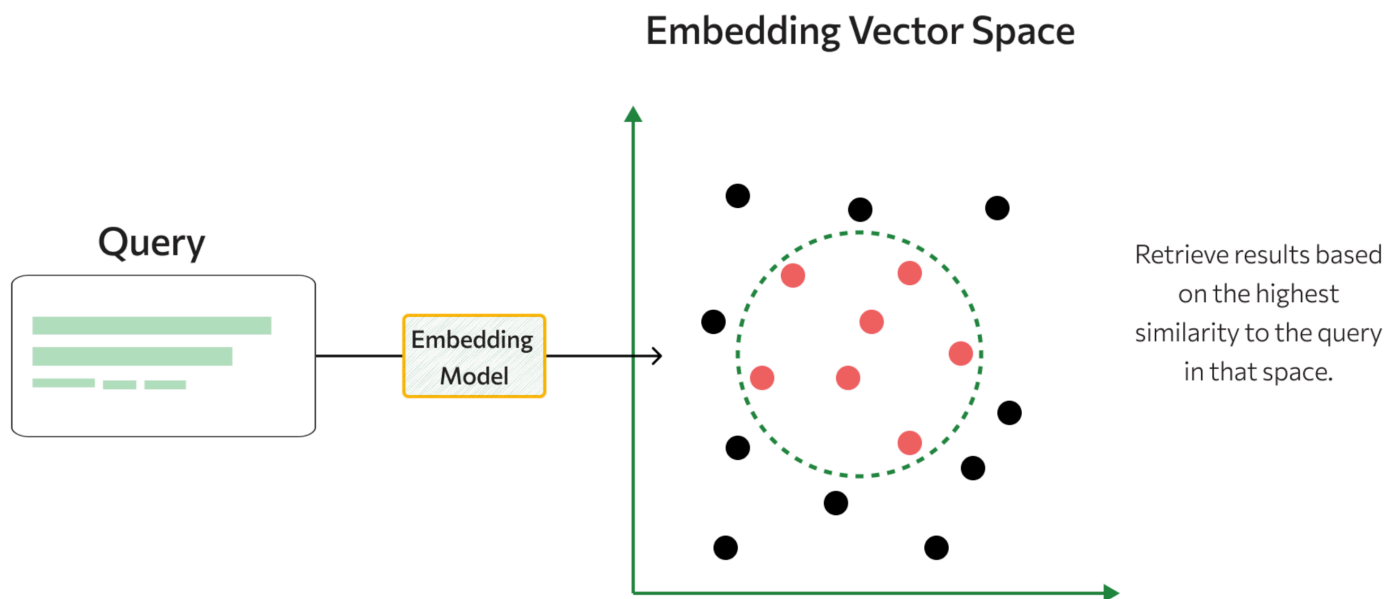


Demo Exercise - Understanding Vector Space Models

In this exercise, we'll experiment with how different types of queries affect the our toy semantic search Word2Vec model. In this notebook, we'll build that model again from scratch, using the top 1000 passages that we downloaded about the transatlantic slave trade.

In order to create and train our Word2Vec models, we'll have to make use of some fairly basic linear algebra. Thankfully, there's a Python library that will do much of the math for us.

Remember, when we submit our queries, we'll be checking to see which aggregate word vectors in the dataset are closest, in terms of cosine similarity, to our aggregate query vector. This is the rough example that I showed you previously:



Let's get started!

Step 1: Install gensim

The library we will use to create and train our Word2Vec model is gensim, the same Python library we used for our lesson on topic modeling. gensim is a Python library for topic modeling, document indexing and similarity retrieval

```
!pip install gensim
```

```

Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: numpy<2.0, >=1.18.5 in /usr/local/lib/python3.11/di
Requirement already satisfied: scipy<1.14.0, >=1.7.0 in /usr/local/lib/python3.11/
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (

```

!! - This is important


Once you have installed gensim, you **must** make sure you restart this session. It's easy to do so, but if you skip this step, a conflict with NumPy will produce some pretty frustrating errors.

In the menu above, go to "Runtime" and click on "Restart session." Once your notebook restarts, skip Step 1 and proceed directly to Step 2.

Step 2: Import Necessary Libraries

As usual, we'll begin by importing the libraries that will help us streamline our code. You have used all of these in the past, except for gensim's Word2Vec and the linalg model in NumPy.


```
import pandas as pd
import os
import numpy as np
import gensim as gensim
# Word2Vec is the model from gensim that we'll use to create word embeddings
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import re
# numpy.linalg is a module in NumPy that provides functions for linear algebra operat
from numpy.linalg import norm
from google.colab import files
print("Libraries imported.")
```

 Libraries imported.

Step 3: Upload the CSV File

Now, let's get the CSV file that contains all those passages we downloaded about the slave trade. You'll have this stored on your local machine - ideally it should be in our project folder for this lesson.

```
uploaded = files.upload()
filename = next(iter(uploaded))
print(f"Successfully uploaded file: {filename}")
```

 loading_passages.csv

- **loading_passages.csv**(text/csv) - 6296460 bytes, last modified: 5/4/2025 - 100% done

Saving loading_passages.csv to loading_passages.csv
Successfully uploaded file: loading_passages.csv

Step 4: Load the CSV into a Pandas DataFrame

This step loads your data into a dataframe, which will make it easier to access and manipulate.

```
try:
    df = pd.read_csv(filename, sep=None, encoding='utf-8', engine='python')
    print("CSV file loaded successfully!")
    print(df.head()) # Print the first few rows
    print(df.columns) # Print column names
except FileNotFoundError:
    print(f"Error: File '{filename}' not found. Please make sure you have uploaded th
```

```
⇒ CSV file loaded successfully!

      document_id      volume_title \
0  0606000402_000011  The history, civil and commercial, of the Brit...
1  0004900300_000002  The history civil and commercial, of the Briti...
2  0151301400_000003  A short account of the African slave trade, co...
3  0072002600_000028  A review of the principal proceedings of the P...
4  1150400200_000029  A review of the principal proceedings of the P...

      volume_author \
0      Bryan Edwards, 1743-1800
1      Bryan Edwards, 1743-1800
2  Norris, Robert      1791
3                      NaN
4                      NaN

      highlighted_passages \
0  defcription of the African Coaq/. forts and Fa...
1  CHARA.BEE ISLANDS . CHAPTER I. BARBADOS. Firt ...
2  bloody and unrelenting Arm of Tyranny is etern...
3  which expressly authorise and encourage the sl...
4  islands . Is Parliament prepared to strike off...

      bibliographic_information
0  Bryan Edwards, 1743-1800 Dublin : Luke White, ...
1  Bryan Edwards, 1743-1800 London : printed for ...
2  Norris, Robert      1791 Liverpool : prin...
3  None London : [ printed for J. Stockdale, and ...
4  None London : printed for R. Edwards, No. 142,...
Index(['document_id', 'volume_title', 'volume_author', 'highlighted_passages',
      'bibliographic_information'],
      dtype='object')
```

Excellent. If that worked, you can see that our column "highlighted_passages" contains all the text we'll be using to build our vector model.

Step 5: Specify Column with Passages and Output Directory

We need to specify that we're particularly interested in that one column of "highlighted passages". If, for some reason, you've changed the name of that column, you should use the column that contains all your text.

In Step 6, we'll create an output directory to store all these passages as .txt files, so we may as well specify that now, while we're creating these variables.

```
passage_column = 'highlighted_passages' # REPLACE WITH YOUR COLUMN NAME, IF DIFFEREN
output_directory = 'passages'
```

Step 6: Create the Output Directory

Here, we create that new output directory. You should be pretty used to this by now.

```
if not os.path.exists(output_directory):
    os.makedirs(output_directory)
    print(f"Created directory: {output_directory}")
else:
    print(f"Directory already exists: {output_directory}")
```

➡ Directory already exists: passages

Step 7: Extract Passages and Save as .txt Files Here, we extract all the passages. If you click on the folder icon to the left, you can open the passages folder. Since our index starts at 0, our passages names will only run up to 999, but don't worry - all 1000 should be there.

```
try:
    for index, row in df.iterrows():
        passage_text = str(row[passage_column]) # Get the passage text and convert t
        filename = f"passage_{index}.txt"
        filepath = os.path.join(output_directory, filename)
        with open(filepath, 'w', encoding='utf-8') as f:
            f.write(passage_text)
        print(f"Created file: {filepath}")
except Exception as e:
    print(f"Error processing spreadsheet rows: {e}")
    exit() # Stop if there's an error with row processing
```

```
import nltk
nltk.download('punkt_tab')
```

➡ [nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Package punkt_tab is already up-to-date!
True

Step 8: Load Passages and Tokenize for Word2Vec

Here, we'll load each of the extracted passages from the text files and tokenize them into individual "words". **You'll remember from our text processing lesson that there are lots of different ways that we can manipulate these tokens. In this case, we're just using them as-is, but that should set of warning bells! On your own, add a line of code to stem these words.**

```

corpus = [] # List to store tokenized passages
try:
    for filename in os.listdir(output_directory):
        if filename.endswith(".txt"):
            filepath = os.path.join(output_directory, filename)
            try:
                with open(filepath, 'r', encoding='utf-8') as f:
                    passage_text = f.read()
                    # Tokenize and lowercase the passage
                    passage_words = [word.lower() for word in word_tokenize(passage_text)]
                    corpus.append(passage_words)
            except Exception as e:
                print(f"Error reading or tokenizing file {filepath}: {e}")
except Exception as e:
    print(f"Error processing files in {output_directory}: {e}")

print(f"Number of tokenized passages: {len(corpus)}")

```

➡ Number of tokenized passages: 1000

Step 9: Train Word2Vec Model

Here's where things start to get interesting! This step trains your Word2Vec model from the tokenized passages. There are a number of different parameters to manipulate, and, as you've done in the past, you should absolutely experiment with different settings.

Read this article, if you need a refresher on how our model is built:

<https://medium.com/@manansuri/a-dummys-guide-to-word2vec-456444f3c673>

vector_size:

In this case, each word has 100 vectors. That's a pretty low level of dimensionality, but it's a start, and you can increase it if you want to attempt to encode more complex semantic relationships, but this will be more computationally burdensome. Remember that our corpus is very small, only one million words.

window

A window of 5 is quite specific. The model will consider that many words to the left and right to make predictions. You might want to pull up a passage and read through it: how much of a window captures related semantic context?

min_count

You should definitely experiment with increasing this. Remember, our goal is to eliminate as much bad OCR and meaningless function words as possible, without tossing out meaningful semantic context.

workers


This mainly affects training speed. If you find that your model is taking forever to load, you may want to stop the process and increase this number.

epochs

This is the number of passes that are made over the data, just as we saw in topic modeling. Ten is a reasonable number, and there's probably not much reason to increase this, since we're dealing with such a small dataset. If it was larger, we could increase this to capture more specific semantic information.

```
#Parameters
vector_size = 100
window = 5
min_count = 1
workers = 4
epochs = 10

model = Word2Vec(sentences=corpus, vector_size=vector_size, window=window, min_count=
model.save("word2vec.model")
print("Word2Vec model trained and saved successfully!")
```

 Word2Vec model trained and saved successfully!

Step 10: Define Query and Parameters

This is where you can be creative! You'll build the query vector here, either from words or passages. In the a couple of cells, you'll run this vector against your model.

This cell is built to give you options. You can build your vector either from words or passages. For this exercise, in your group, develop at least one of each of the following types:

Query 1: A single word.

Query 2: A collection of words.

Query 3: At least three passages of bad OCR, drawn from our top search results, which you judge to be relevant.

Query 4: At least three passages of clean text (you have to type it yourself, if you can't find the volume online!), drawn from our top search results, which you judge to be relevant.

Query 5: At least three passages of AI-generated text.

If you really want to experiment, I suggest you to try "pushing" the concept of slavery in various ways. For instance, you might try to find passages that specifically express the conditions that people endured during the middle passage. Or, you might want to try to find passages that describe people's lives before they were enslaved.


```
# CHOOSE YOUR QUERY METHOD: Set ONE of these to True, the other to False
USE_QUERY_WORDS = True # Set to True to use query words, False for passage(s)
USE_QUERY_PASSAGES = False # Set to True to use full passage(s), False for words

# Load the model
model = Word2Vec.load("word2vec.model")

if USE_QUERY_WORDS:
    query_words = ["africa", "carry", "child", "coast", "colony", "condition", "count"]
    #query_words = ["slavery"]
    # Tokenize and clean the query words - remember to stem your words, if you did so
    query_words = [word.lower() for word in query_words]
    # Create a query vector by averaging the word vectors of the query words
    query_vector = np.mean([model.wv[word] for word in query_words if word in model.wv])
    print("Query vector created using query words.")

elif USE_QUERY_PASSAGES:
    query_passages = [
        """defcription of the African Coaq/. forts and Facories. -- Exports front Gre
        """"trade of this island , it is proper to begin with the Negroe trade , which
        """"bloody and unrelenting Arm of Tyranny is eternally held up, ready to cut i
    ] # Replace with your query passages, and add more as needed.

    all_passage_words = []
    for passage in query_passages:
        # Clean the passage: lowercase, remove punctuation, etc. (adjust regex as nee
        cleaned_passage = re.sub(r'[^w\s]', '', passage.lower())
        passage_words = cleaned_passage.split()
        all_passage_words.extend(passage_words) # Accumulate words from all passages
    query_vector = np.mean([model.wv[word] for word in all_passage_words if word in m
    print("Query vector created using passages.")
```

 Query vector created using query words.

Step 11: Function to Calculate Cosine Similarity

Here, we'll define a function to assess cosine similarity, the aggregate difference between vectors. This is what you're using to match your queries to documents.

For this function, `v1` is the first vector and `v2` is the second vector.

```
def cosine_similarity(v1, v2):
    return np.dot(v1, v2) / (norm(v1) * norm(v2))
```

Step 12: Calculate and Rank Passages by Similarity, storing the passage number

```
passage_similarities = []
all_similarity_values = []
for filename in os.listdir(output_directory):
    if filename.endswith(".txt"):
```

```

11 filename = filename.replace(" ", "_")
    filepath = os.path.join(output_directory, filename)
    try:
        with open(filepath, 'r', encoding='utf-8') as f:
            passage_text = f.read()
        # Tokenize and lowercase the passage
        passage_words = [word.lower() for word in word_tokenize(passage_text)]
        # Get the passage vector
        passage_vector = np.mean([model.wv[word] for word in passage_words if word

    if passage_vector is not None: # Ensure passage_vector is not None
        # Calculate Cosine Similarity
        similarity = cosine_similarity(query_vector, passage_vector)
        # Extract Passage Number
        passage_number = int(filename.replace("passage_", "").replace(".txt", ""))
        passage_similarities.append((passage_number, similarity)) # Store pas
        all_similarity_values.append(similarity) # NEW: collect similarity
    else:
        print(f"No words from passage {filename} were found in the Word2Vec mo

except Exception as e:
    print(f"Error processing {filename}: {e}")

# Sort Results, Extract Passage Numbers Starting from 1
passage_similarities.sort(key=lambda x: x[1], reverse=True)
ranked_passage_numbers = [passage_number + 1 for passage_number, similarity in passage
print("Ranked Passage Numbers (Starting from 1):")

average_similarity = sum(all_similarity_values) / len(all_similarity_values)
print(f"\nAverage Cosine Similarity across all passages: {average_similarity:.4f}")

```

➦ Ranked Passage Numbers (Starting from 1):

Average Cosine Similarity across all passages: 0.7708

Step 13: Create a Pandas Dataframe of New Passage Ranks from Word2Vec, Save and Download as CSV

```

# --- Step 13: Create a Pandas Dataframe, Save and Download as CSV ---
rankings_df = pd.DataFrame({'rankings': ranked_passage_numbers})
csv_filename = "passage_rankings.csv"
rankings_df.to_csv(csv_filename, index=False) # index=False prevents writing the index
files.download(csv_filename)
print(f"Downloaded CSV file: {csv_filename}")

```

➦ Downloaded CSV file: passage_rankings.csv

