Feature Extraction in NLP

Table of Contents

What is Feature Extraction in NLP?

Why Do We Need Feature Extraction?

After **text preprocessing**, we still have **raw text**, which **machines cannot understand directly**. We need to **convert text into numerical features** so that machine learning models can process it.

What is Feature Extraction?

Feature extraction is the process of **converting text data into numerical vectors**. It is a crucial step before applying machine learning models. The two most common techniques are:

- **Bag of Words (BoW)**
- **TF-IDF (Term Frequency-Inverse Document Frequency)**

Feature Extraction: Bag of Words (BoW)

What is Bag of Words?

The **Bag of Words model** represents text as a **collection of word counts** without considering the order of words.

How It Works?

1️⃣ **Create a vocabulary** of all unique words in the dataset.
2️⃣ **Count how many times each word appears** in a sentence/document.
3️⃣ **Represent this as a numerical vector**.

Example:

Let's take **three movie reviews**:

Review	Sentence
1	"The movie was good and we really like it"
2	"The movie was good but the ending was boring"
3	"We did not like the movie as it was too lengthy"

The vocabulary extracted:

```
['movie', 'good', 'like', 'boring', 'lengthy', 'ending', 'really', 'did', 'not']
```

Now, we count the **frequency of each word** in each sentence:

	movie	good	like	boring	lengthy	ending	really	did	not
Review 1	1	1	1	0	0	0	1	0	0
Review 2	1	1	0	1	0	1	0	0	0
Review 3	1	0	1	0	1	0	0	1	1

Implementing Bag of Words in Python

```
from sklearn.feature_extraction.text import CountVectorizer

# Sample Reviews
reviews = [
    "The movie was good and we really like it",
    "The movie was good but the ending was boring",
    "We did not like the movie as it was too lengthy"
]

# Initialize BoW Model
vectorizer = CountVectorizer(stop_words="english")
bow_matrix = vectorizer.fit_transform(reviews)

# Convert BoW to a DataFrame
import pandas as pd
bow_df = pd.DataFrame(bow_matrix.toarray(),
columns=vectorizer.get_feature_names_out())

# Display the DataFrame
print(bow_df)
```

Key Observations:

- The words are converted into numerical values.

- **Order of words is lost**, but **frequency information is preserved**.

Feature Extraction: TF-IDF (Term Frequency-Inverse Document Frequency)

What is TF-IDF?

While **Bag of Words** counts the **frequency of words**, **TF-IDF** goes a step further: ✅**It assigns more importance to words that are unique to a document**
✅**It reduces the importance of words that appear frequently across all documents (e.g., "the", "is")**

How is TF-IDF Calculated?

TF-IDF is computed using two values:

1️⃣**Term Frequency (TF)** = `(Number of times a word appears in a document) / (Total words in that document)`
2️⃣**Inverse Document Frequency (IDF)** = `log(Total number of documents / Number of documents containing the word)`

Final Formula:

$$\text{TF-IDF} = \text{TF} \times \log(\frac{N}{d})$$

Where:

- **N** = Total number of documents
- **d** = Number of documents containing the word

Example:

Let's calculate **TF-IDF manually** for the word `"lengthy"`.

Term	TF (Review 1)	TF (Review 2)	TF (Review 3)	IDF	TF-IDF
lengthy	0	0	1	0.239	**0.239**
boring	0	1	0	0.159	**0.159**

As you can see, **"lengthy"** has a higher TF-IDF value because it is **more unique to Review 3**.

Implementing TF-IDF in Python

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample Reviews
reviews = [
    "The movie was good and we really like it",
    "The movie was good but the ending was boring",
    "We did not like the movie as it was too lengthy"
]

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words="english")
tfidf_matrix = vectorizer.fit_transform(reviews)

# Convert TF-IDF to a DataFrame
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(),
columns=vectorizer.get_feature_names_out())

# Display the DataFrame
print(tfidf_df)
```

Key Observations:

- Words like `"lengthy"` have a **higher TF-IDF value** than common words.
- Words that appear **in all sentences** (e.g., `"movie"`) get **lower scores**.

Bag of Words vs TF-IDF: Which One to Use?

Feature	Bag of Words (BoW)	TF-IDF
Captures Frequency?	✅Yes	✅Yes
Considers Importance of Words?	❌No	✅Yes
Handles Common Words?	❌No, treats all words equally	✅Reduces the impact of common words
When to Use?	Simple models like Naïve Bayes	More advanced models like SVM, Neural Networks

✅**Use BoW when word frequency matters more than importance.**
✅**Use TF-IDF when you want to reduce the impact of frequent words and focus on rare, meaningful words.**

Summary

Feature extraction converts text into numerical vectors for machine learning.
Bag of Words counts word occurrences but doesn't account for importance.
TF-IDF reduces common words' importance and highlights unique terms.
TF-IDF is generally more effective for NLP tasks like text classification and sentiment analysis.