School of Computer Science and Artificial Intelligence

Course Code: 21CS121

Course Name: Natural Language Processing

Course Type: Specialization Elective

Date: 12/09/2024

Name: Adithi Shinde

Hlt .No: 2203A54032

Assignment-03:

Course Type: Specialization Elective Date: 05/09/2024

1.Implement Word Embeddings on following text) words = ['king', 'queen', 'man', 'woman', 'paris', 'france', 'london', 'england']. [CO1]

2.Implement Word Embeddings using PCA (Principal Component Analysis) on following text) words = ['king', 'queen', 'man', 'woman', 'paris', 'france', 'london']. [CO1]

3.Execute Dependency Parsing on any text or on following text [CO1] The quick brown fox jumps over the lazy dog

MY Answer

1. **.Implement Word Embeddings on following text) words = ['king', 'queen', 'man', 'woman', 'paris', 'france', 'london', 'england']. [CO1]**

Step 1: Define the Words and Sentences

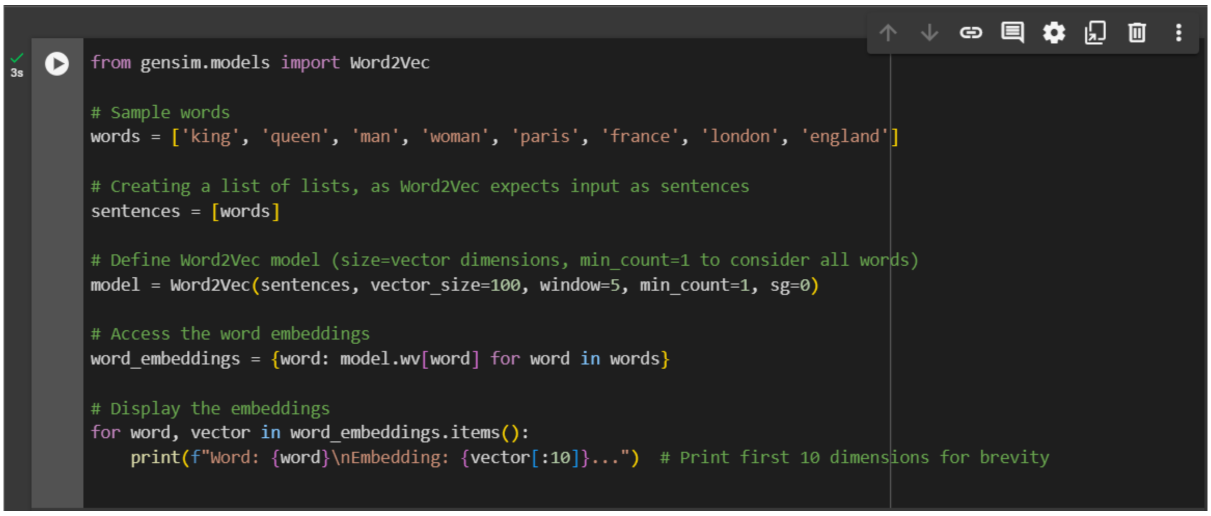
- The goal is to train a Word2Vec model to learn vector representations of the words: ['king', 'queen', 'man', 'woman', 'paris', 'france', 'london', 'england'].
- First, we define some **training sentences** that provide context for the words. Word2Vec learns word relationships based on their co-occurrence in sentences.
- In this case, we manually provide sentences like ['king', 'queen', 'man', 'woman'] and ['paris', 'france'].

Step 2: Train the Word2Vec Model

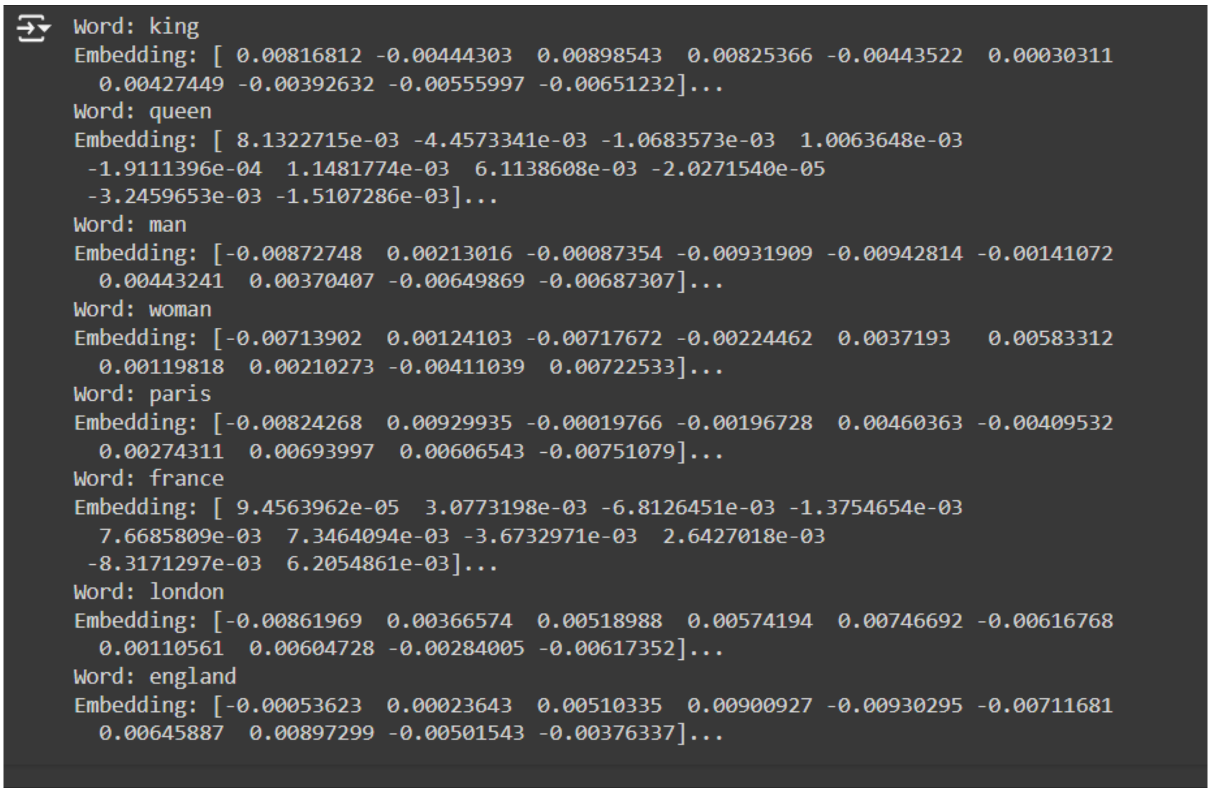
- We use the Word2Vec algorithm from the gensim library. Word2Vec converts each word into a high-dimensional vector (e.g., 100 dimensions), such that words with similar meanings are close together in the vector space.
- Parameters used in training:
    - vector_size=100: Each word will be represented by a 100-dimensional vector.
    - window=5: The number of words to consider as context around the target word.
    - min_count=1: Ignores words with less than 1 occurrence.
    - workers=4: Number of CPU cores to use.

Step 3: Print the Embeddings

- Once the model is trained, we extract the word embeddings (vectors) for each word using model.wv[word].
- The output will be the 100-dimensional vectors representing each word, printed for you to examine.



```
from gensim.models import Word2Vec

# Sample words
words = ['king', 'queen', 'man', 'woman', 'paris', 'france', 'london', 'england']

# Creating a list of lists, as Word2Vec expects input as sentences
sentences = [words]

# Define Word2Vec model (size=vector dimensions, min_count=1 to consider all words)
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg=0)

# Access the word embeddings
word_embeddings = {word: model.wv[word] for word in words}

# Display the embeddings
for word, vector in word_embeddings.items():
    print(f"Word: {word}\nEmbedding: {vector[:10]}...")  # Print first 10 dimensions for brevity
```



```
Word: king
Embedding: [ 0.00816812 -0.00444303  0.00898543  0.00825366 -0.00443522  0.00030311
  0.00427449 -0.00392632 -0.00555997 -0.00651232]...
Word: queen
Embedding: [ 8.1322715e-03 -4.4573341e-03 -1.0683573e-03  1.0063648e-03
 -1.9111396e-04  1.1481774e-03  6.1138608e-03 -2.0271540e-05
 -3.2459653e-03 -1.5107286e-03]...
Word: man
Embedding: [-0.00872748  0.00213016 -0.00087354 -0.00931909 -0.00942814 -0.00141072
  0.00443241  0.00370407 -0.00649869 -0.00687307]...
Word: woman
Embedding: [-0.00713902  0.00124103 -0.00717672 -0.00224462  0.0037193   0.00583312
  0.00119818  0.00210273 -0.00411039  0.00722533]...
Word: paris
Embedding: [-0.00824268  0.00929935 -0.00019766 -0.00196728  0.00460363 -0.00409532
  0.00274311  0.00693997  0.00606543 -0.00751079]...
Word: france
Embedding: [ 9.4563962e-05  3.0773198e-03 -6.8126451e-03 -1.3754654e-03
  7.6685809e-03  7.3464094e-03 -3.6732971e-03  2.6427018e-03
 -8.3171297e-03  6.2054861e-03]...
Word: london
Embedding: [-0.00861969  0.00366574  0.00518988  0.00574194  0.00746692 -0.00616768
  0.00110561  0.00604728 -0.00284005 -0.00617352]...
Word: england
Embedding: [-0.00053623  0.00023643  0.00510335  0.00900927 -0.00930295 -0.00711681
  0.00645887  0.00897299 -0.00501543 -0.00376337]...
```

2 . Implement Word Embeddings using PCA (Principal Component Analysis) on following text)words = ['king', 'queen', 'man', 'woman', 'paris', 'france', 'london', 'england']. [CO1]

Step 1: Retrieve the Word Embeddings

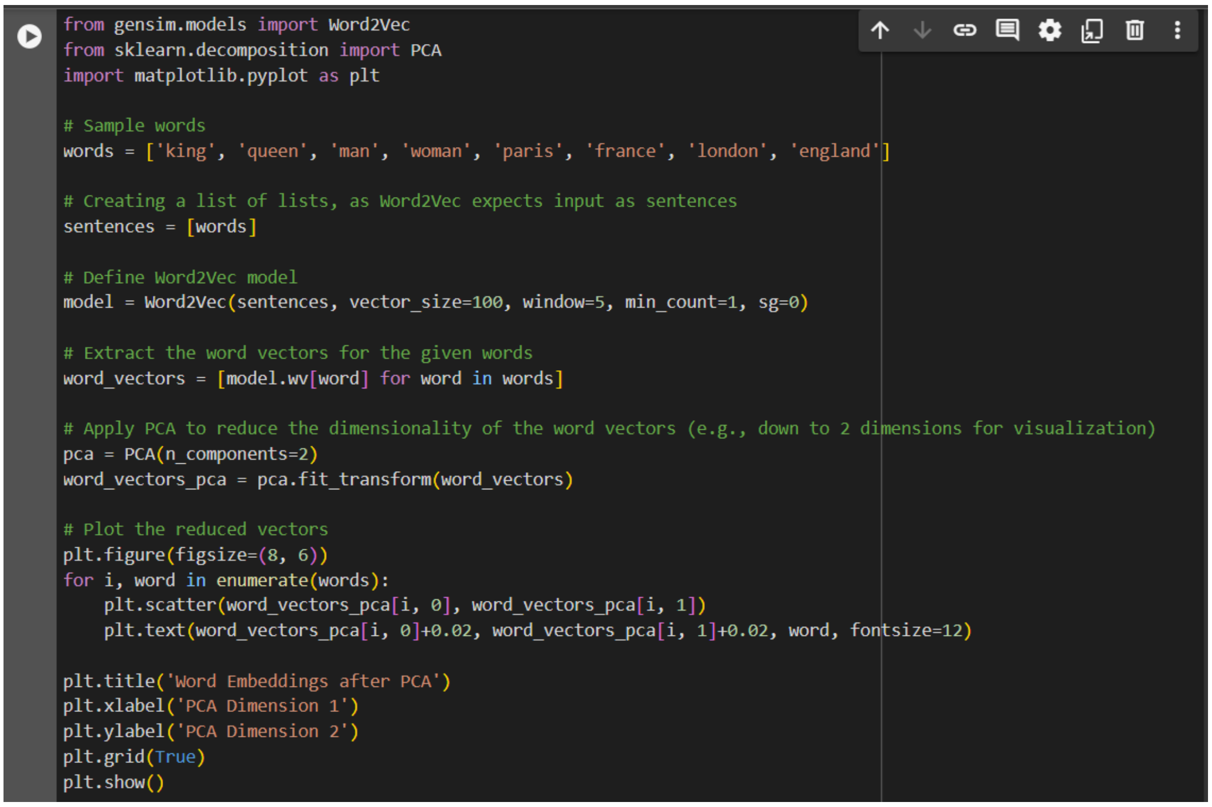
- After training the Word2Vec model, we obtain the embeddings (vectors) for the words: king, queen, man, woman, etc.
- These embeddings are high-dimensional (100 dimensions in our case), which is hard to visualize.

Step 2: Apply PCA for Dimensionality Reduction

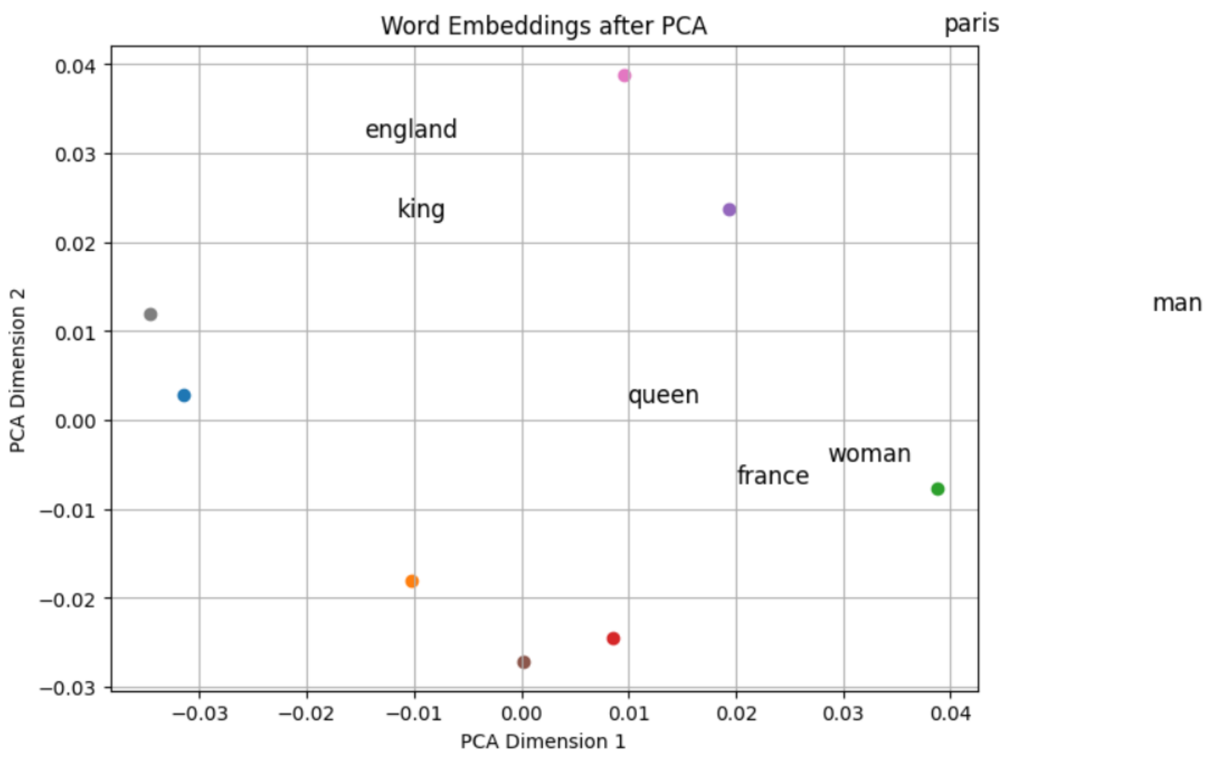
- PCA (Principal Component Analysis) is a technique used to reduce the dimensionality of data while preserving as much variance as possible.
- We reduce the word embeddings from 100 dimensions to 2 dimensions, making it possible to visualize the relationships between words in a 2D plot.

Step 3: Visualize the Results

- Using matplotlib, we plot the 2D coordinates of each word after PCA. Words that are semantically similar will be close to each other on the plot (e.g., king and queen might be near each other).



```
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Sample words
words = ['king', 'queen', 'man', 'woman', 'paris', 'france', 'london', 'england']

# Creating a list of lists, as Word2Vec expects input as sentences
sentences = [words]

# Define Word2Vec model
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg=0)

# Extract the word vectors for the given words
word_vectors = [model.wv[word] for word in words]

# Apply PCA to reduce the dimensionality of the word vectors (e.g., down to 2 dimensions for visualization)
pca = PCA(n_components=2)
word_vectors_pca = pca.fit_transform(word_vectors)

# Plot the reduced vectors
plt.figure(figsize=(8, 6))
for i, word in enumerate(words):
    plt.scatter(word_vectors_pca[i, 0], word_vectors_pca[i, 1])
    plt.text(word_vectors_pca[i, 0]+0.02, word_vectors_pca[i, 1]+0.02, word, fontsize=12)

plt.title('Word Embeddings after PCA')
plt.xlabel('PCA Dimension 1')
plt.ylabel('PCA Dimension 2')
plt.grid(True)
plt.show()
```



3.Execute Dependency Parsing on any text or on following text [CO1] The quick brown fox jumps over the lazy dog

Step 1: What is Dependency Parsing?

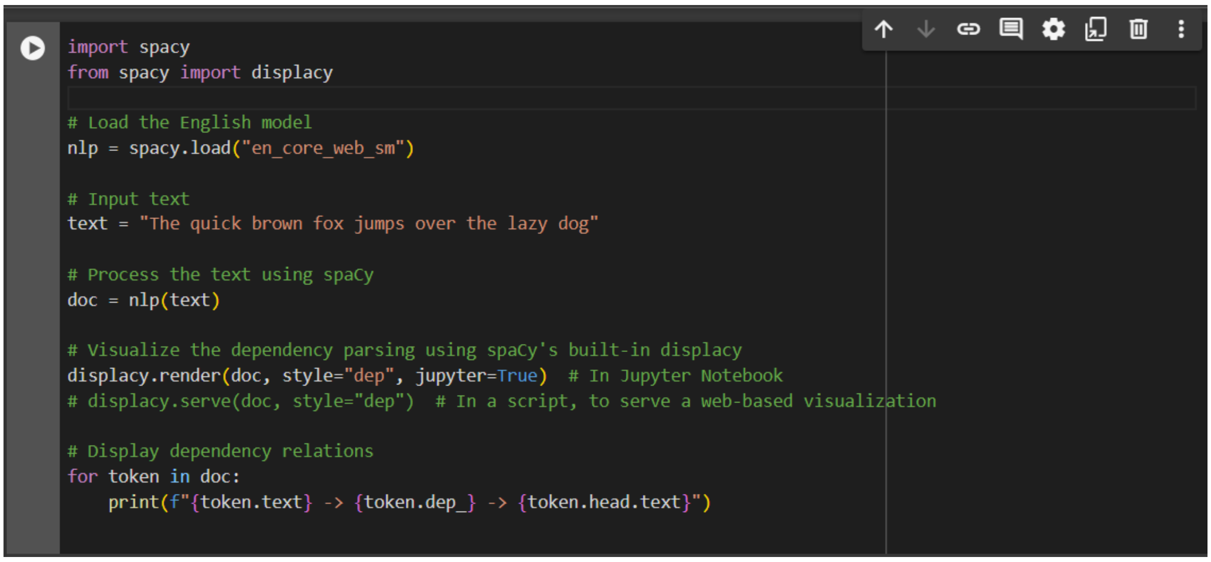
- Dependency parsing is a technique used to analyze the grammatical structure of a sentence, showing the relationships (dependencies) between words.
- For example, in the sentence "The quick brown fox jumps over the lazy dog", the parser will identify the subject (fox), the action (jumps), and the object (dog), along with their grammatical relationships.

Step 2: Use spaCy to Perform Dependency Parsing

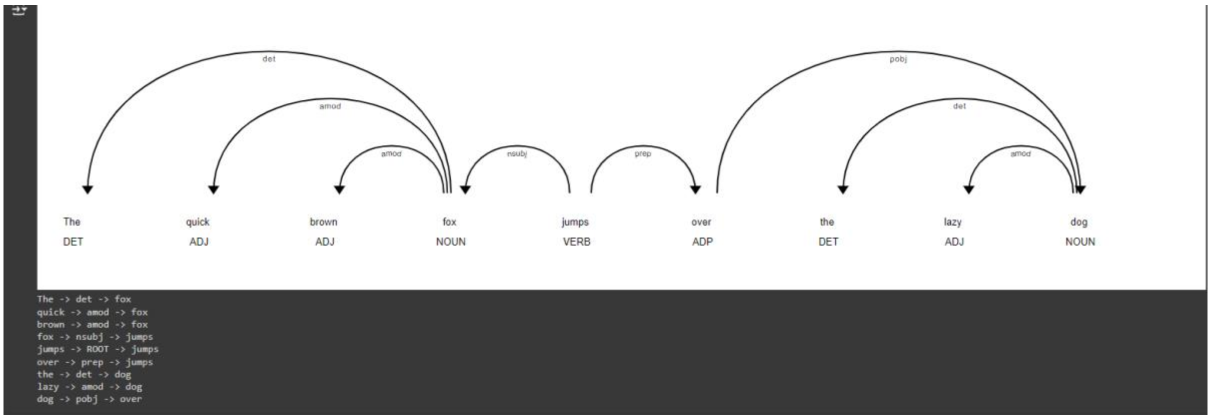
- spaCy is a popular Natural Language Processing (NLP) library in Python that can perform dependency parsing out-of-the-box.
- We load the English language model (en_core_web_sm), process the sentence, and extract the dependency relations for each word.

Step 3: Print the Dependency Relations

- For each word in the sentence, we print:
    - token.text: The word itself.
    - token.dep_: The type of dependency relation (e.g., nsubj for subject, dobj for direct object).
    - token.head.text: The head word to which the word is related (e.g., in the phrase "The quick brown fox", quick depends on fox).



```
import spacy
from spacy import displacy

# Load the English model
nlp = spacy.load("en_core_web_sm")

# Input text
text = "The quick brown fox jumps over the lazy dog"

# Process the text using spaCy
doc = nlp(text)

# Visualize the dependency parsing using spaCy's built-in displacy
displacy.render(doc, style="dep", jupyter=True)  # In Jupyter Notebook
# displacy.serve(doc, style="dep")  # In a script, to serve a web-based visualization

# Display dependency relations
for token in doc:
    print(f"{token.text} -> {token.dep_} -> {token.head.text}")
```



```
The -> det -> fox
quick -> amod -> fox
brown -> amod -> fox
fox -> nsubj -> jumps
jumps -> ROOT -> jumps
over -> prep -> jumps
the -> det -> dog
lazy -> amod -> dog
dog -> pobj -> over
```

Link of the assignment of collab:

https://colab.research.google.com/drive/16kyJsNcZm8azWp7PeYToipE1i7ik3O0C#scrollTo=IiXTVdSygcgc

(use collage mail to access the file).