CSE 390 Assignment 1

Due: February 25th, 2017 by 11:59 pm

This assignment has a main programming component and two written questions.

You will build bigram language models using MLE, Laplace Smoothing, Interpolation, and Katz backoff methods. You will train the models on a training corpus and evaluate them on a test corpus. You will also build a command-line application that will output the bigram conditional probability of a specific pair of words.

Programming Language Choice:

- We recommend that you implement in Python. This is a good scripting language to be familiar with if you want to build NLP applications. Many standard frameworks are built in Python.
- You can implement the program in any language of choice. No penalties for implementing in a different language.
- However, if you do implement in a different language you will have to demo it with the TA.

1 Preprocessing

As with many NLP applications, the first step is to prepare the corpus. This involves the following three steps.

1. Break the text into sentences. This is a non-trivial task, which requires some smarts. However, for this assignment you will build a dumb sentence segmentation function, which splits the text on the dot symbols. For example, if the text was "Mr. John likes to tip a $1.50.", your function should output three sentences: 1) Mr, 2) John likes to tip a $1, and 3) 50.
2. Break the sentences into words. This step is referred to as Tokenization. This is also a non-trivial task but we will do a dumb tokenizer. You tokenizer will simply split the words on blank spaces. For the sentence, "John likes to tip a $1" your function should output a sequence of 6 words [John, likes, to, tip, a, $1].
3. Lower case the words.

2 Applications [90 points]

You will produce three applications.

1. Language Model Builder [30 points] – This application should take in a text file (train.txt) and run it through the pre-processing steps and output three files (i) bigram language model, (ii) unigram language model and (iii) top Laplace bigrams file.

   (a) bigram.lm – This bigram language model file should contains all bigrams seen in training along with their observation counts, the MLE, Interpolated smoothing, and Katz conditional and joint probabilities for the bigram. The idea is to have the training bigrams stored so we don't spend time computing and instead just look them up during test time. So each line in the language model file should include the following information:

   - x – first word
   - y – second word
   - $\#(x, y)$ – number of times y followed the word x
   - $Pr_{MLE}(y|x)$
   - $Pr_L(y|x)$
   - $Pr_{Int}(y|x)$
   - $Pr_{AD}(y|x)$

   Note 1: $Pr_K(y|x) = Pr_{AD}(y|x)$ for bigrams seen in training, which is why we don't need a separate entry for Pr_K.

   See next section for details on how to compute each of these probabilities.

   (b) unigram.lm – This unigram language model file should contain all the unigrams seen in the training along with their observation counts. So each line in the language model file should include the following information:

   - x – word
   - $\#(x)$ – number of times word was observed in training.

   (c) top-bigrams.txt – This top bigrams file should contain the top 20 bigrams (i.e., word pairs) according to their joint probability with Laplace smoothing $Pr_L(x, y)$. You can use $Pr_L(x, y) = Pr_L(x)Pr_L(y|x)$ to compute the joint probability.

2. Bigram Query Application – This should be an interactive command line application, which allows the user to interactively specify a pair of words (x, y) and type of estimate desired – *MLE*, *Laplace*, *Interpolated*, or *Katz* – and should output the corresponding estimate. Specifically, the program will be invoked with the following command:

   `python bigram-query.py ⟨/path/to/bigram/lm/file⟩ \`
   `⟨/path/to/unigram/lm/file⟩ ⟨x⟩ ⟨smoothing: M, L, I, K⟩`

This command should return the top ten words that is likely to follow the word x according to the corresponding bigram conditional probability $Pr(y|x)$. Each line should include the word and the bigram conditional probability.

3. Language Model Evaluator – This application should take the language model file and a test file (test.txt) as inputs. It should output perplexities under the (1) *Laplace bigram*, and (2) *Interpolated bigram* methods, and (3) *Laplace* unigram smoothing methods. **I haven't listed how to compute the unigram perplexities but you can figure it out yourself.**

   (a) Apply the same sentence segmenting and word tokenization steps on the test file first.

   (b) Insert the start ⟨s⟩ and end ⟨/s⟩ tokens at the beginning and end of each sentence.

   (c) Create a single sequence of tokens that concatenates all the words in all the sentences. Suppose there were m sentences in total in the test set. Now you are left with a long sequence of words:
   $Test = w_0^{(1)}, w_1^{(1)}, \cdots w_{n_1+1}^{(1)}, w_0^{(2)}, w_1^{(2)}, \cdots, w_{n_m+1}^{(m)}$. Suppose there are a total of N words in this sequence (including all start and end symbols).

   (d) Then, you compute perplexity as follows:

   $$\text{PP(B; test)} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 Pr_B(w_i|w_{i-1})}$$

   where, B is one of the smoothing methods: *MLE*, *Laplace*, or *Interpolated*.

   (e) This application should run by evoking the following command:
   `python perplexity.py ⟨/path/to/bigram/lm/file⟩ \`
   `⟨/path/to/unigram/lm/file⟩ \`
   `⟨/path/to/test/file⟩`

   It should return the three perplexities, *Laplace bigram*, *Interpolated bigram*, and *Laplace unigram* models.

3 Estimation methods

1. MLE – Implement a function that takes as input two words (x, y) outputs the MLE conditional probability of a bigram i.e., $Pr_{MLE}(y|x)$.

2. Laplace Smoothing (Add-1 smoothing) – This function should output Laplace smoothed bigram conditional probability.

   $$Pr_L(y|x) = \frac{\#(x,y)+1}{\#(x)+V'}$$

   where $V' = V + 1$, to account for unseen words.

3. Interpolated Smoothing – Smooth the bigram probability with the unigram estimate.

$$Pr_{Int}(y|x) = \lambda Pr_{MLE}(y|x) + (1-\lambda)Pr_L(y)$$

You need to figure out a good setting for λ. You can do this by evaluating the perplexity of the interpolated method with different lambda settings ($\lambda \in 0.1, 0.3, 0.5, 0.7, 0.9$ on the development set (dev.txt). Pick the setting with the lowest perplexity and use that to report perplexity on the test set.

Note 2: The second term is using Laplace smoothed estimate of unigrams. Otherwise, we will run into the zero probability issue when y is an unseen word. As with the bigram case, we use $V' = V + 1$ to account for test words that are unseen in training.

4. Katz Back-off – Implement the back-off method using the AD probability, as the discounted probability estimate – $Pr^*(y|x) = Pr_{AD}(y|x) = \frac{\#(x,y)-D}{\#(x)}$, with $D = 0.5$.

$$Pr_K(y|x) = \begin{cases} Pr_{AD}(y|x) & \text{if } \#(x,y) > 0 \\ \alpha(x)\beta(y) & \text{if } \#(x,y) = 0 \end{cases} \quad (1)$$

where,

$$\alpha(x) = 1 - \sum_{w:\#(x,w)>0} Pr_{AD}(w|x)$$

$$\beta(y) = \frac{Pr_L(y)}{\sum_{w:\#(x,w)=0} Pr_L(w)}$$

Note 3: This is computation can be slow, since the summation is over words that don't follow x, which is usually most of the vocabulary. Here is a hint to speed things up: Notice that $Pr_L(w)$ is a probability distribution, which means that if you sum up the probabilities going over all w in the vocabulary they should sum up to one. You can see that the summation for β is over a subset of terms in the vocabulary.

4 Data

- train.txt – Training set to be used for estimating the language models and to obtain the top 20 bigrams.
- dev.txt – Development set to be used for figuring out λ.
- test.txt – Test set to be used for evaluating the language models.

All files are available as part of the Blackboard assignment.

5 Submission

1. You should submit the source code via Blackboard. Please name the functions clearly, and add some description of what the methods do. Please include a README on how to run your programs.
2. Run the LM Builder on the training data and for MLE, Laplace, and Interpolation smoothing methods, use the LM Evaluator to obtain its perplexity on the test set
3. Write a brief report describing your work. It should include the following.
   - Perplexity of the Laplace, and Interpolation bigram models on the test set.
   - Perplexity of the unigram Laplace model on the test set.
   - Explain the perplexity numbers you observe. Why is one higher than the other?
   - Include the top 20 bigrams for Laplace.
4. **Written assignment [10 points]:** Should not take you more than one page each.
   (a) Question 1: Specify how to compute $Pr_{AD}(y|x)$ for cases where $(x, y) = 0$, so that it is a valid probability distribution. You should provide that with your specification, $Pr_{AD}(y|x)$ is indeed a valid probability distribution.
   (b) Question 2: Please show that $Pr_L(y|x)$ is a valid probability distribution.

6 Scoring

Scoring specifics will be set by the grader. It will involve the following criteria.

- Each application should run without crashing for appropriate inputs and outputs.
- The top bigrams file should contain entries that are reasonably close to the entries our program outputs. We won't provide the expected output before submission. So you will have to do your own testing.
- The perplexity numbers should be close to the number returned by our implementation.
- The bigram query application should return similar outputs for the words that we test on.