TEXT REPRESENTATION - NLP LECTURE 4
Bag of Words | Tf-Idf | N-grams, Bi-grams and Uni-grams | OHE

Author : Harshavardhan

INSTRUCTOR: Nitish Sir, CampusX

Text Representation

Text representation in natural language processing (NLP) refers to the process of converting textual data into a numerical format that can be understood and processed by machine learning algorithms. This is necessary because machines can only work with numerical data.

These numerical representations allow machines to analyze and understand the text, perform tasks like sentiment analysis, text classification, and machine translation. By converting text into a numerical format, text representation enables machines to process and interpret human language.

Techniques covered in this lecture:

- One Hot encoding
- Bag of words
- N-grams, Bi-grams and Uni-grams
- Tf-Idf
- Custom features

Why is Text representation important ?

- It enables machines to understand and analyze human language.
- It allows for effective information extraction and retrieval.
- It helps in tasks like sentiment analysis, text classification, and machine translation.
- It plays a crucial role in various applications, including search engines, recommendation systems, and automated content generation.

Why is Text representation difficult ?

- Human language is complex with synonyms and ambiguity.
- Text data lacks a predefined structure.
- Sentence structures and grammar vary.
- Figurative language and idioms complicate understanding.
- Different techniques have limitations.
- Contextual understanding is subjective and challenging.

Key Terms

1. Corpus (C) :

A Corpus refers to a collection of texts or documents that are used for analysis and study. It can be a large collection of books, articles, web pages, or any other written material.

2. Vocabulary (V) :

Vocabulary refers to the set of unique words or tokens present in a given text or corpus. It is created by extracting and collecting all the distinct words from the text data.

3. Document (D) :

A Document refers to a piece of text or a collection of text that is being analyzed or processed. It can be a single sentence, a paragraph, an article, or even a book.

4. Word (W) :

A Word refers to a unit of language that carries meaning and is typically separated by spaces or punctuation marks. It is the basic building block of a sentence or a piece of text.

One Hot Encoding

Let's consider the following Documents

D1	people watch campusx
D2	campusx watch campusx
D3	people write comment
D4	campusx write comment

Corpus : people watch campusx campusx watch campusx people write comment campusx write comment

Vocabulary : people watch campusx write comment

V = 5

D1	people	watch	campusx	write	comment
people	1	0	0	0	0
watch	0	1	0	0	0
campusx	0	0	1	0	0

D1:

[[1, 0, 0, 0, 0] represents the word "people"

[0, 1, 0, 0, 0] represents the word "watch"

[0, 0, 1, 0, 0]] represents the word "campusx"

D2:

[[0, 0, 1, 0, 0]   # represents the word "campusx"

[0, 1, 0, 0, 0]   # represents the word "watch"

[0, 0, 1, 0, 0]]   #represents the word "campusx"

D3:

[[1, 0, 0, 0, 0]   # represents the word "people"

[0, 0, 0, 1, 0]   # represents the word "write"

[0, 0, 0, 0, 1]]   # represents the word "comment"

D4:

[[0, 0, 1, 0, 0]   # represents the word "campusx"

[0, 0, 0, 1, 0]   # represents the word "write"

[0, 0, 0, 0, 1]]   # represents the word "comment"

Advantages of OHE :

- One-hot encoding is simple and easy to understand and Implement.

Disadvantages of OHE :

- **Increased Sparsity:** Since most elements in the one-hot encoded vector are zeros, it can lead to increased sparsity in the data. This can affect the efficiency of certain algorithms and models, especially those that are not optimized for sparse data.
- **High Dimensionality:** One-hot encoding can lead to high-dimensional feature vectors, especially when dealing with large vocabularies. This can increase the computational complexity and memory requirements of the model.
- **Lack of Semantic Information:** One-hot encoding treats each word as independent and does not capture any semantic relationships between words. This can limit the model's ability to understand context and meaning in natural language.
- **Out-of-vocabulary (OOV)**:One-hot encoding requires a fixed vocabulary, so any words outside of this vocabulary are considered out-of-vocabulary (OOV).

Bag of Words

Bag of Words (BoW) is a simple and popular technique used in natural language processing (NLP) to represent text data. It involves treating a piece of text as a "bag" or collection of individual words, disregarding grammar and word order.

Let’s consider the same example

	TEXT	OUTPUT
D1	people watch campusx	1
D2	campusx watch campusx	1
D3	people write comment	0
D4	campusx write comment	0

V = 5

Now count the number of times words appear in each document

DOCUMENT	people	watch	campusx	write	comment
D1	1	1	1	0	0
D2	0	1	2	0	0
D3	1	0	0	1	1
D4	0	0	1	1	1

The core intuition behind the bag of words (BoW) model is to represent text data by focusing solely on the presence and frequency of individual words, without considering their order or meaning. This approach is based on the assumption that the occurrence of words in a document provides valuable information about its content.

The key intuition is that words carry information and by counting the occurrences of words in a document, we can create a numerical representation that captures some aspects of the document's content. This representation can then be used for various tasks such as text classification, sentiment analysis, or information retrieval.

Bag of Words using CountVectorizer

```
import pandas as pd
import numpy as np
```

```
# Importing the pandas module
import pandas as pd

# Creating a DataFrame with 'text' and 'output' columns
df = pd.DataFrame({'text': ['people watch campusx',
                            'campusx watch campusx',
                            'people write comment',
                            'campusx write comment'],
                   'output': [1, 1, 0, 0]})
```

```
df
```

	text	output
0	people watch campusx	1
1	campusx watch campusx	1
2	people write comment	0
3	campusx write comment	0

```
# Importing the CountVectorizer module
from sklearn.feature_extraction.text import CountVectorizer

# Initializing the CountVectorizer object with desired parameters
cv = CountVectorizer(
    lowercase=True,  # Convert all characters to lowercase before tokenizing
    binary=False,    # Set binary=True for problems like sentiment analysis
    max_features=None  # If not None, build a vocabulary that only considers
                       # the top max_features ordered by term frequency across
                       # the corpus. Otherwise, all features are used.
)
```

```
# Transforming the 'text' column of the DataFrame using the CountVectorizer
bow = cv.fit_transform(df['text'])
```

```
# Printing the vocabulary

# The 'cv.vocabulary_' attribute contains a dictionary where the keys are the unique words
# in the documents and the values are the corresponding indices assigned to each word.
print(cv.vocabulary_)
```

```
{'people': 2, 'watch': 3, 'campusx': 0, 'write': 4, 'comment': 1}
```

```
# Converting sparse matrix to array

# The 'toarray()' method is used to convert the sparse matrix 'bow[0]'
# to a dense array representation.

print(bow[0].toarray())
print(bow[1].toarray())
```

```
[[1 0 1 1 0]]
[[2 0 0 1 0]]
```

out of vocab problem is handled here

see how and,of, and other words are handled below they were absent during the training time # Handling Out-of-Vocabulary (OOV) problem When using CountVectorizer, the vocabulary is built based on the words present in the training data. If there are words in the test data that were not present during training,they will not be included in the vocabulary.

Let's see how the words "other" and "book" are handled below: Since these words were absent during the training time, they will not be present in the vocabulary.

```
print(cv.transform(["write other book "]).toarray())
```

```
[[0 0 0 0 1]]
```

Advantages of BOW :

- Simple and intuitive representation
- Language independent
- Efficient feature extraction
- Measures document similarity
- Scalable for large datasets
- Interpretable results

Disadvantages of BOW:

- Sparsity: BOW representation often results in sparse vectors, where most elements are zeros, which can be inefficient in terms of storage and computation.
- Loss of word order: BOW disregards the order of words in a document, potentially losing important contextual information.
- Out-of-vocabulary (OOV) problem: BOW does not handle new or unseen words well, as they are not represented in the initial vocabulary and may be ignored during analysis.
- Lack of semantic meaning: BOW treats each word as an independent feature, ignoring semantic relationships between words.
- Sensitivity to misspellings and variations: BOW is sensitive to misspelled words or variations of the same word, which can lead to inaccurate representations.

N-grams (Bag of n-grams)

N-grams refer to contiguous sequences of n items, usually words or characters, extracted from a text. These n-grams can be used to analyze the structure and patterns within the text data.

Bi-gram : A bigram is an n-gram with n equal to 2, meaning it consists of two consecutive elements, typically words, in a sequence of text.

Consider the sentence is : "I enjoy learning new things", the bi-grams are

- "I enjoy"
- "enjoy learning"
- "learning new"
- "new things"

Let’s consider the another example

	TEXT	OUTPUT
D1	people watch campusx	1
D2	campusx watch campusx	1
D3	people write comment	0
D4	campusx write comment	0

Bag of Bi-grams :

DOCUMENT	people watch	watch campusx	campusx watch	people write	write comment	campusx write
D1	1	1	0	0	0	0
D2	0	1	1	0	0	0
D3	1	0	0	1	1	0
D4	0	0	0	0	1	1

Bag of tri-grams :

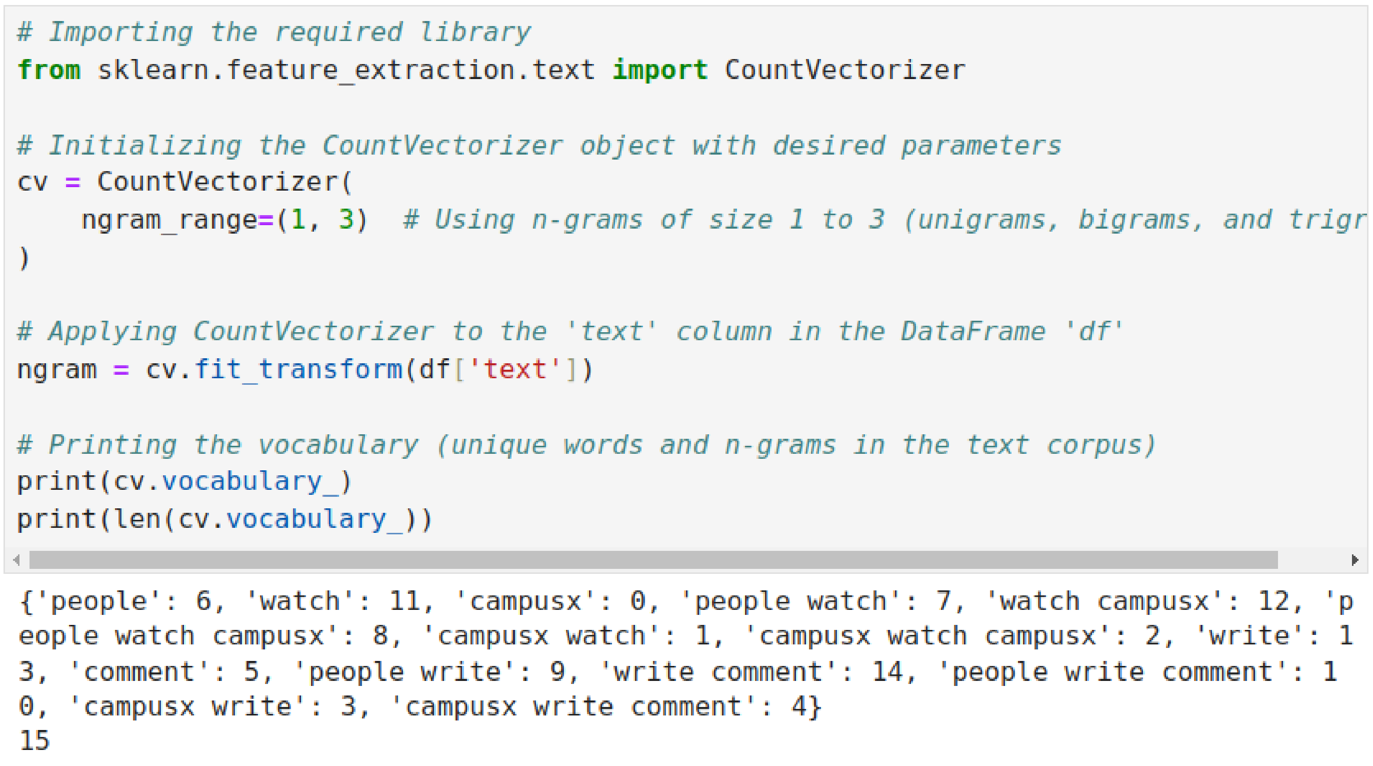
	people watch campusx	campusx watch campusx	people write comment	campusx write comment
D1	1	0	0	0
D2	0	1	0	0
D3	0	0	1	0
D4	0	0	0	1

Implementation in Python

```python
# Importing the required library
from sklearn.feature_extraction.text import CountVectorizer

# Initializing the CountVectorizer object with desired parameters
cv = CountVectorizer(
    ngram_range=(2, 2)  # Using bigrams (ngram_range=(2, 2))
)

# Applying CountVectorizer to the 'text' column in the DataFrame 'df'
bigram = cv.fit_transform(df['text'])

# Printing the vocabulary
print(cv.vocabulary_)
```

```
{'people watch': 2, 'watch campusx': 4, 'campusx watch': 0, 'people write': 3, 'wri
te comment': 5, 'campusx write': 1}
```

Using n-grams of size 1 to 3 (unigrams, bigrams, and trigrams)



```python
# Importing the required library
from sklearn.feature_extraction.text import CountVectorizer

# Initializing the CountVectorizer object with desired parameters
cv = CountVectorizer(
    ngram_range=(1, 3)  # Using n-grams of size 1 to 3 (unigrams, bigrams, and trigr
)

# Applying CountVectorizer to the 'text' column in the DataFrame 'df'
ngram = cv.fit_transform(df['text'])

# Printing the vocabulary (unique words and n-grams in the text corpus)
print(cv.vocabulary_)
print(len(cv.vocabulary_))
```

```
{'people': 6, 'watch': 11, 'campusx': 0, 'people watch': 7, 'watch campusx': 12, 'p
eople watch campusx': 8, 'campusx watch': 1, 'campusx watch campusx': 2, 'write': 1
3, 'comment': 5, 'people write': 9, 'write comment': 14, 'people write comment': 1
0, 'campusx write': 3, 'campusx write comment': 4}
15
```

Advantages of n-grams :

- n-grams can capture some semantic meaning in sentences by preserving word combinations.
- Easy to implement

Disadvantages of n-grams:

- **High Dimensionality:** As the n-gram size increases, the number of features in the dataset grows significantly, leading to higher dimensionality and increased computational requirements.
- **Out-of-Vocabulary Words:** N-grams may not handle out-of-vocabulary words well, as they rely on fixed sequences seen during training.
- **Sparsity Issues with Short Text:** In short texts, n-grams may not provide enough context for meaningful analysis, leading to sparse and less informative feature representations.

Tf-Idf

TF-IDF stands for "Term Frequency-Inverse Document Frequency." It is a numerical representation used in Natural Language Processing (NLP) to measure the importance of a word in a document relative to a collection of documents.

Term Frequency (TF): This represents how frequently a word appears in a document. The more times a word occurs, the higher its term frequency in that document.

$$TF(t,d) = \frac{(Number\ of\ occurrences\ of\ term\ t\ in\ document\ d)}{(Total\ number\ of\ terms\ in\ the\ document\ d)}$$

Inverse Document Frequency (IDF): This measures how unique or rare a word is across all documents in a collection. It helps to downweight common words like "the" and "and" while emphasizing rare and more important words.

$$IDF(t) = log_e \frac{(Number\ of\ documents\ in\ the\ corpus\ )}{(Number\ of\ documents\ with\ term\ t\ in\ them)}$$

The TF-IDF score is calculated by multiplying the term frequency of a word in a document by its inverse document frequency across the entire collection of documents. The resulting score helps identify words that are relatively important and distinctive to a particular document compared to the entire dataset.

Let’s consider the same example

	TEXT	**OUTPUT**
D1	people watch campusx	1
D2	campusx watch campusx	1
D3	people write comment	0
D4	campusx write comment	0

TF

Term / Doc	D1	D2	D3	D4
people	1/3	0	1/3	0
watch	1/3	1/3	0	0
campusx	1/3	2/3	0	1/3
write	0	0	1/3	1/3
comment	0	0	1/3	1/3

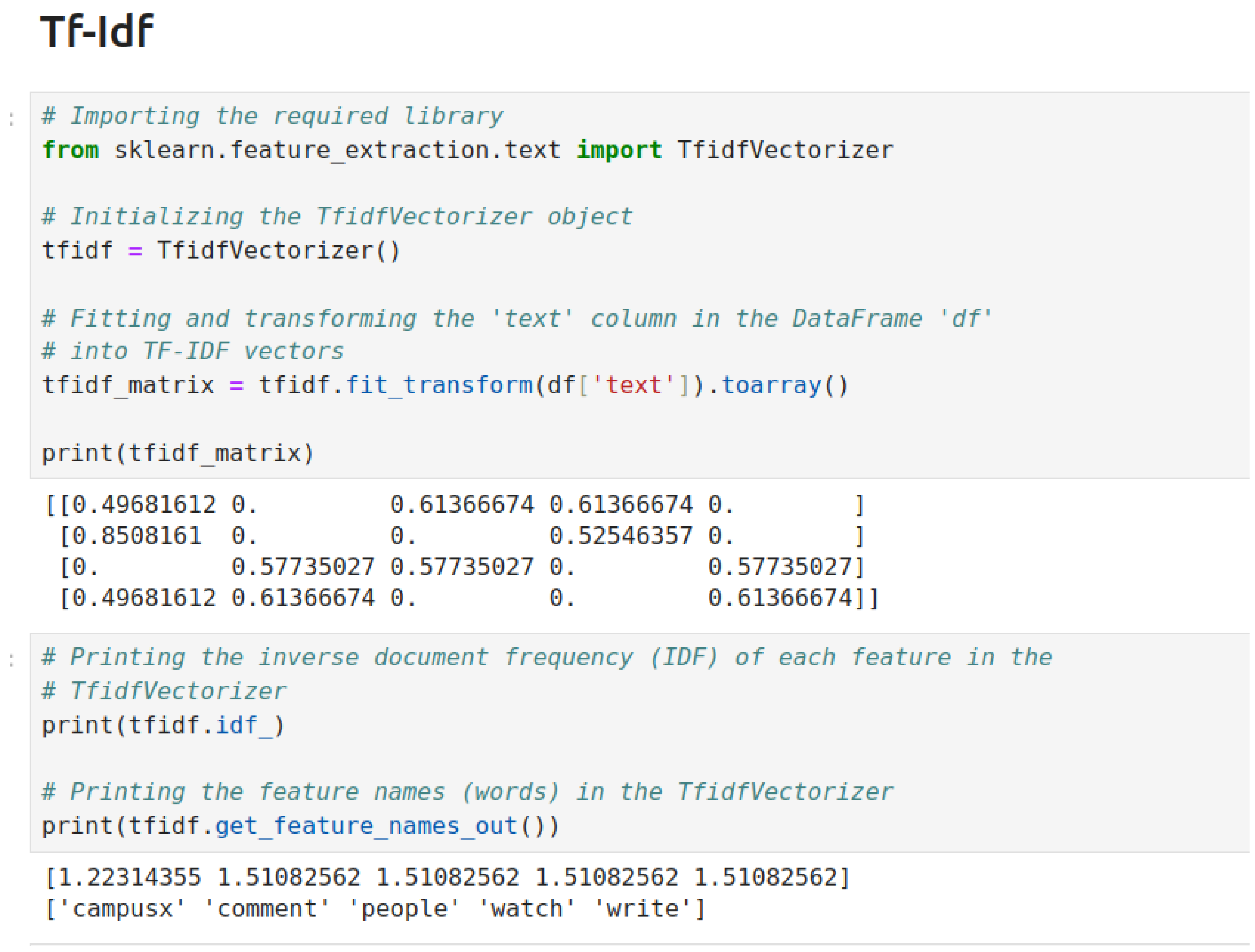
IDF

Term	IDF
people	log(4/2)
watch	log(4/2)
campusx	log(4/3)
write	log(4/2)
comment	log(4/2)

Tf-Idf

Term / Doc	D1	D2	D3	D4
people	1/3 * log(4/2)	0 * log(4/2)	1/3 * log(4/2)	0 * log(4/2)
watch	1/3 * log(4/2)	1/3 * log(4/2)	0 * log(4/2)	0 * log(4/2)
campusx	1/3 * log(4/3)	2/3 * log(4/3)	0 * log(4/3)	1/3 * log(4/3)
write	0 * log(4/2)	0 * log(4/2)	1/3 * log(4/2)	1/3 * log(4/2)
comment	0 * log(4/2)	0 * log(4/2)	1/3 * log(4/2)	1/3 * log(4/2)

Python implementation



Tf-Idf

```
# Importing the required library
from sklearn.feature_extraction.text import TfidfVectorizer

# Initializing the TfidfVectorizer object
tfidf = TfidfVectorizer()

# Fitting and transforming the 'text' column in the DataFrame 'df'
# into TF-IDF vectors
tfidf_matrix = tfidf.fit_transform(df['text']).toarray()

print(tfidf_matrix)
```

```
[[0.49681612 0.         0.61366674 0.61366674 0.        ]
 [0.8508161  0.         0.         0.52546357 0.        ]
 [0.         0.57735027 0.57735027 0.         0.57735027]
 [0.49681612 0.61366674 0.         0.         0.61366674]]
```

```
# Printing the inverse document frequency (IDF) of each feature in the
# TfidfVectorizer
print(tfidf.idf_)

# Printing the feature names (words) in the TfidfVectorizer
print(tfidf.get_feature_names_out())
```

```
[1.22314355 1.51082562 1.51082562 1.51082562 1.51082562]
['campusx' 'comment' 'people' 'watch' 'write']
```

Note that scikit-learn (sklearn) uses a slightly different formula to calculate TF-IDF compared to the standard formula presented earlier.

In scikit-learn, the formula for TF-IDF is as follows:

$$TF - IDF = (TF + 1) * log((N + 1) / (DF + 1)) + 1$$

Why do we use log in IDF ?

We use the logarithm in the Inverse Document Frequency (IDF) calculation for Term Frequency-Inverse Document Frequency (TF-IDF) in Natural Language Processing (NLP) to dampen the scale of IDF scores and improve the representation of term importance.

The main reasons for using the logarithm in IDF are as follows:

- Manage Scale: The IDF values can have a wide range, especially for terms that appear frequently in the document collection (high document frequency). By taking the logarithm, we compress the range of IDF scores, making them more manageable and preventing very large values.
- Equalize Term Weights: In a large document collection, some terms may appear frequently in many documents (high document frequency), and their raw IDF scores can be significantly higher than those of rare terms. Using the logarithm equalizes the weights of common and rare terms, ensuring that neither category dominates the TF-IDF calculation.
- Penalize Common Terms: Common words (e.g., "the," "and," "is") that appear in many documents are usually less informative for distinguishing between documents. By applying the logarithm, we reduce the IDF for such common terms, making their contribution to the TF-IDF score less significant.

Advantages of Tf-Idf

- Term importance highlighting.
- Dimensionality reduction with sparse vectors.
- Effective for text classification tasks.
- Useful in information retrieval systems.

Disadvantages of Tf-Idf

- Ignores word order and semantics.
- Limited context understanding.
- Doesn't handle out-of-vocabulary words.
- Sensitive to document length variations.

Custom Features

Creating custom features in NLP involves manually engineering additional features from the raw text data to enhance the performance of machine learning models. These features are designed to capture specific linguistic, semantic, or syntactic characteristics of the text, enabling the model to learn more effectively.

Examples of custom feature are :

- # of positive words
- # of negative words
- Ratio of positive to negative words
- Word count
- Character count