/c/cis530/hw4/dev_input`). Your summary should be written into `outputfile`, one line per sentence. A list of stopwords is provided at `/home1/c/cis530/hw4/stopwords.txt`.

Again, it is ok to go slightly above exactly 100 words, as long as your summarizer stops adding sentences when your summary is already above the length limit. Please follow the same naming convention as in problem 2.1 but use different directory names. *As part of your submission, submit a single summary file that you have created for one of the document clusters.*

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

2.3 Evaluating the summaries (15 points)

We will use the ROUGE system[2] to evaluate the two summarizers. ROUGE is the most commonly used way to evaluate content selection quality because it is cheap, fast and objective, 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 a set of human summaries. It is a recall-oriented metric, which is more suitable for the summarization task given the variation in human content selection. ROUGE also has numerous parameters, including word stemming, stopwords removal and n-gram size.

ROUGE can be found under `/home1/c/cis530/hw4/rouge`. To score your summaries with Rouge, you simply need to call the script `ROUGE-1.5.5.pl`, which will run Rouge automatically on Eniac. The default settings which we will be using is as follows:

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

It is fine if the summaries are slightly more than 100 words, because ROUGE will automatically truncate the summaries to 100 words with command -l 100.

The configuration file, `config.xml`, tells ROUGE what peers (systems that you are evaluating) and models (human summaries) being used each time. We have prepared a `config.xml` in the `rouge` folder for you; to evaluate your summaries, you need to change the `<PEER-ROOT>` field to the corresponding folder containing your summary outputs. The `<PEERS>` field specifies the file names for the summaries. You can follow these names so you don't need to make more changes to `config.xml`.

Run rouge for your baseline and greedy KL summarizers. Create and submit a file `results.txt` and paste the rouge output in there like the following:

```
baseline
---------------------------------------------
baseline ROUGE-1 Average_R: xxx (95\%-conf.int. xxx)
baseline ROUGE-1 Average_P: xxx (95\%-conf.int. xxx)
baseline ROUGE-1 Average_F: xxx (95\%-conf.int. xxx)
---------------------------------------------
baseline ROUGE-2 Average_R: xxx (95\%-conf.int. xxx)
baseline ROUGE-2 Average_P: xxx (95\%-conf.int. xxx)
baseline ROUGE-2 Average_F: xxx (95\%-conf.int. xxx)

greedy kl
---------------------------------------------
baseline ROUGE-1 Average_R: xxx (95\%-conf.int. xxx)
baseline ROUGE-1 Average_P: xxx (95\%-conf.int. xxx)
baseline ROUGE-1 Average_F: xxx (95\%-conf.int. xxx)
---------------------------------------------
baseline ROUGE-2 Average_R: xxx (95\%-conf.int. xxx)
baseline ROUGE-2 Average_P: xxx (95\%-conf.int. xxx)
baseline ROUGE-2 Average_F: xxx (95\%-conf.int. xxx)
```

[2]Chin yew Lin. Rouge: a package for automatic evaluation of summaries. pages 2526, 2004.