Unit 3 Assignment

Name: Dallin Moore

DATA 5420/6420

In this third unit assignment you will apply methods learned during unit 3, which include document summarization, text similarity, and document clustering. We will combine these steps to create one finished product whereby we begin with clustering documents into groups, making recommendations based on their similarity and then generating an even shorter summary of the recommended document...

This assignment will be fairly open-ended, but I'd like you to try combining your results or insights from your Unit 2 assignment with at least two of the unsupervised methods from Unit 3 to create what should be getting towards a MVP (minimum viable product) -- meaning I should be starting to get an idea of what your final project will look like in terms of its functionality.

Here are some examples:

Does it take your categorized documents (from either document classification or sentiment analysis) extract out topics, and then summarize the key insights from each category?

Does it use your categorized labels as a filter (e.g., only positive documents) in a recommendation system that also provides high-level summaries?

Be creative!

Begin, as always, with importing your required dependencies...This will vary depending on the type of feature engineering, and unsupervised algorithms you choose.

```
!pip install contractions
!pip install kneed
!pip install spacy
```

```
import pandas as pd
import re
import contractions
from string import punctuation
import spacy
import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import MiniBatchKMeans
from collections import Counter
from kneed import KneeLocator
import networkx as nx
from sklearn.metrics.pairwise import cosine_similarity
from nltk.tokenize import sent_tokenize
import matplotlib.pyplot as plt
nltk.download('stopwords')
nltk.download('punkt')
```

Import your dataset

Are you using the same dataset from the Unit 2 assignment, or a different dataset?

Yes, but not the dataset containg the information of themes. It was just too small to be useable for what I want to do. The dataset that I am using contains song name, artist, and lyrics.

```
df_original = pd.read_csv("./Downloads/data_5420/data_5420/Spotify-Million-Song-Dataset.csv",engine='c')
df_original.head()
```

	artist	song	link	text
0	ABBA	She's My Kind Of Girl	/a/abba/ahes+my+kind+of+girl_20598417.html	Look at her face, it's a wonderful face \nAnd...
1	ABBA	Andante, Andante	/a/abba/andante+andante_20002708.html	Take it easy with me, please \nTouch me gentl...
2	ABBA	As Good As New	/a/abba/as+good+as+new_20003033.html	I'll never know why I had to go \nWhy I had t...

Outline your Steps

Give me a high-level walk through (bullets) of the steps you will take in this notebook, including things like text preprocessing, feature engineering, the unsupervised methods you'll choose, etc.

1. **Data Preprocessing**: Clean and preprocess the text data, including removing stopwords, punctuation, and any irrelevant information. You may also want to tokenize the lyrics.
2. **Vectorization**:
   - Convert the text data into numerical vectors with TF-IDF.
3. **Document Clustering**:
   - Apply KMeans to group similar songs together based on their lyrics.
   - Evaluate the clusters to ensure they are meaningful and coherent.
4. **Text Similarity / Information Retrieval**:
   - Implement cosine similarity to compute similarity between songs.
5. **Recommendation Systems**:
   - Develop a recommendation system that suggests songs to users based on their similarity to songs the user already likes.
6. **Document Summarization**:
   - Implement document summarization techniques to generate shorter summaries of the lyrics for recommended songs.
7. **Evaluation**:
   - Evaluate the effectiveness.

Data Cleaning

```
def cleaning(df, text_column):
  cleaned_df = df.copy()
  def clean_text(text):
    # Remove special characters and line breaks
    text = re.sub(r'\s+', ' ', text)

    # Remove punctuation (excluding single quotes)
    text = re.sub(r'[^\w\s\']', '', text)

    return text

  # Apply the cleaning function to the specified text column
  cleaned_df[text_column] = cleaned_df[text_column].apply(clean_text)

  return cleaned_df


def preprocess_dataframe(df, text_column):
    # Load spaCy language model with lemmatization
    nlp = spacy.load("en_core_web_sm", disable=["ner", "parser"])

    # Define a function for lemmatization
    def lemmatize_text(text):
        doc = nlp(text)
        return ' '.join([word.lemma_ if word.lemma_ != '-PRON-' else word.text for word in doc])

    # Define the preprocessing function
    def preprocess_text(text):
        # Casefolding
        text = text.lower()

        # Contraction expansion
        contract = contractions.contractions_dict
        for contract, expansion in contract.items():
            text = re.sub(fr'\b{contract}\b', expansion, text)

        # Lemmatization using spaCy
        text = lemmatize_text(text)

        # Tokenization and stemming using spaCy
        doc = nlp(text)
        preprocessed_tokens = [token.lemma_ for token in doc if token.is_alpha and token.lemma_ not in stopwords.words('english')]
        preprocessed_text = ' '.join(preprocessed_tokens)

        return preprocessed_text

    # Copy the original DataFrame to avoid modifying it in place
    preprocessed_df = df.copy()

    # Batch processing using spaCy's pipe method
    preprocessed_texts = list(nlp.pipe(preprocessed_df[text_column], batch_size=1000))
    preprocessed_texts = [' '.join([token.lemma_ for token in doc if token.is_alpha and token.lemma_ not in stopwords.words('english')]) for

    # Update the DataFrame with preprocessed text
    preprocessed_df[text_column] = preprocessed_texts

    return preprocessed_df


# this takes a long time to run, so I'm downloading the same preprocessed text from unit 1 assignment
df = pd.read_csv('./Downloads/data_5420/data_5420/preprocessed-spotify-dataset.csv')
df.head()
```

	artist	song	text
0	ABBA	She's My Kind Of Girl	look face wonderful face mean something specia...
1	ABBA	Andante, Andante	take easy I please touch I gently like summer ...
2	ABBA	As Good As New	I never know I go I put lousy rotten show boy ...
3	ABBA	Bang	make somebody happy question give take learn s...
4	ABBA	Bang-A-Boomerang	make somebody happy question give take learn s...

Have you made any changes to the way you are cleaning/normalizing the text since the Unit 1 or 2 assignments? If so, describe the changes in steps and explain why you've decided to make these changes:

No, I'm still using casefolding, contraction expansion, stopword removal, lemmatization, and stemming. Hopefully this way the theme is able ot be identified and songs can be grouped together.

Feature Engineering

```
stop_words = nltk.corpus.stopwords.words('english')
stop_words = stop_words + ['la']

# Split data into text and labels
songs = df['song']+', '+df['artist']
lyrics = df['text'].tolist()

# set parameters for tf-idf for unigrams and bigrams
tf = TfidfVectorizer(stop_words = stop_words, ngram_range=(1,2),min_df=2, max_df=0.8)
# extract tfidf features from norm_corpus
tfidf_matrix = tf.fit_transform(lyrics)
tfidf_matrix.shape
```

```
(57650, 441173)
```

How are you setting parameters in your feature engineering (e.g., min_df, max_df, ngram_range)? Is this different from the feature engineering you employed in the Unit 2 assignment, why or why not?

I don't have any reason to change from what I did for he last assignment so I will use the same parameters **min_df**: Since genre classification might involve specialized language or jargon that appears only in a few documents, setting min_df too high might cause you to miss out on important features, so it is set to 1 to start.

min_df: It will be set to 1. Setting min_df too high may result in excluding infrequent but potentially meaningful words but, setting it too low may lead to noise from very rare terms.

max_df: A value of max_df=0.8 will be used because we want to exclude words that appear too frequently across all genres.

ngram_range: For lyrics, considering single words (unigrams) and pairs of words (bigrams) might capture more meaningful phrases and patterns in the text. I will start with a range that includes unigrams and bigrams.

Unsupervised Method 1 - Document Clustering:

```
kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 300
    #"random_state": 42,
}
# Calculate Sum of Squared Errors (SSE) for different values of k
sse = []
for k in range(3, 20):
    kmeans = MiniBatchKMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(tfidf_matrix)
    sse.append(kmeans.inertia_)

# Find the elbow point using the KneeLocator
kl = KneeLocator(range(3, 20), sse, curve='convex', direction='decreasing')
elbow_point = kl.elbow

# Perform MiniBatchKMeans clustering with the optimal number of clusters
kmeans = MiniBatchKMeans(n_clusters=elbow_point, **kmeans_kwargs)
kmeans.fit(tfidf_matrix)

# Count the number of samples in each cluster
cluster_counts = Counter(kmeans.labels_)
print("Cluster Counts:", cluster_counts)
```

```
Cluster Counts: Counter({4: 24477, 1: 15272, 0: 8825, 5: 5835, 2: 1972, 3: 1269})
```

Which method did you select and how do you see it being incorporated in your overall project?

I used MiniBatchKMeans clustering to group the songs so that they can now be summarized. In my final project, MiniBatchKMeans can efficiently cluster songs based on their textual representations, enabling the generation of diverse playlists tailored to user input. By leveraging MiniBatchKMeans, the application can organize songs into clusters, generate playlists representative of different themes or styles, and provide summaries for each cluster.

Unsupervised Method 2 - Summarization:

```
# Create clusters of songs
song_clusters = {}
for i, cluster_label in enumerate(kmeans.labels_):
    if cluster_label not in song_clusters:
        song_clusters[cluster_label] = []
    song_clusters[cluster_label].append(songs[i])

# Function to summarize text using TextRank
def textrank_summarize(text, top_n=4):
    original_sentences = sent_tokenize(text)

    vectorizer = TfidfVectorizer()
    sentence_vectors = vectorizer.fit_transform(original_sentences)

    similarity_matrix = cosine_similarity(sentence_vectors, sentence_vectors)

    scores = nx.pagerank(nx.from_numpy_array(similarity_matrix))

    ranked_sentences = sorted(((scores[i], i) for i in range(len(original_sentences))), reverse=True)

    summary_indices = [index for _, index in ranked_sentences[:top_n]]
    summary = [original_sentences[i] for i in sorted(summary_indices)]

    return ' '.join(summary)

# Generate summaries for each cluster
cluster_summaries = {}
for cluster_label, songs in song_clusters.items():
    concatenated_lyrics = ' '.join(songs)
    summary = textrank_summarize(concatenated_lyrics)
    cluster_summaries[cluster_label] = summary

# Print summaries for each cluster
for cluster_label, summary in cluster_summaries.items():
    print(f"Cluster {cluster_label} Summary:")
    print(summary)
    print()
```

```
uster 1 Summary:
 Had To Be You, Harry Connick, Jr. Standing In The Shadows, Hank Williams Jr. That's How I Wanted It To Be, Hank Williams Jr. If I Were A

uster 4 Summary:
's The Most Wonderful Time Of The Year, Harry Connick, Jr. Jingle Bells, Harry Connick, Jr. Junco Partner, Harry Connick, Jr. Little Farl

uster 0 Summary:
Love You, Bette Midler Shining Star, Bette Midler The Girl Is On To You, Bette Midler The Glory Of Love, Bette Midler The Last Time, Bett

uster 3 Summary:
ving Proof, Hank Williams Jr. Mary's Boy Child, Harry Belafonte A Million Suns, Hillsong All The Heavens, Hillsong By Your Side, Hillsong

uster 5 Summary:
, Insane Clown Posse Crossing The Bridge, Insane Clown Posse Deadbeat Moms, Insane Clown Posse Dear ICP, Insane Clown Posse Dog Beats, Ir

uster 2 Summary:
 Janis Joplin Kozmic Blues ( In Album Woodstock ), Janis Joplin My Baby, Janis Joplin Try, Janis Joplin Kickin' With You, Jason Mraz Danc
```

I wanted to base the summary off of the song lyrics, but I tried multiple ways that ended up crashing when they needed more memory than was available. I have some more ideas to try for the final project to avoid this issue. Then, the summary will be more about what the song lyrics are about instead of just the song titles that are most similar across the clusters.

Which second method did you select and how do you see it being incorporated?

In the final project, TextRank summarization will be employed to generate concise summaries for each song's lyrics within the playlist. Initially, the lyrics of each song will undergo preprocessing, including tokenization and noise removal. Subsequently, the TextRank algorithm will rank the importance of sentences within the lyrics based on their semantic similarity, effectively identifying key phrases and themes. The top-ranked sentences will then be selected to form a summary for each song, capturing its essence and main ideas. Finally, these individual song summaries will be aggregated to create a comprehensive summary for the entire playlist, providing users with a quick overview of the playlist's content and enabling them to make informed decisions about their listening preferences. This approach ensures that users receive relevant and informative summaries tailored to their input text, enhancing their overall engagement and satisfaction with the playlist recommendation system.

Walk me through what you've created or completed in this Unit 3 assignment. I'd also like for you to explain what you think is going well at this point, and what still needs to be done to have what you feel is a complete MVP for the final project -- it's fine if this is still speculative.

For my final project, I want the user to enter in a song or description, then a playlist with a summary will be output. Here are my plans for this project: