CS4740: Intro to NLP
Project 1: Language Modeling

Part 1 due electronically by Thu 9/17 @ 11:59pm
Part 2 due electronically by Fri 9/25 @ 11:59pm

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 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 (e.g. what part was coded/analyzed by whom). Please form groups using CMS. You can find partners using Piazza.

2 Programming Part

1. **Unsmoothed n-grams.** 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. Assume you have raw text, so you may need to do tokenization, based on the design decisions you make. You **may** use existing tools just for the purpose of tokenization and preprocessing whereas you are **not allowed** to do this for computing $n$grams, as mentioned.

   (a) **Data and preprocessing.** You are given a selection of books from gutenberg.org which are separated according to their genres: children's books, crime/detective and history/biography. You will use the books as your corpora to train your language models. Preferably, you should consider every genre (folder) as a single corpus, and generate a language model for each, but you may play with training over the entire set of books or just a single book to observe what type of different behaviors you get. As mentioned above, the files are raw text (.txt), so you will need to perform sentence splitting and/or tokenization (with a method of your choice).

2. **Random sentence generation.** Write code for generating random sentences based on a unigram or bigram language model (as was illustrated in class). Experiment with the random sentence generator after training on

each of the corpora. Also experiment with seeding, i.e., starting from an incomplete sentence of your choice and completing it by generating from your language model, instead of generating from scratch. 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 and unknown words.** Implement Good-Turing smoothing as well as a way to handle **unknown words** in the test data. Explain why you need this for steps 4 and 5.

4. **Perplexity.** Implement code to compute perplexity of a test set. Compute the perplexity of each of the language models (trained on each of the corpora) on each of the test corpora. Compute and report perplexity as follows:

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

   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.
   Also observe the second definition, perplexity is a function of the average (per-word) log probability. This will help you avoid underflow errors.

5. **Genre Classification.** In this part of the project you will need to determine a method that uses your language models to classify books from the test sets according to their genres. Use the folder structure to determine the training and test sets, as well as the class labels (e.g. all the books in `train_books/crime/` are part of the training set with a class label of *crime/detective* (or shortly, *crime*). You should use only the training data for developing your model. The project report should describe the approach you employed for predicting genres. The test data should be reserved for use only at the end for producing the evaluation result. Report how many of the test books you got right and interpret your results if you misclassify any.

6. **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. Measure perplexity of both the training and the test corpora as you increase n. Interpret the results you observe regarding the changes in perplexity.

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. Redo part (or parts) of the exercise at the sentence level, i.e. by modeling sentences out of their context instead of modeling the whole text (books). In this case you will need to define a way to measure the probability of a text that is larger than sentence, explain your assumptions regarding this. Explain the advantages / disadvantages of modeling text this way.

7. An $n$gram language model is just a function from n words to a real number (probability) that is implemented as a lookup table (alternatively, you can see it as a function of $n-1$ words to a distribution over your vocabulary). Define this function some other way than a lookup table (e.g. many machine learning methods that compute a function of a sequence of words to a real number), and describe a method to estimate that function from data. Implement this method.

8. Redo part (or parts) of the exercise using Brown clusters (`http://metaoptimize.com/projects/wordreprs/`) and collapsing the word types that belong to a cluster into a single word type. For step 2, you will need to define a way to generate sentences, since you will need to generate words from clusters. You may try Brown clusters at different granularities and explain the advantages / disadvantages. **You don't need to "learn" the Brown clusters themselves i.e. you don't need to run the Brown clustering method. Just use pretrained Brown clusters.**

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 Programming

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 smoothing, unknown word approach, and perplexity calculation (10%)
- design and implementation of your selected extension (10%)
- design and implementation of deception detection system (10%)
- experiment design and methodology; clarity and quality of the report (50%)

6 What to submit

By 11:59pm, September 17, submit the code for Part 1 and examples from your random sentence generator.

By 11:59pm, September 25, 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