

Music-ML-Magic

- A music analysis system that classifies music into different genres, making use of the GTZAN dataset.

Step by Step explanation of code(as in github repository):

1. Dataset:

- I used the GTZAN dataset (from kaggle).
- Content:
 - Genres original: A collection of 10 genres with 100 audio files each, all having a length of 30 seconds(in .wav format)
 - Images original: A visual representation for each audio file. (Which can be used for CNN)
 - 2 CSV files: containing the audio file's features.
 - A mean and variance computed over numerous features that can be recovered from an audio stream are contained in one file, one for each song (30 seconds long).
 - The other file is identical structurally, but the songs were previously divided into 3 second audio files, increasing the amount of data we feed into our classification algorithms by a factor of 10.
 - Problem:
 - Small dataset
 - Corrupted files
 - Overlap between genres - boundaries not well defined, leading to instances where a particular audio clip could be interpreted as belonging to multiple genres simultaneously.

Step 1: Data preprocessing and splitting of data

- It demonstrates loading audio data, extracting MFCC features, and splitting the data for machine learning tasks.

- Libraries used:
 - 'os' : Used for handling file paths.
 - 'pandas'(pd): Used for loading and manipulating the metadata from a CSV file.
 - 'numpy'(np): Used for working with arrays of audio features.
 - 'librosa': Used for loading audio files and extracting MFCC features.
 - 'tqdm': Used for displaying progress bars in loops.
 - 'sklearn.model_selection': Used for splitting the data into training and testing sets.
- I have defined the file paths to the audio dataset and the 30 sec feature CSV file. I used pandas to read the CSV file and load it into a DataFrame named feat_30.
 - Change the paths for the above files accordingly considering in which directories it is present.
- The function, feature_extractor, takes a file path as input, loads the audio using Librosa, extracts MFCC features, computes the mean of these features, and returns the resulting feature vector.
 - The res_type parameter specifies the resampling method to be used when loading the audio file.
 - Resampling involves changing the sample rate of the audio while preserving its pitch.
 - I used 'kaiser_fast', but 'librosa.load' offers a few other options as well, though for analysis of audio data, it's often useful to have a consistent sample rate.
- 'tqdm', 'taqaddum' in Arabic means progress. It provides a way to visualise the progress of a loop, making it easier to estimate how much time is remaining for the loop to complete. The basic usage of tqdm involves wrapping an iterable (such as a loop) with the tqdm function.
- In the next loop, I'm iterating through each row in the feat_30 DataFrame.

- For each row, I'm extracting the label and file name, creating the full file path, and calling the `feature_extractor` function to extract MFCC features.
- The extracted features, along with the class label, are then appended to the `extracted_features` list.
- If any errors occur during this process, they are caught and printed. (One of the jazz files is corrupted, so to handle this)
- `Value_counts()` tells the number of audio files in each genre(jazz has 99)
- Then the `extracted_features` list is converted to a new dataframe `extracted_features_df`, which has two columns, 'feature' for the extracted features and 'class' for the class labels.
- After that comes the splitting of the dataset into training and testing. I imported the 'train_test_split' function from the scikit learn library.
 - The `test_size` parameter specifies the proportion of the data to be used for testing(20% in this case) and 'random state' ensures reproducibility of the split.

Types of Algorithms in Machine Learning:

1. Supervised Learning Algorithms
 - These algorithms learn from labelled training data, where each data point is associated with a target label or outcome. The goal is to learn a mapping from input features to the correct output labels.
 - Regression Algorithms: Predicts continuous numerical values.
 - Classification Algorithms: Assigns input data to predefined categories or classes.
2. Unsupervised Learning Algorithms

- These algorithms work with unlabeled data and aim to uncover hidden patterns, structures, or relationships within the data.
 - Clustering Algorithms: Groups similar data points into clusters or segments.
 - Dimensionality Reduction Algorithms: Reduces the number of features while preserving as much relevant information as possible.
- 3. Reinforcement Learning Algorithms
 - In this type of learning, an agent learns to perform actions in an environment to maximise a cumulative reward signal. It learns through trial and error, adjusting its actions based on the outcomes.
- 4. Deep Learning Algorithms
 - These are a subset of machine learning algorithms that involve neural networks with multiple layers. They excel at tasks involving complex data like images, audio, and text.
 - Convolutional Neural Networks (CNNs): Especially suited for image data.
 - Recurrent Neural Networks (RNNs): Designed for sequential data.
 - Transformers: Efficient for sequences, widely used in natural language processing.
 - I trained the models using the various classification algorithms from basic to advanced, and I selected the best model according to the accuracy I got.

Step 2: Training with basic Machine Learning models

- **K Nearest Neighbours:**
 - Assigns a class to a data point based on the majority class of its k closest neighbours.
 - Libraries imported:
 - `from sklearn.neighbors import KNeighborsClassifier`
 - `from sklearn.metrics import accuracy_score`

- Installed library → 1. Threadpool Ctl
 - Required for the KNN algorithm.
- The `n_neighbors` parameter can be changed accordingly to get the highest accuracy.
- Observations:
 - I got an accuracy of 56% for testing data and 72.8% for training data.
 - As the number of neighbours increases, accuracy of the testing data generally decreases.
- **Naive Bayes:**
 - Uses Bayes' theorem to predict the class of a data point based on its feature probabilities.
 - Libraries imported:
 - `from sklearn.naive_bayes import GaussianNB`
 - `from sklearn.preprocessing import LabelEncoder`
 - `from sklearn.metrics import accuracy_score`
 - A Label Encoder is a preprocessing technique that converts categorical labels (text-based) into numerical values. It assigns a unique integer to each category, allowing algorithms to process categorical data as numerical input.
 - To train a model, first a label encoder is used, and then, for calculating the accuracy, it is converted back to the original labels.
 - Observations:
 - Accuracy for the testing data is 41.5% and for the training data is 49.4%.
- **Decision Tree:**
 - Divides data into branches of yes/no decisions to make predictions.
 - Libraries imported:
 - `from sklearn.tree import DecisionTreeClassifier`
 - `from sklearn.metrics import accuracy_score`
 - `from sklearn import tree`

- The tree function is imported from the sklearn library to show the tree diagram.
- The criterion is a measure used to evaluate the quality of a split when building the tree.
- The most common criteria are Gini impurity and Entropy. These criteria help the decision tree algorithm determine the optimal feature and value to split the data on at each node.
 - Gini impurity measures the probability of incorrectly classifying a randomly chosen element if it were randomly labelled according to the distribution of labels in the node
 - Entropy measures the level of disorder or impurity in a set of elements. It's used as a criterion to decide how to split data points into different classes.
- I've used the Gini impurity as the criterion.
- Accuracy for the testing data is 51.5% and for the training dataset is 100%.
- **Logistic Regression:**
 - Models the probability of a binary outcome using a logistic function.
 - Libraries imported:
 - `from sklearn.linear_model import LogisticRegression`
 - `from sklearn.metrics import accuracy_score`
 - The 'max_iter' refers to the maximum number of iterations that the optimization algorithm will perform to find the optimal coefficients for the logistic regression model.
 - Logistic regression uses iterative optimization techniques to find the best-fitting parameters that minimise the loss function.
 - Observations:
 - Accuracy for testing data is 59% and for training data is 70.5%.

- I used max_iter as 1000, because I was getting maximum accuracy by that.
- Applying the concept of the serialisation and deserialisation increases the accuracy of the model.

Step 3: Training with intermediate Machine Learning models:

- **Support vector Machine:**

- Finds a hyperplane that best separates data points of different classes by maximising the margin.
- Libraries Imported:
 - from sklearn.svm import SVC
 - from sklearn.metrics import accuracy_score
 - from sklearn.datasets import make_classification
 - import matplotlib.pyplot as plt
- Accuracy for the testing model is 45.5% and for the training model it is 47%.
- The next code demonstrates the use of the Support Vector Machines for binary classification and visualises the decision boundary along with the support vectors using Matplotlib.
- Various parameters are initialised, like the kernel, regularisation parameter, number of samples, number of features, etc. which can be changed.
- It basically includes, creating a synthetic dataset, fitting an SVM model, creating a meshgrid for plotting, predicting for each point in the meshgrid, creating a scatter plot, highlighting support vectors, adding plots and showing the plot.

- **Random Forest:**

- Constructs multiple decision trees and combines their predictions for more accurate results.
- Imported Libraries:
 - from sklearn.ensemble import RandomForestClassifier
 - from sklearn.metrics import accuracy_score

- Changing the parameter number of estimators affects the accuracy, I got the highest accuracy for n_estimators equal to 300.
- The accuracy for the training dataset is 100% and for the testing dataset it is 67.8%.
- The Highest Accuracy by far from all the basic and intermediate Classification algorithms.
- **Gradient Boosting(XGBoost):**
 - Builds an ensemble of weak models sequentially, each correcting the errors of the previous one.
 - Imported Libraries:
 - from xgboost import XGBClassifier
 - from sklearn.metrics import accuracy_score
 - from sklearn.preprocessing import LabelEncoder
 - To train a model, first a label encoder is used, and then, for calculating the accuracy, it is converted back to the original labels.
 - Parameters:
 - n_estimators: This parameter specifies the number of boosting rounds or decision trees that the XGBoost model will use. I used n_estimators = 200 for getting the highest accuracy score.
 - learning rate: The learning rate controls the step size at which the algorithm adjusts the model's parameters during each iteration. A smaller learning rate makes the algorithm more conservative, leading to slower convergence but potentially better generalisation.
 - Accuracy for the testing data is 61.5% and for the training data is 100%.

Step 4: Finding the model with the highest Accuracy using Advanced level algorithm (deep learning algorithm)

- **Convolutional Neural Networks (CNN):**

- What is it and how it works:
 - Specialised neural networks for image classification, using convolutional layers to capture local patterns.
 - We import the necessary modules from TensorFlow.
 - We create a sequential model, which allows you to stack layers on top of each other sequentially.
 - We need to provide appropriate values for height, width, channels, and num_classes based on your specific dataset and task.
 - Then we add a convolutional layer - is the core building block of a CNN, applying filters to extract different features and helps the network learn hierarchical representations of features from low-level to high-level.
 - Then we add a max-pooling layer - reduces the spatial dimensions of the data, helping to extract important features.
 - Then we repeat the above steps to add more convolutional and pooling layers if needed.
 - We then flatten the 2D output into a 1D vector.
 - We add dense for making the final predictions.
 - The model is then compiled, where you specify the optimizer, loss function and evaluation metrics.
 - optimizer: The optimizer is a mathematical algorithm that adjusts the parameters (weights and biases) of the model during training in order to minimise the loss function. "adam", "sgd", "rmsprop", etc are few optimizers.
 - Loss Function: The loss function quantifies how well the model's predictions match the actual target values in the training data. You might use "binary_crossentropy," and for multi-class classification, "categorical_crossentropy" is often used.

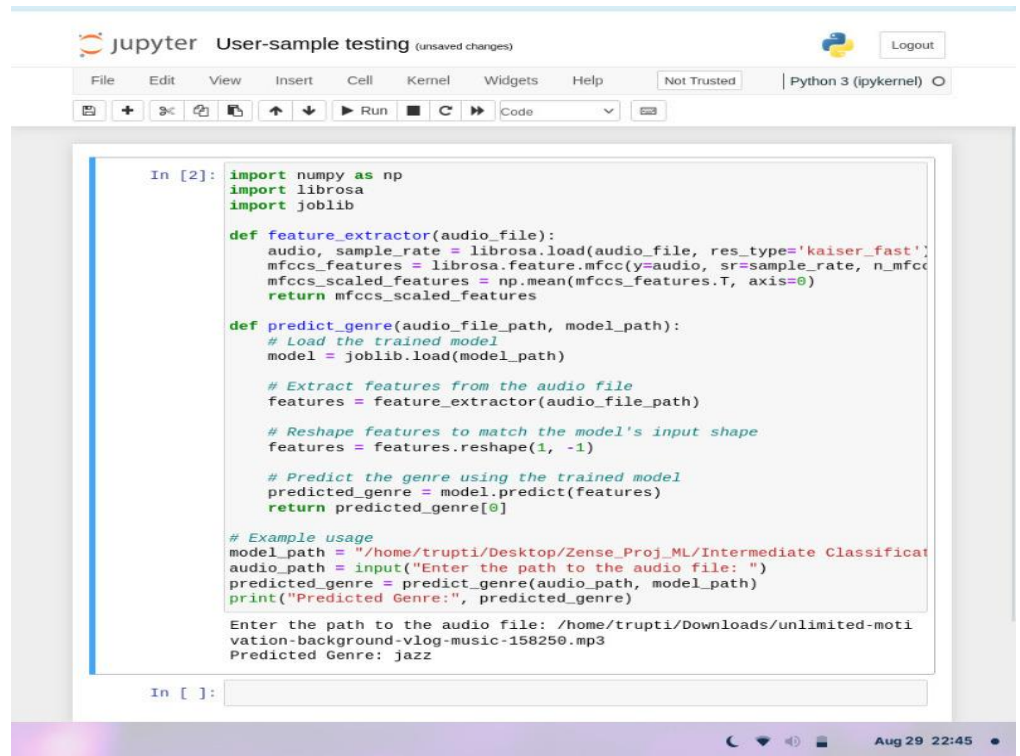
- Metrics: Metrics are used to evaluate the performance of the model during training and testing. Common metrics include "accuracy" for classification tasks, "mean_absolute_error" and "mean_squared_error" for regression tasks, etc.
- So basically, a Convolutional Neural Network (CNN) is a deep learning architecture composed of layers designed to automatically learn and extract hierarchical features from input data, particularly suited for tasks involving images and spatial patterns.
- Since it is a deep learning model, it generally gives higher accuracy.
- Steps I followed for training my model in CNN:
 - Importing necessary libraries
 - Loading a CSV file named 'features_3_sec.csv' using pandas into a DataFrame named 'df'.
 - I dropped the 'filename' column from the DataFrame 'df', as it's not needed for model training.
 - Extracting the target labels (class labels) from the last column of the DataFrame, then using scikit-learn's LabelEncoder to transform these categorical labels into numerical format.
 - Using a StandardScaler to standardise the feature columns (excluding the last column, which is the target). This is done to bring all features to a similar scale and improve convergence during training. I have converted the feature DataFrame to a numpy array and applied the transformation.
 - Splitting the dataset into training and testing sets using scikit-learn's train_test_split function. 33% of the data will be used for testing, and a random seed of 42 is set for reproducibility.

- building a sequential neural network model using Keras. It consists of multiple layers: fully connected (Dense) layers with ReLU activation, and dropout layers to prevent overfitting. The last layer has 10 units (equal to the number of classes) and a softmax activation for classification.
- Defining a function that compiles and trains the model using given parameters. It compiles the model with an optimizer, loss function, and evaluation metrics. Then it trains the model on the training data, using validation data for monitoring. It returns the training history.
- Using the trained model to make predictions on the training and testing data. predict returns class probabilities, and argmax is used to convert these probabilities into class labels.
- Calculating the accuracy of the model predictions using scikit-learn's accuracy_score function and then printing the training and testing accuracies.

Step 5: User-sample testing:

- After this, I imported the joblib module, to load the saved model.
- Then, I Extracted the features from the audio file
- Reshaped the features to match the model's input shape
- Prediction of genre using the trained model
- Since I was getting highest Accuracy with the Random Forest Classifier, in the intermediate classifying algorithms, I saved and loaded this model for the user sample testing, but it can be done for any model.
- Users need to provide the path to the audio file they want to predict the genre for, when it asks for "Enter the path to the audio file: "
- The genre is predicted.

- Sample testing:



The image shows a Jupyter Notebook interface with the title "User-sample testing (unsaved changes)". The notebook contains a single code cell with the following Python code:

```
In [2]: import numpy as np
import librosa
import joblib

def feature_extractor(audio_file):
    audio, sample_rate = librosa.load(audio_file, res_type='kaiser_fast')
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=13)
    mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)
    return mfccs_scaled_features

def predict_genre(audio_file_path, model_path):
    # Load the trained model
    model = joblib.load(model_path)

    # Extract features from the audio file
    features = feature_extractor(audio_file_path)

    # Reshape features to match the model's input shape
    features = features.reshape(1, -1)

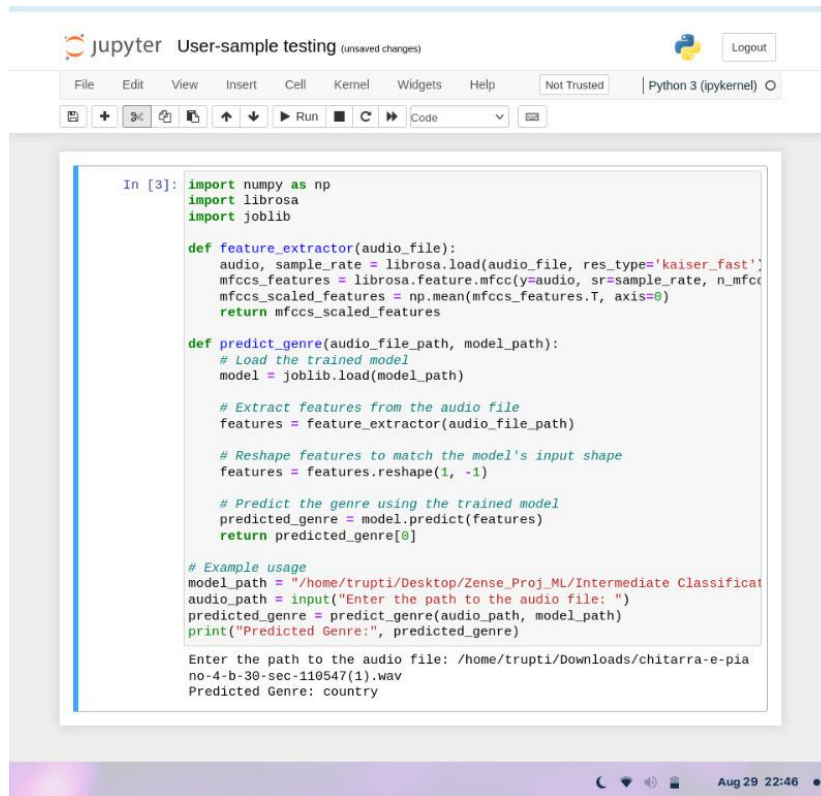
    # Predict the genre using the trained model
    predicted_genre = model.predict(features)
    return predicted_genre[0]

# Example usage
model_path = "/home/trupti/Desktop/Zense_Proj_ML/Intermediate Classification Models/genre_model.joblib"
audio_path = input("Enter the path to the audio file: ")
predicted_genre = predict_genre(audio_path, model_path)
print("Predicted Genre:", predicted_genre)

Enter the path to the audio file: /home/trupti/Downloads/unlimited-motivation-background-vlog-music-158250.mp3
Predicted Genre: jazz
```

The output of the code cell shows the predicted genre as "jazz". The Jupyter Notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom indicating the current time as "Aug 29 22:45".

2.



The screenshot shows a Jupyter Notebook window titled "User-sample testing (unsaved changes)". The interface includes a top bar with the Jupyter logo, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a status bar (Python 3 (pykernel)). Below the menu bar is a toolbar with icons for file operations, cell execution, and output viewing. The main area contains a code cell with the following Python code:

```
In [3]: import numpy as np
import librosa
import joblib

def feature_extractor(audio_file):
    audio, sample_rate = librosa.load(audio_file, res_type='kaiser_fast')
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=13)
    mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)
    return mfccs_scaled_features

def predict_genre(audio_file_path, model_path):
    # Load the trained model
    model = joblib.load(model_path)

    # Extract features from the audio file
    features = feature_extractor(audio_file_path)

    # Reshape features to match the model's input shape
    features = features.reshape(1, -1)

    # Predict the genre using the trained model
    predicted_genre = model.predict(features)
    return predicted_genre[0]

# Example usage
model_path = "/home/trupti/Desktop/Zense_Proj_ML/Intermediate_Classification/genre_model.joblib"
audio_path = input("Enter the path to the audio file: ")
predicted_genre = predict_genre(audio_path, model_path)
print("Predicted Genre:", predicted_genre)

Enter the path to the audio file: /home/trupti/Downloads/chitarra-e-pia-no-4-b-b-39-sec-110547(1).wav
Predicted Genre: country
```

The bottom status bar shows the date and time: Aug 29 22:46.

- END -