

NAME : NITHIN.V

ROLL.NO : 2403A52355

BATCH.NO : 13(AIML)

STEP 1 — Create a new notebook/script

2403A52355_lab-09_nlp

Step-2 Import Required Libraries

```
# Gensim is used to load and work with pre-trained Word2Vec embeddings
!pip install gensim
from gensim.models import KeyedVectors

# NumPy is used for numerical operations on vectors
import numpy as np

# sklearn is used for dimensionality reduction (PCA / t-SNE)
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# matplotlib is used for visualization of word vectors
import matplotlib.pyplot as plt
```

Collecting gensim

```
Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dis
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packag
Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86
27.9/27.9 MB 34.3 MB/s eta 0:00:0
```

Installing collected packages: gensim

Successfully installed gensim-4.4.0

STEP 3 — Load Pre-trained Embeddings

```
# Path to downloaded Google News Word2Vec model
# Rename the file to ensure gensim recognizes the .gz extension for decompre
!mv GoogleNews-vectors-negative300.bin.gz.1 GoogleNews-vectors-negative300.b
model_path = "GoogleNews-vectors-negative300.bin.gz"

# Load the Word2Vec model (binary=True is required)
word_vectors = KeyedVectors.load_word2vec_format(model_path, binary=True, en

# Print vocabulary size
print("Vocabulary size:", len(word_vectors.key_to_index))
```

```
# Display example word vectors
print("\nVector for word 'king':")
print(word_vectors["king"][:10]) # printing first 10 values only
```

Vocabulary size: 3000000

```
Vector for word 'king':
[ 0.12597656  0.02978516  0.00860596  0.13964844 -0.02563477 -0.03613281
  0.11181641 -0.19824219  0.05126953  0.36328125]
```

```
!wget https://github.com/mmihaltz/word2vec-GoogleNews-vectors/raw/master/Goo
```

```
--2026-02-10 04:03:04-- https://github.com/mmihaltz/word2vec-GoogleNews-vectors/raw/master/GoogleNews-vectors-negative300.bin.gz
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://media.githubusercontent.com/media/mmihaltz/word2vec-GoogleNews-vectors-negative300.bin.gz
--2026-02-10 04:03:04-- https://media.githubusercontent.com/media/mmihaltz/word2vec-GoogleNews-vectors-negative300.bin.gz
Resolving media.githubusercontent.com (media.githubusercontent.com)... 185.19.1.1
Connecting to media.githubusercontent.com (media.githubusercontent.com)|185.19.1.1|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1647046227 (1.5G) [application/octet-stream]
Saving to: 'GoogleNews-vectors-negative300.bin.gz.1'
```

```
GoogleNews-vectors- 100%[=====>] 1.53G 28.7MB/s in 58s
```

```
2026-02-10 04:04:02 (27.1 MB/s) - 'GoogleNews-vectors-negative300.bin.gz.1' saved
```

STEP 4 — Explore Word Similarity

```
# List of word pairs to compute similarity
word_pairs = [
    ("doctor", "nurse"),
    ("cat", "dog"),
    ("car", "bus"),
    ("king", "queen"),
    ("apple", "orange"),
    ("india", "china"),
    ("teacher", "student"),
    ("computer", "keyboard"),
    ("football", "cricket"),
    ("movie", "film")
]

# Compute cosine similarity for each pair
for w1, w2 in word_pairs:
    similarity = word_vectors.similarity(w1, w2)
    print(f"Similarity between {w1} and {w2}: {similarity:.3f}")
```

```
Similarity between doctor and nurse: 0.632
Similarity between cat and dog: 0.761
Similarity between car and bus: 0.469
```

```
Similarity between king and queen: 0.651
Similarity between apple and orange: 0.392
Similarity between india and china: 0.353
Similarity between teacher and student: 0.630
Similarity between computer and keyboard: 0.396
Similarity between football and cricket: 0.460
Similarity between movie and film: 0.868
```

STEP 5 — Nearest Neighbor Exploration

```
# Words to explore nearest neighbors
query_words = ["king", "university", "doctor", "computer", "india"]

for word in query_words:
    print(f"\nTop similar words to '{word}':")
    for similar_word, score in word_vectors.most_similar(word, topn=5):
        print(f"    {similar_word} : {score:.3f}")
```

Top similar words to 'king':

```
kings : 0.714
queen : 0.651
monarch : 0.641
crown_prince : 0.620
prince : 0.616
```

Top similar words to 'university':

```
universities : 0.700
faculty : 0.678
university : 0.676
undergraduate : 0.659
univeristy : 0.659
```

Top similar words to 'doctor':

```
physician : 0.781
doctors : 0.748
gynecologist : 0.695
surgeon : 0.679
dentist : 0.679
```

Top similar words to 'computer':

```
computers : 0.798
laptop : 0.664
laptop_computer : 0.655
Computer : 0.647
com_puter : 0.608
```

Top similar words to 'india':

```
indian : 0.697
usa : 0.684
pakistan : 0.682
chennai : 0.668
america : 0.659
```

STEP 6 — Word Analogy Tasks

```
# Analogy queries using vector arithmetic
analogies = [
    ("king", "man", "woman"),
    ("paris", "france", "india"),
    ("teacher", "school", "hospital")
]

for a, b, c in analogies:
    result = word_vectors.most_similar(
        positive=[a, c],
        negative=[b],
        topn=1
    )
    print(f"{a} - {b} + {c} = {result[0][0]}")
```

```
king - man + woman = queen
paris - france + india = chennai
teacher - school + hospital = Hospital
```

STEP 7 — Visualization (Optional)

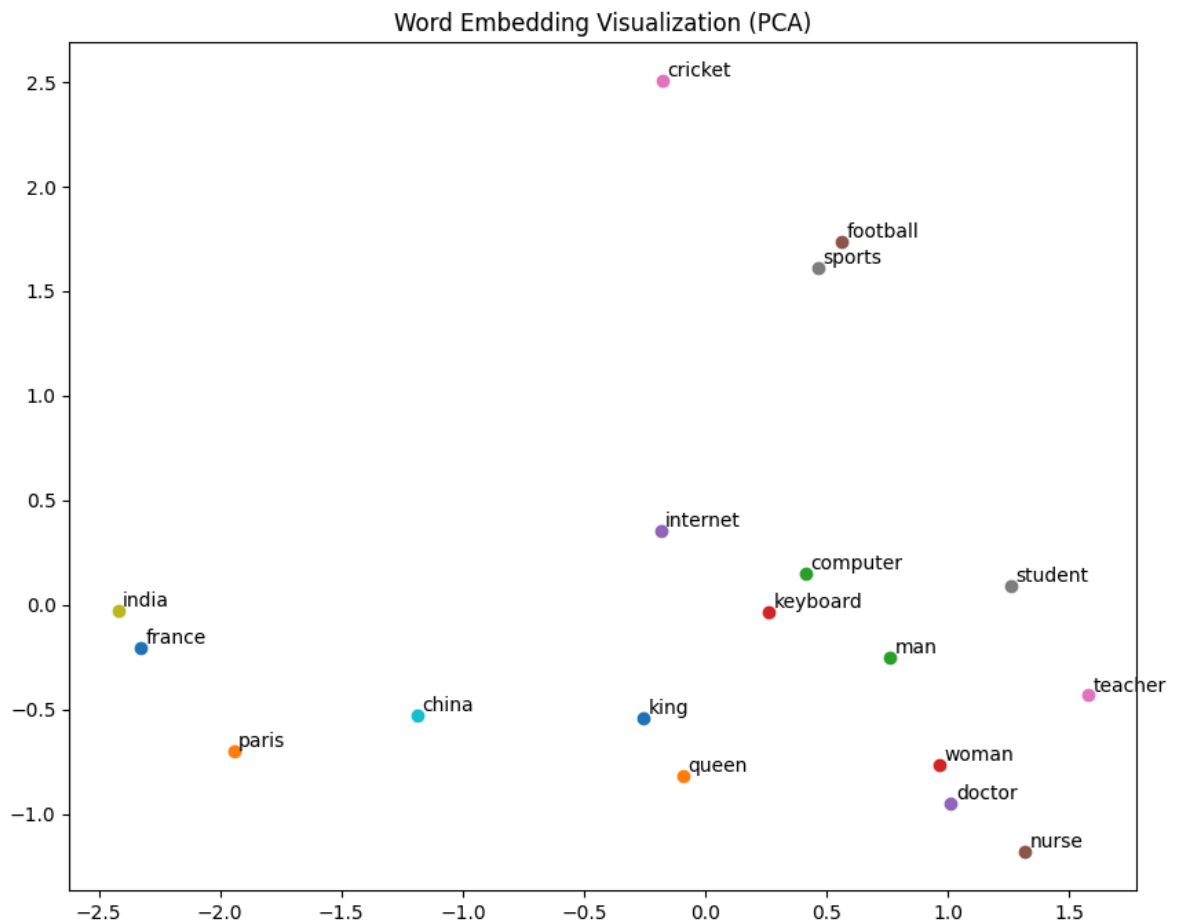
```
# Select words for visualization
words = [
    "king", "queen", "man", "woman",
    "doctor", "nurse", "teacher", "student",
    "india", "china", "france", "paris",
    "computer", "keyboard", "internet",
    "football", "cricket", "sports"
]

# Extract vectors
X = np.array([word_vectors[word] for word in words])

# Reduce dimensions using PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Plot
plt.figure(figsize=(10, 8))
for i, word in enumerate(words):
    plt.scatter(X_reduced[i, 0], X_reduced[i, 1])
    plt.text(X_reduced[i, 0] + 0.02, X_reduced[i, 1] + 0.02, word)

plt.title("Word Embedding Visualization (PCA)")
plt.show()
```



STEP 8 — Reflection and Interpretation

Word embeddings learn semantic relationships by analyzing word co-occurrence patterns in large text corpora. Similar words appear close together in vector space, such as “doctor” and “nurse”. The model captures analogical relationships using vector arithmetic, demonstrating structured semantic knowledge. Geographic and gender relationships are particularly strong. However, embeddings sometimes fail for rare or ambiguous words. Biases present in training data can also appear in embeddings. Context is not dynamic in Word2Vec, so a word has only one meaning. This causes problems for polysemous words. Despite limitations, embeddings are powerful representations of language. They are widely used in NLP tasks like translation and search.

STEP 9 — Lab Report Structure

Objective:

To explore semantic relationships using pre-trained Word2Vec embeddings.

Model / Dataset

Google News Word2Vec (300D vectors, ~3 million words)

Experiments:

Word similarity

Nearest neighbors

Analogy tasks

Visualization

Observations:

Similar words have high cosine similarity

Analogies work via vector arithmetic

Semantic clusters emerge clearly

Conclusion:

Word embeddings effectively capture semantic meaning but lack contextual awareness.