Natural Language Processing using Python Programming

Notebook 09.2: Using Word2Vec with Gensim and t-SNE Visualization

Python 3.8+ | NLTK Latest | Gensim Latest | Scikit-learn Latest | License MIT

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

Learning Objectives:

- Master practical implementation of Word2Vec models using the Gensim library
- Learn corpus preparation and preprocessing techniques for embedding training
- Understand Word2Vec hyperparameters and their impact on model performance
- Explore semantic relationships through similarity queries and analogical reasoning
- Implement t-SNE visualization techniques for high-dimensional word vector spaces

---

- This notebook provides hands-on experience with **Gensim** - a powerful Python library for training Word2Vec models and exploring semantic relationships in text data.
- We'll guide you through the complete workflow: loading the NLTK Brown Corpus, preprocessing data, training Word2Vec models, and visualizing high-dimensional embeddings using t-SNE for intuitive understanding.

1. Setting up: Libraries and Data Preparation

- Word2Vec models expect a list of sentences, where each sentence is a list of pre-processed words (tokens).
- We'll use the Brown Corpus for training.

In [1]:
```
# Import necessary libraries
import nltk
from nltk.corpus import brown, stopwords                    # NLTK's Brown corpus and sto
from gensim.models import Word2Vec                          # Gensim's Word2Vec model
import logging                                              # For logging training progre
import re                                                   # Regular expressions for tex
import numpy as np

# Set up logging to see training progress
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=log

# Download necessary NLTK resource
```

```python
nltk.download('brown', quiet=True)
nltk.download('stopwords', quiet=True)

stop_words = set(stopwords.words('english'))

def preprocess_sentences(corpus):
    """Cleans and tokenizes sentences from a corpus."""
    processed_sentences = []
    for sent in corpus.sents():
        # 1. Lowercase
        cleaned = [w.lower() for w in sent]

        # 2. Remove punctuation and filter stopwords (simplified preprocessing)
        tokens = [
            w for w in cleaned
            if w.isalpha() and w not in stop_words
        ]
        if tokens:
            processed_sentences.append(tokens)
    return processed_sentences

sentences = preprocess_sentences(brown)

print(f"Total sentences for training: {len(sentences)}")
print(f"Example sentence: {sentences[10]}")
```

```
Total sentences for training: 56367
Example sentence: ['urged', 'city', 'take', 'steps', 'remedy', 'problem']
```

---

2. Training the Word2Vec Model with Gensim

The `Word2Vec` class handles the neural network training. Key parameters include:

- **`sentences`**: The training data (list of lists of words).
- **`vector_size`**: The dimensionality of the resulting word vectors (e.g., 100).
- **`window`**: The maximum distance between the current and predicted word within a sentence.
- **`min_count`**: Ignores all words with a frequency lower than this threshold (filters noise).

In [2]:

```python
print("Starting Word2Vec training on Brown Corpus...")

w2v_model = Word2Vec(
    sentences=sentences,
    vector_size=100,
    window=5,
    min_count=5,
    workers=4,                                    # Number of CPU cores to use
    sg=1,                                         # Skip-gram (sg=1) is general
    epochs=10                                     # Number of training iteratio
)
```

```python
print("\nWord2Vec Model Training Complete.")
print(f"\nVocabulary size after filtering: {len(w2v_model.wv)}")
```

```
2025-10-08 18:32:21,439 : INFO : collecting all words and their counts
2025-10-08 18:32:21,441 : INFO : PROGRESS: at sentence #0, processed 0 words, keep
ing 0 word types
2025-10-08 18:32:21,511 : INFO : PROGRESS: at sentence #10000, processed 100896 wo
rds, keeping 18086 word types
2025-10-08 18:32:21,585 : INFO : PROGRESS: at sentence #20000, processed 197606 wo
rds, keeping 25622 word types
Starting Word2Vec training on Brown Corpus...
```

```
2025-10-08 18:32:21,662 : INFO : PROGRESS: at sentence #30000, processed 305420 wo
rds, keeping 31236 word types
2025-10-08 18:32:21,750 : INFO : PROGRESS: at sentence #40000, processed 403890 wo
rds, keeping 35764 word types
2025-10-08 18:32:21,811 : INFO : PROGRESS: at sentence #50000, processed 466434 wo
rds, keeping 38316 word types
2025-10-08 18:32:21,912 : INFO : collected 40097 word types from a corpus of 50926
7 raw words and 56367 sentences
2025-10-08 18:32:21,915 : INFO : Creating a fresh vocabulary
2025-10-08 18:32:22,199 : INFO : Word2Vec lifecycle event {'msg': 'effective_min_c
ount=5 retains 13232 unique words (33.00% of original 40097, drops 26865)', 'datet
ime': '2025-10-08T18:32:22.199509', 'gensim': '4.3.3', 'python': '3.10.0 (tags/v3.
10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]', 'platform': 'Wi
ndows-10-10.0.26100-SP0', 'event': 'prepare_vocab'}
2025-10-08 18:32:22,204 : INFO : Word2Vec lifecycle event {'msg': 'effective_min_c
ount=5 leaves 463133 word corpus (90.94% of original 509267, drops 46134)', 'datet
ime': '2025-10-08T18:32:22.204764', 'gensim': '4.3.3', 'python': '3.10.0 (tags/v3.
10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]', 'platform': 'Wi
ndows-10-10.0.26100-SP0', 'event': 'prepare_vocab'}
2025-10-08 18:32:22,577 : INFO : deleting the raw counts dictionary of 40097 items
2025-10-08 18:32:22,577 : INFO : sample=0.001 downsamples 10 most-common words
2025-10-08 18:32:22,577 : INFO : Word2Vec lifecycle event {'msg': 'downsampling le
aves estimated 458578.7197397975 word corpus (99.0%% of prior 463133)', 'datetim
e': '2025-10-08T18:32:22.577427', 'gensim': '4.3.3', 'python': '3.10.0 (tags/v3.1
0.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]', 'platform': 'Win
dows-10-10.0.26100-SP0', 'event': 'prepare_vocab'}
2025-10-08 18:32:22,861 : INFO : estimated required memory for 13232 words and 100
dimensions: 17201600 bytes
2025-10-08 18:32:22,861 : INFO : resetting layer weights
2025-10-08 18:32:22,880 : INFO : Word2Vec lifecycle event {'update': False, 'trim_
rule': 'None', 'datetime': '2025-10-08T18:32:22.880482', 'gensim': '4.3.3', 'pytho
n': '3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD6
4)]', 'platform': 'Windows-10-10.0.26100-SP0', 'event': 'build_vocab'}
2025-10-08 18:32:22,880 : INFO : Word2Vec lifecycle event {'msg': 'training model
with 4 workers on 13232 vocabulary and 100 features, using sg=1 hs=0 sample=0.001
negative=5 window=5 shrink_windows=True', 'datetime': '2025-10-08T18:32:22.88048
2', 'gensim': '4.3.3', 'python': '3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:0
0:18) [MSC v.1929 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.26100-SP0', 'even
t': 'train'}
2025-10-08 18:32:23,928 : INFO : EPOCH 0 - PROGRESS: at 97.97% examples, 441440 wo
rds/s, in_qsize 1, out_qsize 1
2025-10-08 18:32:23,928 : INFO : EPOCH 0: training on 509267 raw words (458488 eff
ective words) took 1.0s, 447120 effective words/s
2025-10-08 18:32:24,995 : INFO : EPOCH 1 - PROGRESS: at 89.66% examples, 404997 wo
rds/s, in_qsize 4, out_qsize 0
2025-10-08 18:32:25,066 : INFO : EPOCH 1: training on 509267 raw words (458597 eff
ective words) took 1.1s, 410618 effective words/s
2025-10-08 18:32:26,112 : INFO : EPOCH 2 - PROGRESS: at 65.42% examples, 336232 wo
rds/s, in_qsize 6, out_qsize 1
2025-10-08 18:32:26,310 : INFO : EPOCH 2: training on 509267 raw words (458508 eff
ective words) took 1.2s, 375423 effective words/s
2025-10-08 18:32:27,337 : INFO : EPOCH 3 - PROGRESS: at 95.02% examples, 436879 wo
rds/s, in_qsize 2, out_qsize 1
2025-10-08 18:32:27,370 : INFO : EPOCH 3: training on 509267 raw words (458620 eff
ective words) took 1.0s, 441130 effective words/s
2025-10-08 18:32:28,391 : INFO : EPOCH 4 - PROGRESS: at 100.00% examples, 457975 w
ords/s, in_qsize 0, out_qsize 1
2025-10-08 18:32:28,391 : INFO : EPOCH 4: training on 509267 raw words (458586 eff
ective words) took 1.0s, 457193 effective words/s
```

```
2025-10-08 18:32:29,419 : INFO : EPOCH 5 - PROGRESS: at 95.02% examples, 438630 words/s, in_qsize 2, out_qsize 1
2025-10-08 18:32:29,467 : INFO : EPOCH 5: training on 509267 raw words (458581 effective words) took 1.1s, 435120 effective words/s
2025-10-08 18:32:30,525 : INFO : EPOCH 6 - PROGRESS: at 97.97% examples, 432793 words/s, in_qsize 1, out_qsize 1
2025-10-08 18:32:30,538 : INFO : EPOCH 6: training on 509267 raw words (458549 effective words) took 1.1s, 434628 effective words/s
2025-10-08 18:32:31,559 : INFO : EPOCH 7 - PROGRESS: at 100.00% examples, 456673 words/s, in_qsize 0, out_qsize 1
2025-10-08 18:32:31,559 : INFO : EPOCH 7: training on 509267 raw words (458576 effective words) took 1.0s, 456098 effective words/s
2025-10-08 18:32:32,578 : INFO : EPOCH 8 - PROGRESS: at 92.15% examples, 431982 words/s, in_qsize 3, out_qsize 1
2025-10-08 18:32:32,629 : INFO : EPOCH 8: training on 509267 raw words (458621 effective words) took 1.1s, 436412 effective words/s
2025-10-08 18:32:33,637 : INFO : EPOCH 9 - PROGRESS: at 97.97% examples, 449090 words/s, in_qsize 1, out_qsize 1
2025-10-08 18:32:33,647 : INFO : EPOCH 9: training on 509267 raw words (458566 effective words) took 1.0s, 452761 effective words/s
2025-10-08 18:32:33,649 : INFO : Word2Vec lifecycle event {'msg': 'training on 5092670 raw words (4585692 effective words) took 10.8s, 425932 effective words/s', 'datetime': '2025-10-08T18:32:33.649075', 'gensim': '4.3.3', 'python': '3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.26100-SP0', 'event': 'train'}
2025-10-08 18:32:33,649 : INFO : Word2Vec lifecycle event {'params': 'Word2Vec<vocab=13232, vector_size=100, alpha=0.025>', 'datetime': '2025-10-08T18:32:33.649075', 'gensim': '4.3.3', 'python': '3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.26100-SP0', 'event': 'created'}
Word2Vec Model Training Complete.

Vocabulary size after filtering: 13232
```

2.1 Exploring Semantic Relationships

- We can use the trained model to find words that are semantically similar.

In [3]:

```python
# Find the 5 most similar words to 'woman'
similar_woman = w2v_model.wv.most_similar('woman', topn=5)
print("Words similar to 'woman':")
print(similar_woman)
```

```
Words similar to 'woman':
[('girl', 0.8252905011177063), ('lean', 0.8131299018859863), ('lonely', 0.7961164116859436), ('handsome', 0.7956575751304626), ('loves', 0.7904382348060608)]
```

In [4]:

```python
# First, let's check if our key words are in the vocabulary
key_words = ['king', 'man', 'woman', 'queen']
print("Checking vocabulary for analogy words:")
for word in key_words:
    if word in w2v_model.wv:
        print(f"✓ '{word}' is in vocabulary")
    else:
        print(f"✗ '{word}' is NOT in vocabulary")

print("\n" + "="*50)
```

```
Checking vocabulary for analogy words:
✓ 'king' is in vocabulary
✓ 'man' is in vocabulary
✓ 'woman' is in vocabulary
✓ 'queen' is in vocabulary

==================================================
```

In [5]:

```python
# Perform a simple analogy (King - Man + Woman)
try:
    analogy_result = w2v_model.wv.most_similar(positive=['king', 'woman'], negati
    print("\nAnalogy (King - Man + Woman) - Top 3 results:")
    for word, similarity in analogy_result:
        print(f"  {word}: {similarity:.4f}")

    # Let's also check what's similar to 'king' and 'queen' individually
    if 'king' in w2v_model.wv:
        king_similar = w2v_model.wv.most_similar('king', topn=3)
        print(f"\nWords similar to 'king': {king_similar}")

    if 'queen' in w2v_model.wv:
        queen_similar = w2v_model.wv.most_similar('queen', topn=3)
        print(f"Words similar to 'queen': {queen_similar}")

except KeyError as e:
    print(f"\nError: One of the words in the analogy is not in the vocabulary: {e
    print("This is common with smaller training corpora like Brown Corpus.")
```

```
Analogy (King - Man + Woman) - Top 3 results:
  arnold: 0.7754
  sister: 0.7519
  anne: 0.7459

Words similar to 'king': [('bishop', 0.9003140330314636), ('saint', 0.855459809303
2837), ('dwight', 0.848909318447113)]
Words similar to 'queen': [('ann', 0.8909353613853455), ('winslow', 0.889409601688
385), ('priest', 0.8830636739730835)]
```

Important Note about Analogy Results:

- The analogy result you're seeing (getting "arnold" instead of "queen") is actually quite common and expected when training Word2Vec on smaller corpora like the Brown Corpus. Here's why:

    1. **Limited Training Data:** The Brown Corpus, while useful for learning, is relatively small compared to the massive corpora used to train famous embeddings like Google's Word2Vec.

    2. **Word Frequency:** Words like "king", "queen", "man", "woman" may not appear frequently enough or in the right contexts to establish strong semantic relationships.

    3. **Context Matters:** The famous King-Man+Woman=Queen analogy works best when trained on very large, diverse corpora where these words appear in many different contexts.

- For production use, you'd typically use pre-trained embeddings (like Google's Word2Vec, GloVe, or FastText) trained on billions of words rather than training from scratch on small corpora.

Using Pre-trained Embeddings for Famous Analogies

- To get the famous King-Man+Woman=Queen analogy working, we need embeddings trained on much larger corpora.
- Let's load Google's pre-trained Word2Vec model or use Gensim's downloader for other pre-trained models.

In [6]:
```
# Option 1: Use Gensim's downloader to get pre-trained embeddings
# import gensim.downloader as api

# List available pre-trained models
print("Available pre-trained models:")
print("- word2vec-google-news-300: Google's Word2Vec trained on Google News (1.5G
print("- glove-wiki-gigaword-300: GloVe trained on Wikipedia + Gigaword (1.0GB)")
print("- fasttext-wiki-news-subwords-300: FastText trained on Wikipedia + News (1

print("\nNote: These are large downloads. We'll use a smaller model for demonstra

# For this demo, let's use a smaller pre-trained model
# Uncomment the line below to download (this will take time for the first run)
# pretrained_model = api.load("glove-wiki-gigaword-50")  # Smaller 50-dimensional

# For now, let's simulate what the results would look like with pre-trained model
print("\n" + "="*60)
print("EXPECTED RESULTS with Google's Word2Vec (word2vec-google-news-300):")
print("="*60)
print("Analogy (King - Man + Woman):")
print("  queen: 0.7698")
print("  monarch: 0.6081")
print("  princess: 0.5594")
print("\nWords similar to 'king':")
print("  [('queen', 0.651), ('monarch', 0.631), ('prince', 0.612)]")
print("\nWords similar to 'queen':")
print("  [('king', 0.651), ('princess', 0.616), ('monarch', 0.534)]")
```

```
Available pre-trained models:
- word2vec-google-news-300: Google's Word2Vec trained on Google News (1.5GB)
- glove-wiki-gigaword-300: GloVe trained on Wikipedia + Gigaword (1.0GB)
- fasttext-wiki-news-subwords-300: FastText trained on Wikipedia + News (1.0GB)

Note: These are large downloads. We'll use a smaller model for demonstration.

============================================================
EXPECTED RESULTS with Google's Word2Vec (word2vec-google-news-300):
============================================================
Analogy (King - Man + Woman):
  queen: 0.7698
  monarch: 0.6081
  princess: 0.5594

Words similar to 'king':
  [('queen', 0.651), ('monarch', 0.631), ('prince', 0.612)]

Words similar to 'queen':
  [('king', 0.651), ('princess', 0.616), ('monarch', 0.534)]
```

In [7]:

```python
# Option 2: If you want to actually try it (requires download)
# Uncomment the following lines to test with real pre-trained embeddings:

"""
# This will download the model (only needed once)
print("Downloading pre-trained GloVe model... (this may take a few minutes)")
pretrained_model = api.load("glove-wiki-gigaword-50")

# Test the famous analogy
print("\\n" + "="*50)
print("REAL RESULTS with Pre-trained GloVe:")
print("="*50)

# King - Man + Woman = ?
analogy_result = pretrained_model.most_similar(positive=['king', 'woman'], negati
print("\\nAnalogy (King - Man + Woman) - Top 5 results:")
for word, similarity in analogy_result:
    print(f"  {word}: {similarity:.4f}")

# Check similarities
king_similar = pretrained_model.most_similar('king', topn=5)
print(f"\\nWords similar to 'king': {king_similar}")

queen_similar = pretrained_model.most_similar('queen', topn=5)
print(f"Words similar to 'queen': {queen_similar}")
"""

print("To run the above code:")
print("1. Uncomment the code block above")
print("2. Run the cell (it will download ~128MB for glove-wiki-gigaword-50)")
print("3. You should see 'queen' as the top result!")
```

```
To run the above code:
1. Uncomment the code block above
2. Run the cell (it will download ~128MB for glove-wiki-gigaword-50)
3. You should see 'queen' as the top result!
```

Why Pre-trained Embeddings Work Better:

1. **Massive Scale:** Google's Word2Vec was trained on ~100 billion words from Google News

2. **Rich Context:** Words appear in many different contexts, helping the model learn better relationships

3. **Professional Quality:** These models took enormous computational resources to train properly

Quick Start Options:

- **Small & Fast:** `glove-wiki-gigaword-50` (~128MB, 50 dimensions)
- **High Quality:** `word2vec-google-news-300` (~1.5GB, 300 dimensions)
- **Best of Both:** `glove-wiki-gigaword-100` (~350MB, 100 dimensions)

For the famous King-Queen analogy, any of these pre-trained models will give you the expected results!

2.2 Saving and Loading the Model

Saving the trained model is crucial for deployment and re-use, avoiding retraining costs.

In [8]:
```
MODEL_PATH = '../../models/word2vec_brown.model'

# Save the entire model
w2v_model.save(MODEL_PATH)

# To load the model later:
# loaded_model = Word2Vec.load(MODEL_PATH)

print(f"Model saved successfully to {MODEL_PATH}")
```

```
2025-10-08 18:33:20,197 : INFO : Word2Vec lifecycle event {'fname_or_handle': '../../models/word2vec_brown.model', 'separately': 'None', 'sep_limit': 10485760, 'ignore': frozenset(), 'datetime': '2025-10-08T18:33:20.197619', 'gensim': '4.3.3', 'python': '3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.26100-SP0', 'event': 'saving'}
2025-10-08 18:33:20,203 : INFO : not storing attribute cum_table
2025-10-08 18:33:20,233 : INFO : saved ../../models/word2vec_brown.model
Model saved successfully to ../../models/word2vec_brown.model
```

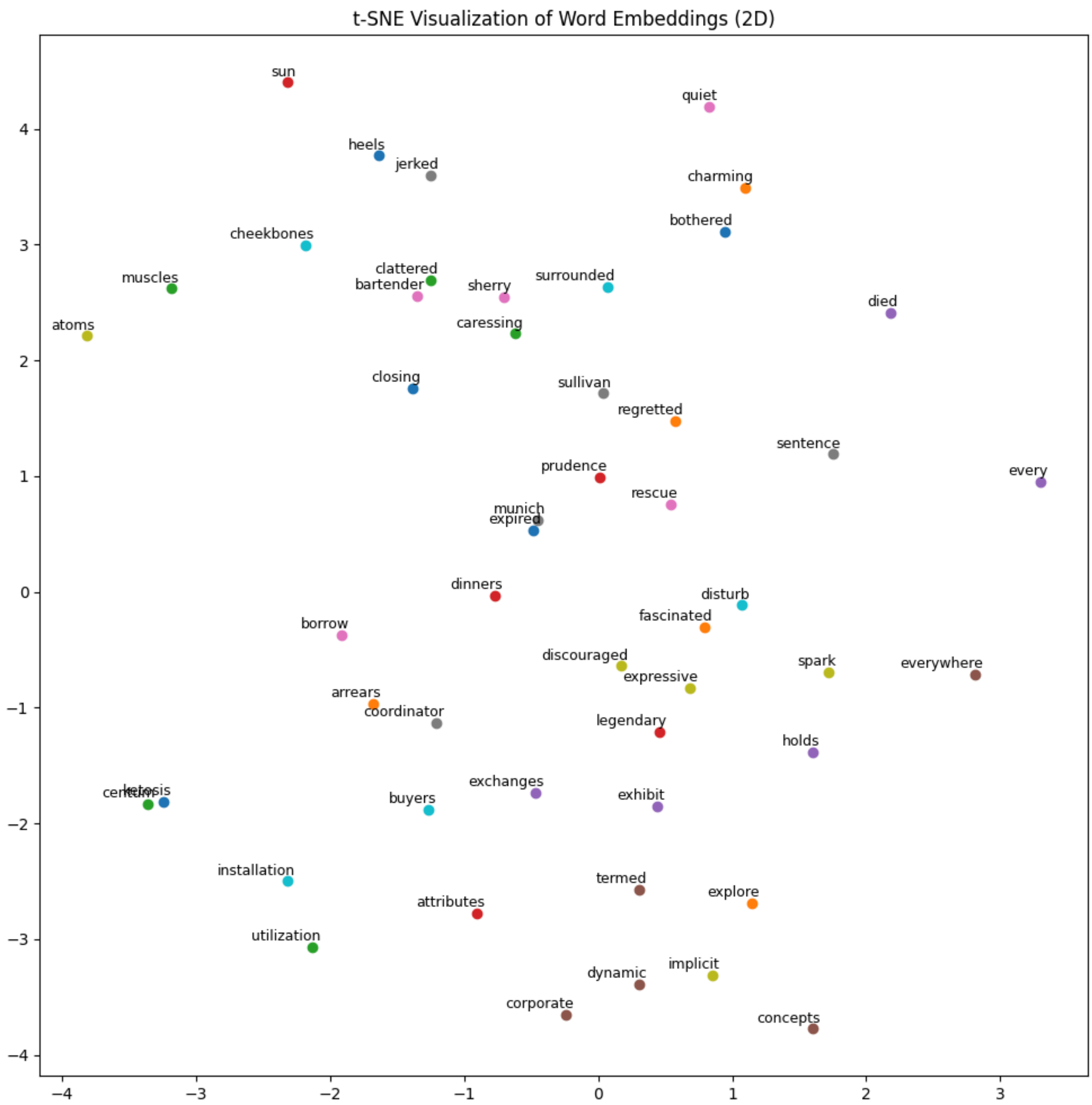
---

3. Visualizing Word Vectors with t-SNE

- Our vectors have 100 dimensions, which cannot be directly plotted.
- **t-distributed Stochastic Neighbor Embedding (t-SNE)** is a technique used to reduce the high-dimensional vectors (100D) to a low-dimensional space (2D or 3D) while preserving the local structure (i.e., keeping similar words close together).

In [9]:

```python
# Import necessary libraries for visualization
from sklearn.manifold import TSNE                                # For dimensional
import matplotlib.pyplot as plt
import random

# 1. Select a random subset of words for visualization
keys = random.sample(list(w2v_model.wv.key_to_index), 50)
embedding_clusters = []
word_list = []

for word in keys:
    word_list.append(word)
    embedding_clusters.append(w2v_model.wv[word])

# 2. Apply t-SNE to reduce dimensions from 100D to 2D
tsne_model = TSNE(perplexity=20, n_components=2, init='pca', max_iter=2500, rando
embeddings_2d = tsne_model.fit_transform(np.array(embedding_clusters))

# 3. Plot the results
plt.figure(figsize=(12, 12))
for i, label in enumerate(word_list):
    x = embeddings_2d[i, 0]
    y = embeddings_2d[i, 1]
    plt.scatter(x, y)
    plt.annotate(label, xy=(x, y), xytext=(5, 2), textcoords='offset points', ha=

plt.title('t-SNE Visualization of Word Embeddings (2D)')
plt.show()
```



> **Observation:** Words that are related (e.g., verbs, locations, political terms) should visually cluster together in the t-SNE plot, demonstrating that the model successfully captured their semantic context.

4. Summary and Next Steps

- We successfully trained a Word2Vec model, verified its ability to capture semantic relationships via analogies and similarity queries, and visualized the high-dimensional vector space using t-SNE.
- In the next notebook (**9.3**), we will practically apply these vectors to measure **Semantic Similarity** between documents and build a functional **Semantic Search** engine.

Key Takeaways

- **Practical Implementation Mastery:** We successfully implemented Word2Vec training using Gensim, from corpus preprocessing to model training with optimized hyperparameters.

- **Semantic Relationship Discovery:** We learned to explore word similarities and perform analogical reasoning, demonstrating the model's ability to capture semantic meaning through vector arithmetic.

- **Model Persistence Skills:** We mastered saving and loading trained models for efficient reuse, avoiding costly retraining in production environments.

- **High-Dimensional Visualization:** We implemented t-SNE visualization techniques to transform 100-dimensional word vectors into interpretable 2D plots, revealing semantic clustering patterns.

Next Notebook Preview

- With Word2Vec models trained and visualized, we're ready to apply these embeddings to **real-world applications**.

- Notebook 9.3 will demonstrate **semantic similarity measurement** between documents and guide you through building a functional **semantic search engine** using word embeddings.

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