Word2Vec with NLTK Reuters Corpus and Visualization using t-SNE

Lab Activity

Import Required Libraries

```
import nltk
from nltk.corpus import reuters, stopwords
from gensim.models import Word2Vec
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import pandas as pd
```

Download the Reuters corpus (if not already downloaded)

```
nltk.download('reuters')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
```

Step 1: Load the Reuters Corpus

Extract sentences from the Reuters corpus Explanation: The Reuters corpus contains a collection of financial news articles. Each sentence will be tokenized into words to create the vocabulary for Word2Vec.

```
corpus_sentences = []
for fileid in reuters.fileids():
    raw_text = reuters.raw(fileid)
    tokenized_sentence = [word for word in nltk.word_tokenize(raw_text) if word.isalnum() and word
    corpus_sentences.append(tokenized_sentence)
print(f"Number of sentences in the Reuters corpus: {len(corpus_sentences)}")
```

Step 2: Train a Word2Vec Model

Explanation: Word2Vec will learn vector representations for each word in the vocabulary.

Parameters:

- vector_size: Size of the word embedding vectors.
- window: Context window size for training.
- min_count: Minimum frequency for a word to be included in the vocabulary.
- workers: Number of threads for parallel processing.

```
model = Word2Vec(sentences=corpus_sentences, vector_size=100, window=5, min_count=5, workers=4)
# Print vocabulary size
print(f"Vocabulary size: {len(model.wv.index_to_key)}")
```

Step 3: Extract Word Embeddings for Visualization

```
import numpy as np
# Extract the learned word vectors and their corresponding words for visualization.
words = list(model.wv.index_to_key)[:200]  # Limit to top 200 words for better visualization
word_vectors = np.array([model.wv[word] for word in words])  # Convert to NumPy array for compatib
```

Step 4: Reduce Dimensionality with t-SNE

```
# Use t-SNE to project the high-dimensional word embeddings into a 2D space.
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
word_vectors_2d = tsne.fit_transform(word_vectors)
```

Step 5: Visualize the Word Embeddings

```
# Plot the 2D t-SNE visualization of the word embeddings with their labels.
def plot_embeddings(vectors, labels):
    plt.figure(figsize=(16, 12))
    for i, label in enumerate(labels):
        x, y = vectors[i]
        plt.scatter(x, y, color='blue')
        plt.text(x + 0.1, y + 0.1, label, fontsize=9)
    plt.title("Word2Vec Embeddings Visualized with t-SNE")
    plt.xlabel("t-SNE Dimension 1")
    plt.ylabel("t-SNE Dimension 2")
    plt.show()

plot_embeddings(word_vectors_2d, words)
```

Notes:

1. The Word2Vec model captures semantic relationships between words based on their co-occurrence in the text.
2. Similar words tend to cluster together in the t-SNE visualization.
3. t-SNE helps reduce the dimensionality for better interpretability, but the visualization may slightly vary with different runs due to its stochastic nature.

Key Questions for Students:

- What do you observe about the clusters in the t-SNE plot?
- How do you think the choice of parameters (e.g., window size, vector size) affects the embeddings?
- What are the limitations of using Word2Vec and t-SNE for NLP tasks?

Information Retreival Task:

Task: Build a Document Retrieval System using Word2Vec

1. Given a query string, find the most relevant documents from the Reuters corpus using Word2Vec embeddings.
2. Steps:

   a. Preprocess the query string by tokenizing and removing stop words.

   b. Compute the average Word2Vec embedding for the query string.

   c. Compute the average Word2Vec embedding for each document in the Reuters corpus.

   d. Use cosine similarity to find the top N most relevant documents for the query.
3. Display the top N document IDs and their similarity scores.

[redacted]