Q9) Write a program to perform n-gram analysis on Amazon review data set and also compare result while performing different type of n-gram analysis on the given dataset

To perform n-gram analysis on an Amazon review dataset, Python with the nltk and sklearn libraries for natural language processing. Below is a sample program to perform unigrams, bigrams, and trigrams (n-grams) analysis and compare the results. It uses CountVectorizer from sklearn to extract the n-grams from the dataset.

python

Copy code

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from collections import Counter

import matplotlib.pyplot as plt

import nltk

from nltk.corpus import stopwords

import re

# Download stopwords from nltk

nltk.download('stopwords')

stop\_words = set(stopwords.words('english'))

# Load the Amazon review dataset (assuming the data is in a CSV file)

# Replace 'amazon\_reviews.csv' with the actual path to the dataset.

df = pd.read\_csv(‘Test.csv')

# Preprocessing: clean the text, remove non-alphabetic characters and convert to lowercase

def preprocess\_text(text):

text = re.sub(r'[^a-zA-Z\s]', '', text) # Remove non-alphabetic characters

text = text.lower() # Convert to lowercase

text = ' '.join([word for word in text.split() if word not in stop\_words]) # Remove stopwords

return text

df['cleaned\_review'] = df['reviewText'].apply(preprocess\_text)

# Function to perform n-gram analysis

def ngram\_analysis(text\_data, n):

vectorizer = CountVectorizer(ngram\_range=(n, n))

ngrams = vectorizer.fit\_transform(text\_data)

ngram\_counts = ngrams.sum(axis=0).A1

ngram\_features = vectorizer.get\_feature\_names\_out()

ngram\_freq = dict(zip(ngram\_features, ngram\_counts))

return Counter(ngram\_freq)

# Perform unigram, bigram, and trigram analysis

unigrams = ngram\_analysis(df['cleaned\_review'], 1)

bigrams = ngram\_analysis(df['cleaned\_review'], 2)

trigrams = ngram\_analysis(df['cleaned\_review'], 3)

# Get the top 10 n-grams for each analysis

def get\_top\_ngrams(ngram\_counter, top\_n=10):

return ngram\_counter.most\_common(top\_n)

top\_unigrams = get\_top\_ngrams(unigrams)

top\_bigrams = get\_top\_ngrams(bigrams)

top\_trigrams = get\_top\_ngrams(trigrams)

# Display the results

print("Top 10 Unigrams:")

for gram, freq in top\_unigrams:

print(f'{gram}: {freq}')

print("\nTop 10 Bigrams:")

for gram, freq in top\_bigrams:

print(f'{gram}: {freq}')

print("\nTop 10 Trigrams:")

for gram, freq in top\_trigrams:

print(f'{gram}: {freq}')

# Visualize the results

def plot\_ngrams(top\_ngrams, title):

labels, values = zip(\*top\_ngrams)

plt.figure(figsize=(10, 5))

plt.bar(labels, values)

plt.title(title)

plt.xticks(rotation=45, ha='right')

plt.show()

# Plot top n-grams for comparison

plot\_ngrams(top\_unigrams, "Top 10 Unigrams")

plot\_ngrams(top\_bigrams, "Top 10 Bigrams")

plot\_ngrams(top\_trigrams, "Top 10 Trigrams")

**Explanation:**

1. **Dataset Loading:** Load the Amazon reviews dataset using pandas. This assumes the dataset is in a CSV file containing a column reviewText with the reviews.
2. **Preprocessing:** Text is cleaned by removing non-alphabetic characters, converting to lowercase, and removing stopwords.
3. **N-gram Analysis:** Using CountVectorizer from sklearn, unigrams, bigrams, and trigrams are extracted. The ngram\_range is set to (n, n) to perform the specific n-gram analysis (e.g., 1 for unigrams, 2 for bigrams, etc.).
4. **Frequency Calculation:** The total frequency of each n-gram is calculated and stored in a dictionary using Counter.
5. **Top N-grams:** The most frequent unigrams, bigrams, and trigrams are displayed.
6. **Visualization:** Bar plots of the top 10 unigrams, bigrams, and trigrams are created to visually compare the n-gram distributions.

**Results:**

The program will output and plot the top 10 unigrams, bigrams, and trigrams from the Amazon review dataset. This allows you to observe how the frequency and relevance of words/phrases change as you move from unigrams to trigrams.