Roll No.- 2024201043

Report on the Language Model Built:

Tokenizer:

Tokenizing is done using regular expressions. It primarily focuses on:

1. **Building a Vocabulary**:
    - Extracts words from a given corpus of text, filters them based on a minimum frequency threshold, and constructs a vocabulary set.
    - Filters out words that occur less than min_freq times. For this assignment, min_freq is set to 2.
    - Adds the <UNK> (unknown) token to the vocabulary for handling out-of-vocabulary (OOV) words.
2. **Tokenizing Text**:
    - Processes input text to handle special patterns (like URLs, hashtags, mentions, etc.) and splits it into tokenized sentences for further usage.

N-Gram Model:

1. read_corpus Function:

Reads the text from a given file path and returns it as a string.

2. generate_ngrams Function:

Generates N-grams from a list of tokenized sentences and counts their occurrences.

Language Model:

It implements various **N-gram language models** using smoothing techniques to calculate probabilities and perplexities for given input sentences. The main focus is on:

1. **Laplace Smoothing**

2. **Good-Turing Smoothing**
3. **Linear Interpolation**

It supports unigram, trigram, and five-gram models, allowing comparisons of their performance on tokenized corpora.

1. **Laplace Smoothing**

- **Purpose**:
  - Avoids zero probabilities for unseen N-grams by adding 1 to the count of each N-gram.
- **Implementation**:
  - Applies Laplace smoothing by modifying the formula:

$$\text{P(N-gram)} = \frac{count(N-gram)+1}{count\big((N-1)-gram\big)+V}$$

where V is the vocabulary size.

- **Strengths**:
  - Simple to implement and ensures non-zero probabilities.
- **Limitations**:
  - May disproportionately affect rare N-grams.

2. **Good-Turing Smoothing**

- **Purpose**:
  - Redistributes probability mass from observed N-grams to unseen N-grams based on frequency counts.
- **Implementation**:
  - Fits a regression model on the log of frequency counts to smooth the estimates for unseen or rare N-grams.
- **Output**:

- Smoothed probabilities for observed N-grams and a default probability for unseen N-grams.

- **Strengths**:
  - Effective for handling sparsity in the dataset.
- **Limitations**:
  - Computationally expensive due to regression fitting.

3. **Linear Interpolation**

- **Purpose**:
  - Combines probabilities from unigram, trigram, and fivegram models using weighted contributions (lambdas).
- **Implementation**:
  - Calculates interpolated probabilities using:

$$P(sentence) = \lambda_1 . P_{unigram} + \lambda_2 . P_{trigram} + \lambda_3 . P_{fivegram}$$

where, $\lambda_1 + \lambda_2 + \lambda_3 = 1$

- **Strengths**:
  - Balances the strengths of multiple models to produce better results.
- **Limitations**:
  - Requires careful tuning of lambda values.

Perplexity Calculation

- **Purpose**:
  - Measures the model's ability to predict the input sentence by calculating the exponential of the average negative log probability:

$$Perplexity = e^{-\frac{1}{N}\sum \log P}$$

where N is the number of tokens in the input.

- **Strengths**:
  - Quantifies model performance in a mathematically sound manner.
- **Limitations**:
  - Affected by tokenization quality and sentence length.

Generation Model:

1. Input and Initialization

The program is run with three command-line arguments:

- lm_type: Specifies the language model type (l for Laplace, g for Good-Turing, and i for Interpolation).
- corpus_path: Path to the corpus file for training the model.
- k: Number of top predictions to output.

2. Preprocessing

- The corpus is read and tokenized into sentences.
- Vocabulary and n-gram counts (up to 5-grams) are generated.
- Smoothing probabilities are precomputed for Good-Turing and Linear Interpolation methods.

3. Next Word Prediction

The user inputs a partial sentence. Based on the chosen model:

a. Laplace Smoothing

- Computes probabilities for all candidate words by considering their context n-gram.
- Ranks and returns the top words based on probabilities.

b. Good-Turing Smoothing

- Retrieves smoothed probabilities for valid n-grams from the precomputed values.
- Outputs the most probable words.

c. Linear Interpolation

- Combines n-gram probabilities for each candidate word using predefined weights.
- Predicts the words with the lowest perplexity (highest probability).

Analysis of Outputs:

LM1:

1. **Unigram Model**:
   - The unigram model considers each word independently, ignoring the context provided by surrounding words. This lack of context leads to less accurate predictions, as the probability of a word depends only on its overall frequency in the corpus rather than its position or relationship with other words.
   - Since the unigram model does not capture dependencies between words, its perplexity is higher because it cannot represent the structure or syntax of the language effectively.
2. **Trigram Model**:
   - The trigram model accounts for two words of context, significantly improving predictions by considering local dependencies. This richer representation of context allows it to assign probabilities more accurately, resulting in lower perplexity compared to the unigram model.
   - Additionally, the trigram model balances the trade-off between complexity and data sparsity, as most corpora are large enough to provide sufficient trigram coverage.
3. **Fivegram Model**:
   - While the fivegram model captures a wider context (four preceding words), it suffers from **data sparsity**. As the n-gram size increases, the likelihood of encountering unseen or infrequent fivegrams grows, even with Laplace smoothing. This results in higher perplexity compared to the trigram model.

- Moreover, Laplace smoothing adds a uniform probability to all possible fivegrams, which can disproportionately affect rare sequences. This smoothing technique tends to overly flatten probabilities, which can degrade the performance of higher-order n-grams like the fivegram model when the training data is insufficient to reliably estimate probabilities.

LM2:

Observed trend in perplexity-

unigram perplexity > trigram perplexity > fivegram perplexity

1. Unigram Model:

- **Context-Free Limitation**: The unigram model, like in the Laplace smoothing case, does not account for any contextual information, relying solely on the overall frequency of words in the corpus. This results in poor performance because language inherently relies on word dependencies and context to form meaningful sequences.
- **Good-Turing Smoothing Impact**: Good-Turing smoothing adjusts probabilities for unseen and rare events, which improves unigram predictions for rare words. However, the lack of context in the unigram model still fundamentally limits its ability to accurately model the language, leading to higher perplexity compared to trigram and fivegram models.

2. Trigram Model:

- **Balanced Context**: The trigram model incorporates two preceding words as context, allowing it to better capture local word dependencies and predict probabilities more accurately. This contextual richness significantly reduces perplexity compared to the unigram model.
- **Effectiveness of Good-Turing Smoothing**: Good-Turing smoothing is particularly effective here because trigrams are more frequent in most corpora, ensuring that rare and unseen events are smoothed without overly distorting probabilities. This leads to more reliable predictions.

3. Fivegram Model:

- **Higher Contextual Depth**: The fivegram model considers four preceding words, providing even greater context than the trigram model. This allows it to make more informed predictions, which naturally results in lower perplexity.
- **Good-Turing Smoothing Advantage**: Good-Turing smoothing excels in handling unseen or low-frequency fivegrams by redistributing probabilities from frequent events to rare ones. Unlike Laplace smoothing, Good-Turing smoothing avoids assigning overly uniform probabilities, which helps the fivegram model maintain its predictive strength even in sparse data scenarios.
- **Sufficient Training Data**: In your case, the corpus likely contains enough data to support reliable estimation of fivegram probabilities. With proper smoothing, the fivegram model outperforms the trigram model because the additional context allows it to make more precise predictions.

LM3:

In the **linear interpolation model**, where multiple n-gram models (unigram, trigram, and fivegram) are combined using different sets of lambda values, the average perplexity values remain in a similar range across the tested combinations: **[0.4, 0.4, 0.2]**, **[0.3, 0.5, 0.2]**, and **[0.2, 0.4, 0.4]**. Let me try to analyze why:

1. Balanced Contribution Across N-Grams:

- Linear interpolation assigns weights (lambdas) to each n-gram model, combining their probabilities to compute the final prediction. Regardless of the lambda configuration, all three sets place a significant emphasis on the **trigram model**, followed by a lower contribution from the **fivegram model** and a modest reliance on the **unigram model**.
- The trigram model is generally the most reliable n-gram for prediction because it balances context richness and data availability. Thus, the final interpolated probability heavily depends on the trigram component, keeping the perplexity within a consistent range across lambda values.

2. Impact of Lambda Variations:

- While the weights for unigram, trigram, and fivegram models vary slightly between configurations, their overall contributions to the final prediction remain similar due to the smoothing effect of interpolation. For example:
    - In **[0.4, 0.4, 0.2]**, the trigram model (0.4) and unigram model (0.4) dominate, with the fivegram model contributing a smaller weight (0.2).
    - In **[0.3, 0.5, 0.2]**, the trigram model (0.5) plays a slightly more dominant role, but the unigram model (0.3) and fivegram model (0.2) still contribute enough to balance the predictions.
    - In **[0.2, 0.4, 0.4]**, the emphasis shifts slightly toward the fivegram model (0.4), but the trigram model (0.4) still retains significant influence. The unigram model's lower weight (0.2) has minimal impact since it contributes less informative context.

Since the trigram consistently contributes the largest or near-largest weight across all lambda sets, the average perplexity values do not diverge significantly.

3. Smoothing Across N-Grams:

- Linear interpolation inherently smooths predictions by combining probabilities from all n-grams, ensuring that no single model dominates or entirely dictates the final probability. This smoothing effect reduces the sensitivity of perplexity to variations in lambda values.
- Even if a lambda configuration places more weight on a specific n-gram (e.g., the fivegram in **[0.2, 0.4, 0.4]**), the smoothing effect ensures that the overall predictions remain similar across lambda sets, leading to comparable perplexity values.

4. Training Data Characteristics:

- The consistency in perplexity values suggests that the corpus has sufficient data to train all three n-gram models effectively. If the data

were sparse, higher weights on higher-order n-grams (e.g., fivegrams) might have resulted in more variability in perplexity due to unreliable probability estimates for rare n-grams.

- Since this is not observed, the corpus likely has enough coverage for trigram and fivegram models to make meaningful contributions regardless of lambda weights.

5. Insights from the Results:

- **Robustness of Linear Interpolation**: The linear interpolation model performs reliably across different lambda configurations, indicating its robustness in balancing contextual richness (via fivegram and trigram models) and broad coverage (via the unigram model).
- **Dominance of Trigram**: The trigram model consistently plays the central role in prediction, and minor variations in lambda weights for unigram and fivegram models do not significantly impact the overall perplexity.

The linear interpolation model shows consistent average perplexity values across lambda configurations ([0.4, 0.4, 0.2], [0.3, 0.5, 0.2], and [0.2, 0.4, 0.4]) due to the smoothing effect of combining n-gram models. The trigram model remains the dominant contributor, balancing context richness and data reliability. Variations in weights for unigram and fivegram models have minimal impact, as the interpolation ensures robust and balanced predictions. Additionally, the adequacy of the training corpus allows all n-gram models to contribute meaningfully, further reducing variability in perplexity. This highlights the effectiveness of linear interpolation in achieving stable performance regardless of lambda configurations.