Natural Language Processing - Project 1
CSE 398/498-013
Prof. Sihong Xie
Due Date: 11:59pm, Sep 24, 2018

1 Problem Statement

Given a training corpus, you will estimate bigram language models with proper smoothings and use the models to find mis-spellings in a test corpus.

1.1 Bigram Language Models

Recall that a bigram model is a bunch of conditional probabilities $p(w|v)$, where $w, v \in V$ and V is a given vocabulary. The simpliest method to estimate these probabilities (or to learn the language model) is to compute the frequencies of tokens (w, v) and v, and take their ratio:

$$P(w|v) = \frac{C(w,v)}{C(v)}. \tag{1}$$

We have learned that there are issues in this estimation, such as zero counts.

1.2 Smoothing

Smoothing is a set of techniques to handle the zero counts. The idea is to shave some probability mass off the seen tokens for those that do not appear. For example, Laplacian smoothing adds one to each count, and we got

$$P_L(w,v) = \frac{C(w,v)+1}{N+|V|}, \tag{2}$$

where N is the total number of all tokens (two consecutive words (w, v) in this case) in the corpus. The **estimated** frequencies of the token (w, v) is:

$$C^*(w,v) = P_L(w,v) \times N. \tag{3}$$

However, this smoothing gives too much probability mass to the unseen ones if the number of unseen tokens is too large. A better technique called Good-Turing smoothing uses the following estimation:

$$C^* = (C+1) \times \frac{N_{C+1}}{N_C}, \quad \text{if } C < k, \tag{4}$$

where N_C is the frequency of tokens with the frequency C (frequencies of frequencies, or ff for short). If $C \geq k$, then no smoothing is applied and C is retained.

Note that N_C is a function of C and roughly follows the following relationship:

$$\log N_C = a + b \log C \tag{5}$$

Figure 1 shows an example plot of the two quantities with a linear regression model[1].

[1] If you don't know what this is, Google it. We use the Apache SimpleRegression class to find the relationship between $\log N_C$ and $\log C$.

Figure 1: Plot of frequencies of frequencies in the log scale, with a line fitted to the points.

1.3 Spell checking using bigrams

For a pair of confusing words, say, *advice vs. advise*, if we are not sure about which word is the correct one under some context, we can compared the bigram proabilities $p(\text{advise}|\text{previous word})$ and $p(\text{advice}|\text{previous word})$. If a word appeared in the text has lower bigram probability than its alternative, then we regard the word appeared an error and should be replaced by the alternative.

2 Questions

Answer the following two questions in your report

- Show that it makes sense to set $p(w, v) = \frac{N_1}{N}$ for those unseen tokens ((w, v) with $C(w, v) = 0$).
- Calculate the probability mass reserved for the unseen tokens when GT smoothing is used, and compare the mass to the mass reserved when Laplacian smoothing is used.

3 Experiments

In the experiment, training and test corpus will be given to you. Training data is obtained from reviews of a product, and we assume that there is not mis-spelling. The test data is polluted by changing every word that is in our confusing word list (also given to you) to its alternative.

3.1 Data Exploration

- Find N_0, the number of zero frequency tokens (w, v).
- Find the probability mass reserved for the zero frequency tokens (w, v) in Laplacian and GT smoothings.

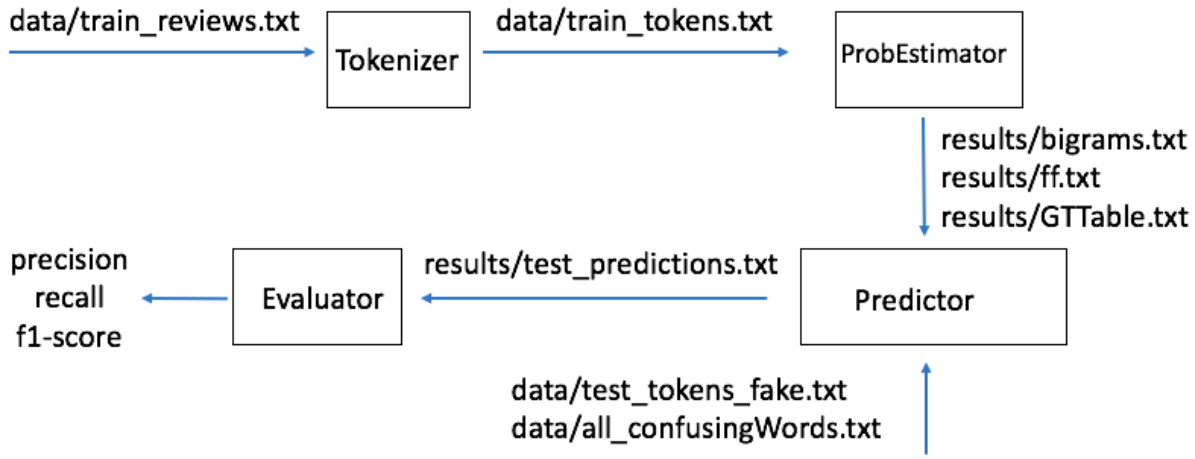

Figure 2: Over project design

- Use the training data to find N_C for each C. Take the logorithm of both quantities and plot a figure like Figure 1 (Matlab codes provided if you want to use Matlab for plotting and line fitting). Add the figure in your report.

3.2 Spell checking

Train two bigram models using Laplacian and GT smoothing, respectively. Find the locations of the errors in the test data using two models, and report their precisions, recalls and F1 scores.

$$precision = \frac{\#\text{correctly detected errors}}{\#\text{all detected errors}}, \tag{6}$$

$$recall = \frac{\#\text{correctly detected errors}}{\#\text{all actual errors}} \tag{7}$$

$$f1 = \frac{precision \times recall}{precision + recall} \tag{8}$$

3.3 Code Design

The overall design of the project is shown in Figure 2. Data flow over the arrows into and out of individual classes/programs (given in boxes). Tokenizer takes data/train_reviews.txt as input and outputs data/train_tokens.txt, which is fed to ProbEstimator to produce three files (shown in the figure) in the results folder. Predictor takes the previous three files, plus data/test_tokens_fake.txt, to predict the locations where there is an error, stored in the results/file test_predictions.txt. Lastly, Evaluator takes the results/test_predictions.txt and data/test_ground_truth.txt to calculate performance. See the attached bash script to understand the flow.

You're going to program two classes, the ProbEstimator and Predictor, in Java, and the Tokenizer.java and Evaluator.java are given to you. Bash script files (build_java.sh and run_java.sh) are provided to compile and run the programs. The necessary third party packages are in the folder "jars" in the zipped file. The file data/all_confusingWords.txt gives you pairs of words that are confusing. Your predictor decides, for each token that is in data/test_tokens_fake.txt and in any confusing pair, which word in that pair should be the correct one to use. If a wrong word is used, output the error location to results/test_predictions.txt. When grading your project, I will run the bash scripts to compile and run your codes on a Sunlab machine.

3.4 Data formats

Refer to the corresponding files in the zipped file to learn the formats of data/train_reviews.txt and data/test_tokens_fake.txt. Each line of the results/test_predictions.txt file looks like

```
5481:114,
5608:8,
5620:66,73,
```

indicating that the 5481-th sentence has an error in token location 114, and the 5620-th sentence has two errors in token locations 66 and 73. All sentence numbers and token locations are 0-based. The ground truth error locations are formatted similarly, although the file is not given to you. You can design the formats of the other files.

4 Deliverables

Download the provided zip file, add your ProbEstimator.java and Predictor.java files to the src folder. Also add README.txt and report_p1.pdf to the root folder (one level above src). Then zip the whole folder to <your Lehigh id>_p1.zip and upload it to coursesite.

The README.txt describes what works and what does not, any improvements you think that should earn you extra credits. The report_p1.pdf file contains the answers to questions in Section 2 and 3.1. Before you submit your project, compile and run it on a Sunlab machine where your projects will be graded.

5 Grading metrics

Correctness of the implementation will be the most importnt factor for grading. Besides, how good your GT estimators are, how fast it runs and how much memory it takes will be used to rank your project against others' and determine one third of your total grade of this project.