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 collections import Counter, defaultdict
import gensim.downloader as api

!pip install gensim
nltk.download('punkt')

# Load Wikipedia dataset
wiki_data = api.load("text8")

# Preprocess words
tokens = [w.lower() for sent in wiki_data for w in sent if w.isalpha()]
print("Total words:", len(tokens))

# Build trigram model
N = 3
model = defaultdict(Counter)
for i in range(len(tokens)-N+1):
    context = tuple(tokens[i:i+N-1])
    target = tokens[i+N-1]
    model[context][target] += 1

# Predict next word and show all possible
def predict_next(context):
    context = tuple(context.lower().split())
    if context in model:
        most_common = model[context].most_common(1)[0][0]
        return most_common, dict(model[context])
```

```
    else:
        return "No prediction", {}

# Test
context_text = "the united"
prediction, possible = predict_next(context_text)
print("\nContext:", context_text)
print("Predicted Next Word:", prediction)

print("\nAll possible next words with counts:")
for w,c in possible.items():
    print(f"{w} : {c}")

print("\nTop 5 next words:")
for w,c in Counter(possible).most_common(5):
    print(f"{w} : {c}")
```

Output:

```
Total words: 17005207

Context: the united
Predicted Next Word: states

All possible next words with counts:
kingdom : 1967
states : 7910
nations : 634
arab : 56
locomotive : 1
daughters : 1
farmers : 2
provinces : 39
hun : 1
state : 4
front : 6
sates : 1
presbyterian : 1
methodist : 28
```

Learning Outcome:
Upon completion of this experiment, the bigram and trigram probability distributions for next-word prediction was implemented and evaluated using the Brown Corpus.

Ex1b: Hidden Markov Model (HMM) based Predictive Text System

Learning Objective:
To implement a predictive text system using Hidden Markov Model (HMM), enabling students to understand contextual probability, POS transitions, and next-word prediction using the Brown Corpus.

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. Load the tagged version of the corpus (brown.tagged_sents) which provides the "Hidden States" (POS tags) linked to the "Observations" (words).
5. Train the HMM Model. Use the HiddenMarkovModelTrainer to perform Supervised Learning. Calculate the internal parameters of the HMM: Initial State Probabilities, Transition Probabilities, Emission Probabilities
6. Build POS Transition Probabilities
7. Build Emission Probabilities
8. Define Prediction Function
9. Test the Model
10. Verify POS prediction on certain words.

Program:

```
import nltk
import numpy as np
from collections import Counter
import re
from hmmlearn import hmm
```

```
nltk.download('brown')
nltk.download('punkt')
from nltk.corpus import brown

print("Loading Brown Corpus...")
sentences = brown.sents()

VOCAB_SIZE = 2000
UNK_TOKEN = "<UNK>"

print("Building Vocabulary...")

all_words = [word.lower() for sent in sentences for word in sent if re.match(r'^[a-z]+$',
word)]


word_counts = Counter(all_words)
most_common = word_counts.most_common(VOCAB_SIZE - 1) # -1 to save room for
UNK
vocab = {word for word, count in most_common}
vocab.add(UNK_TOKEN)


word_to_id = {word: i for i, word in enumerate(sorted(vocab))}
id_to_word = {i: word for word, i in word_to_id.items()}
unk_id = word_to_id[UNK_TOKEN]

print(f"Vocabulary limited to {len(vocab)} words (optimized for speed).")


print("Encoding sequences...")
sequences = []
lengths = []
X_list = []

for sent in sentences[:5000]:
  seq = []
  for word in sent:
    w_lower = word.lower()
    if not re.match(r'^[a-z]+$', w_lower): continue

    # Use specific ID if word in vocab, else use UNK ID
    idx = word_to_id.get(w_lower, unk_id)
    seq.append(idx)
```

```
    if len(seq) >= 2:
        sequences.append(seq)
        lengths.append(len(seq))
        X_list.extend(seq)

X = np.array(X_list).reshape(-1, 1)

# 3. TRAIN HMM
# ========================================
print(f"Training HMM on {len(X)} tokens...")
# n_components=10 is enough for a quick test. Increase to 50 later if needed.
model = hmm.CategoricalHMM(n_components=10, n_iter=10, random_state=42,
verbose=True)
model.fit(X, lengths=lengths)


print("        HMM training completed!")


def predict_next_word(context_words, top_k=3):
    """Predicts next word using the trained HMM"""

    context_ids = [word_to_id.get(w.lower(), unk_id) for w in context_words]

    if not context_ids: return []


    logprob, state_path = model.decode(np.array(context_ids).reshape(-1, 1),
algorithm="viterbi")
    last_hidden_state = state_path[-1]

    # 2. Look at the Transition Matrix: What hidden state comes next?
    next_state_probs = model.transmat_[last_hidden_state]
    best_next_state = np.argmax(next_state_probs)

    # 3. Look at Emission Matrix: What word does that next state emit?
    emission_probs = model.emissionprob_[best_next_state]

    # Get top K words
    top_indices = np.argsort(emission_probs)[-top_k:]

    results = []
    for idx in reversed(top_indices):
        word = id_to_word[idx]
        prob = emission_probs[idx]
```

```
        if word != UNK_TOKEN:
            results.append((word, prob))

    return results


# 5. TEST

test_phrases = ["the teacher", "good", "united", "he", "she"]

print("\n--- Predictions ---")
for phrase in test_phrases:
    preds = predict_next_word([phrase])
    print(f"Input: '{phrase}' -> Predicted: {preds}")
```

Output:

```
Building vocabulary...
Vocabulary limited to 2000 words (optimized for speed).
Encoding sequences...
Training HMM on 90708 tokens...
         1 -696245.74834861            +nan
         2 -426630.91588623 +269614.83246238
         3 -424364.61706964   +2266.29881659
         4 -421813.33313883   +2551.28393081
         5 -418886.61054640   +2926.72259243
         6 -416143.90411604   +2742.70643036
         7 -413879.81027245   +2264.09384359
         8 -412036.39669122   +1843.41358124
         9 -410515.25918771   +1521.13750351
✅ HMM training completed!

--- Predictions ---
Input: 'the teacher' -> Predicted: [('in', np.float64(0.1262511396317139)), ('and', np.float64(0.08313082402468734)), ('that', np.float64(0.070923055103
Input: 'good' -> Predicted: [('said', np.float64(0.012118465419133525)), ('year', np.float64(0.01180363316213765))]
Input: 'united' -> Predicted: [('said', np.float64(0.012118465419133525)), ('year', np.float64(0.01180363316213765))]
Input: 'he' -> Predicted: [('said', np.float64(0.012118465419133525)), ('year', np.float64(0.01180363316213765))]
Input: 'she' -> Predicted: [('said', np.float64(0.012118465419133525)), ('year', np.float64(0.01180363316213765))]
        10 -409238.28696761   +1276.97222009
```

Learning Outcome:

Upon completion of this experiment, HMM based model was built to predict the next word with POS-based suggestions, and evaluated the effectiveness of context-driven predictive text generation using the Brown Corpus.