CS 1671 - Homework 2

due date: 11:59pm 2/18/2016

In this homework, you will be experimenting with language models, and running intrinsic and extrinsic evaluations to assess the performance of your language model. Describe your implementational and design decisions for Part I, II, III, and report the performance of Part III in your write-up.

Part I: Building N-Gram Models.

For the basic part of this assignment, you need to implement three N-gram models: unigram, bigram, and trigram models. You need to interpolate them to get a smoothed N-gram model, as we've talked about in class. You also have to deal with out-of-vocabulary words as discussed in class. Finally, your code should apply the N-gram models to the test strings and compute the models' per-word perplexity.

Part II: Intrinsic Evaluation (for a Toy Problem)

Run your program on a toy problem and compute the per-word perplexity for the strings in the test file. Here is the expected program behavior (using Python as an example):

```
Ngram.py  <1|2|2s|3|3s> <trainfile> <devfile> <testfile>
```

- the first argument specifies whether you are building (1) a unigram model that supports "<UNK>" handling; (2) an unsmoothed MLE bigram; (2s) a bigram model interpolated with unigram model; (3) an unsmoothed MLE trigram model; (3s) a smoothed trigram model that is interpolated with bigram and unigram models.
- the second argument specifies the name of the train file (it may not be preprocessed with tokenization or sentence boundary detection).
- the third argument specifies the name of the development file (which you can use for setting the threshold for "<UNK>" replacement and for figuring out the lambda parameters for the interpolated smoothing model). This and the test file are preprocessed -- they are sentence segmented and marked with markers <s> and </s>. Notice that each sentence does not end with a period. </s> functions as the period. (This is mirrors the format of the "real" data set in Part III)
- the fourth argument specifies the name of the test file.

The program should output the per-word perplexity of each test sentence to standard out, one per line. Here is a sample training file, a sample development file, and a sample test file.

Part III: Extrinsic Evaluation

In this part, you will evaluate the performance of your language model through a *sentence completion* task, where you have to find the most appropriate word to fill in the blank. Here is an example:

The metal work was in the form of a double ring , but it had been bent
and _____ out of its original shape.

a) marched
b) faded
c) wriggled
d) poured
e) twisted

You can use a language model to determine which of the five candidates is the most appropriate filler for the blank (i.e., resulting in a complete sentence with the lowest perplexity). To do so, you need to adapt your implementation of N-gram models to work with real-world texts; this involves drastically increase your model's ability to handle large vocabulary, sparse data storage, and other efficiency related issues.

The test set we will use for this part of the assignment comes from the Microsoft Research Sentence Completion Challenge. This is manually designed to be difficult for N-gram models, so you should expect your N-gram models' performances to be quite low. Nonetheless, your objective is to determine whether/how much the performance improves as you use higher-order N-gram models. You can also try varying the ways in which you set the lambda parameters for the interpolation portion. (You'd need to describe what you do carefully in the write up).

Evaluation Data

The data discussed in this section is for evaluation (i.e., development and testing). Download the data here. The following 6 files are provided:

1. `Holmes.human_format.answers.txt`
2. `Holmes.human_format.questions.txt`
3. `Holmes.lm_format.answers.txt`
4. `Holmes.lm_format.questions.txt`
5. `Holmes.machine_format.answers.txt`
6. `Holmes.machine_format.questions.txt`

These question and answer files are in 3 formats:

Human-readable Format

Files 1 and 2 have the questions and answers in a human-readable form. File 1 consists of questions in the format shown in Figure 1. File 2 consists of the answer for each question. These files are provided to help you understand the task.

Language Model Format

Files 3 and 4 are provided to support training a language model. They are in a convenient language model format, where each sentence has a leading `<s>` token and an ending `</s>` token. Notice that each sentence does not end with a period. `</s>` functions as the period.

The lines of File 3 come as groups of five. In each group, there is a sentence generated by putting a candidate word into the blank. So your language model can simply score each sentence to obtain the score for each candidate.

File 4 has the correct answer for each group of sentences in File 5. The number of lines in File 6 is a fifth of that of File 3.

Machine Readable Format

Files 5 and 6 are the questions and answers in a format suitable for reading by your program. The word where the blank should be has a bracket around it. Each sentence is labeled by *XY*, where *X* (an integer) is the question number, and *Y* (a letter ranging from a to e) is the candidate number. This is provided for more general purposes, in case you want to add more processing in addition to language modeling.

For more information about how the data is generated, read this document.

Development & Testing Split

The original sentence completion task recommended using the first 520 sentences as the development set and last 520 sentences as the test set. Follow this splitting for convenience, as the evaluation script provided for you (described below) uses this splitting. You will have to modify the script or write your own if you want to split the evaluation data in another way.

Tune your model using only the development set. After you are done tuning, report your final performance on the test set.

Evaluation Metrics

Report the **accuracy** as the performance of your model on the test set. Accuracy here means out of all the questions, how many did your model answer correctly. Formally,

$$Accuracy = \frac{\#correctly\ answered\ questions}{\#questions}$$

To calculate accuracy, you are recommended to use the scripts provided in the test data bundle by the original challenge. To use the script, your model should generate output of the following format:

```
<s> John comprehends a cake </s>  -98.4
<s> John ate a cake </s>     -86.12
<s> John read a cake </s>    -00.2
<s> John punished a cake </s>     -100.4
<s> John could a cake </s>   -99.28
... (other groups of sentences)
```

In this format, each line contains two parts that are separated by a tab. The first part is the sentence, the second part is the score produced by your model. Every 5 lines should correspond to one sentence (each line has a sentence for each candidate). Name this file `output.txt`. In your terminal, change directory to where the scripts `bestof5.pl` and `score.pl` are. Executing the following two commands will produce the accuracy on the evaluation set.

```
cat output.txt | ./bestof5.pl > tmp.txt
./score.pl tmp.txt Holmes.lm_format.answers.txt
```

Replace `Holmes.lm_format.answers.txt` with the path to where you store the answer file in LM format. The script `bestof5.pl` selects the sentence in each group with the highest score. The script `score.pl` compares the best sentences to the answer. Report your accuracy on the test set by reading the output (the bolded line):

```
...
372 of 1040 correct
```

```
Overall average: 35.7692307692308%
dev: 36.3461538461538%
test: 35.1923076923077%
...
```

Training data
To train your language model, you must use a different corpus from the evaluation set. A collection of 19th-century novels from Project Gutenberg is provided (download here). Note that the training files have not been preprocessed at all, and are not tokenized in a way that matches the evaluation data.

Bonus (Do not work on this until you are done with the main assignment)

In the bonus portion, you are invited to try to improve the performance by using off-the-shelf packages (see links provided at the end of the assignment) that support higher order N-grams (N=5, say) and different smoothing technique, or other factors that you find helpful for the end task. Moreover, you are not limited to use merely language models. You may process the sentences in more advanced ways to gain more performance. Experiment and compare different approaches, then discuss and analyze the results in the write-up.

Depending on the quality of the approach and/or result, you may get up to 7% extra bonus on this assignment (i.e., if you got a 4 on the regular assignment, getting 7% will make your grade effectively a 5; and if you got a 5 originally, it will make your actual score value to be 107%.).

What to Turn In

- README.txt -- this file is for telling the grader how to run your program and any known bugs or issues. (Please try to run your program with the toy problem on a SENSQ lab machine to make sure that the grader won't run into any issues with missing libraries etc.) In particular, you should describe your Part III pipeline carefully and in detail -- if you're using off-the-shelf tokenizers or sentence splitters, say which ones and how; if you're doing some other preprocessing of the training data, say how.
- code -- For Part I and II, you just need to turn in your Ngram.py (or equivalent) file. For Part III, submit all programs that you wrote yourself, but not scripts or packages you've downloaded or libraries you've used. Do not submit your train file either -- they are all too large.
- Report write-up -- your report should address the following questions:
    1. What problems can occur (or *have occurred* in your experiments, if there is any) when the *N*-gram language model you implemented in Part I is trained on a *large* training data such as the Project Gutenberg? Given that you have access to the development data, how did it help you to adapt and/or train your models?
    2. How did your models perform? Were they as you expected? Why wasn't the *N*-gram language model alone good enough for the sentence completion task? What additional tools or techniques do you think are necessary? Can the language model itself be changed to account for more ambiguities?

3. If you did the bonus part -- what different strategies did you try? Discuss why these strategies worked or didn't work.

Grading

The assignment will be graded on a non-linear 5-point scale.

- 5 (equivalent to 100): Excellent work. The program for Part I works for the sample toy problem, the intrinsic evaluation for Part II is correct; the program and evaluation also works as expected for Part III extrinsic evaluation; the report is clear and insightful.
- 4 (equivalent to 93): Very good work. The program for Part I works for the sample toy problem, the intrinsic evaluation for Part II is correct; substantial effort has been made to adapt the program for the extrinsic evaluation in Part III, but there are some minor problems (e.g., too slow, or suboptimal design choices); the report addresses all the issues we asked about.
- 3 (equivalent to 80): OK work. The program for Part I works for the sample toy problem, the intrinsic evaluation for Part II is correct; some effort has been made to adapt the program for the extrinsic evaluation in Part III, but there are some major problems; the report only cursorily addresses the issues we've asked about.
- 2 (equivalent to 60): Basic work. The program for Part I mostly works for the sample toy problem; the per-word perplexity calculations for Part II are mostly reasonable, minimal attempt at Part III.
- 1 (equivalent to 40): Incomplete work. The program for Parts I and II are partially completed (e.g., no smoothing).
- 0 (equivalent to 0): no submission.

Useful Links

- SRILM. http://www.speech.sri.com/projects/srilm/
- IRSTLM. http://sourceforge.net/projects/irstlm/
- A tokenization script from the Moses machine translation system. https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl
- A Python project for cleaning the texts from Project Gutenberg. https://github.com/c-w/Gutenberg
- The original MSR Sentence Completion Challenge page. http://research.microsoft.com/en-us/projects/scc/
- Computational Approaches to Sentence Completion. http://research.microsoft.com/apps/pubs/default.aspx?id=163344