Sparse and Dense Bag of Words (BOW) Vectors

Estimated time: 15 Min

Objectives

- Describe Bag of Words (BOW) in Natural Language Processing (NLP).
- Explain applications of Sparse and Dense BOW Vectors.
- Explore practical examples to illustrate Sparse and Dense BOW vectors.

An overview of Bag of Words (BOW) in Natural Language Processing (NLP)

Bag of Words (BOW) is a fundamental technique in Natural Language Processing (NLP) used to represent text data. It disregards grammar and word order, focusing solely on the presence and frequency of words in a document. Each document is depicted as a vector, with each dimension corresponding to a distinct word present in the entire corpus. The value assigned to each dimension signifies the frequency of that particular word within the document.

Sparse vs. Dense BOW Vectors

BOW vectors can be represented in two main forms: **Sparse** and **Dense**.

Sparse BOW Vectors: In a Sparse representation, each document's vector is typically large and contains mostly zeros. Each dimension corresponds to a unique word in the entire corpus, and the value represents the frequency of that word in the document. Sparse vectors are memory-efficient but may pose computational challenges due to their high dimensionality.

Dense BOW Vectors: Dense representations aim to address the high dimensionality of Sparse vectors by mapping words to a lower-dimensional continuous space, typically through techniques like Word Embeddings. In Dense vectors, each dimension carries continuous values, making them more computationally efficient but potentially requiring more memory. Dense vectors capture semantic relationships between words and can encode more information within fewer dimensions compared to Sparse vectors.

Applications of Sparse and Dense BOW Vectors

Sparse BOW Vectors

1. Text Classification: Sparse BOW vectors are widely used in text classification tasks like sentiment analysis, spam detection, and topic categorization. They provide a concise representation of textual data, allowing machine learning algorithms to efficiently classify documents into predefined categories.

2. Information Retrieval: Sparse BOW vectors are instrumental in information retrieval systems where documents need to be ranked based on their relevance to a given query. By representing documents as sparse vectors, search engines can quickly identify relevant documents that contain words similar to the query terms.

3. Document Clustering: Sparse BOW vectors facilitate document clustering where documents are grouped into clusters based on their similarity. Clustering algorithms like K-means or hierarchical clustering can effectively partition documents into clusters by analyzing the similarity between their sparse vector representations.

Dense BOW Vectors

1. Semantic Similarity: Dense BOW vectors are valuable in tasks requiring the measurement of semantic similarity between words or documents. By capturing semantic relationships in a continuous vector space, dense representations enable more nuanced comparisons compared to sparse representations. Applications include semantic search, duplicate detection, and recommendation systems.

2. Natural Language Understanding: Dense BOW vectors play a crucial role in natural language understanding tasks such as named entity recognition, part-of-speech tagging, and syntactic parsing. By embedding words into a continuous vector space, dense representations facilitate the extraction of meaningful linguistic features, enabling more accurate language understanding by machine learning models.

3. Machine Translation: Dense BOW vectors are essential in machine translation systems where words and phrases need to be accurately translated between languages. By representing words in a continuous vector space, dense representations enable machine translation models to capture subtle semantic nuances and syntactic structures, leading to more accurate translations.

Explore practical examples to illustrate Sparse and Dense BOW vectors

Sparse BOW Vectors

In this example, we will use CountVectorizer from scikit-learn to convert a collection of text documents into a matrix of token counts. In this representation, every row stands for a document, while each column represents a word within the vocabulary. The value in each cell represents the frequency of that word in the corresponding document. This results in a Sparse BOW representation where most values are zero.

1. Importing Required Library

```
1
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

Copied!

This line imports the `CountVectorizer` class from the `sklearn.feature_extraction.text` module. The CountVectorizer is a tool in scikit-learn designed to transform a set of text documents into a matrix containing the counts of tokens.

2. Sample Documents

```
1
2
3
4
5
```

```
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?",
```

Copied!

This is a list of sample `documents`. Each string in the list represents a document.

3. Initializing CountVectorizer

```
1
```

```
vectorizer = CountVectorizer()
```

Copied!

This line initializes a `CountVectorizer` object. By default, `CountVectorizer` converts text to lowercase and tokenizes it.

4. Fit and Transform Documents

```
1
```

```
sparse_bow = vectorizer.fit_transform(documents)
```

Copied!

`fit_transform()` method is called on the `vectorizer` object with the `documents` list as input. This method fits the vectorizer to the documents (learns the vocabulary and builds the document-term matrix) and transforms the documents into a sparse matrix of token counts.

5. Get Feature Names

```
1
```

```
feature_names = vectorizer.get_feature_names()
```

Copied!

`get_feature_names()` method retrieves the feature (word) names from the vectorizer, which represent the columns of the sparse matrix.

6. Print Sparse BOW Representation

```
1
```

```
print(sparse_bow.toarray())
```

Copied!

`sparse_bow.toarray()` converts the sparse matrix `sparse_bow` into a dense representation (numpy array) for printing. Each row in the array corresponds to a document, and each column represents a word within the vocabulary. The value in each cell indicates the frequency of that word in the corresponding document.

7. Print Feature Names

```
1
```

```
print("Feature Names:", feature_names)
```

Copied!

This prints the list of feature names (words) extracted from the documents.

Code Output

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
Feature Names: ['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

The output displays Sparse Bag-of-Words (BOW) vectors, with each row depicting a document, each column representing a word, and the values indicating the frequency of each word within the document.

The matrix `[[0 1 1 1 0 0 1 0 1], [0 2 0 1 0 1 1 0 1], [1 0 0 1 1 0 1 1 1], [0 1 1 1 0 0 1 0 1]]` represents the Sparse BOW vectors for the given documents.

Each row in the matrix corresponds to a document from the provided sample documents:
1.The first row represents the first document: "This is the first document."
2.The second row represents the second document: "This document is the second document."
3.The third row represents the third document: "And this is the third one."
4.The fourth row corresponds to the fourth document: "Is this the first document?"

Every column in the matrix denotes a distinct word drawn from the vocabulary list:
`'and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this'`

The values in the matrix indicate the frequency of each word in the corresponding document. For example:
1.In the first document, the word "document" appears once, "first" appears once, "is" appears once, "the" appears once, and "this" appears once.
2.In the second document, the word "document" appears twice, "is" appears once, "second" appears once, "the" appears once, and "this" appears once.
3.In the third document, the word "and" appears once, "first" appears once, "is" appears once, "one" appears once, "the" appears once, "third" appears once, and "this" appears once.
4.In the fourth document, the word "document" appears once, "first" appears once, "is" appears once, "the" appears once, and "this" appears once.

The Feature Names list provides the vocabulary of the documents, where each feature name corresponds to a unique word present in the documents.

Dense BOW Vectors (Using Word Embeddings)

In this example, we will use pre-trained Word Embeddings (Word2Vec) from the gensim library to obtain Dense BOW vectors for a set of words. Word Embeddings map words to continuous vector representations in a lower-dimensional space where semantically similar words are closer together. This results in a Dense BOW representation where each word is represented by a continuous vector of fixed size.

1. Installing Gensim Library

```
1
```

```
!pip install gensim  # Install gensim library
```

Copied!

This line installs the `Gensim` library using pip. `Gensim`is an open-source Python library designed for natural language processing (NLP) and machine learning tasks, particularly for topic modeling, document similarity analysis, text summarization, and word vector representations. The library is widely used in academia and industry for various text analysis tasks.

2. Importing Required Libraries

```
1
```

```
import gensim.downloader as api
```

Copied!

This line imports the `api` module from the `gensim.downloader` package. Gensim provides a downloader module `(gensim.downloader)` to access and download pre-trained models and datasets.

3. Loading Pre-trained Word2Vec Model

```
1
```

```
word_vectors = api.load("word2vec-google-news-300")
```

Copied!

This line loads a pre-trained Word2Vec model from the Gensim downloader. `word2vec-google-news-300` specifies the name of the pre-trained model to load. This particular model contains word embeddings trained on a large Google News dataset.

4. Defining Example Words

```
1
```

```
words = ["king", "queen", "man", "woman"]
```

Copied!

This line defines a list of example words for which we want to obtain dense Bag of Words (BOW) representations.

5. Obtaining Dense BOW Representation

```
1
```

```
dense_bow = [word_vectors[word] for word in words]
```

Copied!

This line retrieves the dense representations (word embeddings) for each word in the `words` list from the pre-trained Word2Vec model loaded earlier. It iterates over each word in the list and retrieves its corresponding vector representation.

6. Print the Dense BOW Representation

```
1
```

```
print(dense_bow)
```

Copied!

This line prints the dense Bag of Words (BOW) representations obtained for the example words. Each word in the words list is represented by a continuous vector of fixed size, obtained from the Word2Vec model.

Code Output

```
[array([ 1.25976562e-01,  2.97851562e-02,  8.60595703e-03,  1.39648438e-01,
        -2.56347656e-02, -3.61328125e-02,  1.11816406e-01, -1.98242188e-01,
         5.12695312e-02,  3.63281250e-01, -2.42187500e-01, -3.02734375e-01,
        -1.77734375e-01, -2.49023438e-02, -1.67968750e-01, -1.69921875e-01,
         3.46679688e-02,  5.21850586e-03,  4.63867188e-02,  1.28906250e-01,
         1.36718750e-01,  1.12792969e-01,  5.95703125e-02,  1.36718750e-01,
         1.01074219e-01, -1.76757812e-01, -2.51953125e-01,  5.98144531e-02,
         3.41796875e-01, -3.11279297e-02,  1.04492188e-01,  6.17675781e-02,
         1.24511719e-01,  4.00390625e-01, -3.22265625e-01,  8.39843750e-02,
         3.90625000e-02,  5.85937500e-03,  7.03125000e-02,  1.72851562e-01,
         1.38671875e-01, -2.31445312e-01,  2.83203125e-01,  1.42578125e-01,
         3.41796875e-01, -2.39257812e-02, -1.09863281e-01,  3.32031250e-02,
        -5.46875000e-02,  1.53198242e-02, -1.62109375e-01,  1.58203125e-01,
        -2.59765625e-01,  2.01416016e-02, -1.63085938e-01,  1.35803223e-03,
        -1.44531250e-01, -5.68847656e-02,  4.29687500e-02, -2.46582031e-02,
         1.85546875e-01,  4.47265625e-01,  9.58251953e-03,  1.31835938e-01,
         9.86328125e-02, -1.85546875e-01, -1.00097656e-01, -1.33789062e-01,
        -1.25000000e-01,  2.83203125e-01,  1.23046875e-01,  5.32226562e-02,
        -1.77734375e-01,  8.59375000e-02, -2.18505859e-02,  2.05078125e-02,
        -1.39648438e-01,  2.51464844e-02,  1.38671875e-01, -1.05468750e-01,
         1.38671875e-01,  8.88671875e-02, -7.51953125e-02, -2.13623047e-02,
         1.72851562e-01,  4.63867188e-02, -2.65625000e-01,  8.91113281e-03,
         1.49414062e-01,  3.78417969e-02,  2.38281250e-01, -1.24511719e-01,
        -2.17773438e-01, -1.81640625e-01,  2.97851562e-02,  5.71289062e-02,
        -2.89306641e-02,  1.24511719e-02,  9.66796875e-02, -2.31445312e-01,
         5.81054688e-02,  6.68945312e-02,  7.08007812e-02, -3.08593750e-01,
        -2.14843750e-01,  1.45507812e-01, -4.27734375e-01, -9.39941406e-03,
         1.54296875e-01, -7.66601562e-02,  2.89062500e-01,  2.77343750e-01,
        -4.86373901e-04, -1.36718750e-01,  3.24218750e-01, -2.46093750e-01,
        -3.03649902e-03, -2.11914062e-01,  1.25000000e-01,  2.69531250e-01,
         2.04101562e-01,  8.25195312e-02, -2.01171875e-01, -1.60156250e-01,
        -3.78417969e-02, -1.20117188e-01,  1.15234375e-01, -4.10156250e-02,
        -3.95507812e-02, -8.98437500e-02,  6.34765625e-03,  2.03125000e-01,
         1.86523438e-01,  2.73437500e-01,  6.29882812e-02,  1.41601562e-01,
        -9.81445312e-02,  1.38671875e-01,  1.82617188e-01,  1.73828125e-01,
         1.73828125e-01, -2.37304688e-01,  1.78710938e-01,  6.34765625e-02,
         2.36328125e-01, -2.08984375e-01,  8.74023438e-02, -1.66015625e-01,
```

The output consists of four lists, each representing a word from the input list `["king", "queen", "man", "woman"]`. Each list contains a 300-dimensional dense vector representation of the corresponding word.

Words with similar semantic meanings tend to have similar vector representations.

The dense vector representations capture various semantic and syntactic properties of the words. Although they are not directly interpretable by humans, they encode information about word contexts and relationships learned from the training data (Google News articles in this case).

Each vector has a length of 300 dimensions. This means that each word is represented by a vector in a 300-dimensional space, where each dimension corresponds to some learned feature or aspect of the word. These high-dimensional representations enable the model to capture complex relationships between words.

Conclusion

Understanding the distinction between Sparse and Dense BOW vectors is fundamental in Natural Language Processing (NLP) and text analysis. Sparse representations offer efficiency and interpretability, making them suitable for traditional machine learning tasks, while dense representations provide semantic richness and expressiveness, enabling more sophisticated language processing tasks. By leveraging the strengths of both sparse and dense representations, practitioners can effectively address a wide range of NLP challenges and applications.

Author(s)

Pratiksha Verma

Other Contributors

Malika Singla