Assignment 1

https://github.com/gadarsh043/NLP_N-gramsmodel.git

Group06

Adarsh Gella	Akhila Susarla	Vijaya Sai Latha Pulipati	Abhiram Reddy Madapa
AXG240019	AXS240035	VXP230093	AXM240036

OBJECTIVE:

In this assignment, we focused on developing an n-gram-based language model using hotel reviews. We created both unigram and bigram models, experimenting with different smoothing techniques and methods for handling unknown words. Our approach involved exploring various design strategies to achieve the desired outcomes. We followed the standard procedures for model creation and testing that are typical in Natural Language Processing.

```python
if __name__ == "__main__":
    file_path = 'train.txt'
    test_path = 'validation.txt'

    train_data = load_corpus(file_path)
    test_data = load_corpus(test_path)

    train_corpus = preprocess(train_data)
    test_corpus = preprocess(test_data)

    train_corpus_flat = list(itertools.chain(*train_corpus))
    test_corpus_flat = list(itertools.chain(*test_corpus))

    train_data_split, validation_data = train_test_split(train_data, train_size=0.9, random_state=192)
    val_corpus = preprocess(validation_data)
    val_corpus_flat = list(itertools.chain(*val_corpus))

    print("=== N-gram Language Model Evaluation ===")
    print("Testing different smoothing values and unknown word thresholds\n")

    for min_freq in [1, 2, 3]:
        print(f"--- Unknown Word Threshold: {min_freq} ---")

        train_processed, unknown_token = handle_unknown_words(train_corpus, min_freq)
        val_processed, _ = handle_unknown_words(val_corpus, min_freq)
        test_processed, _ = handle_unknown_words(test_corpus, min_freq)

        train_flat = list(itertools.chain(*train_processed))
        val_flat = list(itertools.chain(*val_processed))
        test_flat = list(itertools.chain(*test_processed))

        for k in [0, 0.5, 1.0]:
            print(f"\nSmoothing k = {k}:")

            unigram_model = UnigramModel(train_flat, k)
            bigram_model = BigramModel(train_flat, k)

            unigram_perplexity_val = unigram_model.unigram_perplexity(val_flat)
            bigram_perplexity_val = bigram_model.bigram_perplexity(val_flat)

            unigram_perplexity_test = unigram_model.unigram_perplexity(test_flat)
            bigram_perplexity_test = bigram_model.bigram_perplexity(test_flat)

            print(f"  Validation Set - Unigram: {unigram_perplexity_val:.4f}, Bigram: {bigram_perplexity_val:.4f}")
            print(f"  Test Set - Unigram: {unigram_perplexity_test:.4f}, Bigram: {bigram_perplexity_test:.4f}")

        print()
```

```python
class UnigramModel:
    def __init__(self, corpus, k=0):
        self.corpus = corpus
        self.word_counts = defaultdict(int)
        self.total_words = 0
        self.vocab = set()
        self.unknown_token = "<UNK>"
        self.add_k = k
        self.vocab_size = 0
        self.build_unigram_model()

    def build_unigram_model(self):
        for word in self.corpus:
            if word not in self.vocab:
                self.vocab.add(word)
            self.word_counts[word] += 1
            self.total_words += 1
        self.vocab_size = len(self.vocab)

    def get_count(self, word):
        return self.word_counts.get(word, 0)

    def get_probability(self, word):
        count = self.get_count(word)
        prob = (count + self.add_k) / (self.total_words + self.add_k * self.vocab_size)
        return max(prob, 1e-10)

    def unigram_perplexity(self, test_corpus):
        log_prob_sum = 0
        test_words = len(test_corpus)

        for word in test_corpus:
            prob = self.get_probability(word)
            log_prob_sum += math.log(prob)

        perplexity = math.exp(-log_prob_sum / test_words)
        return perplexity
```

1. Implementation Details

1.1 Unigram and bigram probability computation

We implemented both unigram and bigram models. For the unigram model, the probability of each word is independent of the previous word, whereas the bigram model calculates the probability of a word given the preceding word.

```
Unigram probability: P (word) = ntrain (word) / Ntrain
```

```
Bigram probability: P (word | previous word) = ntrain
(previous word, word) / ntrain (previous word)
```

Before training the unigram and bigram models, we preprocess the data by converting all words to lowercase, removing punctuation using regular expressions, and applying lemmatization to convert words to their root forms using the NLTK WordNetLemmatizer. The tokenization was already done as the data was space-separated.

1.2 Smoothing

We implemented the Add-k smoothing technique. Smoothing prevents zero probabilities for unseen words by modifying word counts during probability computation.

```
Add-k Smoothing: PAdd-k (word|previous word) = C(previous
word, word) + k / C(previous word) + kV
```

1.3 Smoothing

Unknown words are handled by replacing words that appear less than a specified frequency threshold with a special token, <UNK>. We experimented with different thresholds (1, 2, 3) where words appearing less than the threshold number of times in the training corpus are replaced with <UNK>. This helps improve model generalization on unseen data and prevents the model from being confused by rare words.

1.4 Implementation of perplexity

Perplexity is used to evaluate the model's performance, quantifying the uncertainty in predictions. The formula for perplexity is:

```
PP = exp(-1/N * Σ log P (wi |wi-1 , . . . , wi-n+1))
```

2. Eval, Analysis and Findings

In this section, we present the evaluation of the model using N-gram-based methods, showing the results for different unknown word thresholds and smoothing values on both Validation and Test sets.

Unknown Word Threshold:
1 (No unknown words) Validation Set - Unigram: 58.13, Bigram: 1.77 Test Set - Unigram: 32417.82, Bigram: 259289385.13

Unknown Word Threshold:
2 Validation Set - Unigram: 1.93, Bigram: 2.55 Test Set - Unigram: 3.03, Bigram: 2.73

Unknown Word Threshold:
3 Validation Set - Unigram: 1.50, Bigram: 1.89 Test Set - Unigram: 2.24, Bigram: 2.13

Table 1:
Evaluation results showing perplexity scores for different unknown word thresholds and smoothing values.

Our results show significant improvements when implementing unknown word handling. Without unknown word handling (threshold 1), the perplexity scores are extremely high on the test set, especially for bigram models, because the models encounter many unseen words and bigrams. When we implemented unknown word handling with thresholds 2 and 3, we observed dramatic improvements in perplexity scores. The best performance was achieved with unknown word threshold 3 and no smoothing (k=0), achieving perplexity scores of 1.50 and 2.24 for unigram and bigram models respectively on the test set. This demonstrates that proper unknown word handling is crucial for model generalization and that bigram models generally perform better than unigram models when properly implemented.

3. Others

3.1 Libraries Used

• from nltk.stem import WordNetLemmatizer - to
• from sklearn.model_selection import train_test_split - to split the training data into training and validation sets.
• import re - to remove punctuations from the text.
• import math - to use logarithm and power functions.

• import itertools - to flatten the tokens generated during processing.
• from collections import defaultdict, Counter - for efficient counting and data structures.

3.2 Individual Contributions

• Adarsh Gella: Code implementation, unknown word handling, and model evaluation.

• Akhila Susarla: Preprocessing, smoothing techniques, and report writing.

• Vijaya Sai Latha Pulipati: Bigram model implementation.

• Abhiram Reddy Madapa: Perplexity Calculation, Error analysis, testing, and report writing.

3.3 FeedBack

We found the project to be medium level. Initially we found it easy to implement but we have seen the perplexity scores were too high without proper unknown word handling. Later we invested significant time to enhance the model by concentrating more on preprocessing steps and unknown word handling techniques. Though the smoothing techniques are straight from the textbook we took our time to learn the math behind it and implement. We found the project to be interesting because it challenged our understanding of language modeling and the importance of proper data preprocessing.

3.4 References

1. https://web.cs.hacettepe.edu.tr/~ilyas/Courses/CMP711/lec03-LanguageModels.pdf
2. https://jofrhwld.github.io/teaching/courses/2022_lin517/lectures/ngram/02_smoothing.html
3. NLTK Documentation https://www.nltk.org
4. Jurafsky & Martin - Speech and Language Processing, Chapter 3: N-gram Language Models