Lab 8.4: N-Gram Language Model and Perplexity

2403A52006

Md.Mustafa

Import Required Libraries

```python
import nltk
import math
import string
from collections import Counter, defaultdict

nltk.download('punkt')
nltk.download('punkt_tab') # Added to download the missing resource
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.

True
```

Dataset Creation

```python
corpus = """
Artificial intelligence is transforming modern technology.
Machine learning helps computers learn from data.
Natural language processing allows machines to understand human
language.
Education plays a vital role in personal and social development.
Healthcare systems depend on accurate medical data.
Technology improves communication across the world.
Students learn programming to solve real world problems.
Data science combines statistics and computing.
Health awareness improves quality of life.
Research and innovation drive economic growth.
""" * 150   # repeat to exceed 1500 words

corpus = corpus.lower().translate(str.maketrans('', '',
string.punctuation))
tokens = nltk.word_tokenize(corpus)

print("Total words in corpus:", len(tokens))
print("Sample tokens:", tokens[:20])
```

```
Total words in corpus: 10650
Sample tokens: ['artificial', 'intelligence', 'is', 'transforming',
'modern', 'technology', 'machine', 'learning', 'helps', 'computers',
'learn', 'from', 'data', 'natural', 'language', 'processing',
'allows', 'machines', 'to', 'understand']
```

Train–Test Split

```
split_index = int(0.8 * len(tokens))
train_tokens = tokens[:split_index]
test_tokens = tokens[split_index:]

print("Training tokens:", len(train_tokens))
print("Testing tokens:", len(test_tokens))

Training tokens: 8520
Testing tokens: 2130
```

Build Unigram, Bigram, Trigram Models

```
# Unigrams
unigram_counts = Counter(train_tokens)

# Bigrams
bigrams = list(nltk.ngrams(train_tokens, 2))
bigram_counts = Counter(bigrams)

# Trigrams
trigrams = list(nltk.ngrams(train_tokens, 3))
trigram_counts = Counter(trigrams)

vocab_size = len(unigram_counts)

print("Vocabulary size:", vocab_size)

Vocabulary size: 61
```

Add-One (Laplace) Smoothing :

Sentence Probability Calculation

```
def unigram_prob(sentence):
    prob = 1
    words = nltk.word_tokenize(sentence.lower())
    for word in words:
        prob *= (unigram_counts[word] + 1) / (len(train_tokens) +
vocab_size)
    return prob


def bigram_prob(sentence):
    words = nltk.word_tokenize(sentence.lower())
    prob = 1
    for i in range(len(words) - 1):
        prob *= (bigram_counts[(words[i], words[i+1])] + 1) / \
                (unigram_counts[words[i]] + vocab_size)
    return prob


def trigram_prob(sentence):
    words = nltk.word_tokenize(sentence.lower())
    prob = 1
    for i in range(len(words) - 2):
        prob *= (trigram_counts[(words[i], words[i+1], words[i+2])] +
1) / \
                (bigram_counts[(words[i], words[i+1])] + vocab_size)
    return prob
```

Test Sentences & Probabilities

```
test_sentences = [
    "artificial intelligence improves technology",
    "students learn programming",
    "healthcare systems use data",
    "machine learning processes data",
    "education improves society"
]

for sent in test_sentences:
    print(f"\nSentence: {sent}")
    print("Unigram Probability :", unigram_prob(sent))
    print("Bigram Probability  :", bigram_prob(sent))
    print("Trigram Probability :", trigram_prob(sent))
```

```
Sentence: artificial intelligence improves technology
Unigram Probability : 1.568386942290891e-07
Bigram Probability  : 1.2270484890013356e-05
Trigram Probability : 9.05715062041481 8e-05

Sentence: students learn programming
Unigram Probability : 5.584368610704621e-06
Bigram Probability  : 0.2687358895761825
Trigram Probability : 0.668508287292817 6

Sentence: healthcare systems use data
Unigram Probability : 9.748242732856041e-10
Bigram Probability  : 6.05478024900659e-05
Trigram Probability : 9.057150620414818e-05

Sentence: machine learning processes data
Unigram Probability : 9.748242732856041e-10
```

```
Bigram Probability  : 6.05478024900659e-05
Trigram Probability : 9.057150620414818e-05

Sentence: education improves society
Unigram Probability : 4.615180670003819e-08
Bigram Probability  : 1.835502285200345e-05
Trigram Probability : 0.01639344262295082
```

Perplexity Calculation

```
def perplexity(prob, sentence):
    N = len(nltk.word_tokenize(sentence))
    return pow(1/prob, 1/N)

print("Perplexity Results:\n")

for sent in test_sentences:
    print(f"Sentence: {sent}")
    print("Unigram Perplexity :", perplexity(unigram_prob(sent),
sent))
    print("Bigram Perplexity  :", perplexity(bigram_prob(sent), sent))
    print("Trigram Perplexity :", perplexity(trigram_prob(sent),
sent))
    print("-" * 40)
```

```
Perplexity Results:

Sentence: artificial intelligence improves technology
Unigram Perplexity : 50.25007282859572
Bigram Perplexity  : 16.896025984329306
Trigram Perplexity : 10.250666458988285
----------------------------------------
Sentence: students learn programming
Unigram Perplexity : 56.36488703051952
Bigram Perplexity  : 1.5496184431851456
Trigram Perplexity : 1.1436621148993464
----------------------------------------
Sentence: healthcare systems use data
Unigram Perplexity : 178.96512839484458
Bigram Perplexity  : 11.336406322267871
Trigram Perplexity : 10.250666458988285
----------------------------------------
Sentence: machine learning processes data
Unigram Perplexity : 178.96512839484458
Bigram Perplexity  : 11.336406322267871
Trigram Perplexity : 10.250666458988285
----------------------------------------
Sentence: education improves society
Unigram Perplexity : 278.78565998178414
Bigram Perplexity  : 37.90952622185059
```

```
Trigram Perplexity : 3.936497183102173
----------------------------------------
```

Conclusion

This experiment demonstrated how N-gram language models work and how perplexity is used to evaluate them. Unigram models are simple but weak, Bigram models improve prediction using local context, and Trigram models capture richer structure. Perplexity effectively measures model quality, with lower values indicating better language understanding.