Unit 2 Assignment: Feature Engineering & Supervised Classification

DATA 5420/6420

Name: Dallin Moore

In this second assignment you will be tasked with training your own supervised classification model, this could be to do document classification of some sort, or a sentiment analysis. You will first be tasked with selecting a labeled text dataset to train a supervised classifier, then you will apply it to your dataset from Unit 1.

Next, you will find a pretrained supervised model from Hugging Face, which has a larger collection of pretrained document classification and sentiment analysis models. You will investigate the results of the model you trained against the pretrained model and compare their performances. This will help you decide how you might incorporate some form of either document classification or sentiment analysis into your final product.

General breakdown of steps:

1. Select a labeled dataset to perform document classification or sentiment analysis
2. Train at least two different models on the dataset, compare performance
3. Apply the classification model to your dataset from Unit 1
4. Examine results, speak to how well it appears to perform
5. Apply a pretrained transformer model to your dataset from Unit 1
6. Examine results, speak to how well it appears to perform
7. Compare and contrast your trained model vs the pretrained model

Some suggested datasets for document classification:

- Brown Corpus -- accesible through NLTK
- 20 News Groups -- accessible through scikit learn
- Yelp Reviews Dataset

Some suggetsed datasets for sentiment analysis:

- IMDB movie reviews
- Sentiment140
- Yelp Reviews Dataset - linked above

You are by no means limited to these datasets, Kaggle has lots of datasets available for document classification and sentiment analysis, so you may find something more relevant to your dataset there. Just make sure it it labeled data (i.e., has a labeled class like positive, negative).

Pretrained Models:

You can find pretrained models for sentiment analysis and document classification on the models page for HuggingFace. Remember, tools like Poe, ChatGPT, Claude, etc. are excellent resources for developing code for implementing models such as these!!

Try something like: *I need a pretrained model from hugging face to do XYZ, can you provide python code*

```
# import dependencies
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import re
from transformers import pipeline
```

```
# load in selected labeled dataset
df_songs = pd.read_csv('/content/Top-1000-Songs-To-Hear-Before-You-Die.csv')
# load in the lyrics dataset (to add lyrics to the genre)
df_lyrics = pd.read_csv('/content/preprocessed-spotify-dataset.csv')
df_songs.head()
```

	Artist	Theme	Title	Year (Top 1000 Songs To Hear Before You Die.csv)
0	!!!	Protest	Me and Giuliani Down By the Schoolyard (A True...	2003
1	808 State	Party songs	Pacific State	1989
2	A R Rahman	Love	Kehna hi kya	1995
3	Aaliyah	Sex	Try Again	2000
4	Abba	Heartbreak	The Winner Takes It All	1980

Next steps: View recommended plots

Will you be performing document classification or sentiment analysis? What is your outcome variable (i.e., positive, negative, genre type, etc.)

Document classification predicting the genre of spefic songs. Because the dataset selected doesn't have lyrics, I will create a new DataFrame with the name, artists, lyrics, and list of genres

Which dataset did you decide to go with and why?

I went with a dataset containing the top 1,000 songs to listen to before you die (the guardian). It includes the them that I want for classifying my lyrics.

```
# Function to preprocess strings
def preprocess_string(s):
    # Remove non-alphanumeric characters
    s = re.sub(r'[^a-zA-Z0-9\s]', '', s)
    # Lowercase the string
    s = s.lower()
    return s.strip()

# Function to match song and artist names
def match_song_and_artist(song_name, artist_name, df_lyrics):
    song_name_normalized = preprocess_string(song_name)
    artist_name_normalized = preprocess_string(artist_name)
    match = df_lyrics[(df_lyrics['song'].apply(preprocess_string) == song_name_normalized) &
                      (df_lyrics['artist'].apply(preprocess_string) == artist_name_normalized)]
    if not match.empty:
        return match.iloc[0]['text']
    else:
        return None

# Add matched lyrics column to df_songs
df_songs['Matched_Lyrics'] = df_songs.apply(lambda row: match_song_and_artist(row['Title'], row['Artist'],

# Filter records to only contain those with both genre and lyrics
df = df_songs.dropna(subset=['Theme', 'Matched_Lyrics'])

# Select only the required columns
df = df[['Title', 'Artist', 'Matched_Lyrics', 'Theme']]

# Rename columns to match the specified names
df.columns = ['Song Name', 'Artist', 'Lyrics', 'Theme']

df
```

	Song Name	Artist	Lyrics	Theme
4	The Winner Takes It All	Abba	I want talk thing go though hurt I history I p...	Heartbreak
5	Dancing Queen	Abba	dance jive time life see girl watch scene digg...	Party songs
12	Dream On	Aerosmith	every time I look mirror line face get clear p...	Life and death
13	Love in an Elevator	Aerosmith	workin like dog fo de boss man workin de compa...	Sex
28	Music to Watch Girls By	Andy Williams	boy watch girl girl watch boy watch girl go ey...	Love
...	...	...	...	...
970	Desire	U2	yeah lover I street go go bright light big cit...	Sex
975	Coney Island	Van Morrison	coney island come downpatrick stop st john poi...	People and places

Next steps: View recommended plots

```
df['Theme'].value_counts()
```

```
Heartbreak          27
People and places   25
Life and death      20
Love                20
Sex                 19
Protest             16
Party songs         14
Name: Theme, dtype: int64
```

What, if any cleaning or text normalization steps did you apply to this dataset and why?

The lyrics data was already cleaned, however the artist and song title needed to be lowered and punctuation removed to be match up the lyrics with the theme.

```
# perform feature engineering on your cleaned corpus
# Split data into text and labels
texts = df['Lyrics'].tolist()
labels = df['Theme'].tolist()

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=.2, random_state=42)
# Create a TF-IDF vectorizer
vectorizer = TfidfVectorizer(stop_words = 'english', min_df=1, max_df=0.8, ngram_range=(1, 2))
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

num_features_train = X_train_vec.shape[1]
num_features_test = X_test_vec.shape[1]
```

Which form of feature engineering did you choose (count or TFIDF) and did you go with unigrams, bigrams, etc.? Why?

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.

max_df: For genre classification, we want to exclude words that appear too frequently across all genres, as they are less likely to be discriminative. A value of max_df=0.8 will be used to start, but I will consider raising it to max_df=0.95 to improve the model.

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 (ngram_range=(1, 2)), but I might experiment with higher values depending on the results.

Next, train your supervised classifier. Remember:

- Create at least a training and a test set (fine if you don't have enough data to do a validation set)
- Perform cross-validation
- Train at least two different supervised classifiers on your training set
- If in the 6420 section, also plan to try out at least two changes to the model parameters

- Apply your best performing model to the test set
- Provide model evaluation metrics

```
models = {
    'Naive Bayes': MultinomialNB(),
    'Linear SVM': LinearSVC(random_state=42),
    'Random Forests': RandomForestClassifier(random_state=42)
}

# Perform cross-validation for each model
for name, model in models.items():
    scores = cross_val_score(model,X_train_vec,y_train,cv=5)
    print(f"{name} Cross-Validation Mean Accuracy: {scores.mean():.4f}")
```

```
Naive Bayes Cross-Validation Mean Accuracy: 0.2506
Linear SVM Cross-Validation Mean Accuracy: 0.3300
Random Forests Cross-Validation Mean Accuracy: 0.2324
```

Which model performed best and how do you know?

The Linear SVM performed the best. Unfortunately, with the small dataset it was only 33% accurate.

Now, bring in your dataset from Unit 1 and apply your best performing model to add labels to this dataset (sentiment or document class). Remember:

- Apply the same cleaning and text normalization steps to this dataset as you did the training data
- Apply the same feature engineering type and parameters
- Use the `.transform()` on your Unit 1 dataset with the vectorizer to ensure you match the number of features used to train your model
- Store the predictions and your text observations in a dataframe

```
param_grid_svc = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'loss': ['hinge', 'squared_hinge'],
    'max_iter': [1000, 5000, 10000]
}

# Grid search for LinearSVC
grid_search_svc = GridSearchCV(LinearSVC(random_state=42), param_grid_svc, cv=5, verbose=10)
grid_search_svc.fit(X_train_vec, y_train)

# After fitting, you would typically print the best parameters as follows:
print("Best parameters for LinearSVC:", grid_search_svc.best_params_)
```

```
[CV 2/5; 26/30] END C=10, loss=hinge, max_iter=5000;, score=0.391 total time=   0.0s
[CV 3/5; 26/30] START C=10, loss=hinge, max_iter=5000..........................
[CV 3/5; 26/30] END C=10, loss=hinge, max_iter=5000;, score=0.273 total time=   0.0s
[CV 4/5; 26/30] START C=10, loss=hinge, max_iter=5000..........................
[CV 4/5; 26/30] END C=10, loss=hinge, max_iter=5000;, score=0.273 total time=   0.0s
[CV 5/5; 26/30] START C=10, loss=hinge, max_iter=5000..........................
[CV 5/5; 26/30] END C=10, loss=hinge, max_iter=5000;, score=0.455 total time=   0.0s
[CV 1/5; 27/30] START C=10, loss=hinge, max_iter=10000.........................
[CV 1/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.261 total time=   0.0s
[CV 2/5; 27/30] START C=10, loss=hinge, max_iter=10000.........................
[CV 2/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.391 total time=   0.0s
[CV 3/5; 27/30] START C=10, loss=hinge, max_iter=10000.........................
[CV 3/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.273 total time=   0.0s
[CV 4/5; 27/30] START C=10, loss=hinge, max_iter=10000.........................
[CV 4/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.273 total time=   0.0s
[CV 5/5; 27/30] START C=10, loss=hinge, max_iter=10000.........................
[CV 5/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.455 total time=   0.0s
[CV 1/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..................
[CV 1/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.261 total time=   0.0s
[CV 2/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..................
[CV 2/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.391 total time=   0.0s
[CV 3/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..................
[CV 3/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.273 total time=   0.0s
[CV 4/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..................
[CV 4/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.273 total time=   0.0s
[CV 5/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..................
[CV 5/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.455 total time=   0.0s
[CV 1/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..................
[CV 1/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.261 total time=   0.0s
[CV 2/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..................
[CV 2/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.391 total time=   0.0s
[CV 3/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..................
[CV 3/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.273 total time=   0.0s
[CV 4/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..................
[CV 4/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.273 total time=   0.0s
[CV 5/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..................
[CV 5/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.455 total time=   0.0s
[CV 1/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.................
[CV 1/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.261 total time=   0.0s
[CV 2/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.................
[CV 2/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.391 total time=   0.0s
[CV 3/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.................
[CV 3/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.273 total time=   0.0s
[CV 4/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.................
[CV 4/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.273 total time=   0.0s
[CV 5/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.................
[CV 5/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.455 total time=   0.0s
Best parameters for LinearSVC: {'C': 10, 'loss': 'hinge', 'max_iter': 1000}
```

```
# Sample 5 random records from the DataFrame
random_records = df.sample(n=5)
```

```
# Vectorize the lyrics of the sampled records
lyrics_vec = vectorizer.transform(random_records['Lyrics'])

# Predict labels for the vectorized lyrics
predicted_labels = grid_search_svc.predict(lyrics_vec)

# Print song title, artist, and the beginning of the lyrics along with predicted labels
for index, (title, artist, lyrics, label) in enumerate(zip(random_records['Song Name'], random_records['Ar
    print(f"Record {index+1}:")
    print(f"Song Title: {title}")
    print(f"Artist: {artist}")
    print(f"Lyrics: {lyrics[:100]}...")  # Print only the beginning of the lyrics (first 50 characters)
    print(f"Predicted Label: {label}\n")
```

```
Record 1:
Song Title: Peace Train
Artist: Cat Stevens
Lyrics: I happy lately think good thing come I believe could something good begin I smile lately dream
Predicted Label: Protest

Record 2:
Song Title: Relax
Artist: Frankie Goes to Hollywood
Lyrics: guess happen hey hey whoa oh hey hey well relax want go relax want come relax want suck relax
Predicted Label: Sex

Record 3:
Song Title: Don’t Think Twice, It’s All Right
Artist: Bob Dylan
Lyrics: use sit wonder babe matter anyhow use sit wonder babe know rooster crow break dawn look window
Predicted Label: Heartbreak

Record 4:
Song Title: Fujiyama Mama
Artist: Wanda Jackson
Lyrics: I nagasaki hiroshima I baby I cause I fujiyama mama I blow top fujiyamayama fujiyama I start e
Predicted Label: Sex

Record 5:
Song Title: If You See Her, Say Hello
Artist: Bob Dylan
Lyrics: see say hello might tangier leave last early spring livin I hear say I I right though thing ge
Predicted Label: Heartbreak
```

Now examine your results, look at some individual observations and investigate whether the model predictions are logical/appear accurate. Describe your findings below:

It's not bad. With a bigger dataset, I'm sure that the outcome could be better. Overall however, it's doing the job and categorizing the songs fairly well.

Now select a pretrained model from Hugging Face (linked above) and make predictions onto your Unit 1 dataset. Compare how it appears to perform against how the model you trained appeared to perform.

```
import pandas as pd
from transformers import pipeline

# Load the pre-trained model for text classification
classifier = pipeline("zero-shot-classification")

# Define possible themes
possible_themes = ["Heartbreak", "People and places", "Life and death", "Love", "Sex", "Party songs", "Pro

# Function to predict theme for a given text
def predict_theme(text):
    # Perform zero-shot classification
    result = classifier(text, possible_themes)
    # Get the predicted label
    predicted_theme = result['labels'][0]
    return predicted_theme

df_copy = df.sample(n=5).copy()

df['Predicted_Theme'] = df['Lyrics'].apply(predict_theme)

df
```

```
No model was supplied, defaulted to facebook/bart-large-mnli and revision c626438 (https://huggingface
Using a pipeline without specifying a model name and revision in production is not recommended.
```

It appears to work well enough, and using this approach instead is a benefit because we don't need a labeled dataset to build the model.

How could you incorporate supervised classification (document or sentiment classification) into a product? -- think about what it could be useful for as we continue to work towards your final project.

If I wanted to build curated playlists, being able to predict the theme of new songs that come in could be helpful to group songs together that share a theme.