Word representation

#2

Introduction

- **Word representation** refers to the process of converting words (which are symbolic and non-numerical) into a numerical format that computers can understand and process.
- The goal of word representation is not just to assign a unique number to each word, but to capture its meaning, relationships with other words, and context in a way that is useful for various NLP tasks.

Word representation

- Broadly classified as:
  - Sparse (or Localist) Representations
  - Dense (or Distributed) Representations / Word Embeddings
  - Contextualized Word Embeddings

1. Sparse (or Localist) Representations:

- Presents words as high-dimensional vectors where most values are zero. Each dimension typically corresponds to a unique word in the vocabulary.

    a. One-hot encoding
    b. Bag-of-Words (BoW)
    c. TF-IDF (Term Frequency-Inverse Document Frequency)

a. One-Hot Encoding:

- Each word is represented by a vector with a length equal to the size of the vocabulary.

  A '1' is placed at the index corresponding to that word, and all other elements are ‘0’

- Example

  Corpus (Collection of Text): "The quick brown fox jumps over the lazy dog."

  - **1.Tokenization:** First, the text is broken down into individual words or "tokens." This typically involves:
    - **Lowercasing:** Converting all words to lowercase to treat "The" and "the" as the same word.
    - **Punctuation removal:** Removing punctuation marks.
  - After tokenization and lowercasing: ["the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]

2. Vocabulary Creation: Next, create a unique list of all words (tokens) present in your corpus. This list forms your "vocabulary."

Vocabulary: ["the", "quick", "brown", "fox", "jumps", "over", "lazy", "dog"]

3. Assigning Indices to Words: Assign a unique integer index to each word in your vocabulary. This index will correspond to the position where the '1' will appear in the one-hot vector.

```
"the": 0
"quick": 1
"brown": 2
"fox": 3
"jumps": 4
"over": 5
"lazy": 6
"dog": 7
```

- **4. Creating One-Hot Vectors:** Now, for each word in your original tokenized text, create a vector of zeros with a length equal to the size of your vocabulary (in this case, 8). Then, place a '1' at the index corresponding to that word.
    - **"the"**: `[1, 0, 0, 0, 0, 0, 0, 0]` (1 at index 0)
    - **"quick"**: `[0, 1, 0, 0, 0, 0, 0, 0]` (1 at index 1)
    - **"brown"**: `[0, 0, 1, 0, 0, 0, 0, 0]` (1 at index 2)
    - **"fox"**: `[0, 0, 0, 1, 0, 0, 0, 0]` (1 at index 3)
    - **"jumps"**: `[0, 0, 0, 0, 1, 0, 0, 0]` (1 at index 4)
    - **"over"**: `[0, 0, 0, 0, 0, 1, 0, 0]` (1 at index 5)
    - **"the"**: `[1, 0, 0, 0, 0, 0, 0, 0]` (1 at index 0, again)
    - **"lazy"**: `[0, 0, 0, 0, 0, 0, 1, 0]` (1 at index 6)
    - **"dog"**: `[0, 0, 0, 0, 0, 0, 0, 1]` (1 at index 7)

The entire sentence would then be represented as a sequence of these one-hot vectors.

Exercise

- Doc 1: new home sales top forecasts
- Doc 2: home sales rise in july
- Doc 3: increase in home sales in july
- Doc 4: july new home sales rise

Forecasts: 0 – [1,0,0,0,0,0,0,0,0]

Home: 1– [0,1,0,0,0,0,0,0,0]

….

Find the representation for Doc3 (Include stop words)

Advantages of One-Hot Encoding in NLP

- **No Implied Ordinality:** It avoids implying any numerical relationship or order between words. For example, if you assigned "cat" as 1 and "dog" as 2, a machine learning model might mistakenly interpret "dog" as being "greater" than "cat." One-hot encoding treats each word as independent.
- **Simple to Understand and Implement:** The concept is straightforward, and it's relatively easy to implement using libraries like Pandas or Scikit-learn in Python.

Disadvantages of One-Hot Encoding in NLP

- **High Dimensionality (Curse of Dimensionality):** As the vocabulary size grows (which happens very quickly in real-world text corpora), the length of each one-hot vector increases dramatically. A vocabulary of 100,000 unique words would result in vectors of length 100,000, most of which are zeros. This leads to:
  - **Increased Memory Usage:** Storing large, sparse matrices requires significant memory.
  - **Increased Computational Cost:** Training models on such high-dimensional data can be computationally expensive and slow.
  - **Sparsity:** The vast majority of values in the vectors are zero, leading to "sparse" data.
- **No Semantic Relationship:** One-hot encoding doesn't capture any semantic similarity or relationship between words. "King" and "Queen" are just as different as "King" and "Apple" in this representation, even though "King" and "Queen" are semantically related. This is a major limitation for many advanced NLP tasks.
- **"Out-of-Vocabulary" (OOV) Words:** If a new word appears in the test data that wasn't in the training data's vocabulary, it cannot be represented by the existing one-hot encoding scheme, leading to an "out-of-vocabulary" problem.

b. Bag of Words (BoW)

- Represents a document (or sentence) as a vector of word counts. The order of words is disregarded. Each dimension corresponds to a word in the vocabulary, and its value is the frequency of that word in the document.
  - **Pros:** Simple and effective for tasks like text classification where word order isn't critical.
  - **Cons:** Loses all information about word order and grammatical structure. Suffers from high dimensionality. Doesn't capture semantic meaning beyond simple co-occurrence.

- Example:
  - Consider the corpus consisting of 3 sentences
    - "The quick brown fox jumps over the lazy dog."
    - "The dog is lazy."
    - "A quick fox and a lazy dog."

- **Step 1: Text Preprocessing**
  - **Tokenization**
  - **Lowercasing**
  - **Removing Punctuation**
  - **Removing Stop Words (Optional)**
  - **Stemming or Lemmatization (Optional)**

- Let us apply tokenization, lowercasing, and punctuation removal to our example sentences:
  - ["the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]
  - ["the", "dog", "is", "lazy"]
  - ["a", "quick", "fox", "and", "a", "lazy", "dog"]
- *(For simplicity in this example, we won't remove stop words, but in real applications, it's often done.)*

Step 2: Create the Vocabulary (Unique Words)

- Collect all **unique words** from the entire preprocessed corpus. This forms the vocabulary, and the size of this vocabulary will be the number of features (dimensions) in your BoW vectors.
- From the example sentences, the unique words are:

  ["the", "quick", "brown", "fox", "jumps", "over", "lazy", "dog", "is", "a", "and"]
- Now assign an arbitrary index to each word in the vocabulary (the order doesn't inherently matter, but it is consistent once established):

```
"the": 0
"quick": 1
"brown": 2
"fox": 3
"jumps": 4
"over": 5
"lazy": 6
"dog": 7
"is": 8
"a": 9
"and": 10
```

Vocabulary Size (Number of Features): 11

Step 3: Generate the Bag-of-Words Vector for Each Document

- Now, for each document (sentence) in the corpus, create a vector where each element represents the count of a word from the vocabulary in that document.
  - The length of each vector will be equal to the vocabulary size (11 in our case).
  - The value at each position i in the vector represents the frequency (count) of the word corresponding to index i in the vocabulary.
- **Document 1: "The quick brown fox jumps over the lazy dog."**
  - Preprocessed: ["the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]
- Counts:
  - "the": 2
  - "quick": 1
  - "brown": 1
  - "fox": 1
  - "jumps": 1
  - "over": 1
  - "lazy": 1
  - "dog": 1
  - "is": 0
  - "a": 0
  - "and": 0

BoW Vector for Document 1: [2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]

Document 2: "The dog is lazy."

- Preprocessed: ["the", "dog", "is", "lazy"]
- Counts:
  - "the": 1
  - "dog": 1
  - "is": 1
  - "lazy": 1
  - All others: 0

BoW Vector for Document 2: [1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0]

Document 3: "A quick fox and a lazy dog."

- Preprocessed: ["a", "quick", "fox", "and", "a", "lazy", "dog"]
- Counts:
  - "a": 2
  - "quick": 1
  - "fox": 1
  - "and": 1
  - "lazy": 1
  - "dog": 1
  - All others: 0

BoW Vector for Document 3: [0, 1, 0, 1, 0, 0, 1, 1, 0, 2, 1]

- **Resulting Representation:**
  - The entire corpus is now represented as a matrix where each row is the BoW vector for a document:

```
Document 1: [2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
Document 2: [1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0]
Document 3: [0, 1, 0, 1, 0, 0, 1, 1, 0, 2, 1]
```

This matrix is the "Bag-of-Words" representation of the corpus

The entire corpus is now represented as a matrix where each row is the BoW vector for a document

Alternative: Arrange the words in sorted order
Easy to index

Word	Doc1	Doc2	Doc3
a	0	0	2
and	0	0	1
brown	1	0	0
dog	1	1	1
fox	1	0	1
is	0	1	0
jumps	1	0	0
lazy	1	1	1
over	1	0	0
quick	1	0	1
the	2	1	0

Question: for the query “lazy fox”, retrieve the documents according to similarity.

$$\text{cosine\_similarity} = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

A = [0, 0, 0, 1, 1, 1, 1, 0]
B = [1, 1, 1, 0, 1, 2, 1, 1]

1. Dot Product (A · B)

$$\begin{aligned} A \cdot B &= (0 \times 1) + (0 \times 1) + (0 \times 1) + (1 \times 0) + (1 \times 1) + (1 \times 2) + (1 \times 1) + (0 \times 1) \\ &= 0 + 0 + 0 + 0 + 1 + 2 + 1 + 0 = 4 \end{aligned}$$

2. Norms

$$\|A\| = \sqrt{0^2 + 0^2 + 0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2} = \sqrt{4} = 2$$

$$\|B\| = \sqrt{1^2 + 1^2 + 1^2 + 0^2 + 1^2 + 2^2 + 1^2 + 1^2} = \sqrt{1 + 1 + 1 + 0 + 1 + 4 + 1 + 1} = \sqrt{10} \approx 3.1623$$

3. Cosine Similarity

$$\text{cosine\_similarity} = \frac{4}{2 \times 3.1623} \approx \frac{4}{6.3246} \approx 0.632$$

Advantages of BoW:

- **Simplicity:** Easy to understand and implement.
- **Effectiveness:** Surprisingly effective for many text classification tasks, especially when large amounts of data are available.

Disadvantages of BoW:

- **Loses Word Order/Context:** The biggest limitation is that it completely disregards the grammatical structure and the order of words. "Dog bites man" and "Man bites dog" would have identical BoW representations.
- **High Dimensionality:** For large vocabularies, the feature vectors become very long and sparse (mostly zeros), leading to the "curse of dimensionality." This can lead to high memory usage and increased computational time for training models.
- **Semantic Blindness:** It doesn't capture any semantic relationships between words. "Good" and "Excellent" are treated as completely independent words.

Despite its simplicity, BoW (often with TF-IDF) remains a baseline and sometimes surprisingly effective model in many NLP applications, particularly in traditional machine learning settings.

Numerical Problems:

- Corpus 1
  - Doc1: Machine learning is a subset of artificial intelligence
  - Doc2: Deep learning is a powerful tool in machine learning
  - Doc3: Artificial intelligence enables machines to mimic human behavior
  - Doc4: Neural networks are used in deep learning models
- Question: Find the similarity between Doc1 and Doc3

- Corpus 2
  - Doc 1: new home sales top forecasts
  - Doc 2: home sales rise in july
  - Doc 3: increase in home sales in july
  - Doc 4: july new home sales rise
- Question: Retrieve the documents for the query “home sales in july”

c. TF-IDF (Term Frequency-Inverse Document Frequency)

- TF-IDF is a numerical statistic that reflects how important a word is to a document in a collection or corpus of documents.
- It is widely used as a weighting factor in information retrieval and text mining.
- The core idea behind TF-IDF is that:
  - Words that appear frequently in a **specific document** are probably important to that document (Term Frequency).
  - Words that appear frequently across **many documents** in the *entire corpus* are probably less distinctive and thus less important for differentiating one document from another (Inverse Document Frequency).
- By multiplying these two components, TF-IDF assigns a higher score to words that are both frequent in a particular document *and* rare across the entire set of documents.

Computation

For a term t in a document d in a corpus D:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

---

1. Term Frequency (TF):

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in } d}{\text{Total number of terms in } d}$$

It represents how often a term appears in a document.

---

2. Inverse Document Frequency (IDF):

$$\text{IDF}(t, D) = \log\left(\frac{N}{1 + \text{DF}(t)}\right)$$

Where:

- N = Total number of documents in the corpus
- $\text{DF}(t)$ = Number of documents containing the term t
- The "+1" in the denominator avoids division by zero (smoothing).

IDF measures how important a term is across the entire corpus. It down-weights terms that appear very frequently across many documents (like "the", "a", "is") and boosts the importance of terms that are rare and thus more distinctive.

Consider a term in the query that is rare in the collection (e.g., *arachnophobia*)
A document containing this term is very likely to be relevant to the query *arachnophobia*
→ We want a high weight for rare terms like *arachnophobia*.

3. TF-IDF Score

Finally, the TF-IDF score for a term `t` in a document `d` is calculated by multiplying its TF and IDF scores:

$$TF - IDF(t, d) = TF(t, d) \times IDF(t)$$

Example

Documents:

- **Doc1:** "The quick brown fox jumps over the lazy dog"
- **Doc2:** "The dog is lazy"
- **Doc3:** "A quick fox and a lazy dog"

Vocabulary:
["a", "and", "brown", "dog", "fox", "is", "jumps", "lazy", "over", "quick", "the"]

Word	Doc1	Doc2	Doc3	DF (in how many docs it appears)
a	0	0	2	1
and	0	0	1	1
brown	1	0	0	1
dog	1	1	1	3
fox	1	0	1	2
is	0	1	0	1
jumps	1	0	0	1
lazy	1	1	1	3
over	1	0	0	1
quick	1	0	1	2
the	2	1	0	2

Compute IDF

$$\mathrm{IDF}(t) = \log\left(\frac{3}{1 + \mathrm{DF}(t)}\right)$$

Word	DF	IDF
a	1	log(3 / 2) ≈ 0.1761
and	1	log(3 / 2) ≈ 0.1761
brown	1	log(3 / 2) ≈ 0.1761
dog	3	log(3 / 4) ≈ -0.1249
fox	2	log(3 / 3) = 0
is	1	log(3 / 2) ≈ 0.1761
jumps	1	log(3 / 2) ≈ 0.1761
lazy	3	log(3 / 4) ≈ -0.1249
over	1	log(3 / 2) ≈ 0.1761
quick	2	log(3 / 3) = 0
the	2	log(3 / 3) = 0

Compute TF for each word in each document

Word counts per document:

- Doc1: 9 words
- Doc2: 4 words
- Doc3: 7 words

For example:

- TF(“the”, Doc1) = 2 / 9 ≈ 0.222
- TF(“dog”, Doc2) = 1 / 4 = 0.25

Final TF-IDF Matrix

Word	**Doc1**	**Doc2**	**Doc3**
a	0	0	(2/7)*0.1761 ≈ **0.0503**
and	0	0	(1/7)*0.1761 ≈ **0.0252**
brown	(1/9)*0.1761 ≈ **0.0196**	0	0
dog	(1/9)*(-0.1249) ≈ **-0.0139**	(1/4)*(-0.1249) ≈ **-0.0312**	(1/7)*(-0.1249) ≈ **-0.0178**
fox	(1/9)*0 = 0	0	(1/7)*0 = 0
is	0	(1/4)*0.1761 ≈ **0.0440**	0
jumps	(1/9)*0.1761 ≈ **0.0196**	0	0
lazy	(1/9)*(-0.1249) ≈ **-0.0139**	-0.0312	-0.0178
over	(1/9)*0.1761 ≈ **0.0196**	0	0
quick	0	0	0
the	(2/9)*0 = 0	(1/4)*0 = 0	0