Ex1a: Text Generation using N-gram Language model

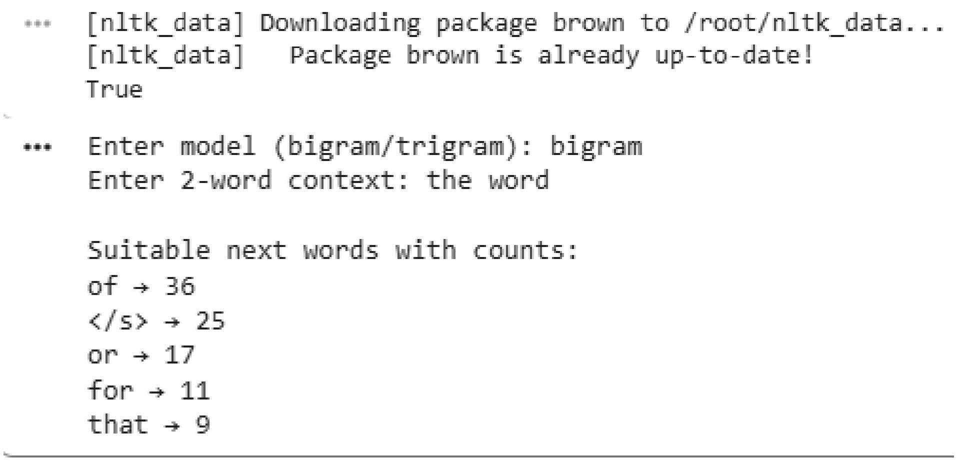
Learning Objective:
Implement an N-Gram–based predictive text model using bigram and trigram probability distributions. This helps to understand how contextual probability influences next-word prediction.

Steps:

1. Initialize the environment by installing and importing the Natural Language Toolkit (nltk). Download the **Brown Corpus** (for text data) and the **Universal Tagset** (for simplified Part-of-Speech tags).
2. Extract raw sentences from the Brown Corpus using brown.sents().
3. **Tokenization:** Convert all words to lowercase and flatten the nested list into a single list of tokens to verify the total volume of training data.
4. Build an N-Gram Model for Bigram and Trigram.
5. Predict Next Word and display Top-5 Suggestions
6. Test Bigram & Trigram Prediction on certain words.

Program:
```
import nltk
from nltk.corpus import brown, stopwords
from nltk.util import ngrams
from collections import defaultdict, Counter
import string
nltk.download('brown')
sentences = brown.sents()

processed = []
for sent in sentences:
    sent = ['<s>'] + [w.lower() for w in sent if w.isalpha()] + ['</s>']
    processed.append(sent)

bigram_counts = defaultdict(Counter)
trigram_counts = defaultdict(Counter)

for sent in processed:
    for w1, w2 in ngrams(sent, 2):
        bigram_counts[w1][w2] += 1

    for w1, w2, w3 in ngrams(sent, 3):
        trigram_counts[(w1, w2)][w3] += 1

def predict_bigram_candidates(context, top_k=5):
    words = context.lower().split()

    if len(words) != 2:
        return "Bigram input must contain exactly 2 words"
    last_word = words[1]

    if last_word not in bigram_counts:
        return "No prediction found"
    return bigram_counts[last_word].most_common(top_k)
def predict_trigram_candidates(context, top_k=5):
    words = context.lower().split()

    if len(words) != 3:
        return "Trigram input must contain exactly 3 words"

    key = (words[1], words[2])

    if key not in trigram_counts:
        return "No prediction found"

    return trigram_counts[key].most_common(top_k)

choice = input("Enter model (bigram/trigram): ").strip().lower()

if choice == "bigram":
    text = input("Enter 2-word context: ")
    results = predict_bigram_candidates(text)

elif choice == "trigram":
    text = input("Enter 3-word context: ")
    results = predict_trigram_candidates(text)

else:
    print("Invalid choice")
    results = []

print("\nSuitable next words with counts:")
if isinstance(results, str):
    print(results)
else:
    for word, count in results:
        print(f"{word} → {count}"
```

Output:



```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
True

Enter model (bigram/trigram): bigram
Enter 2-word context: the word

Suitable next words with counts:
of → 36
</s> → 25
or → 17
for → 11
that → 9
```

RESULT:

Thus the implementation of predictive text system using n-gram has been executed successfully.