transposition

April 1, 2025

[1]:
```python
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import librosa
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import utils
from sklearn.calibration import LabelEncoder
```

[2]:
```python
tracks = utils.load('data/fma_metadata/tracks.csv')
genres = utils.load('data/fma_metadata/genres.csv')
features = utils.load('data/fma_metadata/features.csv')
small = tracks[tracks['set', 'subset'] <= 'small']
```

[3]:
```python
print(tracks.head())
```

```
            album                                                          \
         comments         date_created date_released engineer favorites id
track_id
2               0 2008-11-26 01:44:45    2009-01-05      NaN         4  1
3               0 2008-11-26 01:44:45    2009-01-05      NaN         4  1
5               0 2008-11-26 01:44:45    2009-01-05      NaN         4  1
10              0 2008-11-26 01:45:08    2008-02-06      NaN         4  6
20              0 2008-11-26 01:45:05    2009-01-06      NaN         2  4

                                                                           \
                                        information listens producer tags
track_id
2                                           <p></p>    6073      NaN   []
3                                           <p></p>    6073      NaN   []
5                                           <p></p>    6073      NaN   []
10                                              NaN   47632      NaN   []
20          <p> "spiritual songs" from Nicky Cook</p>    2710      NaN   []
```

```
          …       track                          \
          … information interest language_code
track_id  …
2         …         NaN     4656            en
3         …         NaN     1470            en
5         …         NaN     1933            en
10        …         NaN    54881            en
20        …         NaN      978            en

                                                                 \
                                                  license listens lyricist
track_id
2         Attribution-NonCommercial-ShareAlike 3.0 Inter…    1293      NaN
3         Attribution-NonCommercial-ShareAlike 3.0 Inter…     514      NaN
5         Attribution-NonCommercial-ShareAlike 3.0 Inter…    1151      NaN
10        Attribution-NonCommercial-NoDerivatives (aka M…   50135      NaN
20        Attribution-NonCommercial-NoDerivatives (aka M…     361      NaN


         number publisher tags            title
track_id
2             3       NaN   []             Food
3             4       NaN   []     Electric Ave
5             6       NaN   []       This World
10            1       NaN   []          Freeway
20            3       NaN   []  Spiritual Level

[5 rows x 52 columns]
```

[4]:
```python
# Load the features and metadata
features_df = pd.read_csv('data/fma_metadata/features.csv', index_col=0)
tracks_df = pd.read_csv('data/fma_metadata/tracks.csv', header=[0, 1],␣
↪index_col=0)

# Flatten the multi-level column names in tracks.csv
tracks_df.columns = ['_'.join(col).strip() if isinstance(col, tuple) else col␣
↪for col in tracks_df.columns]

# Filter the small dataset (subset)
small_tracks = tracks_df[tracks_df['set_subset'] == 'small']

# Extract relevant columns: track_id, features, and genre
small_tracks = small_tracks[['track_genre_top']].copy()
small_tracks.index = small_tracks.index.astype(int)  # Ensure track_id is an␣
↪integer
```

```python
# Merge features with genres
mapping_df = features_df.merge(small_tracks, left_index=True, right_index=True)

# Save the mapping to a new CSV file
mapping_df.to_csv('features_genre_mapping_small.csv', index=True)

print("Mapping of features to genres for the small dataset saved to␣
↪'features_genre_mapping_small.csv'.")
```

```
C:\Users\james\AppData\Local\Temp\ipykernel_17884\3966755446.py:2: DtypeWarning:
Columns (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53
,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,8
0,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,1
05,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,1
25,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,1
45,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,1
65,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,1
85,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,2
05,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,2
25,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,2
45,246,247,248,249,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,2
65,266,267,268,269,270,271,272,273,274,275,276,277,278,279,280,281,282,283,284,2
85,286,287,288,289,290,291,292,293,294,295,296,297,298,299,300,301,302,303,304,3
05,306,307,308,309,310,311,312,313,314,315,316,317,318,319,320,321,322,323,324,3
25,326,327,328,329,330,331,332,333,334,335,336,337,338,339,340,341,342,343,344,3
45,346,347,348,349,350,351,352,353,354,355,356,357,358,359,360,361,362,363,364,3
65,366,367,368,369,370,371,372,373,374,375,376,377,378,379,380,381,382,383,384,3
85,386,387,388,389,390,391,392,393,394,395,396,397,398,399,400,401,402,403,404,4
05,406,407,408,409,410,411,412,413,414,415,416,417,418,419,420,421,422,423,424,4
25,426,427,428,429,430,431,432,433,434,435,436,437,438,439,440,441,442,443,444,4
45,446,447,448,449,450,451,452,453,454,455,456,457,458,459,460,461,462,463,464,4
65,466,467,468,469,470,471,472,473,474,475,476,477,478,479,480,481,482,483,484,4
85,486,487,488,489,490,491,492,493,494,495,496,497,498,499,500,501,502,503,504,5
05,506,507,508,509,510,511,512,513,514,515,516,517,518) have mixed types.
Specify dtype option on import or set low_memory=False.
  features_df = pd.read_csv('data/fma_metadata/features.csv', index_col=0)

Mapping of features to genres for the small dataset saved to
'features_genre_mapping_small.csv'.
```

[5]:
```python
mapping_df.describe()
```

[5]:

	chroma_cens	chroma_cens.1	chroma_cens.2	chroma_cens.3	\
count	7918.000000	7918.0000	7918.000000	7918.000000	
unique	7901.000000	7901.0000	7901.000000	7900.000000	
top	2.120035	-0.1481	0.687363	1.388522	
freq	6.000000	6.0000	6.000000	6.000000	

```
        chroma_cens.4  chroma_cens.5  chroma_cens.6  chroma_cens.7  \
count     7918.000000    7918.000000    7918.000000    7918.000000
unique    7900.000000    7900.000000    7900.000000    7901.000000
top         -0.779828      -1.049356      -1.352663      -1.146229
freq         6.000000       6.000000       6.000000       6.000000

        chroma_cens.8  chroma_cens.9  …   tonnetz.40   tonnetz.41  \
count      7918.00000    7918.000000  …  7918.000000  7918.000000
unique     7901.00000    7901.000000  …  7899.000000  7900.000000
top           0.13384       0.494833  …     0.023668     0.022414
freq          6.00000       6.000000  …     6.000000     6.000000

                zcr        zcr.1        zcr.2        zcr.3   zcr.4  \
count   7918.000000  7918.000000  7918.000000  7918.000000  7918.0
unique  7901.000000  1573.000000  7900.000000   343.000000    85.0
top      101.444832     0.385742     0.040447     0.033691     0.0
freq       6.000000    20.000000     6.000000    99.000000  3648.0

              zcr.5        zcr.6  track_genre_top
count   7918.000000  7918.000000             7918
unique  7900.000000  7897.000000                8
top        5.669625     0.026036       Electronic
freq       6.000000     6.000000             1000

[4 rows x 519 columns]
```

[6]:

```python
# Filter relevant features: mfcc_* and spectral_*
# feature_columns = [col for col in mapping_df.columns if col.startswith('mfcc.
 ↪') or col.startswith('spectral_')]
feature_columns = [col for col in mapping_df.columns]

X = mapping_df.drop(columns=['track_genre_top']).values

# Encode the genre labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(mapping_df['track_genre_top'])  # Labels matrix

# Check the shapes of the features and labels
print(f"Features shape: {X.shape}")
print(f"Labels shape: {y.shape}")

# Optional: Save the label encoding mapping for later use
genre_mapping = dict(zip(label_encoder.classes_, label_encoder.
 ↪transform(label_encoder.classes_)))
print("Genre mapping:", genre_mapping)
```

```
Features shape: (7918, 518)
```

```
Labels shape: (7918,)
Genre mapping: {'Electronic': 0, 'Experimental': 1, 'Folk': 2, 'Hip-Hop': 3,
'Instrumental': 4, 'International': 5, 'Pop': 6, 'Rock': 7}
```

[7]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Normalize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert data to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

[8]:
```python
# Define a custom Dataset class
class GenreDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]

# Create Dataset and DataLoader
train_dataset = GenreDataset(X_train, y_train)
test_dataset = GenreDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Define the neural network
```

```python
class GenreClassifier(nn.Module):
    def __init__(self, input_size, num_classes):
        super(GenreClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

[18]:
```python
# Initialize the model, loss function, and optimizer
input_size = 518  # Number of input features
num_classes = 8   # Number of classes
model = GenreClassifier(input_size, num_classes=8)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

def train_model(model, train_loader, criterion, optimizer, epochs):
    model.train()
    total_loss = 0
    correct = 0
    total = 0

    for data, labels in train_loader:
        data, labels = data.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(data)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        _, predicted = outputs.max(1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    train_loss = total_loss / len(train_loader)
    train_acc = correct / total
    return train_loss, train_acc

# Testing loop
def test_model(model, test_loader, criterion, device):
    model.eval()
```

```
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for features, labels in test_loader:
            features, labels = features.to(device), labels.to(device)

            # Forward pass
            outputs = model(features)
            loss = criterion(outputs, labels)

            # Track loss and accuracy
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

    epoch_loss = running_loss / len(test_loader)
    epoch_acc = correct / total
    return epoch_loss, epoch_acc
```

[19]:
```
# Training the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

num_epochs = 20
for epoch in range(num_epochs):
    train_loss, train_acc = train_model(model, train_loader, criterion,␣
 ↪optimizer, epochs=10)
    test_loss, test_acc = test_model(model, test_loader, criterion, device)

    print(f"Epoch {epoch+1}/{num_epochs}")
    print(f"Train Loss: {train_loss:.4f}, Train Accuracy: {train_acc:.4f}")
    print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_acc:.4f}")

# Save the trained model
torch.save(model.state_dict(), 'genre_classifier.pth')
print("Model saved to 'genre_classifier.pth'")
```

```
Epoch 1/20
Train Loss: 1.4521, Train Accuracy: 0.4847
Test Loss: 1.2609, Test Accuracy: 0.5612
Epoch 2/20
Train Loss: 1.1181, Train Accuracy: 0.6080
Test Loss: 1.1971, Test Accuracy: 0.5903
Epoch 3/20
Train Loss: 0.9738, Train Accuracy: 0.6514
```

```
Test Loss: 1.1885, Test Accuracy: 0.5966
Epoch 4/20
Train Loss: 0.8518, Train Accuracy: 0.7024
Test Loss: 1.2233, Test Accuracy: 0.5896
Epoch 5/20
Train Loss: 0.7585, Train Accuracy: 0.7384
Test Loss: 1.2281, Test Accuracy: 0.6061
Epoch 6/20
Train Loss: 0.6513, Train Accuracy: 0.7772
Test Loss: 1.2678, Test Accuracy: 0.5979
Epoch 7/20
Train Loss: 0.5562, Train Accuracy: 0.8118
Test Loss: 1.3424, Test Accuracy: 0.6048
Epoch 8/20
Train Loss: 0.4712, Train Accuracy: 0.8420
Test Loss: 1.4261, Test Accuracy: 0.6067
Epoch 9/20
Train Loss: 0.3850, Train Accuracy: 0.8735
Test Loss: 1.4868, Test Accuracy: 0.5934
Epoch 10/20
Train Loss: 0.3189, Train Accuracy: 0.9020
Test Loss: 1.6245, Test Accuracy: 0.5953
Epoch 11/20
Train Loss: 0.2362, Train Accuracy: 0.9313
Test Loss: 1.7547, Test Accuracy: 0.5922
Epoch 12/20
Train Loss: 0.1807, Train Accuracy: 0.9471
Test Loss: 1.8530, Test Accuracy: 0.5909
Epoch 13/20
Train Loss: 0.1350, Train Accuracy: 0.9651
Test Loss: 2.0017, Test Accuracy: 0.5859
Epoch 14/20
Train Loss: 0.1177, Train Accuracy: 0.9686
Test Loss: 2.1387, Test Accuracy: 0.5859
Epoch 15/20
Train Loss: 0.1009, Train Accuracy: 0.9730
Test Loss: 2.2478, Test Accuracy: 0.5878
Epoch 16/20
Train Loss: 0.0851, Train Accuracy: 0.9779
Test Loss: 2.4011, Test Accuracy: 0.5903
Epoch 17/20
Train Loss: 0.1165, Train Accuracy: 0.9670
Test Loss: 2.6224, Test Accuracy: 0.5789
Epoch 18/20
Train Loss: 0.1336, Train Accuracy: 0.9610
Test Loss: 2.6199, Test Accuracy: 0.5833
Epoch 19/20
Train Loss: 0.0773, Train Accuracy: 0.9769
```

```
Test Loss: 2.5974, Test Accuracy: 0.5890
Epoch 20/20
Train Loss: 0.0266, Train Accuracy: 0.9970
Test Loss: 2.6850, Test Accuracy: 0.5909
Model saved to 'genre_classifier.pth'
```

[20]:

```python
import random

# Function to test the model with 10 random songs
def test_random_songs(model, X, y, metadata_df, device, label_encoder):
    model.eval()  # Set the model to evaluation mode

    # Select 10 random indices from the test dataset
    random_indices = random.sample(range(len(X)), 10)

    # Get the corresponding features, true labels, and metadata
    random_features = X[random_indices]
    true_labels = y[random_indices].numpy()

    # Use the original indices of the test set to retrieve metadata
    metadata = metadata_df.iloc[random_indices]

    # Move features to the appropriate device
    random_features = random_features.to(device)

    # Predict genres
    with torch.no_grad():
        outputs = model(random_features)
        _, predicted_labels = torch.max(outputs, 1)
        predicted_labels = predicted_labels.cpu().numpy()  # Move predictions␣
↪back to CPU

    # Decode the true and predicted labels
    true_genres = label_encoder.inverse_transform(true_labels)
    predicted_genres = label_encoder.inverse_transform(predicted_labels)

    # Display the results
    print(f"{'Track ID':<10} {'Title':<30} {'True Genre':<15} {'Predicted␣
↪Genre':<15}")
    print("-" * 70)
    for i in range(len(random_indices)):
        track_id = metadata.index[i]
        title = metadata.iloc[i]['track_title'] if 'track_title' in metadata.
↪columns else "Unknown"
        true_genre = true_genres[i]
        predicted_genre = predicted_genres[i]
```

```python
        print(f"{track_id:<10} {title:<30} {true_genre:<15} {predicted_genre:
↪<15}")

# Example usage
test_random_songs(model, X_test, y_test, tracks_df, device, label_encoder)
```

```
Track ID   Title                          True Genre      Predicted Genre
----------------------------------------------------------------------
1154       Hello Heartstring              Pop             Pop
416        Smyrna Snow Walk               Rock            Rock
1082       Nam Nhi-tu                     International   International
481        Council Bluffs                 Folk            Folk
891        Your One Mind                  Rock            Rock
564        The Sugar Society              Electronic      International
1620       Track 01                       Experimental    Rock
1525       Scovil                         International   International
458        Hunt Like Devil 4              Pop             Folk
1127       Do I Tingle? Up?               Rock            Pop
```