N-gram Programming Lab 1

Susan W. Brown

Goal for program

- One short sentence describing what you want from the program (What will it output?)
- Every sentence in a file will get a unigram probability, bigram probability, and smoothed bigram probability

What do we need?

- To calculate a sentence's probability, we need the individual unigram and/or bigram probabilities
- To calculate word and bigram probabilities, we need word and bigram counts from a training corpus

Pseudocode

- GetUnigramCounts()
- GetBigramCounts()
- Or GetUnigramAndBigramCounts()
- GetUnigramProbabilities()
- GetBigramProbabilities()
- GetSmoothedProbabilities()
- Or GetBigramAndSmoothedProb()
- GetUnigramSentProbability()
- GetBigramSentProbability()
- GetSmoothedProbability()

Main program running the functions

Planning: What is the goal for the unigram model?

- Use the probability of each word in a sentence to get the probability of that entire sequence.
- What we need:
  - Probabilities for each word
  - Training corpus to calculate a word's probability
  - To calculate a word's probability:
    - Count of the word in the corpus/Total tokens in corpus

Function for counting words

- What information will we use? (What information will we send in to this function?
  - training file or lines from a training file
- What output do we want?
  - each word paired with its count
  - total word count (tokens)

```
def get_unigram_cts(file):
        …
        return some data structure with word-count
                pairs, integer of total tokens
```

body of get_word_counts()

- What structure do we want for the word-count pairs?
  - List with tuples? [(wd1, ct1), (w2, ct2), …]
  - Dictionary? {wd1:ct1, wd2:ct2, …}
- What process for getting counts?

get_word_cts()

```
def get_word_cts(file):
        create structures and variables
                (word_cts = …, total_tokens = …)
        read line
        divide into words; do other prep
        add word to list/dictionary (what if it's
                already there?)
        increment its count
        return word_cts, total_tokens
```

Calculate_probabilities()

- What do we need to send in to the function?
- What output do we want?

```
def calc_probs(word_cts, total_tokens):
        …
        return data structure with word-prob pairs
```

How do we it?

calculate_probabilities()

```
def calc_probs(word_cts, total_tokens):
        Go through the list/dict one by one
                calculate probability—ct/tokens
                put in new structure
        return data structure with word-prob pairs
```

calc_unigram_sent_probs()

- What does this function need?
  - test file line
  - unigram probabilities
- What will it output?
  - float of sentence probability
- How do you get sentence probability?
  - (multiply unigram probabilities)
  - add with the log of the probabilities

Main program

- Call functions with specific arguments (e.g. sys.argv[1])
- Write output
- Details we haven't included yet:
  - lowercase
  - log probabilities
  - round decimals to 4 places

import modules

- sys
- math

Bigram pseudocode

- Same basic structure as unigram
  - get bigram counts
  - calculate probabilities
  - use to get new sent probs
- Different details for counting and calculating probabilities
- Data structure could be two-dimensional (if storing all possible bigrams)
- Or one-dimensional (if only storing seen bigrams)

get_bigram_cts()

- create structures and variables
- go through file, process sentence

Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

get_bigram_probs()

- create a new 2-D structure (for all possible bigrams) to store the probabilities
- or new 1-D structure (for only seen bigrams)
- loop through the structure to fill in

Calculate the bigram probabilities

- use your unigram counts for the denominator
- use bigram counts for numerator

Calculate a new sentence probability

- process the sentence (e.g., a list of tokens)
- move through the list, setting and resetting *prefix* and *word* variables

What about unseen bigrams?

- Depends on how you set up your bigram probability structure
- If you did a matrix of *all words X all words*, you should have lots of zeros
- Test if bigram_probs[prefix][word] is 0; if yes, whole sentence probability will be 0
- Print “Undefined”
- If you only stored found bigrams, make sure the new bigram is there first. If not, “Undefined”

Laplace Smoothing

- Also called Add-One smoothing
- Just add one to all the counts!
- Very simple

- MLE estimate: $$P(w_i) = \frac{c_i}{N}$$

- Laplace estimate: $$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

- Reconstructed counts: $$c_i^* = (c_i + 1)\frac{N}{N + V}$$

Add- k Smoothing

- For bigram smoothing

$$P = \frac{\text{W1 W2 count} + .0001}{\text{W1 count} + V(.0001)}$$

- Same as bigram probability function, but
  - adjust the bigram and unigram counts first
  - no zeros to deal with
  - every sentence should have a probability