Natural Language Processing using Python Programming

Notebook 09.1: Introduction to Word Embeddings

Python 3.8+ NLTK Latest SpaCy Latest Gensim Latest License MIT

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

Learning Objectives:

- Understand the fundamental concepts and advantages of word embeddings over sparse vector representations
- Master the distributional hypothesis and its role in capturing semantic meaning
- Explore vector arithmetic and analogical reasoning capabilities of word embeddings
- Learn about popular embedding models including Word2Vec and GloVe architectures
- Build foundation for practical implementation of word embedding techniques

---

- This notebook introduces the revolutionary concept of **word embeddings** - dense vector representations that capture semantic meaning, moving beyond the limitations of sparse count-based methods like TF-IDF.

- We'll explore how embeddings solve the fundamental problem of semantic similarity and understand the distributional hypothesis that forms the foundation of modern NLP techniques.

1. The Problem with Sparse Vectors (Review)

Sparse Vectors (like TF-IDF) are typically very long (equal to the size of the vocabulary, often 100,000+ dimensions) and mostly filled with zeros. This is inefficient and ignores context.

Feature	TF-IDF (Sparse Vector)	Word Embedding (Dense Vector)
Dimensions	High (e.g., 50,000)	Low (e.g., 50, 100, 300)
Values	Integers or floating-point weights (mostly 0s)	Continuous floating-point numbers
Meaning	Measures word *importance* (count/rarity)	Measures word *meaning* (context/semantics)
Size	Grows with vocabulary size	Fixed size

Why TF-IDF Fails at Meaning:

- If 'good' and 'excellent' never appear in the same document, TF-IDF will place them far apart, even though they mean similar things.
- They have no measurable similarity beyond being two different words.

2. Distributional Semantics: The Core Idea

- The foundation of all modern word embeddings is the **Distributional Hypothesis**:

> **"You shall know a word by the company it keeps."** (Firth, 1957)

- If two words frequently appear in the same contexts (i.e., surrounded by the same words), they are likely to have similar meanings.
  - For example, 'cat' and 'dog' both frequently appear near 'owner', 'feed', and 'pet'.

3. Word Embeddings: How Semantic Space Works

- A word embedding is a multi-dimensional coordinate for a word.
- Words with similar meanings are mapped to positions close to each other in the vector space.

The Analogy Task (Vector Arithmetic)

- The most famous illustration of embeddings is vector arithmetic, which shows that geometric relationships in the vector space correspond to semantic relationships in language:

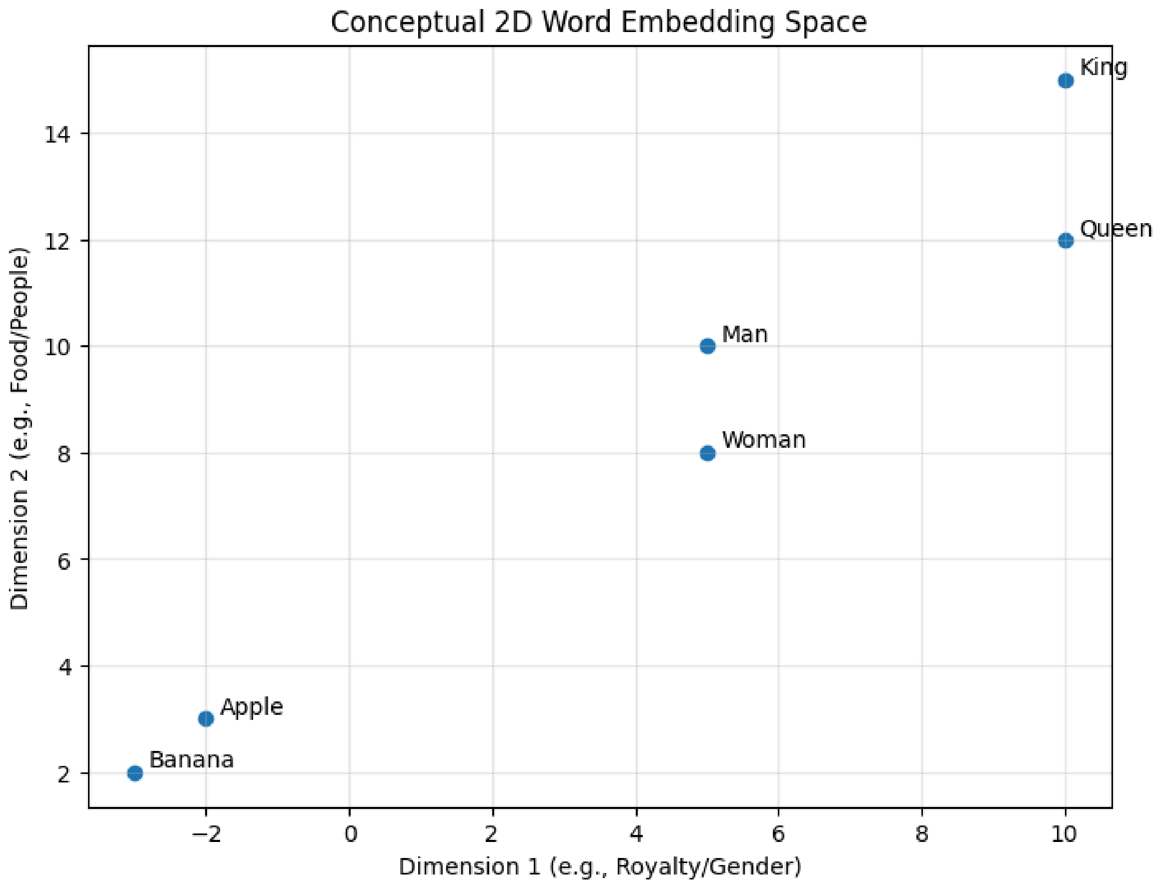
$$\text{Vector}(\text{'King'}) - \text{Vector}(\text{'Man'}) + \text{Vector}(\text{'Woman'}) \approx \text{Vector}(\text{'Queen'})$$

- This works because the 'Royalty' dimension and the 'Gender' dimension are encoded into the dense vector space.

In [1]:

```
# Conceptual visualization of a 2-D vector space
import matplotlib.pyplot as plt

words = ['King', 'Queen', 'Man', 'Woman', 'Apple', 'Banana']
x = [10, 10, 5, 5, -2, -3]
y = [15, 12, 10, 8, 3, 2]

plt.figure(figsize=(8, 6))
plt.scatter(x, y)

for i, word in enumerate(words):
    plt.annotate(word, (x[i] + 0.2, y[i] + 0.1))

plt.title('Conceptual 2D Word Embedding Space')
plt.xlabel('Dimension 1 (e.g., Royalty/Gender)')
```

```python
plt.ylabel('Dimension 2 (e.g., Food/People)')
plt.grid(True, alpha=0.3)
plt.show()

# Observation: 'King' and 'Queen' are close. 'Apple' and 'Banana' are close.
# The arithmetic relationship (King - Man + Woman) should land near Queen.
```



4. Popular Embedding Models (Brief Overview)

4.1 Word2Vec (Mikolov, 2013)

- Developed by Google, Word2Vec uses a shallow neural network to learn word vectors from context.
- It has two main architectures:
  - **CBOW (Continuous Bag-of-Words):** Predicts the current word based on its surrounding context words.
  - **Skip-gram:** Predicts the surrounding context words given the current word (generally performs better).

4.2 GloVe (Global Vectors for Word Representation)

- Developed at Stanford, GloVe combines both global (corpus-wide) and local (contextual) matrix factorization information.
- It's often highly effective and is a popular choice for pre-trained vectors.

5. Summary and Next Steps

- Word embeddings revolutionize NLP by introducing dense, semantic-aware vector representations, fundamentally moving beyond count-based methods.
- In the next notebook (**9.2**), we will gain practical experience by working with **Gensim** to load and train **Word2Vec** models, and visualize these semantic spaces using **t-SNE**.

Key Takeaways

- **Semantic Revolution:** We learned how word embeddings revolutionize NLP by capturing semantic meaning through dense vector representations, solving the fundamental limitations of sparse count-based methods.
- **Distributional Hypothesis Mastery:** We understood that "words are known by the company they keep" - the core principle that enables embeddings to capture contextual similarity and meaning.
- **Vector Arithmetic Wonder:** We explored the fascinating capability of embeddings to perform analogical reasoning through vector arithmetic (King - Man + Woman ≈ Queen).
- **Model Architecture Understanding:** We gained insight into popular embedding models like Word2Vec (CBOW/Skip-gram) and GloVe, understanding their different approaches to learning semantic representations.

Next Notebook Preview

- With theoretical foundations established, we're ready for **hands-on implementation** of word embeddings.
- Notebook 9.2 will provide practical experience with **Gensim** for loading and training Word2Vec models, plus visualization techniques using t-SNE to explore semantic spaces.

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

Built with ❤️ for the Python community