# N-grams

## Objective

The goal of this task was to explore the predictive capabilities of a statistical language model using n-grams. By leveraging the Brown corpus, we aimed to generate next-word predictions given a phrase using simple frequency-based methods.

## Corpus & Preprocessing

We used the Brown corpus available in nltk, which is a balanced collection of texts spanning multiple domains, making it ideal for language modeling tasks.

## Preprocessing steps:

- Converted all tokens to lowercase
- Removed punctuation and numeric tokens using word.isalpha()

This ensured that the resulting tokens were clean and focused strictly on meaningful language units.

```
import nltk
import pandas as pd
import string
from collections import Counter
from nltk.corpus import brown
from nltk.util import ngrams

# Download the corpus
nltk.download('brown')

# Load words and remove punctuation
tokenized_text = [word.lower() for word in brown.words() if word.isalpha()]
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
```

## Methodology

### 1. N-Gram Construction

We used the nltk.util.ngrams function to generate **trigrams (n=3)**. The trigrams were then counted and stored in a DataFrame sorted by frequency.

```python
def generate_ngrams_df(tokens, n):

    """Generate n-gram dataframe with frequency counts."""

    n_gram_list = list(ngrams(tokens, n))
    n_gram_freq = Counter(n_gram_list)

    df = pd.DataFrame(n_gram_freq.items(), columns=['ngram', 'frequency'])
    df.sort_values(by='frequency', ascending=False, inplace=True)
    return df.reset_index(drop=True)
```

**Why trigrams?** I used Trigrams to strike a balance between simplicity and context. Bigrams often lack enough predictive power, while higher-order n-grams can be too sparse.

```python
[ ] ngram_df = generate_ngrams_df(tokenized_text, 3)
    ngram_df.head()
```

| | ngram | frequency |
|---|---|---|
| 0 | (one, of, the) | 404 |
| 1 | (the, united, states) | 337 |
| 2 | (as, well, as) | 238 |
| 3 | (some, of, the) | 179 |
| 4 | (out, of, the) | 174 |

**1. ('one', 'of', 'the') – 404 occurrences**

- This is a very common English phrase structure.
- Appears in both formal and informal texts : for example, *"one of the best"*, *"one of the reasons"*.
- High frequency suggests this trigram is a core syntactic structure in English.

**2. ('the', 'united', 'states') – 337 occurrences**

- This is a named entity (country name) and appears consistently in geopolitical or historical content.
- Brown corpus, being sourced from 20th-century American publications, naturally contains frequent references to The United States**.**

**3. ('as', 'well', 'as') – 238 occurrences**

- A common comparative phrase used in both writing and speech.

- It functions grammatically like a conjunction and appears frequently in expository or academic-style text.

**4. ('some', 'of', 'the') – 179 occurrences**

- Another grammatical pattern frequently used in descriptive sentences.
- Often followed by a noun.
- Indicates partial quantities or subgroups.

**5. ('out', 'of', 'the') – 174 occurrences**

- Likely part of prepositional phrases like *"out of the house"*, *"out of the question"*.
- This is part of idiomatic expressions and is common in narrative or dialogue.

### 2. Next Word Prediction Function

We implemented a function to extract the most frequent next words that follow a given phrase using trigram matches.

```python
def predict_next_words(df, input_text, k=5):
    """
    Given a DataFrame of n-grams and an input string, return top k next-word predictions.
    """
    input_tokens = tuple(input_text.lower().split())
    n = len(df['ngram'].iloc[0])

    if len(input_tokens) != n - 1:
        raise ValueError(f"Input must have {n - 1} words for n={n}")

    # Filter n-grams that match the prefix
    matching = df[df['ngram'].apply(lambda x: x[:-1] == input_tokens)]

    # Return top-k most frequent next words
    top_k = matching.head(k)['ngram'].apply(lambda x: x[-1]).tolist()
    return top_k
```

This function ensures:

- Input validity (right number of tokens for n-1)
- Prefix matching for the last n-1 tokens
- Top k predictions based on frequency

## Example Result

We tested the model with the phrase:

```
[ ]  # Build trigrams (n=3)
     ngram_df = generate_ngrams_df(tokenized_text, 3)

     # Predict next words
     predict_next_words(ngram_df, input_text="the united", k=5)
```

```
['states', 'nations', 'kingdom', 'declining', 'steel']
```

## Interpretation & Insights

| Metric | Comment |
|---|---|
| **Relevance** | All results are semantically and syntactically plausible |
| **Context Awareness** | Completely absent, the model uses no understanding beyond co-occurrence |
| **Bias** | Model reflects frequency bias of the Brown corpus (1950s–60s era) |
| **Performance** | High speed, low complexity, but limited scope for generalization |

**Strengths**:

- Extremely fast and simple to build
- Easy to explain and interpret
- Good baseline for low-resource NLP tasks

**Weaknesses**:

- Doesn’t generalize to unseen sequences
- Purely frequency-driven, lacks understanding of meaning
- No smoothing or backoff mechanism included in current form

## Conclusion

This project demonstrated how n-gram models can provide accurate and interpretable next-word suggestions based purely on frequency statistics. Although they lack deeper understanding, they remain a useful tool in early NLP pipelines or as baselines for more advanced language models.