CS4740: Introduction to NLP
Project 1: Language Modeling

Due electronically by 11:59 PM, Monday, February 24, 2014

Code for Part 1 must be submitted electronically on **Monday, February 17**.

1 Overall Goal

This project is an open-ended programming assignment in which you are to implement a collection of n-gram-based language models. You must work in groups of 2–3 students. Students in the same group get the same grade. In the end of your report, you are to specify the responsibilities of each group member in contributing to the project (i.e. what part was coded/analyzed by whom). Please form groups using CMS. You can find partners using Piazza.

2 Programming Portion

1. **Unsmoothed ngrams.** Write a program that computes unsmoothed unigrams and bigrams for an arbitrary text corpus. You must write all of the code yourself, but you may use any programming language(s) you like.

   (a) There are training, validation, and test sets provided for two corpora via CMS. One corpus is the King James Bible, and the other is a collection of hotel reviews. Note that the hotel review corpus contains additional labels for each review (those will be used for predicting review truthfulness in part 5 of this problem). For all other parts, we are interested in just the text of the reviews. You will need to strip away the labels and aggregate only the text of the reviews for use in developing your language model. The Bible corpus is broken up into documents in an XML-style format, and requires a small amount of pre-processing to extract raw text. Note that individual sentences in the Bible are are separated by newlines. The hotel review corpus, however, requires

you to process sentence boundaries with a sentence segmentation tool of your choice. Include sentence boundaries as tokens in your language model.

2. **Random sentence generation.** Write code for generating random sentences based on the unigram or bigram language model (as was illustrated in class). Experiment with the random sentence generator after training on each of the corpora. **Examples of sentences generated by your system will be part of the final report. Also include them with your Part 1 submission.**

   Steps 1 and 2 constitute Part 1 of the assignment.

3. **Smoothing; unknown words.** Next, implement Good-Turing smoothing as well as an approach for handling **unknown words** in the test data. (You will need this for Step 4.)

4. **Perplexity.** Finally, implement code that computes the **perplexity** of a test set. Compute the perplexity of each of the language models (trained on each of the corpora) on the two test sets. Compute and report perplexity as follows:

$$PP = \left(\prod_{i=0}^{N} \frac{1}{P(w_i|w_{i-1}, \ldots, w_{i-n+1})} \right)^{1/N}$$

   where N is the total number of running tokens in the test corpus and $P(w_i|w_{i-1}, \ldots, w_{i-n+1})$ is $n$gram probability of your model. Note that to properly account for sentence boundaries, you need to include a special sentence boundary token in computing the ngram probabilities (and account for it in the total count of running tokens in the corpus).

5. **Open-ended extension.** Using the n-gram statistics, random sentence generator, and perplexity computation as tools, decide on at least one additional extension to implement. The idea is to identify some aspect of the language model or random sentence generator to improve or generalize, and then to implement an extension that will fix this issue or problem. Section 3 below provides some ideas. In addition, it is important to evaluate your extension quantitatively and/or qualitatively to determine whether or not the extension obtained the expected behavior.

3 Menu of extensions

You must implement one item from this list of extensions or come up with your own extension. Whatever you choose to do, explain it clearly in your writeup.

1. Implement a trigram (or 4-gram, or general n-gram) model.
2. Smoothing. Implement another smoothing method.
3. Interpolation. Implement an interpolation method, e.g. Linear interpolation, Deleted interpolation, Katz's backoff.
4. Nontrivial unknown word handling. Develop and implement a method for better handling of unknown words.
5. Employ the language model in the service of another NLP or speech application.
6. Implement a modification that makes use of the validation set.
7. **Programming Project.** In this part of the project you will predict truthfulness of hotel reviews based on the unigram or bigram language model. Use the HotelReview dataset in CMS for training, validation and testing. The data files have the following format:

   ```
   IsTruthFul,IsPositive,review
   ```

   where `isTruthFul` and `IsPositive` are binary labels and `review` is the text of the review. You should use only the training data and potentially the validation data for developing your model. The project report should describe the approach you employed for predicting review truthfulness and should include a table that shows the accuracy of the approach on the validation data. The test data should be reserved for use only at the end for producing the evaluation result.

   Kaggle will be used to evaluate your system. Instructions and a link for using the system will be provided on Piazza.

4 The Report

You should submit a short document (6 pages will suffice) that describes your work. Your report should contain a section for every Step in the Pro-

gramming Portion section above as well as a short section that explains how you divided up the work for the project.

For each Step, you should describe your approach (or in the case of Step 1, your data structures), include relevant examples where appropriate, as well as snippets of code (see below) that support your explanations. In particular, include examples of the random generator in action; be sure to indicate which smoothing method you implemented and how you handled unknown words; include and discuss the results of the perplexity experiments; describe the extension(s) that you decided to implement and why; can you perform any experiments that support the need for the extension/change? what experiments did you run, or analysis perform, to show whether or not your extension had the desired effect?

Code: Include snippets of your code along with the description of your approach. Include only relevant portions of the code (such as a few lines from a function that accomplish the task that you are describing in a nearby paragraph, to help us understand your implementation). **Do not include boilerplate code or any code that is not directly helping back up your explanations in the report in answering a particular question. If the snippet of code calls any other functions, include only those that are conceptually essential to the implementation (i.e. no functions for processing or loading data, or performing trivial mathematical computations).**.

5 Grading Guide

- Part 1: progress on unigram and bigram table construction algorithms and random sentence generator (10%)
- design and implementation of the unigram and bigram table construction algorithms and random sentence generator (10% of the grade)
- design and implementation of the random sentence generator (5%)
- design and implementation of your selected extension (25%).
- experiment design and methodology; clarity and quality of the report (50%)

6 What to submit

By anytime Monday, February 17, submit the code for Part 1 and examples from your random sentence generator.

By Monday, February 24, 11:59PM, submit the full assignment to CMS. This consists of a .zip or .tar file — one submission per group, containing the following:

- source code and executables
- the report