Lab 8.4: N-Gram Language Model Implementation and Evaluation

Assignment Number: 8.4 / 24
Objective:

- Build Unigram, Bigram, Trigram models
- Apply Laplace smoothing
- Calculate sentence probabilities
- Evaluate models using Perplexity

```
# Text cleaning
import re

# NLP tools
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

# Counting N-grams
from collections import Counter

# Math operations
import math

# Data handling
import pandas as pd
import numpy as np
```

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('gutenberg')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
True
```

```
from nltk.corpus import gutenberg

# Load dataset
corpus = gutenberg.raw('austen-emma.txt')

# Show sample text
print(corpus[:700])
```

```
[Emma by Jane Austen 1816]

VOLUME I

CHAPTER I


Emma Woodhouse, handsome, clever, and rich, with a comfortable home
and happy disposition, seemed to unite some of the best blessings
of existence; and had lived nearly twenty-one years in the world
with very little to distress or vex her.

She was the youngest of the two daughters of a most affectionate,
indulgent father; and had, in consequence of her sister's marriage,
been mistress of his house from a very early period.  Her mother
had died too long ago for her to have more than an indistinct
remembrance of her caresses; and her place had been supplied
by an excellent woman as governess, who had fallen little short
of a mother in affectio
```

Dataset Description

The dataset is taken from the NLTK Gutenberg corpus. It contains natural English text with rich vocabulary. The corpus has more than 1500 words. It includes complete sentences and realistic grammar. The dataset is split into training and testing sets.

```
nltk.download('punkt_tab')
sentences = sent_tokenize(corpus)

split_index = int(0.8 * len(sentences))
train_sentences = sentences[:split_index]
test_sentences = sentences[split_index:]

print("Training sentences:", len(train_sentences))
print("Testing sentences:", len(test_sentences))
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Training sentences: 5994
Testing sentences: 1499
```

```
stop_words = set(stopwords.words('english'))

def preprocess(sentences):
    processed = []
    for sent in sentences:
        sent = sent.lower()                      # lowercase
        sent = re.sub(r'[^a-z\s]', '', sent)    # remove punctuation & numbers
        words = word_tokenize(sent)              # tokenize
        words = [w for w in words if w not in stop_words]  # remove stopwords
        if len(words) > 0:
            processed.append(['<s>'] + words + ['</s>'])  # add tokens
    return processed
```

```
train_data = preprocess(train_sentences)
test_data = preprocess(test_sentences)

print(train_data[0])
```

```
['<s>', 'emma', 'jane', 'austen', 'volume', 'chapter', 'emma', 'woodhouse', 'handsome', 'clever', 'rich', 'comfortable', 'ho
```

```
def build_ngrams(data, n):
    ngrams = []
    for sentence in data:
        for i in range(len(sentence) - n + 1):
            ngrams.append(tuple(sentence[i:i+n]))
    return ngrams
```

```
unigrams = build_ngrams(train_data, 1)
bigrams = build_ngrams(train_data, 2)
trigrams = build_ngrams(train_data, 3)

uni_counts = Counter(unigrams)
bi_counts = Counter(bigrams)
tri_counts = Counter(trigrams)
```

```
pd.DataFrame(uni_counts.most_common(10), columns=["Unigram", "Count"])
```

	Unigram	Count
0	(<s>,)	5936
1	(</s>,)	5936
2	(mr,)	911
3	(could,)	617
4	(would,)	609
5	(emma,)	556
6	(mrs,)	553
7	(miss,)	477
8	(must,)	423
9	(said,)	379

```
pd.DataFrame(bi_counts.most_common(10), columns=["Bigram", "Count"])
```

	Bigram	Count
0	(<s>, mr)	204
1	(mrs, weston)	179
2	(mr, knightley)	176
3	(<s>, emma)	166
4	(mr, elton)	159
5	(oh, </s>)	120
6	(miss, woodhouse)	115
7	(mr, weston)	111
8	(<s>, oh)	110
9	(<s>, mrs)	102

```
pd.DataFrame(tri_counts.most_common(10), columns=["Trigram", "Count"])
```

	Trigram	Count
0	(<s>, oh, </s>)	96
1	(<s>, mr, knightley)	44
2	(<s>, mrs, weston)	44
3	(<s>, mr, elton)	38
4	(<s>, mr, weston)	37
5	(mr, frank, churchill)	30
6	(<s>, miss, woodhouse)	26
7	(mr, john, knightley)	25
8	(<s>, ah, </s>)	23
9	(mr, knightley, </s>)	23

```
vocab = set([w[0] for w in uni_counts])
V = len(vocab)

print("Vocabulary size:", V)
```

```
Vocabulary size: 7810
```

```
def unigram_prob(word):
    return (uni_counts.get((word,), 0) + 1) / (sum(uni_counts.values()) + V)

def bigram_prob(bigram):
    return (bi_counts.get(bigram, 0) + 1) / (uni_counts.get((bigram[0],), 0) + V)

def trigram_prob(trigram):
    return (tri_counts.get(trigram, 0) + 1) / (bi_counts.get(trigram[:2], 0) + V)
```

Smoothing is required to handle unseen words or sequences. Without smoothing, unseen n-grams get zero probability. This makes the entire sentence probability zero. Laplace smoothing assigns a small probability to unseen events.

```
test_sentences_samples = [
    "emma was happy with her friend",
    "the house was full of people",
    "she loved her family",
    "he spoke with confidence",
    "the weather was pleasant"
]
```

```
def sentence_probability(sentence, model="unigram"):
    words = ['<s>'] + sentence.lower().split() + ['</s>']
    prob = 1

    if model == "unigram":
        for w in words:
```

```
            prob *= unigram_prob(w)

    elif model == "bigram":
        for i in range(len(words)-1):
            prob *= bigram_prob((words[i], words[i+1]))

    elif model == "trigram":
        for i in range(len(words)-2):
            prob *= trigram_prob((words[i], words[i+1], words[i+2]))

    return prob
```

```
for s in test_sentences_samples:
    print(s)
    print("Unigram:", sentence_probability(s, "unigram"))
    print("Bigram :", sentence_probability(s, "bigram"))
    print("Trigram:", sentence_probability(s, "trigram"))
    print()
```

```
emma was happy with her friend
Unigram: 2.0138243082998414e-25
Bigram : 1.3133121394522754e-24
Trigram: 4.3147902163212954e-24

the house was full of people
Unigram: 8.662056048116456e-27
Bigram : 2.499665672519294e-27
Trigram: 4.406500226040801e-24

she loved her family
Unigram: 1.823005168353568e-19
Bigram : 9.682273325727428e-20
Trigram: 2.687793284376073e-16

he spoke with confidence
Unigram: 4.649398262030773e-20
Bigram : 3.894683372815798e-20
Trigram: 2.687793284376073e-16

the weather was pleasant
Unigram: 2.0619070553353861e-19
Bigram : 5.817465205291904e-20
Trigram: 2.687793284376073e-16
```

```
def perplexity(sentence, model="unigram"):
    words = ['<s>'] + sentence.lower().split() + ['</s>']
    N = len(words)
    log_prob = 0

    if model == "unigram":
        for w in words:
            log_prob += math.log(unigram_prob(w))

    elif model == "bigram":
        for i in range(len(words)-1):
            log_prob += math.log(bigram_prob((words[i], words[i+1])))

    elif model == "trigram":
        for i in range(len(words)-2):
            log_prob += math.log(trigram_prob((words[i], words[i+1], words[i+2])))

    return math.exp(-log_prob / N)
```

```
for s in test_sentences_samples:
    print(s)
    print("Unigram Perplexity:", perplexity(s,"unigram"))
    print("Bigram Perplexity :", perplexity(s,"bigram"))
    print("Trigram Perplexity:", perplexity(s,"trigram"))
    print()
```

```
emma was happy with her friend
Unigram Perplexity: 1221.7920717408285
Bigram Perplexity : 966.5047776183986
Trigram Perplexity: 832.9713024443595

the house was full of people
Unigram Perplexity: 1810.4951939996508
Bigram Perplexity : 2114.7778803676674
```

```
Trigram Perplexity: 830.7842898230112

she loved her family
Unigram Perplexity: 1328.012052959366
Bigram Perplexity : 1475.71936712763
Trigram Perplexity: 393.64132887513165

he spoke with confidence
Unigram Perplexity: 1667.6332352185163
Bigram Perplexity : 1717.5971359477112
Trigram Perplexity: 393.64132887513165

the weather was pleasant
Unigram Perplexity: 1301.033519881529
Bigram Perplexity : 1606.4887956146968
Trigram Perplexity: 393.64132887513165
```

Comparison and Analysis

The trigram model produced the lowest perplexity in most cases. This shows that trigrams capture better context. However, trigrams sometimes suffer from data sparsity. Unigram models performed poorly due to lack of context. Bigram models provided balanced performance. Unseen words reduced probability values significantly. Laplace smoothing prevented zero probabilities. Overall, trigram and bigram models performed better than unigram.