Veermata Jijabai Technological Institute, Mumbai 400019

Experiment No.: 03

Aim: creating N-Gram models with the help of given corpus. You may use NLTK or Only python to perform the same.

Steps:
1. Access the plain text corpus in your program as input to your model. Preprocess the text. This time without removing stopwords.
2. Write a program to compute unsmoothed unigrams, bigrams and trigrams model.
3. Calculate probability for every N-gram generated.
4. Store all of them in the descending order of probability in respective text files for future use in labs
5. program must be modular so we can reuse them in future.

Name: Kiran K Patil

Enrolment No.: 211070904

Branch: Computer Engineering

Batch: D

Theory:

The experiment you described involves creating N-Gram models using a given corpus. N-Gram models are commonly used in natural language processing and text analysis. Here is an explanation of each step in the experiment:

1. **Access the Plain Text Corpus:**
    - In this step, you will load a plain text corpus into your program. A corpus is a collection of text documents. This corpus will serve as the input for building the N-Gram models. It's important to preprocess the text, which typically includes tasks like tokenization and lowercasing. However, you are explicitly instructed not to remove stopwords in this step.

2. **Compute Unsmoothed N-Gram Models:**
    - In this step, you will write a program to compute unsmoothed unigram, bigram, and trigram models. N-Grams are contiguous sequences of N words. Unigrams are single words, bigrams are sequences of two words, and trigrams are sequences of three words.
    - For each N-Gram, you will count its occurrence in the corpus.

3. **Calculate N-Gram Probabilities:**
    - After counting the occurrences of each N-Gram, you will calculate the probability of each N-Gram. The probability of an N-Gram is the ratio of the count of that N-Gram to the count of the preceding (N-1)-Gram. For example, the probability of a bigram is the count of the bigram divided by the count of the preceding unigram.
    - This step helps you understand how likely each N-Gram is in the given corpus.

4. **Store N-Gram Models and Probabilities:**
    - You will store all the N-Grams, along with their probabilities, in descending order of probability in respective text files. This will enable you to use these models and probabilities for various language modeling tasks in the future.
    - Typically, you will have separate text files for unigrams, bigrams, and trigrams.

5. **Modular Program Design:**
    - The program should be modular, allowing you to reuse the code in the future. Modularity means that each step of the process should be encapsulated in functions or classes, making it easy to modify or extend the program for different corpora or N-Gram types.

This experiment can be implemented using Python and the Natural Language Toolkit (NLTK) or Python alone, depending on your preference. It's an essential task in natural language processing and can be used for tasks like text generation, language modeling, and text prediction. The resulting N-Gram models can be valuable resources for various language-related applications.

Implementation:

```
#Step 1: Access and Preprocess the Corpus

import nltk
nltk.download('punkt')
from nltk.util import ngrams
from collections import Counter

with open('sample_text_corpus2.txt', 'r', encoding='utf-8') as file:
    corpus = file.read()
print(corpus)
```

```
tokens = nltk.word_tokenize(corpus.lower())


#Step 2: Compute Unsmoothed N-Gram Models

unigrams = list(ngrams(tokens, 1))
bigrams = list(ngrams(tokens, 2))
trigrams = list(ngrams(tokens, 3))

#Step 3: Calculate N-Gram Probabilities

unigram_counts = dict(Counter(unigrams))
bigram_counts = dict(Counter(bigrams))
trigram_counts = dict(Counter(trigrams))

total_unigrams = len(unigrams)
total_bigrams = len(bigrams)
total_trigrams = len(trigrams)

unigram_probabilities = {unigram: count / total_unigrams for unigram,
count in unigram_counts.items()}
bigram_probabilities = {bigram: count / total_bigrams for bigram, count
in bigram_counts.items()}
trigram_probabilities = {trigram: count / total_trigrams for trigram,
count in trigram_counts.items()}


#Step 4: Store N-Gram Models in Descending Order of Probability

def save_ngram_model(filename, ngram_probabilities):
    with open(filename, 'w', encoding='utf-8') as file:
        for ngram, probability in sorted(ngram_probabilities.items(),
key=lambda x: -x[1]):
            file.write(f"{ngram}: {probability:.6f}\n")
```

```
save_ngram_model("unigram_model.txt", unigram_probabilities)
save_ngram_model("bigram_model.txt", bigram_probabilities)
save_ngram_model("trigram_model.txt", trigram_probabilities)
```

```
#Step 5: Create a Modular Program

def create_ngram_models(corpus_file):
    # Insert the code from steps 3 to 7 here
    corpus_file = "your_corpus.txt"  # Replace with the path to your
plain text corpus
    create_ngram_models(corpus_file)


#Step 6 : Display output

with open('unigram_model.txt', 'r', encoding='utf-8') as file:
    unigram_output = file.read()

print(unigram_output)
```

```
with open('bigram_model.txt', 'r', encoding='utf-8') as file:
    bigram_output = file.read()

print(bigram_output)
```

```
with open('trigram_model.txt', 'r', encoding='utf-8') as file:
    trigram_output = file.read()

print(trigram_output)
```

Output :

Sample text corpus :

```
The quick brown fox jumps over the lazy dog. The sun is shining brightly in the clear blue sky.
```

Unigram output :

```
('the',): 0.190476
('.',): 0.095238
('quick',): 0.047619
('brown',): 0.047619
('fox',): 0.047619
('jumps',): 0.047619
('over',): 0.047619
('lazy',): 0.047619
('dog',): 0.047619
('sun',): 0.047619
('is',): 0.047619
('shining',): 0.047619
('brightly',): 0.047619
('in',): 0.047619
('clear',): 0.047619
('blue',): 0.047619
('sky',): 0.047619
```

Bigram Output :

```
('the', 'quick'): 0.050000
('quick', 'brown'): 0.050000
('brown', 'fox'): 0.050000
('fox', 'jumps'): 0.050000
('jumps', 'over'): 0.050000
('over', 'the'): 0.050000
('the', 'lazy'): 0.050000
('lazy', 'dog'): 0.050000
('dog', '.'): 0.050000
('.', 'the'): 0.050000
('the', 'sun'): 0.050000
('sun', 'is'): 0.050000
('is', 'shining'): 0.050000
('shining', 'brightly'): 0.050000
('brightly', 'in'): 0.050000
('in', 'the'): 0.050000
('the', 'clear'): 0.050000
('clear', 'blue'): 0.050000
('blue', 'sky'): 0.050000
('sky', '.'): 0.050000
```

Trigram Output :

```
('the', 'quick', 'brown'): 0.052632
('quick', 'brown', 'fox'): 0.052632
('brown', 'fox', 'jumps'): 0.052632
('fox', 'jumps', 'over'): 0.052632
('jumps', 'over', 'the'): 0.052632
('over', 'the', 'lazy'): 0.052632
('the', 'lazy', 'dog'): 0.052632
('lazy', 'dog', '.'): 0.052632
('dog', '.', 'the'): 0.052632
('.', 'the', 'sun'): 0.052632
('the', 'sun', 'is'): 0.052632
('sun', 'is', 'shining'): 0.052632
('is', 'shining', 'brightly'): 0.052632
('shining', 'brightly', 'in'): 0.052632
('brightly', 'in', 'the'): 0.052632
('in', 'the', 'clear'): 0.052632
('the', 'clear', 'blue'): 0.052632
('clear', 'blue', 'sky'): 0.052632
('blue', 'sky', '.'): 0.052632
```

Conclusion:

In this experiment, we successfully created N-Gram models, including unigrams, bigrams, and trigrams, from a given text corpus. We calculated the probabilities for each N-Gram and stored them in descending order of probability in separate text files. The modular design of the program ensures reusability for future projects. N-Gram models are essential tools in natural language processing, and this experiment provides a solid foundation for their creation and application in various language-related tasks.