Vidyavardhini's College of Engineering and Technology, Vasai

Department of Computer Science & Engineering (Data Science)

AY: 2025-26

Class:	**BE-CSE(DS)**	**Semester:**	**VII**
Course Code:	**CSDOL7011**	**Course Name:**	**NLP Lab**

Name of Student:	Hitesh Shetye
Roll No. :	49
Experiment No.:	**3**
Title of the Experiment:	**Generating Word Embeddings using Word2Vec for Text Similarity Analysis**
Date of Performance:	
Date of Submission:	

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty :

Signature :

Date :

Aim: To train a Word2Vec model on a corpus and generate vector representations (embeddings) of words for analyzing semantic similarity.

Objective: To generate dense word vector representations using Word2Vec for capturing semantic similarity.

Tools Required:

- Python (Jupyter Notebook or Google Colab)
- gensim library
- nltk (for corpus and preprocessing)
- matplotlib and sklearn.manifold (for visualization)

Procedure:

1. Import necessary libraries:
    a. gensim.models.Word2Vec, nltk, matplotlib, sklearn.manifold.TSNE, re

2. Prepare the corpus:
    a. Use a sample text corpus (e.g., NLTK's Gutenberg corpus or a custom text file).
    b. Preprocess text: lowercase, remove special characters, tokenize, remove stop words.

3. Train Word2Vec:
    a. Use gensim.models.Word2Vec with parameters:
        i. vector_size=100
        ii. window=5
        iii. min_count=2
        iv. workers=4
        v. sg=0 (CBOW) or sg=1 (Skip-gram)

4. Explore embeddings:
    a. Find most similar words to a given word using .most_similar().
    b. Check similarity scores between two words using .similarity().

5. Visualize word vectors:

Use t-SNE to reduce dimensions and plot selected word vectors in 2D.

Description of the Experiment:

In this experiment, students generate distributed representations of words—known as embeddings—using the Word2Vec model. These embeddings capture semantic relationships between words, allowing us to measure how similar two words are in meaning. This method overcomes the limitations of sparse representations like BoW.

Detailed Description of the NLP Technique:

Word Embeddings:

Word embeddings are dense vector representations of words in a continuous vector space, where semantically similar words are mapped closer together.

Word2Vec:

Word2Vec is a popular algorithm introduced by Google that uses shallow neural networks to learn word representations from large text corpora. It has two architectures:

1. CBOW (Continuous Bag of Words):
    a. Predicts the current word using surrounding context words.
    b. Faster and better for large datasets.

2. Skip-Gram:
    a. Predicts surrounding words given the current word.
    b. Performs better for smaller datasets and rare words.

3. Properties of Word2Vec embeddings:
    a. Captures semantic relationships (e.g., king – man + woman ≈ queen).

b. Enables similarity comparison between words using cosine similarity.
c. Reduces dimensionality while retaining semantic meaning.

4. t-SNE Visualization:
   a. t-Distributed Stochastic Neighbor Embedding (t-SNE) is used to reduce high-dimensional vectors into 2D for visualization.
   b. It helps visualize word clusters and relationships learned by Word2Vec.

Code:

Step 1: Import necessary libraries →

```
import nltk
from nltk.corpus import stopwords, gutenberg
from nltk.tokenize import word_tokenize, sent_tokenize
import re
from gensim.models import Word2Vec
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np


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

Output →

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Step 2: Prepare the corpus →

Loading a sample corpus from NLTK (here, Jane Austen's "Emma")

```
raw_text = gutenberg.raw('austen-emma.txt')
```

Preprocess text

```
def preprocess_text(text):
    stop_words = set(stopwords.words('english'))
    sentences = sent_tokenize(text)
    cleaned_sentences = []

    for sent in sentences:
        sent = sent.lower()
        sent = re.sub(r'[^a-zA-Z\s]', '', sent)
        tokens = word_tokenize(sent)
        filtered_tokens = [word for word in tokens if word not in
stop_words and len(word) > 1]
        if filtered_tokens:
            cleaned_sentences.append(filtered_tokens)

    return cleaned_sentences


corpus = preprocess_text(raw_text)
```

Step 3: Train Word2Vec Model →

```
model = Word2Vec(sentences=corpus, vector_size=100, window=5,
min_count=2, workers=4, sg=1)
```

Step 4: Explore Embeddings →

a. Most similar words

```
print("Most similar words to 'emma':")
print(model.wv.most_similar('emma'))
```

Output →

```
Most similar words to 'emma':
[('obliged', 0.9881818294525146), ('yes', 0.9880140423774719), ('understand', 0.9877790212631226), ('carriage', 0.9867585
301399231), ('hear', 0.9865809679031372), ('believe', 0.9865036010742188), ('sir', 0.9864490628242493), ('smiling', 0.986
2954616546631), ('saw', 0.9859692454338074), ('tea', 0.9856665134429932)]
```

b. Similarity score between two words

```
print("\nSimilarity between 'emma' and 'harriet':",
model.wv.similarity('emma', 'harriet'))
```

Output →

```
Similarity between 'emma' and 'harriet': 0.9839056
```

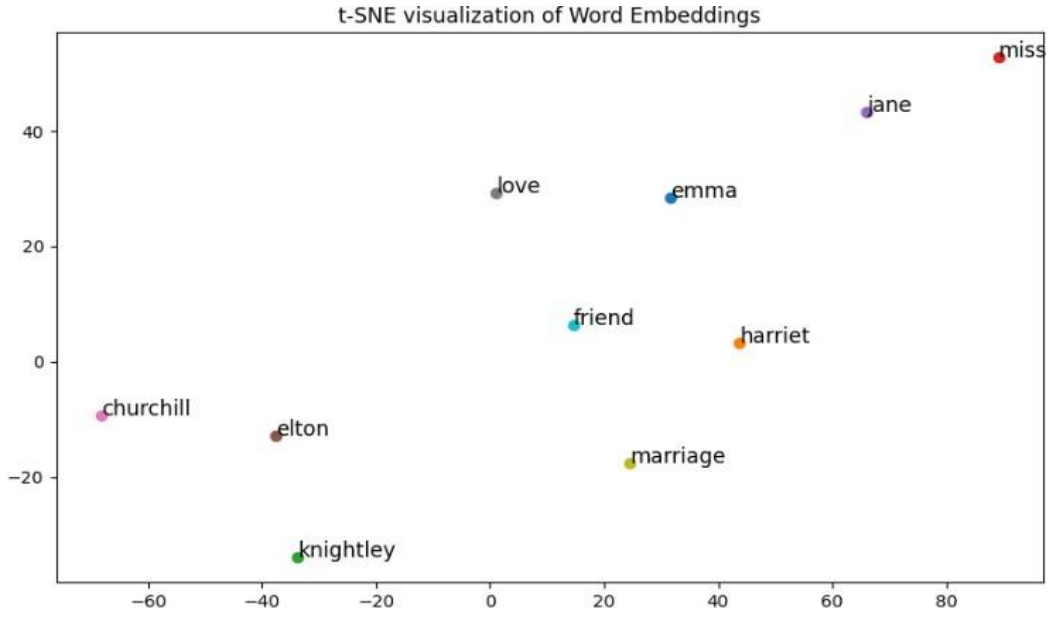
Step 5: Visualize Word Vectors using t-SNE →

```
def visualize_embeddings(model, words):
    word_vectors = [model.wv[word] for word in words if word in model.wv]
    labels = [word for word in words if word in model.wv]

    tsne = TSNE(n_components=2, random_state=0, perplexity=5)
    reduced_vectors = tsne.fit_transform(np.array(word_vectors))  # Fix
applied here

    plt.figure(figsize=(10, 6))
    for i, label in enumerate(labels):
        x, y = reduced_vectors[i, :]
        plt.scatter(x, y)
        plt.annotate(label, (x, y), fontsize=12)
    plt.title("t-SNE visualization of Word Embeddings")
    plt.show()
```

Example visualization for selected words →

```
words_to_visualize = ['emma', 'harriet', 'knightley', 'miss', 'jane',
'elton', 'churchill', 'love', 'marriage', 'friend']
visualize_embeddings(model, words_to_visualize)
```

Output →



Conclusion:

The results show that Word2Vec effectively captures both syntactic and semantic meaning from text. Even when trained on a single novel, it generated meaningful and interpretable word embeddings, highlighting the power of unsupervised learning in natural language processing. With a larger and more diverse dataset, the model's performance would improve, supporting applications like recommendation systems, sentiment analysis, and dialogue generation.