AY: 2025-26

Class:	BE	**Semester:**	VII
Course Code:	CSDOL7011	**Course Name:**	Natural Language Processing

Name of Student:	Konisha Jayesh Thakare
Roll No. :	71
Experiment No.:	3
Title of the Experiment:	Generating Word Embeddings using Word2Vec for Text Similarity Analysis
Date of Performance:	29.07.2025
Date of Submission:	05.08.2025

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 : Dr. Tatwadarshi P. Nagarhalli

Signature :

Date :

Aim: To generate word embeddings using the Word2Vec model and analyze text similarity by comparing vector representations of words.

Theory:

Traditional text representation methods like Bag-of-Words (BoW) and TF-IDF treat words as independent features, ignoring semantics and context. To overcome this, word embeddings represent words as dense, low-dimensional vectors where semantically similar words are placed closer in vector space. Word2Vec, introduced by Mikolov et al. at Google (2013), is one of the most popular methods for generating embeddings. It has two architectures:

1. Continuous Bag-of-Words (CBOW): Predicts a word based on its surrounding context.
2. Skip-Gram: Predicts the surrounding context words given a target word.

By training on large corpora, Word2Vec captures semantic relationships like:

- king - man + woman ≈ queen
- Similar words (e.g., "doctor", "nurse") are placed closer in embedding space.

For text similarity analysis, embeddings allow us to compute similarity using cosine similarity, which measures how close two word vectors are in direction.

Procedure:

1. Prepare or collect a sample text corpus.
2. Preprocess the text (tokenization, stopword removal, lowercasing).
3. Train Word2Vec embeddings using CBOW or Skip-Gram.
4. Retrieve word vectors for given words.
5. Compute similarity between words using cosine similarity.
6. Analyze results and interpret semantic closeness.

Algorithm:

1. Input raw corpus.
2. Tokenize text into sentences and words.
3. Train Word2Vec model using Gensim.
4. Extract vector representation for selected words.
5. Use cosine similarity to compare word embeddings.
6. Output similarity results.

Implementation:

```
import nltk
from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
nltk.download('punkt')
corpus=["Natural Language Processing enables computers to understand human language","Word embeddings capture semantic meaning of words","Word2Vec is an algorithm to generate word vectors","Text similarity can be computed using cosine similarity"]
sentences=[nltk.word_tokenize(sentence.lower()) for sentence in corpus]
model=Word2Vec(sentences,vector_size=50,window=3,min_count=1,sg=0)
vector_nlp=model.wv['language']
print("Word Vector for 'language':\n",vector_nlp)
print("\nMost similar to 'language':\n",model.wv.most_similar('language'))
def get_similarity(word1,word2):
    if word1 not in model.wv or word2 not in model.wv:
            return f"One of the words '{word1}' or '{word2}' not in vocabulary."
    v1=model.wv[word1].reshape(1,-1)
    v2=model.wv[word2].reshape(1,-1)
    return cosine_similarity(v1,v2)[0][0]
print("\nSimilarity between 'language' and 'text':",get_similarity('language','text'))
print("Similarity between 'word' and 'vector':",get_similarity('word','vector'))
print("Similarity between 'word' and 'vectors':",get_similarity('word','vectors'))
```

Output:

```
PS C:\Users\Konisha Thakare\OneDrive\Desktop\Python> python word2vec_embeddings.py
Word Vector for 'language':
 [-1.0724545e-03  4.7286271e-04  1.0206699e-02  1.8018546e-02
 -1.8605899e-02 -1.4233618e-02  1.2917745e-02  1.7945977e-02
 -1.0030856e-02 -7.5267432e-03  1.4761009e-02 -3.0669428e-03
 -9.0732267e-03  1.3108104e-02 -9.7203208e-03 -3.6320353e-03
  5.7531595e-03  1.9837476e-03 -1.6570430e-02 -1.8897636e-02
  1.4623532e-02  1.0140524e-02  1.3515387e-02  1.5257311e-03
  1.2701781e-02 -6.8107317e-03 -1.8928028e-03  1.1537147e-02
 -1.5043275e-02 -7.8722071e-03 -1.5023164e-02 -1.8600845e-03
  1.9076237e-02 -1.4638334e-02 -4.6675373e-03 -3.8754821e-03
  1.6154874e-02 -1.1861792e-02  9.0324880e-05 -9.5074680e-03
 -1.9207101e-02  1.0014586e-02 -1.7519170e-02 -8.7836506e-03
 -7.0199967e-05 -5.9236289e-04 -1.5322480e-02  1.9229487e-02
  9.9641159e-03  1.8466286e-02]

Most similar to 'language':
 [('is', 0.2705516219139099), ('of', 0.2105558067560196), ('text', 0.18602196872234344), ('embeddings', 0.16704076528549194), ('ve
ctors', 0.16078656911849976), ('semantic', 0.150198832154274), ('enables', 0.13204392790794373), ('to', 0.1267625242471695), ('und
erstand', 0.09983965009450912), ('be', 0.07528751343488693)]
```

Conclusion:

This experiment demonstrated how word embeddings generated using Word2Vec can represent semantic meaning of words in a dense vector space. Unlike Bag-of-Words or TF-IDF, Word2Vec captures contextual and semantic relationships, allowing for meaningful similarity computations between words. The results showed that semantically related words, such as "language" and "text," or "word" and "vector," have higher cosine similarity scores, validating the effectiveness of embeddings. Such representations are widely used in modern NLP applications including document similarity, recommendation systems, machine translation, and question answering.