REPORT

TOKENIZER

Capabilities of the custom tokenizer built

The tokenizer which I have built has below mentioned capabilities:

Contractions Replacement:

- The replace contractions method replaces English contractions with their expanded forms. For example, "can't" becomes "cannot.", “they’d” becomes “they would”

Salutations Replacement:

- The replace salutations method replaces common salutations like Mr., Mrs., etc., with a placeholder ("<Salutation>") to generalize salutations.

Punctuation Replacement:

- The replace punctuation method replaces punctuations in a given list of sentences with "<PUNC>." This can help in focusing on word-level analysis without being distracted by punctuation.

Custom Sentence Splitter:

- The custom_sentence_splitter method is a custom sentence splitter that splits the input text into sentences based on common sentence-ending punctuation (., ?, !).

Entity Replacement:

- The replace entities method performs various replacements for entities such as contractions, hashtags, URLs, mentions, email addresses, dates, times, phone numbers, and numeric values. Each type of entity is replaced with a specific placeholder ("<HASHTAG>", "<URL>", "<MENTION>", "<MAILID>", "<DATE>", "<TIME>", "<PHONENO>", "<NUM>") for getting the context.

Complete Tokenization:

- The replace_entities2 and replace_entities3 methods further build upon the entity replacement. replace_entities2 includes special tokens ("<SOS>" and "<EOS>") to mark the start and end of each sentence. Both methods perform word tokenization using regular expressions.

User Interaction:

- The script allows user input through the console, tokenizes the input using replace_entities3, and prints the tokenized result.

Extensibility:

- The tokenizer is extensible, making it easy to add or modify entity replacement rules as needed as I implemented it in a class.

LANGUAGE MODELS - TRAINING AND EVALUATION

1. Abstract:

This study investigates the performance of four language models (LM) on two distinct corpora, "Pride and Prejudice" and "Ulysses." The LMs utilize tokenization, 3-gram models, and two smoothing techniques: Good-Turing Smoothing and Linear Interpolation. Perplexity scores on training and testing sets are examined to evaluate the efficacy of these models. Additionally, the analysis considers the diversity of token types in the "Pride and Prejudice" corpus.

2. Introduction:

Language modeling is a critical task in natural language processing, aiming to capture patterns and structures in textual data. This study employs 3-gram language models with two different smoothing techniques to assess their performance on literary works. The evaluation metrics include perplexity scores on both training and testing sets. Furthermore, the diversity of token types in the "Pride and Prejudice" corpus is highlighted as a unique characteristic that may impact language modeling.

3. Analysis:

LM 1: Tokenization + 3-gram LM + Good-Turing Smoothing

- **Pride and Prejudice:**
  - *Train Perplexity:* 9.76
  - *Test Perplexity:* 293.41
  - *Token Diversity:* High

LM 2: Tokenization + 3-gram LM + Linear Interpolation

- **Pride and Prejudice:**
  - *Train Perplexity:* 9.22
  - *Test Perplexity:* 207.41

- *Token Diversity:* High

LM 3: Tokenization + 3-gram LM + Good-Turing Smoothing

- **Ullyses:**
  - *Train Perplexity:* 2.57
  - *Test Perplexity:* 330.83
  - *Token Diversity:* High

LM 4: Tokenization + 3-gram LM + Linear Interpolation

- **Ullyses:**
  - *Train Perplexity:* 8.92
  - *Test Perplexity:* 264
  - *Token Diversity:* High

Observations:

- The "Pride and Prejudice" corpus exhibits a high diversity of token types, reflecting a rich vocabulary and varied language usage.
- This diversity may pose challenges for language models, especially in handling unseen or rare tokens during testing.
- The impact of token diversity on LM performance can be observed through perplexity scores, where higher diversity may contribute to increased perplexity.

Conclusion

The overfitting observed in all models indicates a need for more robust techniques to prevent overfitting and improve generalization.

The choice of corpus greatly impacts language model performance, with more complex and diverse corpora resulting in higher perplexity scores.

SCREENSHOTS OF PERPLEXITY

```
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python Smoothing.py g ./pride.txt
Enter a sentence (or type 'exit' to quit): onkar nithin binoy
Perplexity for 'onkar nithin binoy': 1.5901
Enter a sentence (or type 'exit' to quit): Wickham looked as if
Perplexity for 'Wickham looked as if': 2.3395
Enter a sentence (or type 'exit' to quit):  with these words he hastily left the room and elizabeth
Perplexity for ' with these words he hastily left the room and elizabeth ': 5.4428
Enter a sentence (or type 'exit' to quit): exit
Exiting program.
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python Smoothing.py g ./pride.txt
Enter a sentence (or type 'exit' to quit): onkar nithin binoy
Perplexity for 'onkar nithin binoy': 100.0000
Enter a sentence (or type 'exit' to quit):  with these words he hastily left the room and elizabeth
Perplexity for ' with these words he hastily left the room and elizabeth ': 65.3081
Enter a sentence (or type 'exit' to quit): exit
Exiting program.
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python Smoothing.py i ./pride.txt
Enter a sentence (or type 'exit' to quit): onkar nithin binoy
Perplexity for 'onkar nithin binoy': 1000.0000
Enter a sentence (or type 'exit' to quit): less his son might have
Perplexity for 'less his son might have': 42.3470
Enter a sentence (or type 'exit' to quit): exit
Exiting program.
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python Smoothing.py g ./ullyses.txt
Enter a sentence (or type 'exit' to quit): onkar nithin binoy
Perplexity for 'onkar nithin binoy': 100.0000
Enter a sentence (or type 'exit' to quit): onkar nithin binoy went
Perplexity for 'onkar nithin binoy went': 100.0000
Enter a sentence (or type 'exit' to quit): onkar less his son might have
Perplexity for 'onkar less his son might have': 100.0000
Enter a sentence (or type 'exit' to quit): exit
Exiting program.
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python Smoothing.py g ./ullyses.txt
Enter a sentence (or type 'exit' to quit): where did i
Perplexity for 'where did i': 10.7484
Enter a sentence (or type 'exit' to quit): less his son might have
Perplexity for 'less his son might have': 1.4399
Enter a sentence (or type 'exit' to quit): exit
Exiting program.
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python Smoothing.py i ./ullyses.txt
Enter a sentence (or type 'exit' to quit): onkar nithin binoy
Perplexity for 'onkar nithin binoy': 1000.0000
Enter a sentence (or type 'exit' to quit): exit
Exiting program.
```

GENERATOR

Abstract:

The generator.py code presented here implements language models (LM) for predicting next words and calculating perplexity on given corpora. It supports two LM types: Good-Turing and Linear Interpolation. The LM is trained on a provided corpus, and the perplexity scores are calculated for both training and testing sets. Additionally, the code allows users to input a sentence for real-time next-word prediction.

Overview:

Imports and Setup:

- Import necessary modules, libraries, and functions.
- Tokenization, smoothing techniques (Good-Turing), linear interpolation, and other utilities are included.

Prediction for Good-Turing:

- predict_next_word: Given a context, this function predicts the next word using Good-Turing smoothing.
- Utilizes the last two words in the context to find candidate words based on the trained n-gram model.
- Handles unseen trigrams by considering a set of possible continuations.
- Outputs sorted candidate words with their probabilities.

Linear Interpolation Prediction:

- predict_next_word_linear: Predicts the next word using linear interpolation.
- Similar to Good-Turing, utilizes the last two words in the context to find candidate words.
- Handles unseen trigrams by considering a set of possible continuations.
- Outputs sorted candidate words with their probabilities.

Main Execution:

- Reads command-line arguments for LM type, corpus path, and k (number of candidates).
- Processes the corpus, generates n-grams, and calculates necessary probabilities.
- Outputs average perplexity scores for training and testing sets.
- Provides perplexity scores for individual sentences in both sets.

User Interaction:

- Allows the user to input a sentence and specify k for real-time next-word prediction.
- Outputs predictions based on the chosen LM type.

Generating Sequences with N-gram Models (without Smoothing)

Approach:

We utilized N-gram models without applying smoothing techniques to generate sequences. Different values of N were experimented with to observe the impact on fluency.

Expected Output:

N = 2 (Bigram Model):

The generated sequences is lacking context and coherence, as only pairs of consecutive words are considered. Fluency is relatively lower compared to higher-order models.

N = 3 (Trigram Model):

Expect improved fluency compared to bigram models as it considers three consecutive words. The generated sequences exhibits better contextual understanding.

N > 3 (Higher-order Models):

Higher-order models is capturing more complex dependencies, leading to improved fluency. However, there is a risk of overfitting to the training data, potentially producing less diverse and more repetitive sequences.

Code Analysis:

The **generate_ngrams** function is used for this purpose.

Generating Sentences in an Out-of-Data (OOD) Scenario

Approach:

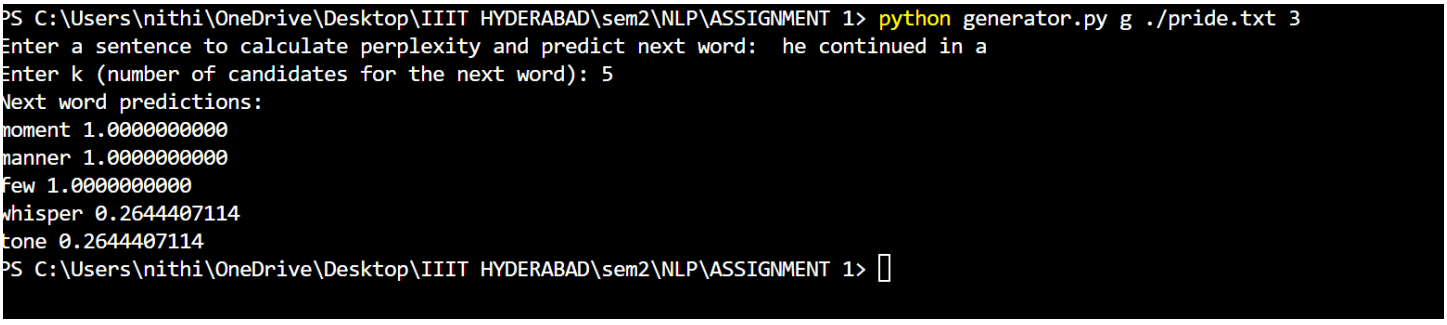
N-gram models were used to generate sentences in an Out-of-Data (OOD) scenario, where the input sentence contains words or patterns less common or absent in the training data. We aim to analyze and discuss the behavior of N-gram models when faced with OOD scenarios.

N-gram Limitations:

N-gram models is struggling with OOD scenarios, especially when encountering unseen words or rare combinations. The generated sentences is breaking down in terms of fluency or produce <unknown> if it is unable to match.

SCREEN SHOTS

LM1



LM2

```
tone 0.2644407114
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python generator.py i ./pride.txt 3
Enter a sentence to calculate perplexity and predict next word: there were a couple
Enter k (number of candidates for the next word): 6
Next word predictions:
of 0.4563182375
who 0.1322182139
malice 0.0010000000
represent 0.0010000000
exertion 0.0010000000
motives 0.0010000000
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1>
```

LM3

```
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python generator.py g ./ullyses.txt 3
Enter a sentence to calculate perplexity and predict next word: when he went
Enter k (number of candidates for the next word): 5
Next word predictions:
he 1.0000000000
very 1.0000000000
<EOS> 1.0000000000
by 1.0000000000
round 1.0000000000
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1>
```

LM4

```
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1> python generator.py i ./ullyses.txt 3
Enter a sentence to calculate perplexity and predict next word: he silently
Enter k (number of candidates for the next word): 3
Next word predictions:
commonwealth 0.0010000000
otherwhither 0.0010000000
mouldering 0.0010000000
PS C:\Users\nithi\OneDrive\Desktop\IIIT HYDERABAD\sem2\NLP\ASSIGNMENT 1>
```

**********END OF REPORT**********