Experiment no.3

Aim: To study N-gram model

Theory:

Given a sequence of N-1 words, an N-gram model predicts the most probable word that might follow this sequence. It's a probabilistic model that's trained on a corpus of text. Such a model is useful in many NLP applications including speech recognition, machine translation and predictive text input.

An N-gram model is built by counting how often word sequences occur in corpus text and then estimating the probabilities. Since a simple N-gram model has limitations, improvements are often made via smoothing, interpolation and backoff. An N-gram model is one type of a Language Model (LM), which is about finding the probability distribution over word sequences.

Consider two sentences: "There was heavy rain" vs. "There was heavy flood". From experience, we know that the former sentence sounds better. An N-gram model will tell us that "heavy rain" occurs much more often than "heavy flood" in the training corpus. Thus, the first sentence is more probable and will be selected by the model.

A model that simply relies on how often a word occurs without looking at previous words is called unigram. If a model considers only the previous word to predict the current word, then it's called bigram. If two previous words are considered, then it's a trigram model.

An n-gram model for the above example would calculate the following probability:

P('There was heavy rain') = P('There', 'was', 'heavy', 'rain') =

P('There')P('was'|'There')P('heavy'|'There was')P('rain'|'There was heavy')

Since it's impractical to calculate these conditional probabilities, using Markov assumption, we

approximate this to a bigram model:

P('There was heavy rain') ~ P('There')P('was'|'There')P('heavy'|'was')P('rain'|'heavy')

In speech recognition, input may be noisy and this can lead to wrong speech-to-text conversions. N-gram models can correct this based on their knowledge of the probabilities. Likewise, N-gram models are used in machine translation to produce more natural sentences in the target language. When correcting for spelling errors, sometimes dictionary lookups will not help. For example, in the phrase "in about fifteen minutes" the word 'minuets' is a valid dictionary word but it's incorrect in this context. N-gram models can correct such errors.

N-gram models are usually at word level. It's also been used at character level to do stemming, that is, separate the root word from the suffix. By looking at N-gram statistics, we could also classify languages or differentiate between US and UK spellings. For example, 'sz' is common in

Czech; 'gb' and 'kp' are common in Igbo.

In general, many NLP applications benefit from N-gram models including part-of-speech tagging, natural language generation, word similarity, sentiment extraction and predictive text input.

Code:

```
import re

from nltk.util import ngrams

s = "Machine learning is an important part of AI " "and AI is going to become inmporant for daily functio

nong "

tokens = [token for token in s.split(" ")]

output = list(ngrams(tokens, 2))

print(output)
```

Output:

Output:

```
[('Machine', 'learning'), ('learning', 'is'), ('is', 'an'), ('an', 'important'), ('important', 'part'), ('part', 'of'),

('of', 'AI'), ('AI', 'and'), ('and', 'AI'), ('AI', 'is'), ('is', 'going'), ('going', 'to'), ('to', 'become'), ('become',

'inmporant'), ('inmporant', 'for'), ('for', 'daily'), ('daily', 'functionong'), ('functionong', '')]
```

Conclusion:

Thus, in the above experiment we have studied regarding N-Gram Model in detail with the help of theory and then tried to implement the code and successfully executed it.