5.N-Grams in Natural Language Processing

Introduction

N-grams are a fundamental concept in Natural Language Processing (NLP) that enhance text representation by capturing word sequences and context. They address the limitations of simple bag-of-words approaches by considering word combinations rather than individual words in isolation.

The Problem with Bag-of-Words

Consider two sentences with opposite meanings:

Sentence 1: "The food is good"
Sentence 2: "The food is not good"

Using a simple bag-of-words approach with stop word removal, the vocabulary becomes:

$$\text{Vocabulary} = \{\text{food}, \text{not}, \text{good}\}$$

The vector representations would be:

$$\text{Sentence}_1 = [1, 0, 1]$$

$$\text{Sentence}_2 = [1, 1, 1]$$

Where:

- First position represents "food" (present in both → 1, 1)
- Second position represents "not" (absent in Sentence 1 → 0, present in Sentence 2 → 1)
- Third position represents "good" (present in both → 1, 1)

Problem: These vectors are nearly identical (differ by only one element) despite the sentences having opposite meanings. This fails to capture the semantic difference between positive and negative sentiment.

What are N-Grams?

N-grams are contiguous sequences of n items (words, characters, or tokens) from a given text. The value of 'n' determines the type of n-gram:

- **Unigram (n=1):** Single words
- **Bigram (n=2):** Two consecutive words
- **Trigram (n=3):** Three consecutive words
- **N-gram (n=N):** N consecutive words

Mathematical Definition

For a sequence of words $W = \{w_1, w_2, w_3, ..., w_m\}$, an n-gram is defined as:

$$\text{n-gram}_i = (w_i, w_{i+1}, w_{i+2}, ..., w_{i+n-1})$$

Where:

- w_i is the word at position i
- n is the length of the sequence
- i ranges from 1 to $(m - n + 1)$
- m is the total number of words in the sequence

Solving the Problem with Bigrams (n=2)

Let's apply bigrams to our example sentences.

Extended Vocabulary

$$\text{Vocabulary} = \{\text{food}, \text{not}, \text{good}, \text{food good}, \text{food not}, \text{not good}\}$$

This vocabulary includes:

- Three **unigrams**: individual words (food, not, good)
- Three **bigrams**: word pairs (food good, food not, not good)

Vector Representation with Bigrams

For **Sentence 1: "The food is good"**

$$\text{Sentence}_1 = [1, 0, 1, 1, 0, 0]$$

Breaking down each position:

- Position 1: "food" is present → 1
- Position 2: "not" is absent → 0
- Position 3: "good" is present → 1
- Position 4: "food good" appears consecutively → 1
- Position 5: "food not" does not appear → 0
- Position 6: "not good" does not appear → 0

For **Sentence 2: "The food is not good"**

$$\text{Sentence}_2 = [1, 1, 1, 0, 1, 1]$$

Breaking down each position:

- Position 1: "food" is present → 1
- Position 2: "not" is present → 1
- Position 3: "good" is present → 1
- Position 4: "food good" does not appear consecutively → 0
- Position 5: "food not" appears consecutively → 1
- Position 6: "not good" appears consecutively → 1

Result

The vectors now differ in **4 out of 6 positions**, making them much more distinguishable. The model can now clearly differentiate between the positive and negative sentences.

N-Gram Range Parameter

In scikit-learn's text vectorization tools, the `ngram_range` parameter controls which n-grams to include:

$$\text{ngram\_range} = (n_{min}, n_{max})$$

Where:

- n_{min} is the minimum n-gram size
- n_{max} is the maximum n-gram size

Examples

1. **ngram_range = (1, 1):** Unigrams only

$$\text{Features} = \{\text{unigrams}\}$$

2. **ngram_range = (1, 2):** Unigrams + Bigrams

$$\text{Features} = \{\text{unigrams}\} \cup \{\text{bigrams}\}$$

3. **ngram_range = (1, 3):** Unigrams + Bigrams + Trigrams

$$\text{Features} = \{\text{unigrams}\} \cup \{\text{bigrams}\} \cup \{\text{trigrams}\}$$

4. **ngram_range = (2, 3):** Bigrams + Trigrams only

$$\text{Features} = \{\text{bigrams}\} \cup \{\text{trigrams}\}$$

Advantages of N-Grams

1. **Contextual Information:** Captures word order and local context
2. **Better Semantic Representation:** Distinguishes between similar sentences with different meanings
3. **Phrase Detection:** Identifies common phrases and expressions
4. **Improved Model Performance:** Enhances classification and sentiment analysis tasks

Practical Considerations

Feature Space Growth

The number of possible n-grams grows exponentially:

$$|\text{Vocabulary}_{n\text{-gram}}| \approx |\text{Vocabulary}_{unigram}|^n$$

Where:

- $|\text{Vocabulary}_{n\text{-gram}}|$ is the total number of possible n-grams
- $|\text{Vocabulary}_{unigram}|$ is the total number of unique words
- n is the n-gram size

This means higher values of n lead to:

- **Larger feature spaces** (higher dimensionality)
- **Increased computational cost**
- **Greater memory requirements**
- **Potential overfitting** due to sparse representations

Optimal N-Gram Selection

Choose n-gram ranges based on:

- **Task requirements:** Sentiment analysis often benefits from bigrams and trigrams
- **Dataset size:** Larger datasets can support higher n values
- **Computational resources:** Balance between performance and efficiency
- **Domain specificity:** Some domains have characteristic multi-word phrases

Summary

N-grams extend basic text representation by incorporating word sequences, enabling models to:

- Capture local word order and context
- Better distinguish between semantically different sentences
- Represent phrases and common expressions
- Improve performance on various NLP tasks

The combination of unigrams, bigrams, and trigrams (ngram_range = (1, 3)) often provides a good balance between contextual information and computational efficiency.