Natural Language Processing (CS60075)

Assignment 1: N-gram Language Model

Deadline: 14th Feb EOD <3

General Guidelines

- This assignment is to be done individually by each student. You should not copy any code from one another, or from any web source. We will use standard plagiarism detection tools on the submissions. Plagiarized codes will be awarded zero for the assignment, and we will not differentiate between who copied from whom.
- In this assignment, you are allowed to use Python libraries like spacy, NLTK, numpy and pandas for performing basic NLP tasks easily. However, **you are NOT allowed to use any feature / library / tool meant specifically for language modeling (or which allows you to build language models using built-in functions).**
- **Follow the submission instructions given at the end. Solutions should be uploaded via the CSE Moodle website (see course website for details).**
- You should organize your code into different functions and add explanatory comments. Make sure your code can be easily read. Codes that cannot be understood will be penalized.

Definition of an n-Gram Model

An **n-gram language model** is a probabilistic model used for predicting the next word in a sequence based on the previous (n−1) words. It relies on the **Markov assumption**, which states that the probability of a word depends only on a fixed number of preceding words, rather than the entire history of the sequence.

A language model assigns a probability to a sequence of words $W = (w_1, w_2, \ldots, w_T)$. Using the **chain rule of probability**, the probability of the full sequence can be decomposed as:

$$P(W) = P(w_1, w_2, \ldots, w_T) = \prod_{i=1}^{T} P(w_i \mid w_1, \ldots, w_{i-1})$$

However, computing $P(w_i|w_1, \ldots, w_{i-1})$ directly is impractical because it requires a vast amount of data to estimate probabilities for all possible word sequences. Instead, we approximate it using the **n-gram assumption**, which simplifies the dependency:

$$P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

where n is a fixed value determining how many preceding words are considered. This assumption leads to the **n-gram model**, where the probability of a sentence is computed as:

$$P(W) \approx \prod_{i=1}^{T} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

Types of n-Gram Models

- **Unigram model ($n = 1$)**: Assumes each word is independent:
  $$P(W) = \prod_{i=1}^{T} P(w_i)$$
- **Bigram model ($n = 2$)**: Each word depends on the previous word:
  $$P(W) \approx \prod_{i=1}^{T} P(w_i|w_{i-1})$$
- **Trigram model ($n = 3$)**: Each word depends on the previous two words:
  $$P(W) \approx \prod_{i=1}^{T} P(w_i|w_{i-2}, w_{i-1})$$
- **Higher-order n-gram models ($n > 3$)**: Consider longer contexts but require more data for reliable estimation.

A. Dataset and Preprocessing

The dataset contains a random selection of wikipedia articles, in two standard CSV files. Each row contains the title of the article along with the text body. For this assignment use only the text field.

- The dataset is already divided into train and test sets
- The 'test.csv' file contains 100 articles.
- The 'train.csv' file contains ~14k articles.

[TASKS]

Download the dataset from this link.

1. Extract 100 articles from the training set and keep them as the validation/development set.
2. Remove punctuations and non-ASCII characters (if any)
3. Stopword removal
4. Stemming/Lemmatization
5. Any other pre-processing steps you feel will improve efficiency and make the data more suitable for this type of language modeling.

B. Estimation Using Maximum Likelihood

To compute $P(w_i|w_{i-(n-1)}, \ldots, w_{i-1})$, we use **Maximum Likelihood Estimation (MLE)**, which calculates probabilities based on relative frequencies:

$$P(w_i|w_{i-(n-1)}, ..., w_{i-1}) = \frac{C(w_{i-(n-1)}, ..., w_i)}{C(w_{i-(n-1)}, ..., w_{i-1})}$$

where:

- $C(w_{i-(n-1)}, ..., w_i)$ is the count of the sequence $(w_{i-(n-1)}, ..., w_i)$ in the training corpus.
- $C(w_{i-(n-1)}, ..., w_{i-1})$ is the count of the preceding $(n-1)$-gram.

Smoothing Techniques

One of the major challenges in n-gram modeling is **data sparsity**. Many valid word sequences may have zero probability because they are not observed in the training data. To address this, **smoothing techniques** are applied.. The simplest approach is **Laplace smoothing**, which adds a constant k (usually 1) to all counts:

$$P_{Lap}(w_i|w_{i-(n-1)}, ..., w_{i-1}) = \frac{C(w_{i-(n-1)}, ..., w_i) + k}{C(w_{i-(n-1)}, ..., w_{i-1}) + kV}$$

where:

- V is the vocabulary size.
- k is a small positive constant (usually $k = 1$).

[TASKS]

Train Unigram, Bigram and Trigram models using MLE

6. When creating the vocabulary of n-grams, select only those n-grams that appear in at least 1% articles of the train set
7. Apply Laplace smoothing to each of the models during MLE, take $k = 1$.

C. Evaluating an n-Gram Model using Perplexity

A key measure of language model performance is **perplexity**, which evaluates how well the model predicts a given test set. It is defined as:

$$PP(W) = \left(\prod_{i=1}^{T} P_{Lap}(w_i | w_{i-(n-1)}, ..., w_{i-1}) \right)^{-\frac{1}{T}}$$

Alternatively, using logarithms:

$$PP(W) = \exp\left(-\frac{1}{T} \sum_{i=1}^{T} \log P_{Lap}(w_i | w_{i-(n-1)}, ..., w_{i-1}) \right)$$

Lower **perplexity** indicates a better model.

[TASKS]

Evaluate the unigram, bigram and trigram models (which you trained above with MLE) on the test set using perplexity

8. Calculate perplexity of each test set article using the exp() formula.
9. Average across all articles in the test set to get overall perplexity for each model

D. Interpolation Model

The interpolation model combines probability estimates from multiple n-gram models, weighted by certain coefficients that are learned or set manually. For example,

$$P_{Int}(w_i | w_{i-2}, w_{i-1}) = \lambda_1 P_{Lap}(w_i | w_{i-2}, w_{i-1}) + \lambda_2 P_{Lap}(w_i | w_{i-1}) + \lambda_3 P_{Lap}(w_i)$$

Here, the model is combining the trigram, bigram, and unigram probabilities, with $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

[TASKS]

10. Train the interpolation model with MLE estimates of the smoothed trigram, bigram and unigram models
11. $\lambda_1, \lambda_2, \lambda_3$ are hyper-parameters, you need to decide on the optimal values based on the given **validation set**. Ensure that they sum to 1.
12. Evaluate the interpolation model on the test set with perplexity, like you did for the individual models

Submission Instructions

Submit one .zip file containing the following. Name the compressed file the same as your roll number. Example: “25CS60R00.zip”'

1. **Codes:** Submit a main.ipynb notebook or a main.py file from where execution will be started. You are free to create other files as per convenience for legibility.
2. Do **NOT** include the **dataset folder**
3. **A report (as a doc or pdf)** containing:
    - Short description of the work
    - Report the time taken (in seconds) for each part of the code to run.
    - Try to improve efficiency and/or performance of the system. For efficiency, both memory and time can be optimized. To this end, explain the different choices you’ve made and the reasons behind them. Marks will be given based on the creativity of the solution.

[Queries, if any, should be addressed to TAs Shounak Paul and Soham Poddar (email ids are given on the course website).]