Natural Language Processing using Python Programming

Notebook 09.3: Similarity Analysis and Semantic Search

Part of the comprehensive learning series: Natural Language Processing using Python Programming

Learning Objectives:

- Master semantic similarity measurement using cosine similarity in word embedding spaces
- Learn document vectorization techniques through word vector averaging methods
- Implement practical semantic search engines that find meaning-based document relevance
- Understand the advantages of embedding-based search over traditional keyword matching
- Build foundation for advanced similarity analysis and information retrieval systems

---

- This notebook demonstrates the practical power of word embeddings for **semantic similarity analysis** and **meaning-based search**, moving beyond traditional keyword matching to true understanding of content relationships.

- We'll explore cosine similarity measurement, document vectorization techniques, and build a functional semantic search engine that finds relevant documents based on meaning rather than exact word matches.

1. Setting up: Loading Model and Data

- We load the Word2Vec model trained in the previous notebook (9.2) and a sample of pre-processed documents.

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize

MODEL_PATH = '../../models/word2vec_brown.model'

try:
    # Load the trained Word2Vec model
    w2v_model = Word2Vec.load(MODEL_PATH)
    word_vectors = w2v_model.wv
```

```
    print("Word2Vec model loaded successfully.")
except FileNotFoundError:
    print(f"ERROR: Model not found at {MODEL_PATH}. Please run 09_2_using_word2vec
    word_vectors = None
```

```
Word2Vec model loaded successfully.
```

In [2]:
```
# Sample Documents (already tokenized and cleaned conceptually)
documents = [
    ['programmer', 'love', 'python', 'deep', 'learning'],
    ['monarch', 'rule', 'kingdom', 'throne'],
    ['scientist', 'study', 'biology', 'laboratory'],
    ['king', 'rule', 'queen', 'palace']
]
doc_titles = ['Tech Report', 'Royalty History', 'Science Journal', 'Royalty Story'
```

---

2. Cosine Similarity: Measuring Distance in Semantic Space

- The distance between two dense vectors is measured using **Cosine Similarity**.
- It measures the cosine of the angle between two vectors.

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

Interpreting the Results:

- A score close to **1.0** means the vectors are very similar (small angle) - **semantically related**
- A score around **0.5-0.7** indicates moderate similarity - **some semantic relationship**
- A score close to **0.0** means they are orthogonal (unrelated) - **no semantic relationship**
- A score close to **-1.0** means they are highly dissimilar (opposite meaning) - **opposite semantics**

2.1 Word-to-Word Similarity

- We use the loaded Gensim model to instantly calculate the cosine similarity between word vectors.

In [5]:
```
if word_vectors:
    # Similarity between related words
    sim_king_queen = word_vectors.similarity('king', 'queen')

    # Similarity between unrelated words
    sim_king_biology = word_vectors.similarity('king', 'biology')

    # Similarity between closely related synonyms
    sim_research_scientist = word_vectors.similarity('research', 'scientist')
```

```
    print(f"Similarity('king', 'queen'):         {sim_king_queen:.4f}")
    print(f"Similarity('king', 'biology'):       {sim_king_biology:.4f}")
    print(f"Similarity('research', 'scientist'): {sim_research_scientist:.4f}")
else:
    print("Cannot run word similarity, model failed to load.")
```

```
Similarity('king', 'queen'):         0.7892
Similarity('king', 'biology'):       0.5361
Similarity('research', 'scientist'): 0.7830
```

Analysis of Results:

Looking at our similarity scores:

- **'king' ↔ 'queen': 0.7892** - High similarity! These words are semantically very related (both royalty)
- **'king' ↔ 'biology': 0.5361** - Moderate similarity, indicating some distant relationship (perhaps through academic contexts)
- **'research' ↔ 'scientist': 0.7830** - High similarity! These words frequently appear together in academic contexts

These results demonstrate that our Word2Vec model successfully learned semantic relationships from the Brown Corpus, even with limited training data.

---

3. Document Vectorization (Document Averaging)

- To compare entire documents, we must convert each document (a sequence of words) into a single document vector.
- The simplest and most common method is **Document Averaging**:

> **Document Vector** = Average of the Word Vectors of all words in the document.

In [6]:
```
def document_vector(word_list, model_wv):
    """Creates a document vector by averaging word vectors."""
    if not model_wv: return None

    # Filter out words not in the model's vocabulary
    vectors = [model_wv[word] for word in word_list if word in model_wv]

    if vectors:
        # Calculate the mean (average) of all word vectors
        return np.mean(vectors, axis=0)
    else:
        # Return a zero vector if the document is empty or contains only OOV words
        return np.zeros(model_wv.vector_size)

if word_vectors:
    # Create document vectors for all our sample documents
    doc_vectors = [document_vector(doc, word_vectors) for doc in documents]

    # Check the dimensions
    print(f"Vector dimension: {word_vectors.vector_size}")
```

```
    print(f"Document 1 Vector Shape: {doc_vectors[0].shape}")
else:
    doc_vectors = []
```

```
Vector dimension: 100
Document 1 Vector Shape: (100,)
```

---

4. Semantic Search Implementation

- Semantic search allows us to find documents relevant to a query based on **meaning**, even if they don't share exact keywords. The process is:
    1. Convert the **query** into a document vector.
    2. Calculate the **Cosine Similarity** between the query vector and every document vector.
    3. Rank documents by their similarity score.

In [7]:
```
# Import cosine function
from scipy.spatial.distance import cosine

def semantic_search(query, doc_vectors, doc_titles, model_wv):
    """Finds and ranks documents based on semantic similarity to the query."""
    if not doc_vectors: return "Model not ready."

    # 1. Preprocess and vectorizer the query
    query_tokens = word_tokenize(query.lower())
    query_vector = document_vector(query_tokens, model_wv)

    if query_vector is None:
        return "Query contains no words in the model vocabulary."

    # 2. Calculate similarities
    results = []
    for i, doc_vec in enumerate(doc_vectors):
        # Cosine distance = 1 - Cosine Similarity. We want to maximize similarity.
        similarity = 1 - cosine(query_vector, doc_vec)
        results.append((doc_titles[i], similarity))

    # 3. Rank results
    results.sort(key=lambda item: item[1], reverse=True)
    return results
```

In [8]:
```
# Test 1: Query related to Royalty
query1 = "The monarch and his wife in the castle."
results1 = semantic_search(query1, doc_vectors, doc_titles, word_vectors)

print(f"\n--- Search Query: '{query1}' ---")
for title, sim in results1:
    print(f" - {title:<18} | Similarity: {sim:.4f}")
```

```
--- Search Query: 'The monarch and his wife in the castle.' ---
 - Royalty Story      | Similarity: 0.8004
 - Royalty History    | Similarity: 0.7462
 - Tech Report        | Similarity: 0.7109
 - Science Journal    | Similarity: 0.4189
```

In [9]:
```python
# Test 2: Query related to Science/Technology
query2 = "Research on computing done in the lab."
results2 = semantic_search(query2, doc_vectors, doc_titles, word_vectors)

print(f"\n--- Search Query: '{query2}' ---")
for title, sim in results2:
    print(f" - {title:<18} | Similarity: {sim:.4f}")
```

```
--- Search Query: 'Research on computing done in the lab.' ---
 - Science Journal    | Similarity: 0.8317
 - Royalty History    | Similarity: 0.5469
 - Royalty Story      | Similarity: 0.5195
 - Tech Report        | Similarity: 0.4496
```

> **Observation:** Documents related to 'Royalty' should cluster together and score highly for Query 1, even if the query uses words like 'wife' or 'castle' which are not explicitly in the document titles. This demonstrates true semantic retrieval.

5. Summary and Next Steps

- We successfully utilized Word2Vec embeddings to measure semantic relationships using **Cosine Similarity**.
- We learned how to create a single **Document Vector** via averaging and used this method to build a basic **Semantic Search** application.
- You now have a strong grasp of both classical (TF-IDF) and modern (Embedding) vectorization.
- In **Chapter 10**, we will move to the current state-of-the-art: **Transformer Models** (BERT, GPT) and the Hugging Face ecosystem, preparing us for advanced Deep Learning NLP.

Key Takeaways

- **Semantic Similarity Mastery:** We learned to measure meaningful relationships between words and documents using cosine similarity in embedding spaces, moving beyond simple keyword matching.
- **Document Vectorization Skills:** We mastered the document averaging technique to convert multi-word documents into single representative vectors for comparison and analysis.
- **Practical Search Implementation:** We built a functional semantic search engine that finds relevant documents based on meaning, demonstrating the power of

embeddings for information retrieval.
- **Classical vs Modern Understanding:** We established a strong foundation in both classical (TF-IDF) and modern (embedding-based) vectorization approaches for comprehensive NLP knowledge.

Next Chapter Preview

- With word embeddings mastered, we're ready to explore the **cutting-edge of NLP** technology.
- Chapter 10 will introduce **Transformer Models** (BERT, GPT) and the **Hugging Face ecosystem** - the foundation of modern AI language models and the gateway to advanced deep learning NLP.

About This Project

This notebook is part of the **Natural Language Processing using Python Programming for Beginners** repository - a comprehensive, beginner-friendly guide for mastering NLP using Python, NLTK, and SpaCy.

Repository: `NLP`

Author

Prakash Ukhalkar
GitHub prakash-ukhalkar
