(An Attempt in) Predicting Genre of Music with Neural Networks

USING SPOTIFY API DATA

Josiah Wu
MATHEMATICS, YEAR 1 | ICDSS DATA BRAWL 2021-22 ENTRY

Contents

Abstract

Spotify created an API that allows developers to access metadata of musical tracks – these data are the "audio features" of a track. This research aims to create a neural network that predicts the genre of a musical track based on these "audio features".

The data is first cleaned by removing irrelevant information and deleting rows containing missing values. Columns of categorical variables are then one-hot encoded into the data set. To train the neural network, the data set is split into two sets of data – the training set and the test set. The training set is used to train the neural network model. After that, the model is used to predict musical genres with training and test data, and they're compared with each other based on classification reports (this is known as cross-validation). This is to ensure no over-fitting has occurred.

The results show that the model can only achieve at most ~54% accuracy. This is not largely due to the fault of the neural network, but the data set itself. The data set fails to account for fusion music – music that contains features from more than one genre, which leads to a great reduction in accuracy. Another source of error may be due to the removal of rows containing unknown values.

Introduction

Spotify is the biggest music streaming provider in the world; it has around 381 million active users across 184 markets[1]. In late 2014, Spotify released a Web API that allowed developers to retrieve Spotify metadata, including data of numerous albums and artists[2]. In particular, the main scope of this research is the metadata for individual tracks' audio features.

According to the Spotify Web API documentation, each audio track on Spotify is equipped with data about the audio track itself[3]. These are referred to as "audio features" of a track; they are computed based on the **elements of music** – dynamics, rhythm, structure, melody, instrumentation, texture and harmony.

The main objective of this research is to utilize a dataset, which contains "audio features" of numerous audio tracks, to train a neural network that effectively predicts the genre of any audio track on Spotify. Unfortunately, I'm only able to train a neural network up to at most 54% accuracy despite using different methods to improve it.

Methodology: Cleaning the Dataset

The table below shows the heading of each column within the given dataset. Descriptions are either found from the original Spotify Web API documentation[3,4] or the source of the dataset[5].

Columns	Description	Data Type & Range
instance_id	Unique ID for each track	$int;\ 20002 \leq x \leq 91759$
artist_name	Name of the artist who composed the track	$string;$
track_name	Name of the track	$string;$
acousticness	A confidence measure of whether the track is acoustic i.e. recorded with only acoustic instruments (e.g. voice, piano).	$float;\ 0 \leq x \leq 1$ 1 represents high confidence the track is acoustic.
danceability	Describes how suitable a track is for dancing based on combinations of musical elements such as tempo, rhythm stability, beat strength and overall regularity.	$float;\ 0 \leq x \leq 1$ 1 represents the track is very suitable for dancing.
duration_ms	The duration of the track in milliseconds. **Outputs -1 if unknown.**	$int;\ 0 < x < +\infty$
energy	A perceptual measure of intensity and activity of a track based on combinations of dynamic range, loudness, timbre, onset rate and general entropy.	$float;\ 0 \leq x \leq 1$ 1 represents the track is very intense and energetic.
instrumentalness	A confidence measure of whether a track contains no vocals.	$float;\ 0 \leq x \leq 1$ 1 represents high confidence that the track contains no vocals.
key	The key of the track using standard Pitch Class notation.	$string;\ x \in \{C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#, B\}$
liveness	A confidence measure of whether a track is performed live (e.g. in a concert).	$float;\ 0 \leq x \leq 1$ 1 represents high confidence that the track is performed live.
loudness	The overall loudness of a track in decibels (dB).	$float;\ -\infty < x < \infty$
mode	Modality of a track.	$string;\ x \in \{Minor, Major\}$
speechiness	The presence of spoken words in a track.	$float;\ 0 \leq x \leq 1$
tempo	The overall estimated tempo of the track in beats per minute (BPM) **Outputs '?' if unknown.**	$float;\ 0 < x < \infty$
obtained_date	Date for which the track metadata is scraped from the Spotify Web API.	$date;$

valence	Describes the musical mood of a track. Higher valence → cheerful, uplifting Lower valence → aggressive, solemn	$float;\ 0 \le x \le 1$
music_genre	The genre of the audio track.	$string;\ x \in \{Electronic, Jazz, Alternative, Country, Rap, Blues, Rock, Classical, Hip-Hop\}$

Below are the first five rows of the dataset:

```
   instance_id             artist_name                track_name  popularity  \
0      32894.0                Röyksopp    Röyksopp's Night Out        27.0
1      46652.0   Thievery Corporation        The Shining Path        31.0
2      30097.0          Dillon Francis                Hurricane        28.0
3      62177.0                Dubloadz                    Nitro        34.0
4      24907.0             What So Not           Divide & Conquer        32.0

   acousticness  danceability  duration_ms  energy  instrumentalness key  \
0       0.00468         0.652         -1.0   0.941           0.79200  A#
1       0.01270         0.622     218293.0   0.890           0.95000   D
2       0.00306         0.620     215613.0   0.755           0.01180  G#
3       0.02540         0.774     166875.0   0.700           0.00253  C#
4       0.00465         0.638     222369.0   0.587           0.90900  F#

   liveness  loudness   mode  speechiness              tempo obtained_date  \
0     0.115    -5.201  Minor       0.0748            100.889         4-Apr
1     0.124    -7.043  Minor       0.0300  115.00200000000001         4-Apr
2     0.534    -4.617  Major       0.0345            127.994         4-Apr
3     0.157    -4.498  Major       0.2390            128.014         4-Apr
4     0.157    -6.266  Major       0.0413            145.036         4-Apr

   valence music_genre
0    0.759  Electronic
1    0.531  Electronic
2    0.333  Electronic
3    0.270  Electronic
4    0.323  Electronic
```

Before the data can be used for actual training, it first has to be cleaned and reorganized.

Firstly, we remove columns that are irrelevant – i.e. columns whose values do not affect the prediction of the genre of a track. These columns include 'instance_id', 'obtained_date', 'artist_name' and 'track_name'. They are irrelevant because they are not related to the elements of music. The data also contains some blank rows (e.g. from rows 10000 to 10004), so these rows are removed too.

```
In [265]: ## remove irrelevant columns
          del df['instance_id'], df['artist_name'], df['track_name'], df['obtained_date']
          print(df.loc[9995:10005])

       popularity  acousticness  danceability  duration_ms  energy  \
9995         39.0      0.030100         0.504     302080.0   0.860
9996         33.0      0.000456         0.517     258480.0   0.868
9997         21.0      0.106000         0.527     134787.0   0.262
9998         44.0      0.030300         0.271     275933.0   0.969
9999         14.0      0.020000         0.573     226374.0   0.921
10000         NaN           NaN           NaN          NaN     NaN
10001         NaN           NaN           NaN          NaN     NaN
10002         NaN           NaN           NaN          NaN     NaN
10003         NaN           NaN           NaN          NaN     NaN
10004         NaN           NaN           NaN          NaN     NaN
10005        44.0      0.006210         0.711     285987.0   0.621

       instrumentalness  key  liveness  loudness   mode  speechiness  \
9995           0.000038   F#     0.254    -4.059  Major       0.1380
9996           0.000594    B     0.183    -3.696  Minor       0.0343
9997           0.167000    F     0.146   -17.812  Major       0.0394
9998           0.000490   C#     0.301    -2.539  Major       0.0678
9999           0.000004   F#     0.325    -3.841  Major       0.0734
10000               NaN  NaN       NaN       NaN    NaN          NaN
10001               NaN  NaN       NaN       NaN    NaN          NaN
10002               NaN  NaN       NaN       NaN    NaN          NaN
10003               NaN  NaN       NaN       NaN    NaN          NaN
10004               NaN  NaN       NaN       NaN    NaN          NaN
10005          0.029700    G     0.159    -7.429  Major       0.0382
```

Before

```
In [266]: ## remove blank rows
          df = df.dropna(how='any',axis=0)
          print(df.loc[9995:10005])

       popularity  acousticness  danceability  duration_ms  energy  \
9995         39.0      0.030100         0.504     302080.0   0.860
9996         33.0      0.000456         0.517     258480.0   0.868
9997         21.0      0.106000         0.527     134787.0   0.262
9998         44.0      0.030300         0.271     275933.0   0.969
9999         14.0      0.020000         0.573     226374.0   0.921
10005        44.0      0.006210         0.711     285987.0   0.621

       instrumentalness key  liveness  loudness   mode  speechiness  \
9995           0.000038  F#     0.254    -4.059  Major       0.1380
9996           0.000594   B     0.183    -3.696  Minor       0.0343
9997           0.167000   F     0.146   -17.812  Major       0.0394
9998           0.000490  C#     0.301    -2.539  Major       0.0678
9999           0.000004  F#     0.325    -3.841  Major       0.0734
10005          0.029700   G     0.159    -7.429  Major       0.0382
```

After

Furthermore, for unknown values in columns 'tempo' and 'duration_ms', they are computed as '?' and -1 respectively. From our computation, ~9.9% of the tracks in the dataset have an unknown duration, whereas ~10.0% of the tracks have an unknown tempo. To solve this problem, there are two possible options:

I. Replace the unknown values with the median/mean value (This is known as **imputation**)
II. Delete rows containing the unknown values.

During the data cleaning process, option II is chosen; this is because if ~10% of the data is replaced with the mean/median value, it can greatly reduce the variance of the dataset, making predictions more difficult. Also, through computation the data remain **balanced** (i.e. each category has approximately the same amount of data rows) after having removed the unknown rows:

```
In [267]: ## calculate proportion of unknown data
          unknown_duration = df[df['duration_ms']==-1].shape[0]
          unknown_tempo = df[df['tempo']=='?'].shape[0]
          print(unknown_duration/(df.shape[0]))
          print(unknown_tempo/(df.shape[0]))

          0.09878
          0.0996

In [268]: ## delete rows containing unknown values
          df = df[df['tempo']!='?']
          df = df[df['duration_ms']!=-1.0]
          df['tempo'] = df['tempo'].astype(float)
          print(df['music_genre'].value_counts())

          Rock           4099
          Hip-Hop        4077
          Anime          4064
          Jazz           4064
          Alternative    4051
          Country        4049
          Blues          4046
          Rap            4042
          Classical      4036
          Electronic     4032
          Name: music_genre, dtype: int64
```

Next, we want to convert the column of music_genre into integers, so that it can be used as training data. The dictionary function is therefore utilized to do so. Note that each musical genre is mapped into a distinct integer as follows:

0	Electronic	5	Rap
1	Anime	6	Blues
2	Jazz	7	Rock
3	Alternative	8	Classical
4	Country	9	Hip-Hop

```
In [269]: ## converting 'music_genre' from str to int
df_genre = df['music_genre']
genre = df_genre.to_numpy()
GenreDict = {
    "Electronic": 0,
    "Anime": 1,
    "Jazz": 2,
    "Alternative": 3,
    "Country": 4,
    "Rap": 5,
    "Blues": 6,
    "Rock": 7,
    "Classical": 8,
    "Hip-Hop": 9,
}
def transform_genres(arr):
    genre_list = []
    for x in range(arr.shape[0]):
        g = GenreDict[arr[x]]
        genre_list.append(g)
    return genre_list
df['music_genre'] = transform_genres(genre)
print(df['music_genre'].head())
```

```
1    0
2    0
3    0
4    0
6    0
Name: music_genre, dtype: int64
```

Finally, we need to convert columns ‘key’ and ‘mode’, both of which are categorical variables, into numerical variables (as training data can only contain float/integer values). The values from ‘key’ and ‘mode’ are first combined to create a new column of values called **‘scale’**, which is then **one-hot encoded** into the data frame using the pandas.getdummies() function.

```
In [270]: ## combine 'key' and 'mode' to 'scale' and then one-hot encode 'scale'
df_key = df[['key','mode']].fillna("")
key = df_key.to_numpy()

KeyDict = {
    "Major": "",
    "Minor": "m",
    "":""
}
def transform_scales(arr):
    scale_list = []
    for x in range(arr.shape[0]):
        rootnote = arr[x][0]
        mode = KeyDict[arr[x][1]]
        if rootnote != "":
            scale = rootnote + mode
        scale_list.append(scale)
    return scale_list
df['scale'] = transform_scales(key)

y = pd.get_dummies(df.scale, prefix='scale')
del df['key'], df['mode']
df = df.drop('scale',axis=1)
df = df.join(y)
print(df.head())
```

```
   popularity  acousticness  danceability  duration_ms  energy  \
1        31.0       0.01270         0.622     218293.0   0.890
2        28.0       0.00306         0.620     215613.0   0.755
3        34.0       0.02540         0.774     166875.0   0.700
4        32.0       0.00465         0.638     222369.0   0.587
6        46.0       0.02890         0.572     214408.0   0.803

   instrumentalness  liveness  loudness  speechiness    tempo  ...  scale_E  \
1          0.950000     0.124    -7.043       0.0300  115.002  ...        0
2          0.011800     0.534    -4.617       0.0345  127.994  ...        0
3          0.002530     0.157    -4.498       0.2390  128.014  ...        0
4          0.909000     0.157    -6.266       0.0413  145.036  ...        0
6          0.000008     0.106    -4.294       0.3510  149.995  ...        0

   scale_Em  scale_F  scale_F#  scale_F#m  scale_Fm  scale_G  scale_G#  \
1         0        0         0          0         0        0         0
2         0        0         0          0         0        0         1
3         0        0         0          0         0        0         0
4         0        0         1          0         0        0         0
6         0        0         0          0         0        0         0

   scale_G#m  scale_Gm
1          0         0
2          0         0
3          0         0
4          0         0
6          0         0
```

The reason columns ‘key’ and ‘mode’ are combined is because, in music, combinations of key and mode form scales, and each scale is harmonically different from one another. For example, B minor is different from C minor, and C major is different from C minor, etc. Hence the two columns must be combined.

Methodology: Training the Neural Network

In Machine Learning, it is common practice to split up the dataset into two sets of data – **Training Set** and **Test Set.** For this research, the dataset is split to the ratio of 8:2, in which the former represents the size of the training set and the latter represents the size of the test set.

```
In [273]: ## Splitting the dataset into TRAINING data and TEST data
          import tensorflow as tf
          import keras
          import pandas as pd
          from sklearn.model_selection import train_test_split

          X = df.drop(columns=['music_genre'])
          Y = df[['music_genre']]
          X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Now we have to make sure the training set is **normalized**, in other words, all data values are between 0 and 1. This is not the case for columns ‘loudness’, ‘duration_ms’, ‘popularity’ and ‘tempo’.

To ensure all the values of these columns lie between 0 and 1, we apply a function $F: \mathbb{R} \to [0,1]$ to any data value $x_j^{(i)}$ in column $X^{(i)}$:

$$F(x_j^{(i)}) = \frac{x_j^{(i)} - \min(X^{(i)})}{\max(X^{(i)}) - \min(X^{(i)})} \to x_j^{(i)}$$

in which $X^{(i)} = \{x_1^{(i)}, x_2^{(i)} \dots x_n^{(i)}\}$. Moreover, all the $\max(X^{(i)})$ and $\min(X^{(i)})$ are recorded so that we can apply the same mapping to the test set for prediction purposes.

In [274]:
```
## normalising the TRAINING data
MaxDict = {
    'duration_ms': X_train['duration_ms'].max(),
    'tempo': X_train['tempo'].max(),
    'popularity': X_train['popularity'].max(),
    'loudness': X_train['loudness'].max()
}
MinDict = {
    'duration_ms': X_train['duration_ms'].min(),
    'tempo': X_train['tempo'].min(),
    'popularity': X_train['popularity'].min(),
    'loudness': X_train['loudness'].min()
}

def NormalizeTrain(c):
    return (X_train[c]-MinDict[c])/(MaxDict[c]-MinDict[c])

X_train['loudness'] = NormalizeTrain('loudness')
X_train['duration_ms'] = NormalizeTrain('duration_ms')
X_train['popularity'] = NormalizeTrain('popularity')
X_train['tempo'] = NormalizeTrain('tempo')
print(X_train.head())
```

```
       popularity  acousticness  danceability  duration_ms  energy  \
49054    0.595960      0.007500         0.796     0.048330   0.357
44947    0.292929      0.760000         0.169     0.057152   0.158
68       0.272727      0.000355         0.814     0.081022   0.877
3934     0.303030      0.005330         0.421     0.033191   0.800
14954    0.333333      0.488000         0.814     0.045129   0.693

       instrumentalness  liveness  loudness  speechiness     tempo  ...  \
49054             0.000    0.0966  0.721870       0.3440  0.622238  ...
44947             0.930    0.1040  0.570650       0.0396  0.422236  ...
68                0.455    0.2060  0.820614       0.0533  0.471449  ...
3934              0.332    0.3810  0.840820       0.3030  0.836814  ...
14954             0.190    0.1080  0.813695       0.0360  0.487600  ...

       scale_Gm  scale_G  scale_G#m  scale_G#  scale_Am  scale_A  scale_A#m  \
49054         0        0          0         0         0        0          0
44947         0        0          0         0         0        0          0
68            0        0          0         0         0        0          0
3934          0        0          0         0         0        0          0
14954         0        0          0         0         0        0          0

       scale_A#  scale_Bm  scale_B
49054         0         0        0
44947         0         0        0
68            0         0        0
3934          0         0        1
14954         0         1        0

[5 rows x 35 columns]
```

We then construct the neural network. For this research, a 35-35-10-10 multilayer perceptron is used with activation function Sigmoid. The choice of the optimizer is Adam, as it consumes relatively little computer memory[7]. As the model is used for multiclass classification, the loss function (or cost function) is chosen to be Sparse Categorical Cross-Entropy[8], and the metric used for deep learning is Sparse Categorical Accuracy.

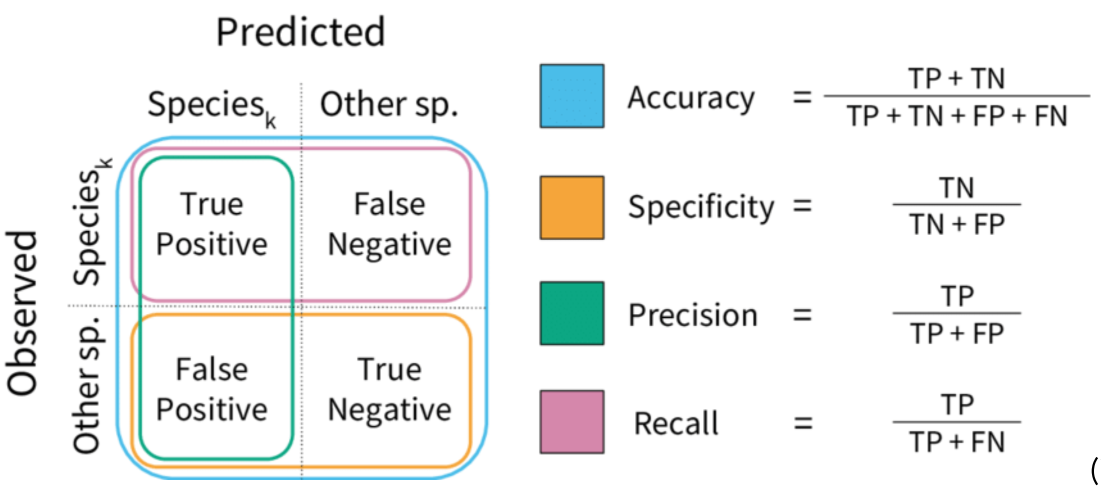
Now we can train the neural network with 15 epochs with the batch size of 16 for computational efficiency:

```
In [289]: ## Constructing the Neural Network
          model = keras.Sequential()

          model.add(keras.layers.Dense(35, input_shape=(35,), activation='sigmoid'))
          model.add(keras.layers.Dense(10, activation='sigmoid'))

          optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
          model.compile(optimizer=optimizer,loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                        metrics=['sparse_categorical_accuracy'])

In [291]: ## Training
          Y_pred_train = model.fit(X_train, Y_train, batch_size=16, epochs=15)

          Epoch 1/15
          2028/2028 [==============================] - 8s 4ms/step - loss: 1.4080 - sparse_categorical_accuracy: 0.4913
          Epoch 2/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.3589 - sparse_categorical_accuracy: 0.5012
          Epoch 3/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.3245 - sparse_categorical_accuracy: 0.5129
          Epoch 4/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.3015 - sparse_categorical_accuracy: 0.5163
          Epoch 5/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.2835 - sparse_categorical_accuracy: 0.5215
          Epoch 6/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.2715 - sparse_categorical_accuracy: 0.5239
          Epoch 7/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.2614 - sparse_categorical_accuracy: 0.5259
          Epoch 8/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.2531 - sparse_categorical_accuracy: 0.5296A: 1s -
          loss: 1.248
          Epoch 9/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.2465 - sparse_categorical_accuracy: 0.5307
          Epoch 10/15
          2028/2028 [==============================] - 7s 3ms/step - loss: 1.2410 - sparse_categorical_accuracy: 0.5338
```

Results

Once we have trained the neural network, we will need to **cross-validate** our model; this is done by computing the **confusion matrix** of the test Data and training Data. A confusion matrix is, in a nutshell, a contingency table between two variables – predictions from the model, and the actual result. In the context of this research, the (i, j)th entry of the matrix represents the number of tracks which the model predicts to be category i, but it's in fact category j.

We can then do a **classification report** using the confusion matrix. In general, a classification report provides different metrics to visualize the efficiency of our model in predicting musical genres. Such metrics include accuracy, precision and recall. The following diagrams demonstrate how these metrics are calculated:



(i)

A classification report for the training data and the test data is essential because we can compare them to check for **over-fitting**. If the metrics of the two reports are completely different, then it shows that the model is ineffective in making predictions with new data. Below are the classification reports for the test data and the training data, respectively:

In [307]:
```
## Predictions
Y_pred_test = model.predict(X_test)
Y_pred_tra = model.predict(X_train)
```

In [308]:
```
## Computing the Confusion Matrix
from sklearn import metrics
import random

predict = []
predict2 = []
for k in range(Y_pred_test.shape[0]):
    verdict = Y_pred_test[k].tolist()
    VerdictGenre = verdict.index(max(verdict))
    predict.append(VerdictGenre)

for k in range(Y_pred_tra.shape[0]):
    verdict2 = Y_pred_tra[k].tolist()
    VerdictGenre2 = verdict2.index(max(verdict2))
    predict2.append(VerdictGenre2)

print('Confusion Matrix - Test Dataset')
print(metrics.confusion_matrix(Y_test, predict))
print(metrics.classification_report(Y_test,predict))

print('Confusion Matrix - Train Dataset')
print(metrics.confusion_matrix(Y_train, predict2))
print(metrics.classification_report(Y_train,predict2))
```

```
Confusion Matrix - Test Dataset
[[469  77 107  64  31  19  33  16   7  20]
 [ 53 520  28  31  26   0  73   4  92   0]
 [107  24 404  17  37   1  79  36  48  29]
 [ 53   4  66 245 153  39  21 151   1  64]
 [ 30  27  44  73 372  11  81 149   0  17]
 [  8   0   5  36  15 365   0  92   0 261]
 [ 39 153 110  25  82   1 339  31  11   1]
 [  7   1  24 100  56  67   0 571   1  13]
 [ 12  36  52  19   6   0  16   5 675   0]
 [ 10   0  12  47  19 350   3  37   0 346]]
              precision    recall  f1-score   support

           0       0.60      0.56      0.58       843
           1       0.62      0.63      0.62       827
           2       0.47      0.52      0.49       782
           3       0.37      0.31      0.34       797
           4       0.47      0.46      0.46       804
           5       0.43      0.47      0.45       782
           6       0.53      0.43      0.47       792
           7       0.52      0.68      0.59       840
           8       0.81      0.82      0.82       821
           9       0.46      0.42      0.44       824

    accuracy                           0.53      8112
   macro avg       0.53      0.53      0.53      8112
weighted avg       0.53      0.53      0.53      8112
```

```
Confusion Matrix - Train Dataset
[[1755  291  416  205  109   69  159   88   23   74]
 [ 160 2145   82   74  106    4  289   11  365    1]
 [ 375  100 1712   77  186   19  365  115  244   89]
 [ 194   23  254 1088  541  170   72  611    5  296]
 [ 118   75  196  209 1605   48  373  553    9   59]
 [  13    2   44  170   55 1571    0  348    0 1057]
 [ 151  529  476  140  277    6 1459  158   52    6]
 [  19   10  106  355  265  239   10 2215    5   35]
 [  74  187  147   67   15    0   65   16 2644    0]
 [  38    0   55  148   67 1346    4  184    0 1411]]
              precision    recall  f1-score   support

           0       0.61      0.55      0.58      3189
           1       0.64      0.66      0.65      3237
           2       0.49      0.52      0.51      3282
           3       0.43      0.33      0.38      3254
           4       0.50      0.49      0.50      3245
           5       0.45      0.48      0.47      3260
           6       0.52      0.45      0.48      3254
           7       0.52      0.68      0.59      3259
           8       0.79      0.82      0.81      3215
           9       0.47      0.43      0.45      3253

    accuracy                           0.54     32448
   macro avg       0.54      0.54      0.54     32448
weighted avg       0.54      0.54      0.54     32448
```

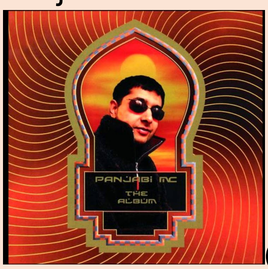
Evidently, the metrics are very similar in both reports, so no over-fitting has likely occurred.

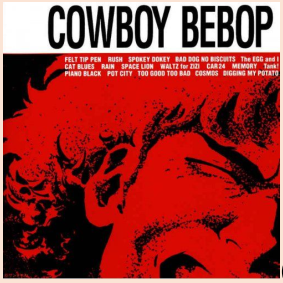
Furthermore, the classification reports show that the model recognizes musical Classical Music (category 8) most effectively, with around 80% precision. However, it is the least effective in recognizing Alternative Music (category 3) and Rap Music (category 5), both of which have merely 40% precision.

Evaluation

Overall, the model is merely ~54% accurate – not reliable enough for actual prediction purposes. I believe the major source for this lack of accuracy is the construction of the data set. A flawed assumption is being made by the dataset about music – and that is, musical genres are mutually exclusive.

In music, any music which contains characteristics from more than one genre is called **fusion music**[9]. Below are a few famous examples of fusion music.

Piece of Music	Genres
Panjabi MC - Mundian To Bach Ke  (ii) https://www.youtube.com/watch?v=DJztXj2GPfk	Electronic[10], Hip-Hop[10] Bhangra[11] (Punjabi dance music)

Radiohead – Karma Police  (iii) https://www.youtube.com/watch?v=1uYWYWPc9HU	Alternative[12], Rock[12]
The Seatbelts – Tank! (OST of 'Cowboy Bebop')  (iv) https://www.youtube.com/watch?v=UFFa0QoHWvE	Anime[13], Jazz[13]

In fact, there are many different genres in music which is a combination of elements from our list of musical genres (e.g. Alternative Rock, Jazz Rock, etc.). Hence our model is susceptible to classifying fusion music incorrectly, which greatly reduces the accuracy.

Additionally, another source of error may originate from the removal of the rows containing unknown values, as ~10% of the training data is lost.

Ultimately, I believe a better model can be constructed by reframing the data set. One way to do this is by labelling every audio track with an integer between 0 and 99 inclusive, where the first digit and the second digit represent the main genre and the secondary genre (if any) of the track respectively. The main and secondary genres of a track are classified based on metadata from Spotify API. That way, any fusion musical track can be identified with any two genres on our list of ten genres. However, this method requires much more data as the number of classes increased from 10 to 100.

References

1	https://newsroom.spotify.com/company-info/
2	https://developer.spotify.com/documentation/web-api/quick-start/
3	https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features
4	https://developer.spotify.com/documentation/web-api/reference/#/operations/get-track
5	https://www.kaggle.com/vicsuperman/prediction-of-music-genre
6	https://towardsdatascience.com/how-to-increase-the-accuracy-of-a-neural-network-9f5d1c6f407d
7	https://machinelearningknowledge.ai/keras-optimizers-explained-with-examples-for-beginners/
8	https://neptune.ai/blog/keras-loss-functions
9	https://www.soundsoftheundergroundtour.com/the-origin-of-fusion-music.html
10	https://www.bbc.co.uk/bitesize/guides/zt2v34j/revision/3
11	https://www.discogs.com/master/84644-Panjabi-MC-Mundian-To-Bach-Ke
12	https://www.allmusic.com/album/karma-police-mw0000937923
13	https://cowboybebop.fandom.com/wiki/Tank!
i	https://www.researchgate.net/figure/Model-performance-metrics-Visual-representation-of-the-classification-model-metrics_fig1_328148379
ii	https://i.pinimg.com/originals/e3/be/2c/e3be2c37291f1796aa294884f9854643.jpg
iii	http://mediad.publicbroadcasting.net/p/shared/npr/styles/placed_wide/nprshared/201705/527922859.jpg
iv	https://www.music-bazaar.com/world-music/album/854400/Cowboy-Bebop-Remastered-Original-Soundtrack/