Lecture 10

One-hot encoding and Word Count

Course Outcome:

1. Apply One-Hot Encoding and Bag of Words to a small dataset.
2. Analyze the Strengths and Limitations of OHE & BoW

Introduction- One-hot encoding and Word Count

Both One-Hot Encoding (OHE) and Word Count (Bag of Words - BOW) are fundamental text representation techniques in Natural Language Processing (NLP).

One-hot encoding

One-Hot Encoding is a technique used in Natural Language Processing (NLP) and Machine Learning to represent categorical data as binary vectors.

📌 Key Idea:

- Each unique word (or category) is represented as a vector of 0s and 1s.
- The word's position in the vocabulary (vocab) is marked as 1, while all other positions remain 0.
- Used in text processing, sentiment analysis, deep learning (Neural Networks, RNNs, Transformers).

Question 1. Given a set of sentences, how do we represent words numerically so that a computer can process them?

Think: What if we assign each word a unique identifier?

Ans: step 1: Create the Vocabulary of Unique Words List:

Sample Dataset

Let's assume we have the following sentences in our dataset:

1. "The phone is excellent."
2. "The battery performance is excellent."
3. "The phone has excellent battery life."

Index	Word	One-Hot Encoding
0	The	[1, 0, 0, 0, 0, 0, 0, 0]
1	phone	[0, 1, 0, 0, 0, 0, 0, 0]
2	is	[0, 0, 1, 0, 0, 0, 0, 0]
3	excellent	[0, 0, 0, 1, 0, 0, 0, 0]
4	battery	[0, 0, 0, 0, 1, 0, 0, 0]
5	performance	[0, 0, 0, 0, 0, 1, 0, 0]
6	has	[0, 0, 0, 0, 0, 0, 1, 0]
7	life	[0, 0, 0, 0, 0, 0, 0, 1]

Step 1: Create the Vocabulary

Extract unique words across all text samples

Vocab: ["The", "phone", "is", "excellent", "battery", "performance", "has", "life"]

Question 2 : How can we represent the first review "The phone is excellent"

Answer:

📌 Each word is assigned a unique vector with 1 where the word appears and 0 elsewhere.

Sentence 2: The battery performance is excellent.

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

Sentence 3: The phone has excellent battery life.

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

Question 2: Can you come up with a different Approach?

🔷 Example: Sentence-Level One-Hot Encoding

📌 **Sentences:** 1️⃣ *"The phone is excellent."*

2️⃣ *"The battery performance is excellent."*

3️⃣ *"The phone has excellent battery life."*

✅ **Step 1: Create a Vocabulary**

👉 Extract **unique words** from all sentences:

📌 **Vocab:** `["The", "phone", "is", "excellent", "battery", "performance", "has", "life"]`

✅ **Step 2: Convert Sentences into One-Hot Encoding Vectors**

Sentence	The	phone	is	excellent	battery	performance	has	life
"The phone is excellent."	1	1	1	1	0	0	0	0
"The battery performance is excellent."	1	0	1	1	1	1	0	0
"The phone has excellent battery life."	1	1	0	1	1	0	1	1

Analogy for One-Hot Encoding: The Word Pool and Vocabulary Selection

🔹 Imagine a Pool of Words (Vocabulary)

Think of all the words in a language as a huge pool of water.

- This pool contains millions of words—both useful and irrelevant for our task.
- But we don't need all words, so we create a small vocabulary (subset of words) that is important for our task.

✅ Example: If we want to analyze sentiment in restaurant reviews, we only select words like: ["delicious", "amazing", "terrible", "bad", "service", "slow"]
🚫 We ignore words like "chair", "table", "fork" because they don't affect sentiment.

🔹 One-Hot Encoding: The Selected Words Become Unique Identifiers

📌 Now, each word in our vocabulary gets a unique identity (vector)—like assigning it a seat in a stadium.

✅ Example: Suppose our vocabulary is:
👉 ["delicious", "amazing", "bad", "terrible", "slow", "great"]

Now, we encode each word as a binary vector where:

- 1 means the word is present.
- 0 means it is absent.

✅ **Sentence-level One-Hot Encoding:** Used for tasks like **text classification and sentiment analysis** (treats sentence as a whole).

✅ **Word-level One-Hot Encoding:** Used for tasks like **word embeddings, RNNs, and sequence modeling** (treats words individually).

Limitations of one-hot encoding?

🚨 Limitations: Why One-Hot Encoding is Being Replaced?

Problem	Why It's an Issue?	Better Alternative
High Dimensionality	If we have 10,000 words, we need a **10,000-column matrix**	✅ Word Embeddings (Word2Vec, BERT)
Doesn't Capture Meaning	"Great" and "Awesome" are treated as separate, unrelated words	✅ Embeddings (Vectors capture meaning)
Memory Inefficiency	Mostly **sparse** (many 0s), wasting memory	✅ Compressed Embedding Models

1. **High Dimensionality (Curse of Dimensionality):** A vocabulary of 10,000 words requires each word to be represented by a 10,000-dimensional vector, with only one active element (1), leading to wasted memory.
2. **Lack of Semantic Meaning:** One-hot encoding treats all words as independent entities without any notion of similarity. Example: The words "**King**" and "**Queen**" would have completely different one-hot vectors, even though they are
semantically related.
3. **Not Scalable for Large Datasets:** In real-world NLP tasks (e.g., training GPT models), vocabularies can reach millions of words, making one-hot encoding impractical.
4. **No Context Awareness:** One-hot encoding does not consider word meaning or word usage in different contexts.
Example:

Sentence 1: "He went to the bank to withdraw money."
Sentence 2: "She sat by the bank of the river."
The word "bank" has different meanings but is treated identically in one-hot encoding.

Applications of One hot encoding:

Used in Text Processing before Moving to Advanced Embeddings

- Before deep learning, One-Hot Encoding and word count was a standard way to represent words for machine learning models.
- Many datasets contain categorical features (e.g., "Yes/No", "Red/Blue/Green"). One-Hot Encoding converts these categories into numbers so that machine learning models can process them.

Code of OHE:

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Given words in reviews
vocab = np.array(["the", "phone", "is", "great", "battery", "bad", "has", "good", "life"]).reshape(-1,
1)

# Initialize One-Hot Encoder
encoder = OneHotEncoder(sparse=False)

# Apply One-Hot Encoding
one_hot_encoded = encoder.fit_transform(vocab)

# Display Encoded Vectors
for word, encoding in zip(vocab.flatten(), one_hot_encoded):
    print(f"{word}: {encoding}")
```

Word Count

Word count is a basic text processing technique used to determine **how many times each word appears in a given document or text corpus**.

1️⃣ Basic Word Count Example

Let's take a simple **sentence**:

💬 **Sentence:**
📌 *"The food was good, and the service was good too."*

🔷 **Step 1: Tokenize (Split) Words**
We remove punctuation and split the sentence into words:

`["the", "food", "was", "good", "and", "the", "service", "was", "good", "too"]`

🔷 **Step 2: Count Word Occurrences**

Word	Count
the	2
food	1
was	2
good	2
and	1
service	1
too	1

✅ **Final Word Count Representation:**

`{"the": 2, "food": 1, "was": 2, "good": 2, "and": 1, "service": 1, "too": 1}`

Review is represented as:

[2,1,2,2,1,2,1,2,2,1]

Applications of Bag of Words (BOW):

1. **Text Classification & Sentiment Analysis**

✅ **BoW is used to classify text into categories** such as spam detection, fake news detection, or sentiment analysis.

💡 **Example:**
🔹 **Task:** Classify movie reviews as **positive or negative**.
🔹 **Dataset:**

- **Review 1:** `"The movie was excellent, I loved it!"` ✅ (Positive)
- **Review 2:** `"Worst movie ever, complete waste of time!"` ❌ (Negative)

📌 **How BoW Helps?**

- Words like **"excellent", "loved"** appear more in **positive** reviews.
- Words like **"worst", "waste"** appear more in **negative** reviews.
- **ML models (Naïve Bayes, Logistic Regression, SVM)** use these BoW features to classify reviews.

2. **Word Cloud Example Using Bag of Words (BoW) in Python**

🚀 **What is a Word Cloud?**
A **Word Cloud** is a **visual representation of text data**, where the size of each word represents its **frequency or importance**. It is still widely used in:

✅ **Customer Feedback Analysis** (Amazon, Flipkart, Google Reviews)
✅ **Social Media Analysis** (Twitter trends, YouTube comments)
✅ **News Analytics** (Finding trending topics in news articles)
✅ **SEO & Content Marketing** (Finding frequently used keywords)

Limitations of BOW?

❌ 1. Ignores Word Order (Loses Context)

📌 **Problem:**
BoW does not consider **word sequence**, so `"not happy"` and `"happy not"` are treated **the same**, even though they have different meanings.

💡 **Example:**

- **Sentence 1:** `"The movie was not good."` ❌ (Negative)
- **Sentence 2:** `"The movie was good."` ✅ (Positive)

🚀 **Why is this bad?**

- Both sentences may have **similar BoW word counts** (`"movie"`, `"was"`, `"good"`), but the **first one is negative while the second is positive**.

✅ **Solution:** Use **N-Grams (bigrams, trigrams) or Transformers** to preserve word order.

❌ 2. High Dimensionality & Sparsity

📌 **Problem:**
BoW creates **huge sparse matrices** because each document is represented as a **vector with as many features as the vocabulary size**.

💡 **Example:**

- If the vocabulary size = **100,000 words**, then each document will be a **100,000-dimensional vector**, mostly filled with **zeros**.

🚀 **Why is this bad?**

- **Consumes a lot of memory**.
- **Slows down training and prediction**.

✅ **Solution:** Use **TF-IDF to reduce feature space** or **dimensionality reduction techniques like PCA**.

❌ 3. Cannot Capture Word Meaning or Synonyms

📌 **Problem:**
BoW **treats synonyms as separate features**, ignoring the fact that `"amazing"`, `"great"`, and `"fantastic"` all have similar meanings.

💡 **Example:**

- `"The food was amazing!"` → `["food", "was", "amazing"]`
- `"The food was great!"` → `["food", "was", "great"]`

🚀 **Why is this bad?**

- `"amazing"` and `"great"` should contribute **similarly to sentiment analysis**, but BoW treats them as completely **different words**.

✅ **Solution:** Use **Word Embeddings (Word2Vec, GloVe, BERT)** that learn word relationships.

❌ 4. Fails for Long Documents & Real-Time Applications

📌 **Problem:**

- Longer documents naturally have **higher word counts**, leading to **biased models**.

- BoW models need **retraining every time new data is added**, making it **inefficient for real-time NLP**.

💡 **Example:**

- A **500-word blog post** will have much higher word counts than a **10-word customer review**, even if both talk about **the same product**.

🚀 **Why is this bad?**

- BoW models **favor longer documents** because they have **more words**, even if the sentiment or topic is the same.

✅ **Solution:**

- Use **TF-IDF (which normalizes word importance instead of raw count)**.
- Use **Neural Networks with Embeddings** instead of BoW for large-scale NLP.

📌 Summary: Comparing OHE & BoW Limitations

Limitation	One-Hot Encoding (OHE)	Bag of Words (BoW)
❌ **High Dimensionality**	Needs large binary vectors	Needs large word count vectors
❌ **Ignores Word Order**	No sequence information	No sequence information
❌ **Does Not Capture Meaning**	`"great"` and `"excellent"` treated separately	`"happy"` and `"joyful"` treated separately
❌ **Fails with Out-of-Vocabulary Words**	Cannot handle new words	Cannot handle new words
❌ **Not Suitable for Large Datasets**	Huge sparse matrices	High memory consumption
❌ **Poor Performance on Long Texts**	All words have equal weight	Longer texts dominate shorter ones

✅ **Better Alternatives:**

- **Word Embeddings (Word2Vec, FastText, BERT)** → Capture meaning & context.
- **TF-IDF** → Reduces high dimensionality & normalizes word importance.
- **Transformers (BERT, GPT-4)** → Handle word relationships, context, and meaning.

Code of BOW

https://colab.research.google.com/drive/1puXZaq6QKGMy1th3CqsDwZQcr7PMom26#scrollTo=3rv5J3eR4B3N