Assignment-1

N-Grams Model, Smoothing and Interpolation

Introduction

This report describes the implementation of an N-gram-based language model designed to analyze and predict text sequences. The model includes multiple preprocessing steps, various smoothing techniques, and evaluation metrics, including perplexity calculations.

Implementation Details

1. Corpus Loading and Preprocessing

The model supports text processing using two distinct cleaning functions:

- **clean_and_tokenize_1()**: Processes text from "Pride and Prejudice" by Jane Austen.
- **clean_and_tokenize_2()**: Processes text from "Ulysses" by James Joyce.

These functions remove unwanted sections, normalize text, and tokenize sentences while preserving punctuation and sentence boundaries. Tokenized sentences are augmented with SOS (start-of-sentence) and EOS (end-of-sentence) markers.

2. N-Gram Model Construction

The model builds N-gram representations for different values of N:

- **generate_ngrams()**: Generates contiguous sequences of tokens of length N.
- **build_ngram_model()**: Constructs frequency distributions of N-grams and (N-1)-grams for probability calculations.

3. Smoothing Techniques

To handle unseen N-grams, the model implements various smoothing methods:

- **No Smoothing** (no_smoothing()): Direct probability computation from raw counts.
- **Laplace Smoothing** (laplace_smoothing()): Adds one to all N-gram counts to avoid zero probabilities.
- **Good-Turing Smoothing** (good_turing_smoothing()): Adjusts probabilities based on frequency-of-frequencies. The method discussed in the research paper has been implemented. We use Turing estimate r* until there is significant difference between Turing and LGT estimates.
- **Linear Interpolation** (linear_interpolation()): Computes weighted probabilities across different N-gram orders using a held-out dataset. Lambdas are determined in linear time by deleted interpolation (as discussed in paper).

4. Model Training

The train_language_model() function trains the model based on the specified smoothing technique. A portion of the corpus is reserved as a held-out dataset for parameter estimation in interpolation-based smoothing.

5. Evaluation Metrics

Perplexity Calculation

The performance of the language model is assessed using **perplexity**, a measure of how well the model predicts unseen text.

- Implemented in perplexity() and sentence_perplexity() functions.
- Lower perplexity values indicate better model performance.

Experimental Setup and Execution

1. The script takes command-line arguments specifying:
    - **Smoothing type** (l for Laplace, g for Good-Turing, i for Interpolation, na for No Smoothing).
    - **N-gram order (N)**.
    - **Corpus file path**.
2. The model preprocesses the selected corpus, splits it into training and test sets, and trains the specified N-gram model.
3. Perplexity is computed for both training and test datasets, and the results are saved to output files.
4. The user can enter a sentence interactively, and the model returns its estimated probability.

Results and Comments:

1. Corpus 1: Pride and Prejudice

Method	**Average Perplexity**					
	N=1		**N=3**		**N=5**	
	Train	Test	Train	Test	Train	Test
Laplace Smoothing	406.68	393.48	493.52	3563.89	355.16	5553.17
Good Turing Smoothing	14602.0	13592.2	11131.6	184.88	109209.4	1471.5
Linear Interpolation	409.85	432.8	49.2	444.20	42.75	468.43

2. Corpus 2: Ulysses

Method	**Average Perplexity**					
	N=1		**N=3**		**N=5**	
	Train	Test	Train	Test	Train	Test
Laplace Smoothing	796.10	680.87	2330.6	16746.49	1862.6	24417.3
Good Turing Smoothing	17656.5	13950.2	51386.9	906.2	184264.9	1866.0
Linear Interpolation	898.6	810.7	60.39	1430.7	54.3	1640.5

For Laplace Smoothing, we observe consistent increase in perplexities as N increases. With sparsity, we expect the probabilities to be smaller and therefore, higher perplexity. For the same reason, the train perplexity is smaller than the test.

In case of Good Turing, as N increases, the sparsity increases thereby, frequently using the probability of N1/N for unseen ngrams. This results in higher perplexities for bigger N. For the same reason, the test perplexity is smaller than the train (for a given N).

In case of linear interpolation, we observe lower train perplexity than test.

Discussion:

- Linear Interpolation performs well across different N-gram models for both the corpuses.
- The second corpus has a lot more punctuations and irregularities than the first one. This means more shorter sentences after pre-processing. Pre-processing such corpus may need more attention to detail and advanced methods.

Conclusion

This implementation of an N-gram language model demonstrates the effectiveness of various smoothing techniques in addressing sparsity issues. The model's performance is evaluated using perplexity, providing insights into language modeling strategies for different corpora and training setups.

Text Generation

Given a sequence of words, the top k most probable words are generated with their corresponding probabilities.

Results:

1. Corpus 1: Pride and Prejudice

```
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py na 1 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. the (Probability: 0.03494910268127439)
2. to (Probability: 0.033763122215346206)
3. of (Probability: 0.02932606243022937)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py na 3 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. girls (Probability: 0.9412384815940812)
2. 15th (Probability: 9.41238481594086e-06)
3. 18th (Probability: 9.41238481594086e-06)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py na 5 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. girls (Probability: 0.9412384815940817)
2. 15th (Probability: 9.412384815940761e-06)
3. 18th (Probability: 9.412384815940761e-06)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py l 1 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. the (Probability: 0.03466872970337545)
2. to (Probability: 0.03349226355539644)
3. of (Probability: 0.029090799295476033)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py l 3 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. girls (Probability: 0.0035585763719962974)
2. 15th (Probability: 0.0001596093902976139)
3. 18th (Probability: 0.0001596093902976139)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py l 5 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. girls (Probability: 0.4981914637087048)
2. 15th (Probability: 8.037939072421836e-05)
3. 18th (Probability: 8.037939072421836e-05)
```

```
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py g 1 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. an (Probability: 0.006458603838965957)
2. if (Probability: 0.006170156316287088)
3. am (Probability: 0.0056473455768171815)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py g 3 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier
The model predictions are weak and non-distinctive. Try some other model or smoothing technique.
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py g 5 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier
The model predictions are weak and non-distinctive. Try some other model or smoothing technique.
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py i 1 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. the (Probability: 0.035169714319097906)
2. to (Probability: 0.03397624750665602)
3. of (Probability: 0.02951117936220468)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py i 3 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. girls (Probability: 0.5162000237297331)
2. coloured (Probability: 0.37408018358481326)
3. the (Probability: 0.003860166863225008)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py i 5 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: had never met with pleasanter people or prettier

Top-3 next word predictions:
1. girls (Probability: 0.561298599766872)
2. coloured (Probability: 0.33907831103767394)
3. the (Probability: 0.0035049441701632275)
```

- Good Turing performs poorly with prediction because of distorted probability mass distribution between seen and unseen words.

- Linear Interpolation and No smoothing perform well when N >= 3.
- When N=1, the stopwords dominate the prediction list

2. Corpus 2: Ulysses

   Similar results were observed for this corpus.

3. OOD instances:

```
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py na 3 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: write my name
The model predictions are weak and non-distinctive. Try some other model or smoothing technique.
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py na 5 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: write my name
The model predictions are weak and non-distinctive. Try some other model or smoothing technique.
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py l 3 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: write my name
The model predictions are weak and non-distinctive. Try some other model or smoothing technique.
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py l 1 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: write my name

Top-3 next word predictions:
1. the (Probability: 0.034668729703375235)
2. to (Probability: 0.03349226355539662)
3. of (Probability: 0.029090799295475846)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py g 1 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: write my name

Top-3 next word predictions:
1. an (Probability: 0.006458603838965983)
2. if (Probability: 0.006170156316287074)
3. am (Probability: 0.005647345576817141)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py g 3 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: write my name
The model predictions are weak and non-distinctive. Try some other model or smoothing technique.
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py i 1 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: write my name

Top-3 next word predictions:
1. the (Probability: 0.03516971431909809)
2. to (Probability: 0.033976247506655774)
3. of (Probability: 0.029511179362204654)
PS C:\Users\NAITIK\Desktop\Semester-4\INLP\Assignments\A1> python generator.py i 3 "Pride and Prejudice - Jane Austen.txt" 3
Input sentence: write my name

Top-3 next word predictions:
1. to (Probability: 0.14101337258314448)
2. of (Probability: 0.13973986564241517)
3. at (Probability: 0.08309872976322095)
```

   For random input like “write my name”, all the models perform poorly with higher order N-gram models as they are more context-rich and offer poor generalization. With unigram models, the predictions were stopwords as their counts are expected to be dominate.