Homework 3: Language modeling (CS 2731 Fall 2023)

Due 2023-11-02, 11:59pm. *Instructions last updated 2023-11-01.*

In this assignment, you will build unigram, bigram, and trigram character language models (both unsmoothed and smoothed versions) for three languages, score a test document with each, and determine the language it is written in based on perplexity. You will also use your English language models to generate texts. You will critically examine all results. The learning goals of this assignment are to:

- Understand how to compute language model probabilities using maximum likelihood estimation.
- Implement basic smoothing and interpolation.
- Use the perplexity of a language model to perform language identification.
- Use a language model to probabilistically generate texts.

Data

The data for this project is available here: hw3_data.zip. It consists of:

- training.en - English training data
- training.es - Spanish training data
- training.de - German training data
- test - test document

1. Train character n-gram language models

To complete the assignment, you will need to write a program (from scratch) that:

- builds the models: reads in training data, collects counts for all **character** 1, 2, and 3-grams, estimates probabilities, and writes out the unigram, bigram, and trigram models (ngram probabilities) into files. Build separate models for each language.
- adjusts the counts: rebuilds the bigram and trigram language models using linear interpolation with lambdas equally weighted

You may make any additional assumptions and design decisions, but state them in your report (see below). For example, some design choices that could be made are how you want to handle uppercase and lowercase letters or how you want to handle digits. The choice made is up to you, we only require that you detail these decisions in your report and consider any implications of them in your results. There is no wrong choice here, and these decisions are typically made by NLP researchers when pre-processing data.

You may write your program in any TA-approved programming language (Python, Java, C/C++).

For this assignment you must implement the model generation from scratch, but you are allowed to use any resources or packages that help you manage your project, i.e. Github or any file i/o packages. If you have questions about this please ask.

Extra credit (3 points): Also implement add-one smoothing.

Deliverables for part 1

In your report, include:

1. a description of how you wrote your program, including all assumptions and design decisions
2. an excerpt of both the unsmoothed and interpolated (and optional add-one) trigram language models for English, displaying all n-grams and their probability with the two-character history *t h*
3. documentation that probability distributions for all language models are valid (sum to 1). Specifically, where w is a character (random variable) and c is the prior context of that character, this means that $\sum_i P(w_i|c) = 1, \forall c$. It is okay for sums to not be exactly 1, but they should be close to 1 (>0.97 and <1.03 is fine). Adding special start and end characters to every line when you read in the file may help probabilities sum to 1. If the prior context c has never been seen before by the model (and thus the sum of the probability distribution over all characters given that context is 0), it can simply be excluded from this calculation.

2. Use character n-gram language models

- Language identification: For all smoothed language models, read in the test document, apply the language model to all sentences (lines) in it separately. Calculate the perplexity of each sentence and the average perplexity across all sentences (lines) in the document. Based on the results, identify the language of the test document. You can use fill out and use the following Python function for perplexity (or something else if you want):

```
import math

def perplexity(text, model, n):
""" Args:
           text: a string of characters
           model: a matrix or df of the probabilities with rows as prefixes, columns as suffixes.
                     You can modify this depending on how you set up your model.
           n: n-gram order of the model

       Acknowledgment:
         https://towardsdatascience.com/perplexity-intuition-and-derivation-105dd481c8f3
         https://courses.cs.washington.edu/courses/csep517/18au/
         ChatGPT with GPT-3.5
    """

    # FILL IN: Remove any unseen characters from the text that have no unigram probability in the language

    N = len(text)
    char_probs = []
    for i in range(n-1, N):
        prefix = text[i-n+1:i]
        suffix = text[i]
        # FILL IN: look up the probability in the model of the suffix given the prefix
        prob =
        char_probs.append(math.log2(prob))
    neg_log_lik = -1 * sum(char_probs) # negative log-likelihood of the text
    ppl = 2 ** (neg_log_lik/(N - n + 1)) # 2 to the power of the negative log likelihood of the words divided by #ngrams
    return ppl
```

- Text generation: For only the bigram and trigram language models trained on English, extend your programs so that you can generate sentences. That is, given any English letter(s) as input and based on the n-gram model you should continue the sentence with the most likely characters. More specifically, given a letter to begin a sentence with, you should choose as the next character that character that yields the highest n-gram count when composed with previous characters into a n-gram. Thus, you will use the previous character for bigrams and previous two characters for trigrams. Your program should continue generating new characters until you have generated a 100 character sentence or you have reached a dead end (all n-gram counts are zero).

Deliverables for part 2

In your report, include:

1. for the linear interpolated model (and optional add-one model), the average perplexity score across all lines in the test document, as well as which language each model estimates the test document is written in (has the lowest perplexity).
2. generated text outputs for the following inputs: bigrams starting with 10 letters of your choice, and trigrams using those 10 letters as the first character with a second meaningful character of your choice. This is for English bigram and trigram models, both unsmoothed and smoothed.
3. critical analysis of your language identification results: e.g., why do your perplexity scores tell you what language the test data is written in? what does a comparison of your unigram, bigram, and trigram scores tell you about which performs best? etc.
4. critical analysis of your generation results: e.g., are there any difference between the sentences generated by bigrams and trigrams, or by the unsmoothed versus smoothed models? Give examples to back up your conclusions.

Submission

Please submit the following items on Canvas:

- Your report with results and answers to questions in Part 1 and Part 2, named `report_{your pitt email id}_hw3.pdf`. No need to include @pitt.edu, just use the email ID before that part. For example: `report_mmy29_hw3.pdf`.
- The code of your program
- Model files (ngram probabilities) ideally in a human-readable format (CSV, JSON, etc)
- A `README.txt` file explaining
  - how to run your code
  - the computing environment you used; what programming language you used and the major and minor version of that language; what packages did you use in case we replicate your experiments (a `requirements.txt` file for setting up the environment may be useful if there are many packages).
  - any additional resources, references, or web pages you've consulted
  - any person with whom you've discussed the assignment and describe the nature of your discussions
  - any generative AI tool used, and how it was used
  - any unresolved issues or problems

This homework assignment is worth 45 points.

Acknowledgments

This assignment is based on a homework assignment by Prof. Diane Litman.