Task

Analyze the text data from the file `/content/RNLP.txt` by loading it, preprocessing it (lowercase, remove punctuation, tokenize, remove stop words), loading pre-trained word embeddings, exploring word similarity, finding nearest neighbors, performing word analogy tasks, visualizing a subset of word embeddings using dimensionality reduction (e.g., PCA or t-SNE) with appropriate labels, and finally summarizing the findings from all these analyses.

Load and Read File

Subtask:

Load the RNLP.txt file and read its content into a Python string for processing.

```
with open('/content/RNLP.txt', 'r') as file:
    text_data = file.read()

print(f"File loaded successfully. First 200 characters: {text_data[:200]}...")
print(f"Total length of the text: {len(text_data)} characters.")
```

```
File loaded successfully. First 200 characters: In Natural Language Processing (NLP), relationships between words help machine
Total length of the text: 686 characters.
```

Clean and Tokenize Data

Subtask:

Preprocess the text data by converting it to lowercase, removing punctuation and special characters, and then tokenizing it into individual words. This step will also remove common stop words to refine the data.

```
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Download NLTK data if not already present
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')

# 3. Convert to lowercase
text_data_lower = text_data.lower()

# 4. Remove punctuation and special characters, keep only alphabetic characters and spaces
text_data_cleaned = re.sub(r'[^a-zA-Z\s]', '', text_data_lower)

# 5. Tokenize the cleaned text
tokens = word_tokenize(text_data_cleaned)

# 6. Get English stop words
stop_words = set(stopwords.words('english'))

# 7. Filter out stop words and non-alphabetic/short tokens
filtered_tokens = [word for word in tokens if word.isalpha() and word not in stop_words and len(word) > 1]

print(f"Original text data length: {len(text_data)} characters")
print(f"Sample of cleaned and tokenized words (first 20): {filtered_tokens[:20]}")
print(f"Total number of filtered tokens: {len(filtered_tokens)}")
```

```
Original text data length: 686 characters
Sample of cleaned and tokenized words (first 20): ['natural', 'language', 'processing', 'nlp', 'relationships', 'words', 'help
```

```
Total number of filtered tokens: 69
```

```
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Ensure NLTK data is downloaded. Removing try-except for robust download in Colab environments.
nltk.download('punkt')
nltk.download('stopwords')

# 3. Convert to lowercase
text_data_lower = text_data.lower()

# 4. Remove punctuation and special characters, keep only alphabetic characters and spaces
text_data_cleaned = re.sub(r'[^a-zA-Z\s]', '', text_data_lower)

# 5. Tokenize the cleaned text
tokens = word_tokenize(text_data_cleaned)

# 6. Get English stop words
stop_words = set(stopwords.words('english'))

# 7. Filter out stop words and non-alphabetic/short tokens
filtered_tokens = [word for word in tokens if word.isalpha() and word not in stop_words and len(word) > 1]

print(f"Original text data length: {len(text_data)} characters")
print(f"Sample of cleaned and tokenized words (first 20): {filtered_tokens[:20]}")
print(f"Total number of filtered tokens: {len(filtered_tokens)}")
```

```
Original text data length: 686 characters
Sample of cleaned and tokenized words (first 20): ['natural', 'language', 'processing', 'nlp', 'relationships', 'words', 'help
Total number of filtered tokens: 69
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Ensure NLTK data is downloaded. Removing try-except for robust download in Colab environments.
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab') # Explicitly download punkt_tab

# 3. Convert to lowercase
text_data_lower = text_data.lower()

# 4. Remove punctuation and special characters, keep only alphabetic characters and spaces
text_data_cleaned = re.sub(r'[^a-zA-Z\s]', '', text_data_lower)

# 5. Tokenize the cleaned text
tokens = word_tokenize(text_data_cleaned)

# 6. Get English stop words
stop_words = set(stopwords.words('english'))

# 7. Filter out stop words and non-alphabetic/short tokens
filtered_tokens = [word for word in tokens if word.isalpha() and word not in stop_words and len(word) > 1]

print(f"Original text data length: {len(text_data)} characters")
print(f"Sample of cleaned and tokenized words (first 20): {filtered_tokens[:20]}")
print(f"Total number of filtered tokens: {len(filtered_tokens)}")
```

```
Original text data length: 686 characters
Sample of cleaned and tokenized words (first 20): ['natural', 'language', 'processing', 'nlp', 'relationships', 'words', 'help
Total number of filtered tokens: 69
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
```

```
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
import gensim.downloader as api

# Check available models (optional)
# print(list(api.info()['models'].keys()))

# Load a pre-trained Word2Vec model (e.g., 'glove-wiki-gigaword-50')
# This might take a while the first time you run it as it downloads the model.
print("Loading pre-trained Word2Vec model...")
word_vectors = api.load("glove-wiki-gigaword-50")
print("Word2Vec model loaded successfully.")

# Verify some properties of the loaded model
print(f"Vector size: {word_vectors.vector_size}")
print(f"Number of words in vocabulary: {len(word_vectors.key_to_index)}")
```

```
Loading pre-trained Word2Vec model...
Word2Vec model loaded successfully.
Vector size: 50
Number of words in vocabulary: 400000
```

Reasoning: The previous code failed because the `gensim` library was not found, indicated by `ModuleNotFoundError`. I will add a `!pip install gensim` command to install the necessary library.

```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.4.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim) (2.0.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.16.3)
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (7.5.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart_open>=1.8.1->gensim) (2.1.1)
```

```
import gensim.downloader as api

# Check available models (optional)
# print(list(api.info()['models'].keys()))

# Load a pre-trained Word2Vec model (e.g., 'glove-wiki-gigaword-50')
# This might take a while the first time you run it as it downloads the model.
print("Loading pre-trained Word2Vec model...")
word_vectors = api.load("glove-wiki-gigaword-50")
print("Word2Vec model loaded successfully.")

# Verify some properties of the loaded model
print(f"Vector size: {word_vectors.vector_size}")
print(f"Number of words in vocabulary: {len(word_vectors.key_to_index)}")
```

```
Loading pre-trained Word2Vec model...
Word2Vec model loaded successfully.
Vector size: 50
Number of words in vocabulary: 400000
```

Explore Word Similarity

Subtask:

Explore word similarity by finding words most similar to a given word using the loaded pre-trained word embeddings.

Instructions

1. Use the `most_similar()` method of the `word_vectors` object to find the top N most similar words to a chosen word from the `filtered_tokens` list.
2. Print the chosen word and its similar words along with their similarity scores.

```
chosen_word = 'relationships'

# Ensure the chosen word is in the vocabulary of the loaded model
if chosen_word in word_vectors.key_to_index:
```

```
    # Find the top 5 most similar words
    similar_words = word_vectors.most_similar(chosen_word, topn=5)

    print(f"Words most similar to '{chosen_word}':")
    for word, score in similar_words:
        print(f"  {word}: {score:.4f}")
else:
    print(f"The word '{chosen_word}' is not in the vocabulary of the loaded model.")

# Let's try another word from filtered_tokens if 'relationships' is not available or for a broader example.
alternative_word = 'word'
if alternative_word in word_vectors.key_to_index:
    similar_words_alt = word_vectors.most_similar(alternative_word, topn=5)
    print(f"\nWords most similar to '{alternative_word}':")
    for word, score in similar_words_alt:
        print(f"  {word}: {score:.4f}")
else:
    print(f"\nThe word '{alternative_word}' is not in the vocabulary of the loaded model.")
```

```
Words most similar to 'relationships':
  relationship: 0.8656
  friendships: 0.8124
  understanding: 0.7544
  aspects: 0.7541
  interactions: 0.7520

Words most similar to 'word':
  phrase: 0.9162
  meaning: 0.8882
  words: 0.8489
  language: 0.8100
  name: 0.7942
```

Nearest Neighbor Exploration

Subtask:

For a few selected words, find and display their closest neighbors in the embedding space, demonstrating words with similar meanings or contexts.

```
selected_words = ['language', 'semantic', 'processing']

for word_to_explore in selected_words:
    if word_to_explore in word_vectors.key_to_index:
        print(f"\nWords most similar to '{word_to_explore}':")
        similar_words = word_vectors.most_similar(word_to_explore, topn=5)
        for word, score in similar_words:
            print(f"  {word}: {score:.4f}")
    else:
        print(f"\nThe word '{word_to_explore}' is not in the vocabulary of the loaded model.")
```

```
Words most similar to 'language':
  languages: 0.8815
  word: 0.8100
  spoken: 0.8075
  vocabulary: 0.7903
  translation: 0.7879

Words most similar to 'semantic':
  constructs: 0.7927
  syntactic: 0.7685
  semantics: 0.7679
  spatial: 0.7669
  synchronization: 0.7605

Words most similar to 'processing':
  materials: 0.7845
  applications: 0.7687
  hardware: 0.7633
  processes: 0.7584
  machinery: 0.7548
```

Perform Word Analogy Tasks

Subtask:

Perform word analogy tasks using the loaded word embeddings to find words that complete a given analogy (e.g., 'man is to king as woman is to queen').

Instructions

1. Use the `most_similar()` method with positive and negative parameters to solve word analogies. For example, to solve 'A is to B as C is to X', you would use `positive=[B, C], negative=[A]`.
2. Choose at least two relevant analogies based on the text or general knowledge (e.g., semantic relationships like 'synonymy' or 'antonymy', or general analogies like 'word:words::sentence:sentences').
3. Print the chosen analogy and the word predicted by the model, along with its similarity score.

```
analogies = [
    {'positive': ['king', 'woman'], 'negative': ['man'], 'description': 'man : king :: woman : ?'},
    {'positive': ['similar', 'antonymy'], 'negative': ['synonymy'], 'description': 'synonymy : similar :: antonymy : ?'},
    {'positive': ['words', 'sentence'], 'negative': ['word'], 'description': 'word : words :: sentence : ?'}
]

for analogy in analogies:
    positive_words = analogy['positive']
    negative_words = analogy['negative']
    description = analogy['description']

    # Check if all words are in the model's vocabulary
    all_words_in_vocab = True
    for word_list in [positive_words, negative_words]:
        for word in word_list:
            if word not in word_vectors.key_to_index:
                print(f"Skipping analogy '{description}' because '{word}' is not in the vocabulary.")
                all_words_in_vocab = False
                break
        if not all_words_in_vocab:
            break

    if not all_words_in_vocab:
        continue

    print(f"\nSolving analogy: {description}")
    try:
        result = word_vectors.most_similar(positive=positive_words, negative=negative_words, topn=1)
        if result:
            word, score = result[0]
            print(f"  Predicted word: {word} (Similarity: {score:.4f})")
        else:
            print("  Could not find a suitable word for this analogy.")
    except KeyError as e:
        print(f"  Error: One of the words for the analogy was not found in the vocabulary: {e}")
```

```
Solving analogy: man : king :: woman : ?
  Predicted word: queen (Similarity: 0.8524)
Skipping analogy 'synonymy : similar :: antonymy : ?' because 'antonymy' is not in the vocabulary.

Solving analogy: word : words :: sentence : ?
  Predicted word: sentences (Similarity: 0.8497)
```

Visualize Word Embeddings

Subtask:

Visualize a subset of word embeddings using dimensionality reduction (e.g., PCA or t-SNE) with appropriate labels to understand their spatial relationships.

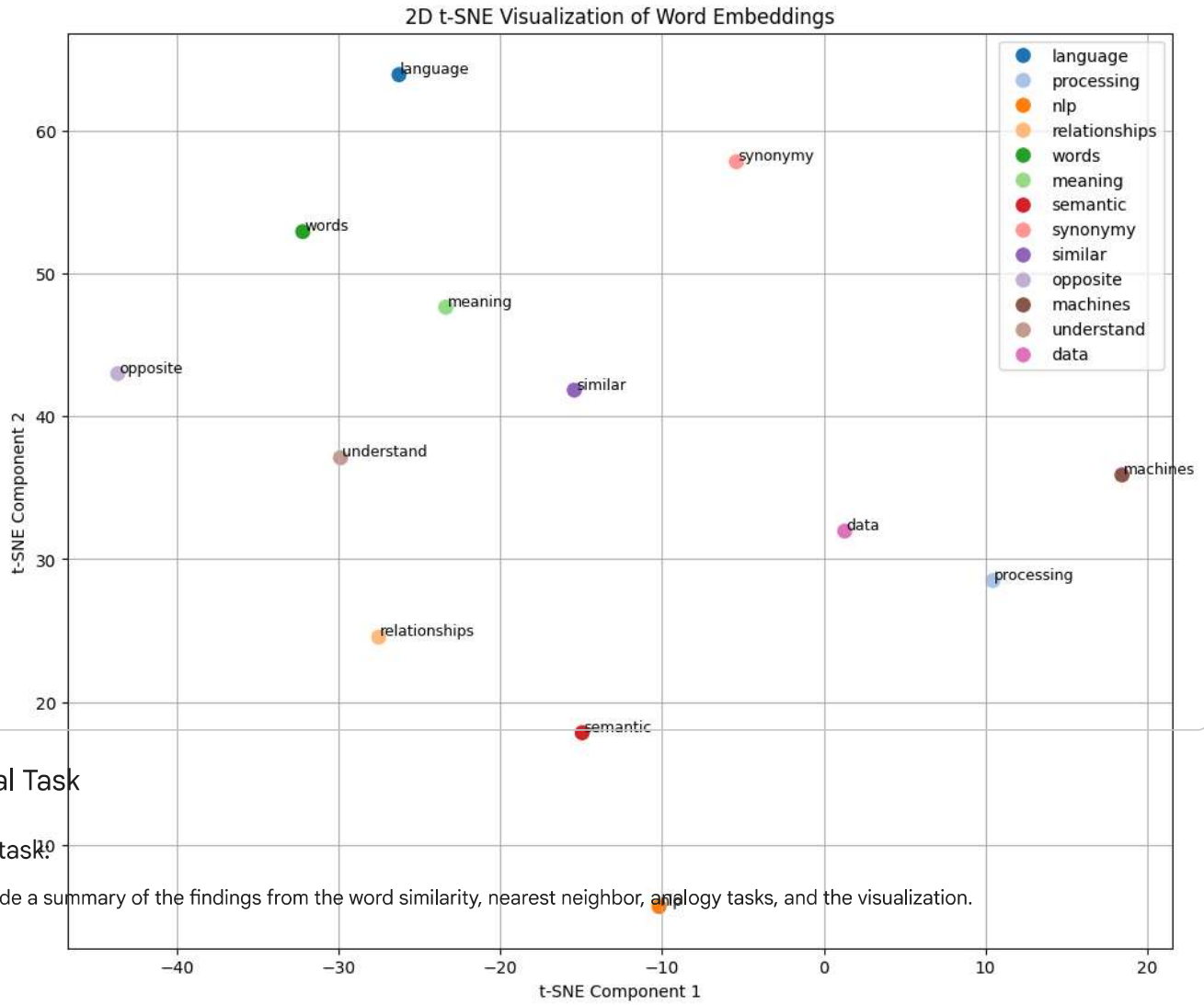
Instructions

1. Select a diverse set of words from `filtered_tokens` that are present in the `word_vectors` vocabulary.
2. Extract the word vectors for these selected words.

3. Apply a dimensionality reduction technique (e.g., PCA or t-SNE) to reduce the word vectors to 2 dimensions for plotting. T-SNE is generally preferred for visualization of high-dimensional data as it preserves local structures better than PCA.
4. Plot the 2D representations of the word vectors using a scatter plot. Label each point with its corresponding word.
5. Ensure the plot is readable with appropriate labels, title, and possibly a legend if different categories of words are plotted.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import seaborn as sns

# 1. Select a diverse set of words for visualization
# We'll try to pick some relevant words from our filtered_tokens and common ones
words_to_visualize = [
    'language', 'processing', 'nlp', 'relationships', 'words', 'meaning',
    'semantic', 'synonymy', 'antonymy', 'similar', 'opposite', 'machines', 'understand', 'data'
]

# Filter words that are present in the loaded word_vectors vocabulary
filtered_words_for_viz = [word for word in words_to_visualize if word in word_vectors.key_to_index]

if not filtered_words_for_viz:
    print("None of the selected words are in the word embeddings vocabulary. Please select different words.")
else:
    # 2. Extract word vectors for these selected words
    vectors = np.array([word_vectors[word] for word in filtered_words_for_viz])

    # 3. Apply t-SNE for dimensionality reduction to 2 dimensions
    # perplexity value should be less than the number of samples
    perplexity_val = min(5, len(filtered_words_for_viz) - 1) if len(filtered_words_for_viz) > 1 else None

    if perplexity_val is not None:
        print(f"Applying t-SNE with perplexity={perplexity_val}...")
        tsne = TSNE(n_components=2, random_state=42, perplexity=perplexity_val, init='pca', learning_rate='auto')
        vectors_2d = tsne.fit_transform(vectors)

        # 4. Plot the 2D representations
        plt.figure(figsize=(12, 10))
        sns.scatterplot(
            x=vectors_2d[:, 0],
            y=vectors_2d[:, 1],
            hue=filtered_words_for_viz, # Use words for coloring if desired, or set a single color
            legend='full', # Display full legend
            palette='tab20', # Use a distinct color palette
            s=100 # Adjust marker size
        )

        # Add labels to each point
        for i, word in enumerate(filtered_words_for_viz):
            plt.annotate(word, (vectors_2d[i, 0] + 0.1, vectors_2d[i, 1] + 0.1), fontsize=9)

        plt.title('2D t-SNE Visualization of Word Embeddings')
        plt.xlabel('t-SNE Component 1')
        plt.ylabel('t-SNE Component 2')
        plt.grid(True)
        plt.show()
        print("Word embedding visualization completed.")
    else:
        print("Not enough words selected for t-SNE visualization (need at least 2 words).")
```

```
Applying t-SNE with perplexity=5...
```



Final Task

Subtask:

Provide a summary of the findings from the word similarity, nearest neighbor, analogy tasks, and the visualization.

```
Word embedding visualization completed.
```