3.One-Hot Encoding: Converting Text to Vectors

Introduction

One-hot encoding is a fundamental technique for converting textual data into numerical vector representations. This method creates sparse binary vectors where each word in the vocabulary is represented by a unique vector.

Note on Text Preprocessing: In practice, text preprocessing steps like stop word removal, lowercasing, and punctuation removal are typically performed before one-hot encoding. This document demonstrates both approaches to illustrate the impact of preprocessing on the resulting vectors.

The Process

Step 1: Build the Vocabulary

Given a corpus (collection of documents), we first extract all unique words to form our vocabulary **V**.

Example Corpus:

- Document 1 (D_1): "the food is good"
- Document 2 (D_2): "the food is bad"
- Document 3 (D_3): "pizza is amazing"

Two Approaches: With and Without Stop Words

Approach A: WITHOUT Stop Word Removal (Original Text)

Vocabulary V: {the, food, is, good, bad, pizza, amazing}

The vocabulary size is:

$$|V| = 7$$

where $|V|$ denotes the total number of unique words in the vocabulary.

Position Mappings:

$$V = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$$

where:

- w_1 = "the" (position 1)
- w_2 = "food" (position 2)
- w_3 = "is" (position 3)
- w_4 = "good" (position 4)
- w_5 = "bad" (position 5)
- w_6 = "pizza" (position 6)
- w_7 = "amazing" (position 7)

Approach B: WITH Stop Word Removal

Stop words identified: {the, is}

Cleaned Corpus:

- Document 1 (D_1): "food good"
- Document 2 (D_2): "food bad"
- Document 3 (D_3): "pizza amazing"

Vocabulary V: {food, good, bad, pizza, amazing}

The vocabulary size is:

$$|V| = 5$$

Position Mappings:

$$V = \{w_1, w_2, w_3, w_4, w_5\}$$

where:

- w_1 = "food" (position 1)
- w_2 = "good" (position 2)
- w_3 = "bad" (position 3)
- w_4 = "pizza" (position 4)
- w_5 = "amazing" (position 5)

Key Observation: Removing stop words reduced vocabulary size from 7 to 5 (28.6% reduction), which decreases vector dimensionality and improves efficiency.

Vector Representation

For any word w_i in the vocabulary, its one-hot encoded vector is defined as:

$$\vec{v}_{w_i} = [0, 0, \ldots, 1, \ldots, 0] \in \mathbb{R}^{|V|}$$

where:

- $\vec{v}_{w_i}$ is the vector representation of word w_i
- The vector has dimension $|V|$ (vocabulary size)
- The i-th position contains 1
- All other positions contain 0
- $\mathbb{R}^{|V|}$ indicates the vector space with dimension equal to vocabulary size

Mathematical Formula:

$$\vec{v}_{w_i}[j] = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{if } j \neq i \end{cases}$$

where:

- $\vec{v}_{w_i}[j]$ is the j-th element of the vector for word w_i
- j ranges from 1 to $|V|$
- i is the position index of the word in the vocabulary

Detailed Examples: Comparing Both Approaches

Approach A: WITHOUT Stop Word Removal

Encoding "the"

Word "the" is at position 1, so:

$$\vec{v}_{\text{the}} = [1, 0, 0, 0, 0, 0, 0]$$

Encoding "food"

Word "food" is at position 2, so:

$$\vec{v}_{\text{food}} = [0, 1, 0, 0, 0, 0, 0]$$

Encoding "is"

Word "is" is at position 3, so:

$$\vec{v}_{\text{is}} = [0, 0, 1, 0, 0, 0, 0]$$

Encoding "good"

Word "good" is at position 4, so:

$$\vec{v}_{\text{good}} = [0, 0, 0, 1, 0, 0, 0]$$

Approach B: WITH Stop Word Removal

Encoding "food"

Word "food" is at position 1, so:

$$\vec{v}_{\text{food}} = [1, 0, 0, 0, 0]$$

Encoding "good"

Word "good" is at position 2, so:

$$\vec{v}_{\text{good}} = [0, 1, 0, 0, 0]$$

Encoding "bad"

Word "bad" is at position 3, so:

$$\vec{v}_{\text{bad}} = [0, 0, 1, 0, 0]$$

Encoding "pizza"

Word "pizza" is at position 4, so:

$$\vec{v}_{\text{pizza}} = [0, 0, 0, 1, 0]$$

Note: Vectors are now 5-dimensional instead of 7-dimensional, reducing memory and computation requirements.

Document Representation

A document containing n words is represented as a matrix by stacking the one-hot vectors of all its words:

$$D = \begin{bmatrix} \vec{v}_{w_1}^T \\ \vec{v}_{w_2}^T \\ \vdots \\ \vec{v}_{w_n}^T \end{bmatrix} \in \mathbb{R}^{n \times |V|}$$

where:

- D is the document matrix
- n is the number of words in the document
- $\vec{v}_{w_i}^T$ is the transposed one-hot vector of the i-th word
- The resulting matrix has shape $n \times |V|$

Approach A: WITHOUT Stop Word Removal

Document 1 - "the food is good"

$$D_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Shape of D_1: 4×7 (4 words, 7-dimensional vectors)

Document 2 - "the food is bad"

$$D_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Shape of D_2: 4×7

Document 3 - "pizza is amazing"

$$D_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Shape of D_3: 3×7

Total elements: $4 \times 7 + 4 \times 7 + 3 \times 7 = 77$ elements

Approach B: WITH Stop Word Removal

Document 1 - "food good"

$$D_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Shape of D_1: 2×5 (2 words, 5-dimensional vectors)

Document 2 - "food bad"

$$D_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Shape of D_2: 2×5

Document 3 - "pizza amazing"

$$D_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Shape of D_3: 2×5

Total elements: $2 \times 5 + 2 \times 5 + 2 \times 5 = 30$ elements

Efficiency Gain: Stop word removal reduced total matrix elements from 77 to 30 (61% reduction in memory usage for this small corpus).

Key Properties

1. Orthogonality

All one-hot vectors are orthogonal to each other:

$$\vec{v}_{w_i} \cdot \vec{v}_{w_j} = 0 \text{ for all } i \neq j$$

where:

- $\vec{v}_{w_i} \cdot \vec{v}_{w_j}$ denotes the dot product of two vectors

- The result is 0 when $i \neq j$, meaning the vectors are perpendicular

2. Unit Norm

Each one-hot vector has a norm (length) of 1:

$$\|\vec{v}_{w_i}\| = \sqrt{\sum_{j=1}^{|V|} \vec{v}_{w_i}[j]^2} = 1$$

where:

- $\|\vec{v}_{w_i}\|$ is the Euclidean norm of the vector
- The sum $\sum_{j=1}^{|V|} \vec{v}_{w_i}[j]^2$ equals 1 because only one element is 1 and the rest are 0

3. Sparsity

The sparsity of a one-hot vector is defined as:

$$\text{Sparsity} = \frac{|V| - 1}{|V|}$$

where:

- Only 1 out of $|V|$ positions is non-zero
- As $|V|$ increases, the vector becomes increasingly sparse

Approach A (without stop word removal): $|V| = 7$

$$\text{Sparsity} = \frac{7 - 1}{7} = \frac{6}{7} \approx 85.7\%$$

Approach B (with stop word removal): $|V| = 5$

$$\text{Sparsity} = \frac{5 - 1}{5} = \frac{4}{5} = 80\%$$

Improvement: Stop word removal reduced sparsity from 85.7% to 80%, though vectors remain highly sparse.

Impact of Stop Word Removal: Comparison Table

Metric	Without Stop Words	With Stop Words	Improvement
Vocabulary Size	7	5	28.6% reduction
Vector Dimension	7	5	28.6% reduction
Sparsity	85.7%	80%	5.7% improvement
Total Matrix Elements	77	30	61% reduction
Semantic Focus	Diluted by stop words	Concentrated on meaning	Better
Computational Cost	Higher	Lower	More efficient

Advantages of One-Hot Encoding

1. Easy to Implement

One-hot encoding is straightforward to implement using standard Python libraries:

In Scikit-learn:

```
from sklearn.preprocessing import OneHotEncoder
```

In Pandas:

```
import pandas as pd
pd.get_dummies(data)
```

The simplicity of implementation makes it an accessible starting point for text vectorization tasks.

Disadvantages of One-Hot Encoding

1. Sparse Matrix Problem

One-hot encoding creates sparse matrices (or sparse arrays) containing mostly zeros with very few ones.

Sparsity Ratio:

$$S = \frac{\text{number of zeros}}{\text{total elements}} = \frac{|V| - 1}{|V|}$$

where:

- S is the sparsity ratio
- $|V|$ is the vocabulary size
- For large vocabularies, $S \approx 1$ (nearly 100% sparse)

Example: For a vocabulary of 50,000 words:

$$S = \frac{50000 - 1}{50000} = \frac{49999}{50000} = 0.99998 = 99.998\%$$

Problem: Sparse matrices lead to **overfitting** in machine learning models, where:

- Training accuracy is high
- Test/new data accuracy is poor
- Model fails to generalize

Mitigation with Stop Word Removal: While stop word removal helps reduce sparsity somewhat, it doesn't fundamentally solve the problem. Even with preprocessing, real-world vocabularies remain large enough to create highly sparse representations.

2. Variable Input Size (Not Fixed-Length)

Machine learning algorithms require fixed-size feature vectors, but one-hot encoding produces variable-sized document representations.

For a document with n words:

$$\text{Shape of } D = n \times |V|$$

where:

- n varies across different documents
- $|V|$ is constant (vocabulary size)

Example from our corpus:

Without stop word removal:

- D_1 : "the food is good" $\rightarrow 4 \times 7$
- D_2 : "the food is bad" $\rightarrow 4 \times 7$
- D_3 : "pizza is amazing" $\rightarrow 3 \times 7$

With stop word removal:

- D_1 : "food good" $\rightarrow 2 \times 5$
- D_2 : "food bad" $\rightarrow 2 \times 5$
- D_3 : "pizza amazing" $\rightarrow 2 \times 5$

Problem: Cannot train ML models with variable input dimensions. All input samples must have identical shape for training.

Note: Stop word removal makes documents shorter and more uniform in length, but doesn't eliminate the variable size problem entirely.

3. No Semantic Meaning Captured

One-hot vectors fail to capture semantic relationships between words. All distinct words are equidistant from each other.

Euclidean Distance between any two different words:

$$d(\vec{v}_{w_i}, \vec{v}_{w_j}) = \sqrt{\sum_{k=1}^{|V|} (\vec{v}_{w_i}[k] - \vec{v}_{w_j}[k])^2} = \sqrt{2}$$

where:

- d is the Euclidean distance
- $\vec{v}_{w_i}[k]$ is the k-th element of word w_i's vector
- The distance is always $\sqrt{2}$ for any $i \neq j$

Cosine Similarity between any two different words:

$$\text{similarity}(\vec{v}_{w_i}, \vec{v}_{w_j}) = \frac{\vec{v}_{w_i} \cdot \vec{v}_{w_j}}{\|\vec{v}_{w_i}\|\|\vec{v}_{w_j}\|} = \frac{0}{1 \times 1} = 0$$

where:

- The numerator is the dot product (always 0 for different words)
- The denominator is the product of vector norms (always 1 × 1 = 1)
- Result is always 0, indicating no similarity

Example: Food-related words

Consider vocabulary: {food, pizza, burger}

- $\vec{v}_{\text{food}} = [1, 0, 0]$
- $\vec{v}_{\text{pizza}} = [0, 1, 0]$
- $\vec{v}_{\text{burger}} = [0, 0, 1]$

All pairwise distances are equal:

$$d(\text{food}, \text{pizza}) = d(\text{food}, \text{burger}) = d(\text{pizza}, \text{burger}) = \sqrt{2}$$

Visualization in 3D space:

- food at point (1, 0, 0)
- pizza at point (0, 1, 0)
- burger at point (0, 0, 1)

All three words are equidistant despite "pizza" and "burger" being more semantically similar to each other than to "food". The encoding cannot capture that:

- "pizza" and "burger" are both food items
- They should be closer to each other than to the generic term "food"
- Semantic relationships like synonymy, similarity, or relatedness are lost

Impact of Stop Word Removal: Removing stop words doesn't help with this problem. Semantic relationships remain completely absent regardless of preprocessing.

4. Out-of-Vocabulary (OOV) Problem

Words not present in the training vocabulary cannot be represented during testing or inference.

Training vocabulary: $V_{\text{train}} = \{w_1, w_2, \ldots, w_{|V|}\}$

Test data contains: $w_{\text{new}} \notin V_{\text{train}}$

Problem: $\vec{v}_{w_{\text{new}}}$ is undefined because w_{new} has no assigned position in the vocabulary.

Example:

Training sentences (without stop words):

- "food good"
- "food bad"
- "pizza amazing"

Training vocabulary: {food, good, bad, pizza, amazing}

Test sentence: "burger bad"

The word "burger" does not exist in V_{train}, therefore:

$$\vec{v}_{\text{burger}} = \text{undefined}$$

This makes it impossible to:

- Create a vector representation for the test sentence
- Make predictions on new data containing unseen words
- Generalize to real-world scenarios with evolving vocabulary

5. High Dimensionality with Large Vocabularies

In real-world applications, vocabulary sizes can be extremely large (10,000 to 100,000+ words), especially when stop words and infrequent words are not removed during preprocessing.

For a vocabulary of size $|V|$:

$$\text{Vector dimension} = |V|$$

Memory requirement for m documents with average length $\bar{n}$:

$$\text{Memory} = m \times \bar{n} \times |V| \times \text{bytes per element}$$

where:

- m is the number of documents
- $\bar{n}$ is the average number of words per document
- Each element typically requires 4-8 bytes (float32 or float64)

Example: For 10,000 documents with average 100 words

Without stop word removal: $|V| = 50,000$

$$\text{Memory} = 10000 \times 100 \times 50000 \times 4 \text{ bytes} = 200 \text{ GB}$$

With stop word removal: $|V| \approx 30,000$ (assuming ~40% reduction)

$$\text{Memory} = 10000 \times 60 \times 30000 \times 4 \text{ bytes} = 72 \text{ GB}$$

Savings: Stop word removal can reduce memory requirements by approximately 64% in this example.

This creates:

- **Computational inefficiency:** Processing large sparse matrices is slow
- **Storage inefficiency:** Most stored values are zeros
- **Scalability issues:** Cannot handle large corpora effectively

Stop word removal helps by reducing both vocabulary size and average document length, but the fundamental scalability problem remains for large datasets.

Best Practices for Stop Word Removal

When using one-hot encoding in practice:

1. **Always remove stop words** to reduce dimensionality and improve efficiency
2. **Use standard stop word lists** (e.g., NLTK, spaCy) as a starting point
3. **Domain-specific considerations:** Some domains may require keeping certain common words that carry meaning
4. **Balance:** Aggressive stop word removal can hurt semantic understanding; keep words that contribute to meaning
5. **Preprocessing pipeline:** Apply stop word removal consistently across training and test data

Summary Comparison

Aspect	Without Stop Words	With Stop Words
Implementation	✓ Very simple	✓ Very simple
Semantic meaning	✗ Not captured	✗ Not captured

Aspect	Without Stop Words	With Stop Words
Fixed input size	✗ Variable	✗ Variable (but shorter)
Sparsity	✗ 85.7% (our example)	✗ 80% (our example)
OOV handling	✗ Cannot handle	✗ Cannot handle
Scalability	✗ Poor	✓ Better (but still limited)
Memory efficiency	✗ Highly inefficient	✓ More efficient
Overfitting risk	✗ High	✓ Lower (less sparsity)
Vocabulary size	Large	Smaller (20-40% reduction)

Conclusion

One-hot encoding, while simple to understand and implement, suffers from fundamental limitations that make it impractical for most real-world NLP applications. **Stop word removal helps mitigate some disadvantages** (dimensionality, memory usage, sparsity) but doesn't solve the core problems of lack of semantic meaning, variable input size, and OOV handling.

More advanced techniques like Bag of Words, TF-IDF, and word embeddings (Word2Vec, GloVe, BERT) address these limitations by creating fixed-size, dense representations that capture semantic relationships. However, understanding one-hot encoding and the impact of preprocessing steps like stop word removal provides essential foundation knowledge for these more sophisticated methods.