Music Genre Classification

A PROJECT REPORT

Under the guidance of
SOURAV GOSWAMI SIR

CALCUTTA INSTITUTE OF ENGINEERING & MANAGEMENT

KOLKATA, WEST BENGAL

(Note: All entries of the Performa of approval should be filled up with appropriate and complete information. Incomplete Performa of approval in any respect will be summarily rejected.)

1. Title of the Project: **MUSIC GENRE CLASSIFICATION**
2. Project Members: **PARIJAT DHAR, SOHAM DUTTA, SUMIT DEY, ABHISHEK SHAW**
3. Name of the guide: **Mr. SOURAV GOSWAMI**
4. Address: **Ardent Computech Pvt. Ltd (An ISO 9001:2008 Certified) CF-137, Sector - 1, Salt Lake City, Kolkata - 700064**

Signature of Student
Date:

Signature of Approver
Date:

Mr. Sourav Goswami Sir

DECLARATION

We hereby declare that the project work being presented in the project proposal entitled “Music Genre Classification” at ARDENT COMPUTECH PVT. LTD, SALTLAKE, KOLKATA, WEST BENGAL, is an authentic work carried out under the guidance of MR. Sourav Goswami. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date:

Name of the Students: **PARIJAT DHAR, SOHAM DUTTA, SUMIT DEY, ABHISHEK SHAW**

Signature of the student

PARIJAT DHAR, SOHAM DUTTA, SUMIT DEY, ABHISHEK SHAW

ACKNOWLEDGEMENT

Success of any project depends largely on the encouragement and guidelines of many others. We take this sincere opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project work.

We would like to show our greatest appreciation to Mr. Sourav Goswami sir, Professor at ARDENT®. We always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at ARDENT®, for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

ABSTRACT

Music genre classification is a crucial task in music information retrieval, enabling efficient organization, recommendation, and retrieval of audio tracks. This project develops a machine learning-based system to classify audio tracks from the GTZAN dataset into ten distinct genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock. Using pre-extracted audio features such as Mel-frequency cepstral coefficients (MFCCs) and spectral characteristics, the dataset is preprocessed with standardization and label encoding. Class imbalances, particularly in underrepresented genres like rock and reggae, are addressed using sample weighting. The CatBoost algorithm, a gradient boosting decision tree model optimized for categorical data and GPU acceleration, is employed for classification. Hyperparameter tuning is performed to maximize validation accuracy, and model performance is evaluated using cross-validation, accuracy, F1-score, and confusion matrices. The final model demonstrates strong classification performance across most genres, with targeted improvements for previously underrepresented classes. This work illustrates the effectiveness of ensemble tree-based methods in audio classification and highlights strategies to mitigate class imbalance and overfitting in multiclass datasets.

INTRODUCTION

In recent years, remarkable progress has been made in computer science and artificial intelligence (AI). Systems such as Watson, Siri, and deep learning models demonstrate that AI can deliver services that are both intelligent and creative. Today, very few companies can operate efficiently without leveraging AI to optimize business processes or reduce costs. AI systems have become indispensable as the world grows increasingly complex, enabling humans to focus on higher-level tasks while computers handle structured, high-quality computational work.However, the rise of AI also raises philosophical and ethical questions. The possibility that machines might possess intelligence unsettles many, as intelligence has traditionally been viewed as a uniquely human trait. If intelligence can be mechanized, what distinguishes humans from machines? The quest to replicate human thought is not new. Throughout history, humans have imagined or even attempted to create artificial intelligences. From the sixteenth century, legends and mechanical devices—such as homunculi, automata, the golem, Mälzel's chess-playing machine, and even Frankenstein—reflect humanity's enduring fascination with reproducing or simulating human intelligence. Ancient Greeks told myths about robotics, while Chinese and Egyptian engineers built intricate automatons.Modern artificial intelligence traces its roots to the symbolic attempts of classical philosophers to formalize human thought. However, AI was not formally recognized as a scientific field until 1956, at the Dartmouth College conference in Hanover, New Hampshire. Pioneering cognitive scientists, including Marvin Minsky, were highly optimistic about AI's potential. As Minsky later wrote in *AI: The Tumultuous Search for Artificial Intelligence*, "In a generation, the problem of artificial intelligence creation will be solved at a significant level."Among the most influential visionaries was Alan Turing (1912–1954). In 1936, Turing demonstrated the theoretical possibility of a universal computing machine, now known as the Turing machine. His insight showed that any problem that can be formulated algorithmically could, in principle, be solved by a machine. This concept suggested that human cognitive processes, if broken down into finite, well-defined steps, could also be executed computationally.Decades later, the first practical digital computers were built, providing the "physical vehicle" for artificial intelligence. Turing's electromechanical machine, a precursor to modern computers, played a crucial role in breaking German military codes during World War II at Bletchley Park. His team deciphered the Enigma code—considered unbreakable by the Germans—and developed Colossus, one of the first

programmable computers, significantly shortening the war by an estimated two years. Turing's pioneering work laid the foundations for the computational intelligence we continue to develop and refine today.

History of Artificial Intelligence

To understand the history of artificial intelligence (AI), we must look back centuries. In Ancient Greece, myths already reflected humanity's fascination with creating artificial beings. Daedalus, for instance, was said to have crafted mechanical figures in mythology. Across other ancient civilizations—such as China and Egypt—engineers built simple automatons, showing that the idea of artificial humans has long captured the human imagination.Modern artificial intelligence began to emerge through attempts by philosophers and mathematicians to define human thought as a symbolic system. In 1834, Charles Babbage designed the Analytical Engine, a mechanical device that laid the foundations for modern computing. Although Babbage realized that his machine could not fully replicate human intelligence, his work became a stepping stone toward the development of intelligent systems.By the mid-20th century, interest in AI gained momentum. In 1950, Claude Shannon proposed that computers could play chess, while Alan Turing introduced the famous **Turing Test** to evaluate whether machines could exhibit human-like intelligence. The official birth of AI as a scientific discipline came in 1956 at the **Dartmouth Conference**, where John McCarthy, Marvin Minsky, and other pioneers formally established the field. McCarthy also developed **LISP** in 1957, a powerful programming language specifically designed for AI research.

Milestones in AI Development

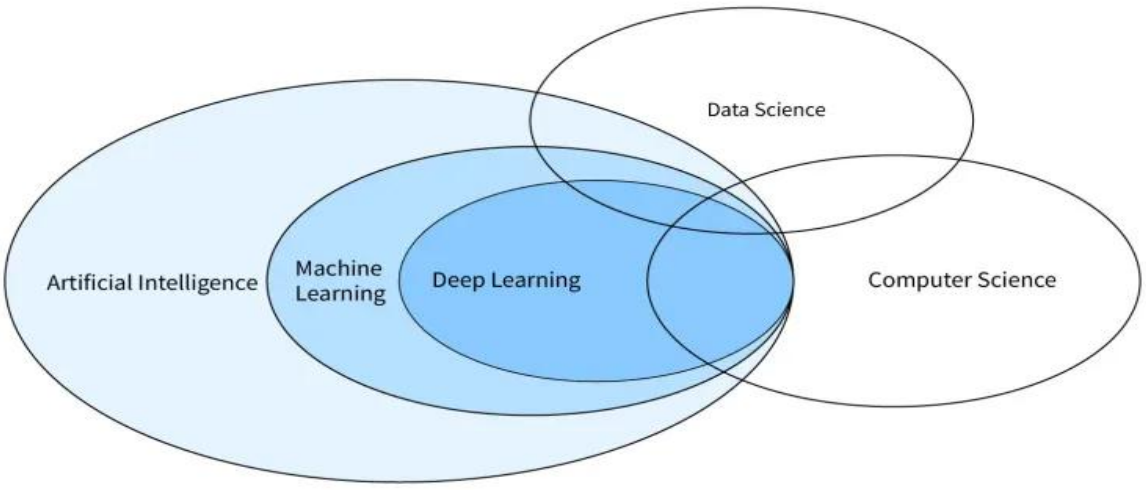
- **1950 – Turing Test:** Alan Turing introduced a test to evaluate machine intelligence by checking if a machine's responses could be indistinguishable from those of a human.
- **1956 – Dartmouth Conference:** The term *Artificial Intelligence* was formally introduced, marking the beginning of AI as a recognized field of research.
- **1957 – LISP Programming Language:** John McCarthy created LISP, a functional programming language that became one of the most influential tools for AI research.
- **1965–1970 – The First AI Winter:** Overly ambitious expectations led to disappointment, as progress in creating intelligent machines proved slower than anticipated. This period became known as a "dark age" for AI.

- **1970–1975 – Revival:** AI regained momentum, particularly in applications such as disease diagnosis, laying the foundations of today's expert systems.
- **1975–1980 – Interdisciplinary Growth:** Researchers began incorporating insights from psychology, cognitive science, and other fields to strengthen AI models.
- **1980s – Practical Applications:** AI systems started to be applied in real-world projects, including industrial automation, medical decision support, and knowledge-based systems.

Since then, AI has continued evolving rapidly. Moving beyond theoretical exploration, it has become a practical tool to solve real-life problems across industries—from healthcare and finance to robotics and natural language processing.

With the development of more efficient software and affordable tools, the use of artificial intelligence has expanded into a much wider range of applications. The concept of *machine intelligence*, driven by algorithms and large-scale data, shows that nearly all technological devices—from the earliest computers to today's smartphones—have been built to emulate human abilities.

Although AI evolved slowly in its early years, steady advancements have led to remarkable breakthroughs. Modern progress has culminated in the creation of highly capable systems, including intelligent robots and adaptive applications, which demonstrate how far AI has advanced from its early beginnings to its present role in everyday life.



Definition of Artificial Intelligence

According to John McCarthy, widely regarded as the *father of Artificial Intelligence*, AI is defined as *"the science and engineering of making intelligent machines, especially intelligent computer programs."* Artificial intelligence represents the broad discipline of creating machines capable of performing tasks that typically require human intelligence.

Early AI research was divided into two major approaches: Symbolic AI and Cybernetic AI. Symbolic AI, focused on logic-based systems, often failed to deliver expected results because robots and programs could not provide accurate or adaptive responses in real-world contexts. On the other hand, Cybernetic AI, rooted in artificial neural networks, also struggled to achieve its promises at the time due to limited computational power and insufficient training data.

These shortcomings led to the emergence of specialized AI, where systems were designed to serve a single, well-defined purpose rather than attempting to replicate the entirety of human intelligence. Despite early failures, continuous research, rational problem-solving, and technological improvements gradually pushed AI towards commercialization. Over time, AI grew into a billion-dollar industry, proving its capacity for real-world success.

Recent developments have also highlighted the central importance of language in AI. Since human thought is closely tied to language, advancements in natural language processing (NLP) have enabled AI to better simulate human communication. As a result, modern AI can now power applications ranging from conversational agents to robots capable of speech and interaction.

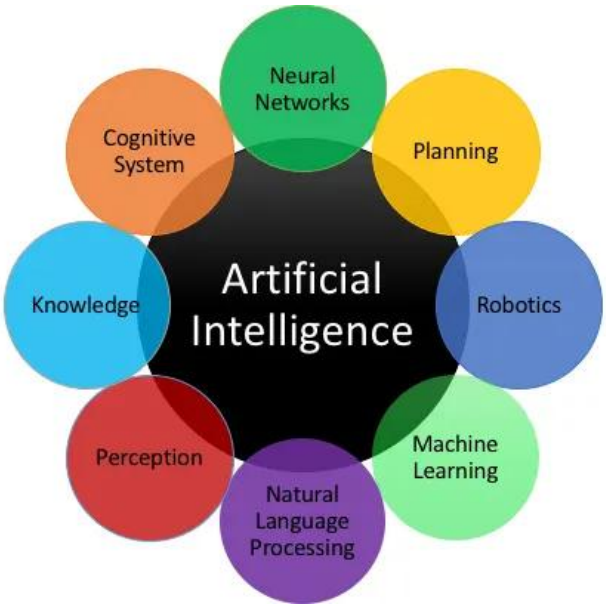
Goals of Artificial Intelligence

The overall research objective of AI is to create technologies that allow machines to function intelligently. This broad aim is broken down into several specific goals:

1. **To Create Expert Systems** – Systems that demonstrate intelligent behavior, explain their reasoning, and provide advice to users.
2. **To Implement Human Intelligence in Machines** – Designing systems that can understand, reason, learn, and behave in ways similar to humans.
3. **Learning and Adaptation** – Enabling systems to improve performance through experience and adapt to new information.
4. **Problem Solving and Planning** – Developing the ability to analyze situations, plan actions, and achieve goals effectively.

5. **Natural Language Processing (NLP)** – Building systems that can understand, interpret, and generate human language.

As Erik Sandwell emphasizes, *the effectiveness of AI lies not only in learning, but in planning and applying knowledge in ways that are relevant to real-world situations.*



Core Areas of Artificial Intelligence Research

Artificial Intelligence is a multidisciplinary field that brings together reasoning, knowledge representation, planning, learning, and the pursuit of general intelligence. The following are the major research areas:

1. Reasoning and Problem Solving

Early AI research focused on algorithms that could mimic human logical reasoning, such as step-by-step problem solving and puzzle-solving. By the late 1980s and 1990s, researchers extended these methods to handle uncertainty and incomplete information, borrowing concepts from probability theory and economics. However, many complex problems face the challenge of combinatorial explosion, where the required computational resources grow exponentially. Therefore, developing more efficient reasoning and problem-solving algorithms remains a central priority.

2. Knowledge Representation

For machines to act intelligently, they must be able to represent and use knowledge. This includes knowledge about objects, properties, categories, relations, situations, events, states, time, and cause-and-effect relationships. One key concept is the ontology, which defines the set of entities and their interrelations within a system.

Upper ontologies attempt to capture the most general forms of knowledge, serving as a foundation for more specialized systems.

3. Planning

An intelligent system must be capable of goal setting and decision-making. This requires the ability to predict future states of the world, evaluate the outcomes of different actions, and choose the course of action that maximizes utility. In classical planning, an agent assumes it is the only actor influencing the world, ensuring predictable outcomes. In real-world environments, however, multiple agents interact simultaneously, requiring systems that can reason under uncertainty, adapt to dynamic changes, and refine predictions.

4. Learning

Machine Learning is a cornerstone of AI, enabling systems to improve performance through experience.

- **Unsupervised learning** identifies hidden patterns and structures within data without labeled outputs.
- **Supervised learning** uses labeled examples for tasks such as classification (assigning categories) and regression (predicting numerical outcomes). These approaches allow AI systems to generalize from past data and apply knowledge to unseen situations.

5. Towards General Intelligence

While current AI systems are mostly narrow AI—designed for specialized tasks—many researchers are working toward Artificial General Intelligence (AGI). AGI would integrate reasoning, planning, knowledge, and learning into a single system, capable of performing across domains and potentially surpassing human abilities. Some researchers even explore the possibility of incorporating anthropomorphic features such as self-awareness or artificial consciousness, although this remains largely theoretical.

Applications of Artificial Intelligence (AI)

Artificial Intelligence has made significant contributions across a wide range of fields, transforming both industry and everyday life. Some of the most prominent applications include:

1. Gaming

AI plays a vital role in modern strategic and simulation games such as chess, poker, and tic-tac-toe. Intelligent algorithms enable machines to evaluate a vast number of

possible moves, assess outcomes, and make decisions based on heuristic knowledge. This makes AI-driven opponents capable of challenging and even surpassing human players.

2. Natural Language Processing (NLP)

With NLP, machines can understand, interpret, and respond to human language. This allows natural communication with computers, enabling applications such as chatbots, virtual assistants, translation systems, and sentiment analysis.

3. Expert Systems

Expert systems combine specialized knowledge bases with reasoning mechanisms to provide advice and decision support. These are widely used in medical diagnosis, engineering troubleshooting, and financial forecasting, offering explanations and recommendations to human users.

4. Vision Systems

AI-based vision systems enable computers to interpret visual input such as images and videos. Applications include:

- **Surveillance systems** for recognizing individuals.
- **Medical imaging systems** for assisting doctors in diagnosis.
- **Aerial reconnaissance**, such as spy drones creating spatial maps from photographs.

5. Speech Recognition

Intelligent speech recognition systems can comprehend spoken language and extract meaning even under challenging conditions such as:

- Accents and dialects
- Informal or slang words
- Background noise
- Variations in human speech due to illness (e.g., cold)

These systems power applications such as voice assistants (e.g., Siri, Alexa) and hands-free controls.

6. Handwriting Recognition

AI systems can read and interpret text written either on paper (using scanners) or on screens (via stylus input). The recognized characters are then converted into editable

and searchable text, used in document digitization, banking (cheque processing), and note-taking applications.

7. Intelligent Robots

Robotics is one of the most visible applications of AI. Intelligent robots are capable of:

- Performing human-assigned tasks
- Detecting environmental factors such as light, temperature, sound, and movement through sensors
- Using processors and memory to learn from experience and adapt to new environments

Such robots are applied in manufacturing, medical surgery, military operations, and household assistance.

Machine Learning

Abstract

Machine Learning (ML) has always been an integral part of Artificial Intelligence (AI), and its methodology has evolved in parallel with the major challenges of the field. As the volume of knowledge and data in modern AI systems increases, researchers have turned to ML to overcome the knowledge acquisition bottleneck. ML techniques are now widely used across disciplines, particularly in bioinformatics, where the high costs and complexities of biological analysis have encouraged the development of sophisticated ML approaches.

This chapter first introduces the fundamental concepts of ML, including feature assessment, supervised vs. unsupervised learning, and classification methods. It then discusses the design of ML experiments and performance evaluation techniques. Finally, it explores common supervised learning algorithms, highlighting their importance in both academic research and real-world applications.

Introduction to Machine Learning

Since the Industrial Revolution, machines have evolved from performing purely manual labor to executing highly complex cognitive tasks. Today, machines are capable of judging song competitions, driving cars, and even outperforming professional chess players. These remarkable achievements, however, have sparked both optimism and fear.

On one hand, machine intelligence demonstrates unprecedented potential. On the other, it raises questions of job security and human survival. Concerns range from fears of the “singularity”—a future where machines surpass human intelligence—to the economic implications of widespread automation. According to a BBC analysis (“Will a Robot Take My Job?”), professions such as accountant, receptionist, taxi driver, and waiter face high risks of automation by 2035.

Yet, such predictions must be taken with caution. While AI is advancing rapidly, its adoption is uneven, slowed by technical challenges, ethical concerns, and workforce readiness. Machine learning, in particular, is not an out-of-the-box solution—it requires skilled professionals, statistical expertise, and iterative experimentation.

The Role of Experts and Statistics

At its core, ML is powered by statistical algorithms developed long before computers existed. Techniques such as linear regression, k-nearest neighbors (k-NN), and

random forests are built upon classical statistics and remain essential to the field today. These algorithms allow machines to learn patterns, adapt to new data, and make predictions with minimal human intervention.

The shortage of professionals with expertise in data science, ML engineering, and algorithm design remains one of the biggest barriers to AI progress. As Charles Green, Director of Thought Leadership at Belatrix Software, explains:

> "It's a huge challenge to find data scientists, people with machine learning experience, or people with the skills to analyze and use the data, as well as those who can create the algorithms required for machine learning."

Thus, while the potential of ML is immense, its future growth depends on both technological innovations and the availability of skilled human resources to guide its development responsibly.

What is Machine Learning?

The term Machine Learning (ML) was first popularized in 1959 by Arthur Samuel, a pioneer at IBM. In his seminal paper, *Some Studies in Machine Learning Using the Game of Checkers*, published in the IBM Journal of Research and Development, Samuel demonstrated that a computer could be programmed to improve its performance over time without explicit reprogramming. Specifically, he designed a checkers-playing program that could learn strategies beyond those initially coded by its creator, thus playing better than the programmer himself.

This work introduced the foundational idea that machines can learn from data and experience, rather than relying solely on hard-coded instructions. Samuel described ML as:

> "The field of study that gives computers the ability to learn without being explicitly programmed."

Modern Definition

Today, ML is broadly understood as a subset of Artificial Intelligence (AI) that focuses on the development of algorithms and models capable of:

- **Learning from data** (experience)
- **Identifying patterns**
- **Making predictions or decisions** with minimal human intervention

In other words, instead of writing step-by-step instructions for every task, developers provide machines with data and algorithms that allow them to automatically improve their performance.

Key Characteristics of Machine Learning

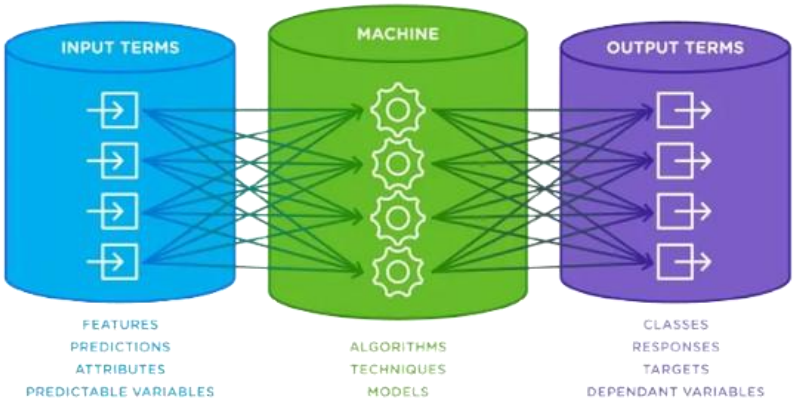
1. **Data-Driven** – ML systems require large volumes of structured or unstructured data to train models.
2. **Adaptive** – Models improve their accuracy and efficiency over time as they are exposed to new data.
3. **Predictive Power** – ML enables forecasting, classification, clustering, and anomaly detection.
4. **Automation** – Once trained, ML systems can make decisions with minimal human supervision.

Historical Perspective of Machine Learning

Arthur Samuel, an IBM researcher, introduced the concept of machine learning in the late 1950s. In his work, he defined it as a subfield of computer science that provides computers with the ability to learn without being explicitly programmed. Nearly six decades later, this definition still holds significant relevance and is widely cited in academic and industrial discussions.

A key feature embedded in Samuel's definition is the notion of self-learning. This refers to the use of statistical modeling and pattern recognition to improve a machine's performance based on data and empirical evidence, rather than through explicit step-by-step instructions.

However, Samuel's description does not suggest that machines make decisions completely independent of programming. On the contrary, machine learning is fundamentally dependent on computer programming, mathematical models, and algorithms. What differentiates it from traditional programming is that the instructions come in the form of data rather than hard-coded commands. In other words, machines learn *from data inputs* rather than direct manual coding for each specific task.



Training and Test Data

In machine learning, available data is typically divided into two parts: training data and test data.

- **Training Data:** This is the initial portion of the dataset used to develop the model. During training, the algorithm learns patterns, relationships, and rules from the data. For instance, in a spam email detection system, the training data may include labeled examples of spam and non-spam emails. If false positives occur—such as legitimate PayPal auto-responses being incorrectly flagged as spam—the model can be refined by introducing new rules or adjustments (e.g., emails from “payments@paypal.com” should be excluded from spam filtering).
- **Test Data:** Once the model has been trained and tuned, it is evaluated using the test data. This separate dataset assesses how well the model generalizes to unseen data. A well-performing model should demonstrate consistent accuracy on both the training and test datasets, minimizing the risks of overfitting (memorizing training examples) and underfitting (failing to capture underlying patterns).

The distinction between training and test data highlights one of the key differences between machine learning and traditional programming. In traditional programming, explicit rules are written by a human developer, whereas in machine learning, rules and decision boundaries emerge from the data itself.

Moreover, machine learning does not exist in isolation. It forms part of the broader landscape of data science, which sits within the parent discipline of computer science. While computer science encompasses all aspects of computational systems, data science specifically focuses on extracting insights and knowledge from data

through algorithms, statistical methods, and computational techniques. Within this framework:

- **Machine Learning** enables systems to automatically learn and improve from data.
- **Data Mining** emphasizes the discovery of hidden patterns and relationships in large datasets, often overlapping with machine learning.
- **Classical Statistics** provides the foundational mathematical principles on which many machine learning algorithms are built.

Thus, understanding the distinction and interrelation between these fields is crucial for grasping the scope of modern artificial intelligence and its practical applications.

Categories of Machine Learning

Machine learning encompasses hundreds of statistical and algorithmic approaches. Selecting the most appropriate algorithm—or a combination of algorithms—for a specific problem is one of the key challenges in the field. Before exploring individual algorithms, it is important to first understand the three overarching categories of machine learning:

1. **Supervised Learning**
2. **Unsupervised Learning**
3. **Reinforcement Learning**

1. Supervised Learning

Supervised learning is the most widely used category of machine learning. It is defined by the use of labeled datasets, where both the input features and the corresponding output (target values) are known.

- **How it works:**
    - Input data is represented as **X** (features).
    - The corresponding known outcome is represented as **y** (label).
    - The learning algorithm identifies patterns and relationships between X and y.
    - Once trained, the algorithm produces a model that can predict outcomes (y) for unseen inputs (X).
- **Example:**
  Suppose we want to predict the market price of a used car. A supervised

learning algorithm can be trained using past sales data, where features (X) include attributes such as car brand, year of manufacture, mileage, and condition. The corresponding label (y) is the actual selling price. By learning the relationship between car attributes and their prices, the model can then predict the price of a new car based on its characteristics.

- **Application Workflow:**
    - Train the algorithm on labeled data.
    - Generate a model that captures the relationship between inputs and outputs.
    - Test the model using unseen (test) data to validate accuracy.
    - Deploy the model in real-world settings for predictive tasks.
- **Common Algorithms in Supervised Learning:**
    - **Regression Analysis** (e.g., linear regression, logistic regression)
    - **Decision Trees**
    - **k-Nearest Neighbors (k-NN)**
    - **Support Vector Machines (SVMs)**
    - **Artificial Neural Networks (ANNs)**
- **Case Study Example:**
  In real estate, supervised learning can be applied to predict house prices. Here, the target variable **y** is the actual selling price, while features **X** include location, land size, number of rooms, and other property characteristics. The trained model can generalize these relationships to estimate the price of new, unseen properties.

2. Unsupervised Learning

Unsupervised learning is used when datasets lack predefined labels. Unlike supervised learning, where both input (X) and output (y) are known, in unsupervised learning the system must uncover hidden patterns and structures in the data without prior classification.

- **How it works:**
    - The algorithm analyzes the dataset and detects relationships between data points.
    - It groups or organizes the data into meaningful clusters or representations.

    - Instead of predicting specific outcomes, the focus is on pattern discovery.
- **Example:**
  Consider grouping businesses based on purchasing behavior. If customer data is analyzed without labels, unsupervised algorithms may naturally split them into **two clusters**:
    - Small and Medium-sized Enterprises (SMEs) with simple purchasing habits (e.g., cloud hosting, CDN).
    - Large enterprises with complex requirements (e.g., advanced infrastructure, distributed systems).
- **Advantages:**
    - Identifies previously unknown structures in data.
    - Serves as a foundation for further supervised analysis after clusters are defined.
- **Applications:**
    - **Clustering:** Discovering distinct customer groups (e.g., using k-means clustering).
    - **Anomaly detection:** Widely applied in fraud detection, where malicious behavior often has no predefined label.
    - **Real-world Example:** DataVisor, a fraud detection company, built its entire business model on unsupervised learning to identify unknown attack patterns.

3. Reinforcement Learning

Reinforcement learning (RL) represents the most advanced category of machine learning. Unlike supervised and unsupervised learning, which operate on static datasets, reinforcement learning focuses on continuous improvement through trial and error. The algorithm learns dynamically by receiving feedback (rewards or penalties) for its actions in a given environment.

- **How it works:**
    - The environment is defined as a set of states (S).
    - The algorithm selects from possible actions (A) at each state.
    - Each action leads to a reward or penalty, updating the algorithm's knowledge.

    - The system seeks to maximize the cumulative reward over time.
- **Analogy:**
  Reinforcement learning can be compared to playing a video game. A player initially tries random moves, learns which strategies avoid failure and yield rewards, and gradually improves performance. Similarly, RL agents adapt through repeated interaction with their environment.
- **Example – Q-Learning:**
  Q-learning is a common RL algorithm.
    - **States (S):** Possible conditions (e.g., in Pac-Man, walls, ghosts, or power pills).
    - **Actions (A):** Movements or strategies (e.g., up, down, left, right).
    - **Q-value (Q):** The score assigned to a state-action pair. Initially set to zero, Q-values increase with positive rewards (e.g., avoiding defeat, eating a power pill) and decrease with penalties (e.g., getting caught by a ghost).
      Over time, the algorithm learns the optimal policy—the best action for each state.
- **Applications:**
    - **Autonomous vehicles**: Rewarded for safe navigation and penalized for crashes.
    - **Game AI:** Mastering complex games like chess or Go.
    - **Robotics:** Teaching robots to adapt to dynamic environments.
- **Key Feature:**
  Unlike supervised or unsupervised learning, reinforcement learning does not stop after training. It is continuous and adaptive, allowing systems to evolve as environments change.

The Machine Learning Toolbox

Just as a web developer relies on specific tools such as programming languages, editors, and hosting services, a machine learning practitioner also works with a set of essential components that can be thought of as compartments within a toolbox.

Compartment 1: Data

Data is the foundation of machine learning. It provides the input variables necessary for forming predictions.

Types of Data

- **Structured data**: Organized into rows and columns, typically stored in tabular form with schema and labels. This is the most suitable starting point for beginners. Examples include spreadsheets and SQL tables.
- **Unstructured data**: Includes text, images, audio, and video. This type of data is more complex and is often used in advanced machine learning tasks.

Organization of Data

- **Row (case, observation, or value)**: Represents one instance in the dataset.
- **Column (feature, attribute, or variable)**: Represents a measurable property of the data.
- **Vector**: A column containing values for a specific feature.
- **Matrix**: A collection of vectors, representing the entire dataset.

In supervised learning, the dependent variable (often denoted as **y**) is usually stored in the final column, while independent variables (features, denoted as **X**) are stored in the preceding columns.

Data Visualization

Visualization is a key element in understanding and preparing data. Scatterplots are commonly used to explore relationships and patterns within a dataset.

- **2D scatterplot**: Uses an x-axis for independent variables (X) and a y-axis for the dependent variable (y), with each data point representing an observation.
- **3D and 4D scatterplots**: Extend this approach by including additional dimensions to capture more complex relationships.

Through scatterplots, practitioners can identify clusters, trends, and outliers, which can guide further data preprocessing and analysis.

Compartment 2: Infrastructure

The second compartment of the machine learning toolbox consists of the **infrastructure**—the platforms, environments, and tools required to process data and build models.

Programming Environments and Languages

Beginners typically start with **Jupyter Notebook**, a web-based interactive environment for writing and running code. The most widely used programming language in machine learning is Python, valued for its simplicity, versatility, and

compatibility with a wide range of libraries. Python can also support related tasks such as data collection (e.g., web scraping) and large-scale data processing (via frameworks like Hadoop and Spark).

Other programming languages used in machine learning include:

- **C and C++**: Provide high performance and are particularly well-suited for GPU-based computation.
- **R**: Optimized for statistics and matrix operations, widely used in data analytics and research.
- **MATLAB and Octave**: Strong in numerical computation, particularly in academic and engineering contexts. MATLAB is commercial, while Octave offers a free open-source alternative.

Machine Learning Libraries

Python users typically rely on three foundational libraries:

1. **NumPy** – Provides efficient operations for handling large datasets, vectors, and matrices.
2. **Pandas** – Offers data manipulation tools similar to spreadsheets, with functions for cleaning, editing, and importing/exporting data (especially from CSV files).
3. **Scikit-learn** – Includes implementations of common machine learning algorithms, such as linear regression, decision trees, and support vector machines.

Visualization Tools

Data visualization is supported by libraries such as Matplotlib and Seaborn, which enable the creation of plots, graphs, and statistical charts. More advanced tools like **Tableau** provide interactive dashboards and visual analytics.

Summary of Infrastructure Workflow

- Load and manage datasets using NumPy.
- Clean, manipulate, and extract data with Pandas.
- Apply machine learning algorithms using Scikit-learn.
- Visualize data and results with Seaborn, Matplotlib, or Tableau.

This infrastructure provides the foundation upon which machine learning models can be developed and tested.

Compartment 3: Algorithms

Once the environment is set up, the next compartment of the toolbox contains algorithms, the core mechanisms that enable learning from data.

Accessing Datasets

Datasets are often provided in CSV (Comma-Separated Values) format and can be easily downloaded from online platforms such as Kaggle, which hosts thousands of free datasets across diverse domains. CSV files can be opened in spreadsheet programs like Excel or imported directly into programming environments for analysis.

Beginner-Friendly Algorithms

For beginners, the following algorithms are commonly used:

- **Supervised Learning**:
    - *Linear Regression* – Models relationships between input variables and a continuous output.
    - *Logistic Regression* – Used for binary classification tasks.
    - *Decision Trees* – Intuitive, tree-structured models that split data into decision paths.
    - *K-Nearest Neighbors (k-NN)* – A simple classification algorithm based on distance between data points.
- **Unsupervised Learning**:
    - *K-Means Clustering* – Groups data into clusters based on similarity.
    - *Dimensionality Reduction* (e.g., Principal Component Analysis) – Reduces the number of variables while preserving patterns in the data.

Progression in Learning

These algorithms form the starting point for understanding machine learning. They help beginners grasp key concepts such as model training, evaluation, and prediction. As skills advance, learners can progress to more sophisticated models such as ensemble methods, support vector machines, and neural networks.

Data Scrubbing

Much like fruit often requires peeling or trimming before being eaten, datasets nearly always require upfront cleaning and transformation before they can be analyzed effectively. In machine learning and data science, this process is called **data**

scrubbing. Scrubbing is the technical procedure of refining a dataset to make it more usable and reliable for model development. It often involves tasks such as:

- Handling missing or incomplete data
- Removing duplicates or irrelevant entries
- Correcting inconsistent or incorrectly formatted values
- Converting categorical/text data into numerical values
- Redesigning or restructuring features for better representation

For most practitioners, data scrubbing is one of the most time-consuming and effort-intensive stages of a project. However, the quality of this step directly impacts the performance of the machine learning model.

Feature Selection

To generate meaningful results, it is critical to identify the **variables (features)** most relevant to the hypothesis or prediction goal. This process, known as **feature selection**, ensures that only useful data contributes to the model while irrelevant or redundant features are removed.

For instance, when analyzing a dataset of endangered languages, the variable *"Name in Spanish"* is unlikely to provide any meaningful insight into why a language becomes endangered. Retaining such irrelevant features risks overcomplicating the dataset and reducing model accuracy. Similarly, when both *"Countries"* and *"Country Code"* appear as columns, one of them can be safely removed since they provide duplicate information.

Another useful approach is feature reduction through merging, where multiple related features are grouped into broader categories. For example, in an e-commerce dataset containing products like protein bars, yoga pants, and fitness trackers, the eight product-specific features could be condensed into three meaningful categories: *Health Food*, *Apparel*, and *Digital*.

This transformation not only simplifies analysis but also reduces computational cost, leading to faster training and cleaner insights. However, it does come with trade-offs. By collapsing features into categories, some fine-grained relationships between individual items are lost. For example, product recommendations may now be based on broader subtypes rather than precise product-to-product associations.

Handling Missing Data

Missing data poses a significant challenge in machine learning, as incomplete datasets can distort analysis and weaken prediction accuracy. A useful analogy is a

jigsaw puzzle with missing pieces—while you may still form a partial picture, the overall outcome is incomplete and less reliable.

Several strategies exist to mitigate the impact of missing values:

1. **Mode Imputation**
    - Replace missing values with the most frequently occurring value (the mode).
    - Best suited for categorical or binary variables.
2. **Median Imputation**
    - Replace missing values with the median (the middle value of the dataset).
    - Works best with integer and continuous variables, as it reduces the effect of outliers compared to the mean.
3. **Row Removal**
    - Remove rows containing missing values.
    - While simple, this approach reduces the dataset size, which may limit the comprehensiveness of the analysis.

Selecting the right method depends on the type of data and the extent of missingness. Whenever possible, imputation is preferred over row removal to preserve valuable information.

Setting Up Your Data

After cleaning and imputing missing values, the next critical step is to split the dataset into training and testing subsets. Testing a model on the same data it was trained on leads to misleadingly high accuracy and poor real-world performance.

- A typical split ratio is either 70/30 or 80/20, where:
    - Training set: 70–80% of the rows, used to teach the model.
    - Test set: 20–30% of the rows, used to evaluate performance.
- **Important Considerations**:
    - **Split by rows, not columns**: Columns represent features; splitting them would break the relationship between input and output variables.
    - **Randomize before splitting**: Datasets are often collected sequentially (e.g., by time). Without randomization, training and test sets may not be

representative, introducing bias. Libraries like **Scikit-learn** provide built-in functions to shuffle and split data efficiently.

Once the dataset is split:

1. The training data is fed into the model along with the expected output (target variable `y`). The algorithm learns the patterns and relationships between the features (`X`) and the target.
2. The test data is then used to evaluate the model's predictive accuracy. A common performance metric is Mean Absolute Error (MAE), which calculates the average difference between predicted and actual values.

- In Scikit-learn, MAE can be calculated by:
    - Using the model's `.predict()` function on the test set.
    - Comparing predictions against the true `y` values.

If the model shows low error across both training and test datasets, it indicates that the algorithm has successfully generalized patterns rather than memorizing data. However, if performance is poor, possible remedies include:

- Ensuring proper randomization of training and test sets.
- Tuning or adjusting hyperparameters (the algorithm's internal settings that control learning speed and pattern detection).

Once the model achieves acceptable accuracy on the test set, it can be confidently applied in real-world scenarios.

Cross Validation

- Splitting into only training and test sets may not always give reliable performance estimates, especially with small datasets.
- Cross Validation solves this by testing models on multiple combinations of data.
    - **Exhaustive Cross Validation**: Tests *all possible* training–testing splits (computationally expensive).
    - **Non-exhaustive (k-Fold Cross Validation)**:
        - Data is divided into **k equal buckets**.
        - In each round, one bucket is used for testing, and the other **(k–1)** are used for training.
        - Repeats for all k buckets, giving a more reliable performance estimate.

Machine Learning Algorithms – Supervised Learning

- **Definition**: Learns a mapping (function) between input data **X** and output labels **Y**.
- **Training data** = labeled examples (X, Y).
- **Supervision** = labels provided by a human or machine supervisor.
    - Human-labeled data is more accurate but costlier.
    - Machine labeling is cheaper but prone to higher error rates.
- **Applications**: sentiment analysis, image recognition, speech detection, tumor detection.
    - Some tasks (e.g., tumor detection) require domain experts for labeling.

Types of Supervised Learning Algorithms

1. **Regression** – Predicts continuous values (e.g., house prices, temperature).
2. **Classification** – Predicts categories (e.g., spam vs. not spam, disease vs. no disease).

Unsupervised Learning

In unsupervised learning, no labels or supervisors are available in the training data. The goal is to uncover hidden structures or patterns in the dataset. This lack of labels may arise due to financial constraints in manual labeling or the inherent complexity of the data itself. With the rapid growth of Big Data, characterized by its volume, velocity, and variety, extracting meaningful insights without supervision has become an essential challenge for machine learning practitioners. Clustering, dimensionality reduction, and anomaly detection are common approaches in unsupervised learning.

Semi-Supervised Learning

Semi-supervised learning combines both labeled and unlabeled data to build predictive models. Typically, labeled data are scarce and expensive to obtain, while unlabeled data are abundant. Semi-supervised methods leverage this imbalance by improving model performance beyond what could be achieved using labeled data alone.
This process can be compared to how a child learns:

1. **Unlabeled data** comes from the environment (raw sensory input).
2. **Labeled data** comes from supervisors (e.g., parents teaching the names of objects).

By combining these two sources of information, semi-supervised learning provides a practical and efficient way to enhance classification accuracy.

Reinforcement Learning

Reinforcement learning (RL) focuses on training intelligent agents to make sequential decisions by interacting with an environment. The agent observes the state of the environment, performs an action, and receives a reward or penalty in return. Over time, the agent refines its policy—a mapping from states to actions—to maximize long-term cumulative rewards.
The RL cycle typically involves the following steps:

1. The agent observes the input state.
2. A decision-making function selects an action.
3. The environment provides a reward or feedback.
4. The agent updates its state–action policy based on this feedback.

Reinforcement learning underpins many modern applications such as robotics, autonomous driving, and game-playing AI (e.g., AlphaGo).

Linear Regression

Linear regression is one of the most fundamental and widely used supervised learning algorithms for predictive analysis. It models the relationship between a dependent variable (Y) and one or more independent variables (X) by fitting a straight line. The general form is:

$Y = a_0 + a_1X + \varepsilon$

Where:

- Y: Dependent (target) variable
- X: Independent (predictor) variable
- a0: Intercept (constant term)
- a1: Coefficient (weight assigned to X)
- ε: Random error

Linear regression assumes a linear relationship between predictors and the target, making it simple yet powerful for forecasting and trend analysis.

Logistic Regression

Logistic regression is a supervised classification algorithm used to predict the probability of a binary outcome. Unlike linear regression, the target variable here is

categorical, with only two possible classes (e.g., Yes/No, 0/1, True/False). Instead of outputting direct class labels, logistic regression estimates probabilities that lie between 0 and 1, modeled using the sigmoid function:

$$P(Y = 1 \vee X) = \frac{1}{1 + e^{-(a_0 + a_1 X)}}$$

This makes logistic regression particularly useful for problems such as spam detection, medical diagnosis, and credit risk assessment.

Logistic Sigmoid Function (Sigmoid Function)

- The sigmoid function is a mathematical function used to map predicted values to probabilities.
- It maps any real number into a value between 0 and 1.
- The output of logistic regression must always lie between 0 and 1, forming an S-shaped curve, also called the Sigmoid (Logistic) function.
- In logistic regression, a threshold value is used to classify outcomes:
    - If probability > threshold → class 1
    - If probability < threshold → class 0

Equation of Logistic Regression:

$$P(Y = 1 \vee X) = \frac{1}{1 + e^{-(a_0 + a_1 X)}}$$

Types of Logistic Regression

1. **Binomial Logistic Regression**
    - Only two possible outcomes for the dependent variable.
    - Examples: {0, 1}, {Pass, Fail}, {Yes, No}.
2. **Multinomial Logistic Regression**
    - Three or more unordered categories of the dependent variable.
    - Example: {Cat, Dog, Sheep}.
3. **Ordinal Logistic Regression**
    - Three or more ordered categories of the dependent variable.
    - Example: {Low, Medium, High}.

Decision Tree

- A Decision Tree is a Supervised Learning method used for both Classification and Regression problems.
- Mostly applied for Classification tasks.
- It is a tree-structured classifier:
    - **Internal nodes** → represent dataset features.
    - **Branches** → represent decision rules.
    - **Leaf nodes** → represent outcomes.
- **Types of nodes:**
    - **Decision Node** → used to make a decision; has multiple branches.
    - **Leaf Node** → represents an output; has no further branches.
- **How it works:**
    - Tests are performed on dataset features.
    - At each step, the dataset is split into subsets based on conditions.
    - The process continues recursively, forming a **tree-like structure**.
- **Algorithm used:**
    - **CART** → Classification and Regression Tree algorithm.
- **Concept:**
    - A decision tree repeatedly asks a question.
    - Based on the answer (Yes/No), the dataset is split further until reaching a leaf (final decision).

Support Vector Machine (SVM)

- Support Vector Machine (SVM) is a popular Supervised Learning algorithm used for both Classification and Regression problems.
- Primarily applied in Classification tasks.
- The main goal of SVM is to find the best decision boundary (hyperplane) that separates data points of different classes in an n-dimensional space.
- Once the hyperplane is determined, new data points can be classified accordingly.

- Support Vectors: The extreme data points that lie closest to the decision boundary, which play a crucial role in defining the hyperplane.

Types of SVM

1. **Linear SVM**
    - Used when the dataset is linearly separable.
    - A single straight line (or hyperplane in higher dimensions) can separate the classes.
2. **Non-linear SVM**
    - Used when the dataset is not linearly separable.
    - Kernel functions (such as Polynomial, RBF) are applied to map data into higher dimensions, making it possible to separate with a hyperplane.

Naïve Bayes

- Naïve Bayes is a Supervised Learning algorithm based on Bayes' Theorem.
- Widely used for classification tasks, especially in text classification with high-dimensional datasets.
- It is a probabilistic classifier, meaning it predicts outcomes based on probability values.
- Known for being fast, simple, and effective for tasks such as:
    - Spam filtering
    - Sentiment analysis
    - Document classification

Why "Naïve Bayes"?

- Naïve: It assumes that all features are independent of each other when predicting the outcome.
    - Example: If a fruit is red, spherical, and sweet, Naïve Bayes treats these features as independent when predicting "Apple."
- **Bayes**: Because it is derived from Bayes' Theorem.

Bayes' Theorem Formula

$$P(A \vee B) = \frac{P(B \vee A) * P(A)}{P(B)}$$

Where:

- P(A|B) → **Posterior Probability**: Probability of hypothesis A given observed event B.
- P(B|A) → **Likelihood**: Probability of evidence B given hypothesis A is true.
- P(A) → **Prior Probability**: Probability of hypothesis before seeing evidence.
- P(B) → **Marginal Probability**: Probability of the evidence.

K-Nearest Neighbour (KNN)

- The **K-Nearest Neighbour (KNN)** algorithm is one of the simplest Machine Learning algorithms based on the Supervised Learning technique.
- KNN assumes similarity between a new case/data point and existing cases, and places the new case into the category that is most similar to the existing ones.
- The algorithm stores all available data and classifies a new point based on similarity measures (e.g., distance metrics). Thus, when new data arrives, it is classified into a suitable category using this similarity.
- KNN can be used for both classification and regression, though it is more commonly used in classification tasks.
- It is a non-parametric algorithm, meaning it does not assume any underlying data distribution.
- KNN is often referred to as a lazy learner because it does not learn from the training set immediately. Instead, it stores the dataset, and computation happens at the time of classification.
- During prediction, KNN calculates the similarity (often using Euclidean distance) between the test point and training data, then assigns the label based on the majority class of its nearest neighbors.
- K-Nearest Neighbour (KNN) is one of the simplest Supervised Learning algorithms.
- It works on the principle of similarity:
    - A new data point is classified into the class most similar to existing data points.
- **Process:**
    - KNN stores the entire dataset.

    - When new data arrives, it calculates the distance to all stored points and assigns the label based on the majority of the k nearest neighbours.
- **Key Features:**
    - Can be used for Regression and Classification (commonly Classification).
    - Non-parametric: makes no assumptions about the underlying data distribution.
    - Known as a lazy learner: does not train in advance; instead, computation happens at classification time.

KNN Mathematical Formula

The **distance** between two points x=(x1,x2,…,xn) and y=(y1,y2,…,yn) in *n*-dimensional space (Euclidean distance) is:

$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

K-Means Algorithm

- K-Means is an unsupervised clustering algorithm.
- It groups data into k clusters based on similarity.
- Each cluster is represented by its centroid (mean of points in that cluster).
- Algorithm steps:
    1. Select k random centroids.
    2. Assign each data point to the nearest centroid.
    3. Update centroids as the mean of assigned points.
    4. Repeat until convergence.

K-Means Objective Function

The algorithm minimizes the within-cluster sum of squared errors (SSE):

$$J = \sum_{j=1}^{k}\sum_{i=1}^{n}\left|x_i^{(j)} - mu_j\right|^2$$

Where:

- xi(j) = data points in cluster j
- μj = centroid of cluster j

Random Forest

- Random Forest is a popular ensemble learning algorithm.
- Works for both Classification and Regression.
- Based on the idea of combining multiple Decision Trees to improve performance.
- **Process:**
    - Multiple decision trees are built on random subsets of the dataset (bagging).
    - Predictions from all trees are combined (majority voting for classification, averaging for regression).
- **Advantages:**
    - Reduces overfitting compared to a single decision tree.
    - Works well with large datasets and high-dimensional features.
    - More trees = higher accuracy.

Random Forest Prediction

For classification:

$\hat{y} = mode\big(h_1(x), h_2(x), \ldots, h_m(x)\big)$

For regression:

$$\hat{y} = \frac{1}{m}\sum_{j=1}^{m} h_j\,(x)$$

Where:

- hj(x) = prediction of the jth tree
- m = total number of trees
- $\hat{y}$ = final output

Machine Learning Algorithms

1. Linear Regression Algorithm

Steps to implement a Linear Regression model:

1. Initialize the model parameters (a0,a1, etc.).
2. Predict the dependent variable (y) given the independent variable (x).
3. Calculate the prediction error for all data points.
4. Compute the partial derivatives of the cost function with respect to parameters a0 and a1.
5. Update parameters iteratively and minimize the overall cost.

2. Logistic Regression Algorithm

Steps to implement Logistic Regression:

1. Perform data preprocessing.
2. Fit the logistic regression model to the training set.
3. Predict the test results using the trained model.
4. Evaluate accuracy using a **confusion matrix**.
5. Visualize the classification boundaries and results.

3. Decision Tree Algorithm

Steps to construct a Decision Tree:

1. Begin with the root node containing the complete dataset.
2. Select the best attribute using an Attribute Selection Measure (ASM).
3. Split the dataset into subsets based on the values of the selected attribute.
4. Create a decision node for the chosen attribute.
5. Recursively build subtrees for each subset until further splitting is not possible.
6. Mark the terminal nodes as leaf nodes (final classification/decision).

4. Naïve Bayes Algorithm

Steps to implement Naïve Bayes:

1. Perform data preprocessing.
2. Fit the Naïve Bayes classifier to the training set.
3. Predict test results using the model.
4. Evaluate accuracy using a confusion matrix.
5. Visualize the classification results.

5. K-Nearest Neighbour (KNN) Algorithm

Steps to implement KNN:

1. Select the number of neighbors (K).
2. Calculate the Euclidean distance between the new data point and all training points.
3. Identify the K nearest neighbors.
4. Count the number of neighbors belonging to each class.
5. Assign the new data point to the class with the maximum votes.
6. The model is ready for predictions.

6. K-Means Clustering Algorithm

Steps to implement K-Means Clustering:

1. Select the number of clusters (K).
2. Randomly initialize K centroids.
3. Assign each data point to the nearest centroid, forming clusters.
4. Recalculate centroids as the mean of points within each cluster.
5. Reassign points to the nearest centroid and update clusters.
6. Repeat steps 4 and 5 until convergence (no reassignment occurs).
7. The clustering model is complete.

7. Random Forest Algorithm

Steps to implement Random Forest:

1. Select random subsets of data points from the training dataset.
2. Build decision trees for each subset.

3. Decide the number of trees (N) to construct.
4. Repeat the sampling and tree-building process until N trees are created.
5. For a new data point, collect predictions from all trees.
6. Assign the final output:
    - Majority voting for classification.
    - Averaging predictions for regression.

MUSIC GENRE CLASSIFICATION

INITIAL TRIALS:

We initially extracted data from the raw audio itself and trained a random forest classifier with it.

Dataset Handling

```
path = kagglehub.dataset_download("andradaolteanu/gtzan-dataset-music-genre-classification")
DATA_DIR = '/kaggle/input/gtzan-dataset-music-genre-classification/Data/genres_original'
```

- GTZAN dataset contains 10 genres, each with 100 WAV files, 30 seconds long.
- os.listdir iterates through genre folders, creating a dataframe of:
    - file_path: location of file
    - label: genre name

Feature Extraction with Librosa

The core part is extract_features(file_path). Each audio signal is converted into feature vectors.

(a) Load Audio

```
y, sr = librosa.load(file_path, duration=AUDIO_DURATION)
```

- y = audio time series = vector of amplitudes
- sr = sample rate (default = 22050 Hz)

(b) MFCC (Mel-Frequency Cepstral Coefficients)

$MFCC_i(t) = \sum_{k=1}^{K} log\big(E_k(t)\big) * cos\big(\Pi * i/K * (k - 1/2)\big)$

Where:

- E_k(t) = energy in the k-th Mel filter at frame t
- K = number of filters

We compute mean, standard deviation, and skewness of MFCCs across time.

c) Delta-MFCC (Δ-MFCC)

Captures temporal variation of MFCCs:

$\Delta MFCC_i(t) = \frac{\sum_{n=1}^{N} n*\big(MFCC_i(t+n) - MFCC_i(t-n)\big)}{2*\sum_{n=1}^{N} n^2}$

d) Chroma Features

Maps frequencies to 12 semitone classes (C, C#, D, … B):

$$Chroma_c(t) = \frac{\sum_{f \in F_c} \left| (X(f,t))^2 \right|}{\sum_f \left| (X(f,t))^2 \right|}$$

Where:

- X(f,t) = Short-Time Fourier Transform magnitude
- F_c = set of frequencies corresponding to pitch class c

e) Spectral Contrast

Measures difference between spectral peaks and valleys:

$$Contrast_j(t) = log\left(P_{max}, j\,(t)/P_{min}\,, j(t)\right)$$

f) Zero Crossing Rate (ZCR)

The rate at which the signal crosses zero amplitude:

$$ZCR = \frac{1}{N-1} * \sum_{n=1}^{N-1} cond(y[n] * y[n-1] < 0,1,0)$$

g) Tempo

Tempo in beats per minute (BPM) is estimated by maximizing the autocorrelation of the onset strength function:

$$R(\tau) = \sum_n^{N-1} o[n] * o[n+\tau]$$

$$\tau^{\square} = argmax_{\tau > \tau_{min}}^{\tau_{max}} \left(R(\tau)\right)$$

$$T = \frac{60.f_0}{\tau^{\square}}$$

where,

o[n] = Original waveform computed via Compute Short-Time Fourier Transform (STFT) of the signal, Measure spectral flux (change in energy) between consecutive frames, Normalize & smooth to get onset envelope

where f_0=sr/hop is the onset envelope sampling rate in frames/second.

Final Feature Vector

For each file, the feature vector includes:

- MFCCs: mean, std, skew = 120
- Δ-MFCCs: mean, std, skew = 120

- Chroma: mean, std = 24
- Spectral contrast: mean, std = 14
- Zero crossing rate: mean, std = 2
- Tempo = 1

Total features approximately 279 per audio file.

Feature Standardization

To normalize features:

$x' = \frac{x-\mu}{\sigma}$

μ = mean of feature

σ = standard deviation

Classification using Random Forest

A **Random Forest classifier** with 150 trees (n_estimators=150) and maximum depth 25 is trained on extracted features. Each decision tree splits data by minimizing **Gini impurity**:

$Gini(S) = 1 - \sum_{i=1}^{C} p_i^2$

where,

- p_i = probability of class i in dataset S
- C = number of classes

Final prediction is made by majority voting across all trees.

Evaluation Metrics

The model is evaluated using accuracy, precision, recall, and F1-score.

a) Accuracy:

$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

b) Precesion

$Precision = \frac{TP}{TP+FP}$

c) Recall

$$Recall = \frac{TP}{TP+FN}$$

d) F1_score

$$F1 = \frac{2*Precision*Recall}{Precision+Recall}$$

where,

TP = true positives

TN = true negatives

FP = false positives

FN = false negatives

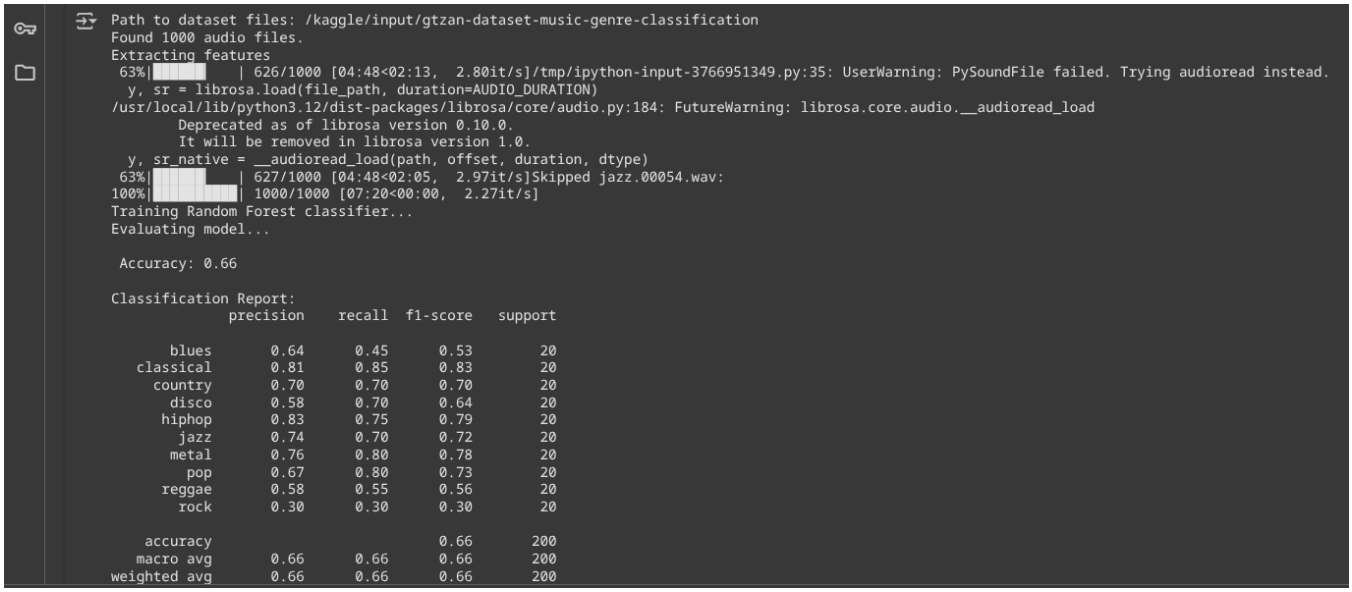
Results

- Extracted ~279 features per file
- Random Forest classifier trained on 80% of dataset
- Achieved accuracy = **66 %**
- Precision, Recall, and F1-scores were reported for each genre

CODE

```
import numpy as np
import pandas as pd
import librosa
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from tqdm import tqdm
import kagglehub
# Download latest version
path = kagglehub.dataset_download("andradaolteanu/gtzan-dataset-music-genre-classification")
print("Path to dataset files:", path)
DATA_DIR = '/kaggle/input/gtzan-dataset-music-genre-classification/Data/genres_original'
AUDIO_DURATION = 30
N_MFCC = 40
file_paths = []
labels = []
for genre in os.listdir(DATA_DIR):
genre_path = os.path.join(DATA_DIR, genre)
if not os.path.isdir(genre_path):
continue
for file in os.listdir(genre_path):
if file.endswith(".wav"):
file_paths.append(os.path.join(genre_path, file))
labels.append(genre)
df = pd.DataFrame({'file_path': file_paths, 'label': labels})
print(f"Found {len(df)} audio files.")
def extract_features(file_path):
try:
y, sr = librosa.load(file_path, duration=AUDIO_DURATION)
features = []
mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=N_MFCC)
delta_mfcc = librosa.feature.delta(mfcc)
for coeffs in [mfcc, delta_mfcc]:
mean = np.mean(coeffs, axis=1).flatten()
std = np.std(coeffs, axis=1).flatten()
skew = np.mean((coeffs - mean[:, np.newaxis])**3, axis=1).flatten()
features.extend(mean)
features.extend(std)
features.extend(skew)
chroma = librosa.feature.chroma_stft(y=y, sr=sr)
features.extend(np.mean(chroma, axis=1).flatten())
features.extend(np.std(chroma, axis=1).flatten())
contrast = librosa.feature.spectral_contrast(y=y, sr=sr)
features.extend(np.mean(contrast, axis=1).flatten())
features.extend(np.std(contrast, axis=1).flatten())
zcr = librosa.feature.zero_crossing_rate(y)
features.append(float(np.mean(zcr)))
```

```
features.append(float(np.std(zcr)))
tempo, _ = librosa.beat.beat_track(y=y, sr=sr)
features.append(float(tempo.item()))
return np.array(features, dtype=np.float32)
except Exception as e:
print(f"Skipped {os.path.basename(file_path)}: {e}")
return None
print("Extracting features ")
features = []
valid_labels = []
expected_length = None
for _, row in tqdm(df.iterrows(), total=len(df)):
f = extract_features(row['file_path'])
if f is not None:
if expected_length is None:
expected_length = len(f)
if len(f) != expected_length:
print(f"Skipping {row['file_path']} (length {len(f)} != expected {expected_length})")
continue
features.append(f)
valid_labels.append(row['label'])
if not features:
raise ValueError("No valid features extracted!")
X = np.array(features)
y = np.array(valid_labels)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
print("Training Random Forest classifier...")
clf = RandomForestClassifier(n_estimators=150, max_depth=25, random_state=42)
clf.fit(X_train, y_train)
print("Evaluating model...")
y_pred = clf.predict(X_test)
print(f"\n Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```



```
Path to dataset files: /kaggle/input/gtzan-dataset-music-genre-classification
Found 1000 audio files.
Extracting features
 63%|██████    | 626/1000 [04:48<02:13,  2.80it/s]/tmp/ipython-input-3766951349.py:35: UserWarning: PySoundFile failed. Trying audioread instead.
  y, sr = librosa.load(file_path, duration=AUDIO_DURATION)
/usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio.__audioread_load
        Deprecated as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
  y, sr_native = __audioread_load(path, offset, duration, dtype)
 63%|██████▎   | 627/1000 [04:48<02:05,  2.97it/s]Skipped jazz.00054.wav:
100%|██████████| 1000/1000 [07:20<00:00,  2.27it/s]
Training Random Forest classifier...
Evaluating model...

 Accuracy: 0.66

Classification Report:
              precision    recall  f1-score   support

       blues       0.64      0.45      0.53        20
   classical       0.81      0.85      0.83        20
     country       0.70      0.70      0.70        20
       disco       0.58      0.70      0.64        20
      hiphop       0.83      0.75      0.79        20
        jazz       0.74      0.70      0.72        20
       metal       0.76      0.80      0.78        20
         pop       0.67      0.80      0.73        20
      reggae       0.58      0.55      0.56        20
        rock       0.30      0.30      0.30        20

    accuracy                           0.66       200
   macro avg       0.66      0.66      0.66       200
weighted avg       0.66      0.66      0.66       200
```

Disadvantages

1. Moderate Accuracy (≈66%)

- The achieved accuracy (≈66%) is not high enough for reliable real-world applications.
- This indicates that the features and classifier are not capturing genre distinctions strongly enough.
- Genres in the GTZAN dataset (e.g., rock vs. metal, jazz vs. blues) have overlapping characteristics, making misclassification common.

2. Large Feature Vector (~279 Features)

- The system extracts a very large number of features (MFCCs, deltas, chroma, contrast, ZCR, tempo).
- High-dimensional feature spaces often lead to:
    - Curse of dimensionality.
    - Overfitting risk, especially with limited dataset size.
    - Increased training time and memory usage.

3. Hand-crafted Features

- Features like MFCC, chroma, contrast are hand-engineered.
- They may not fully capture higher-level musical structure (rhythm patterns, instrumentation, harmony).

4. Genre Ambiguity

- Musical genres are subjective categories — some tracks may belong to multiple genres.
- Strict one-label classification oversimplifies the reality of music styles.

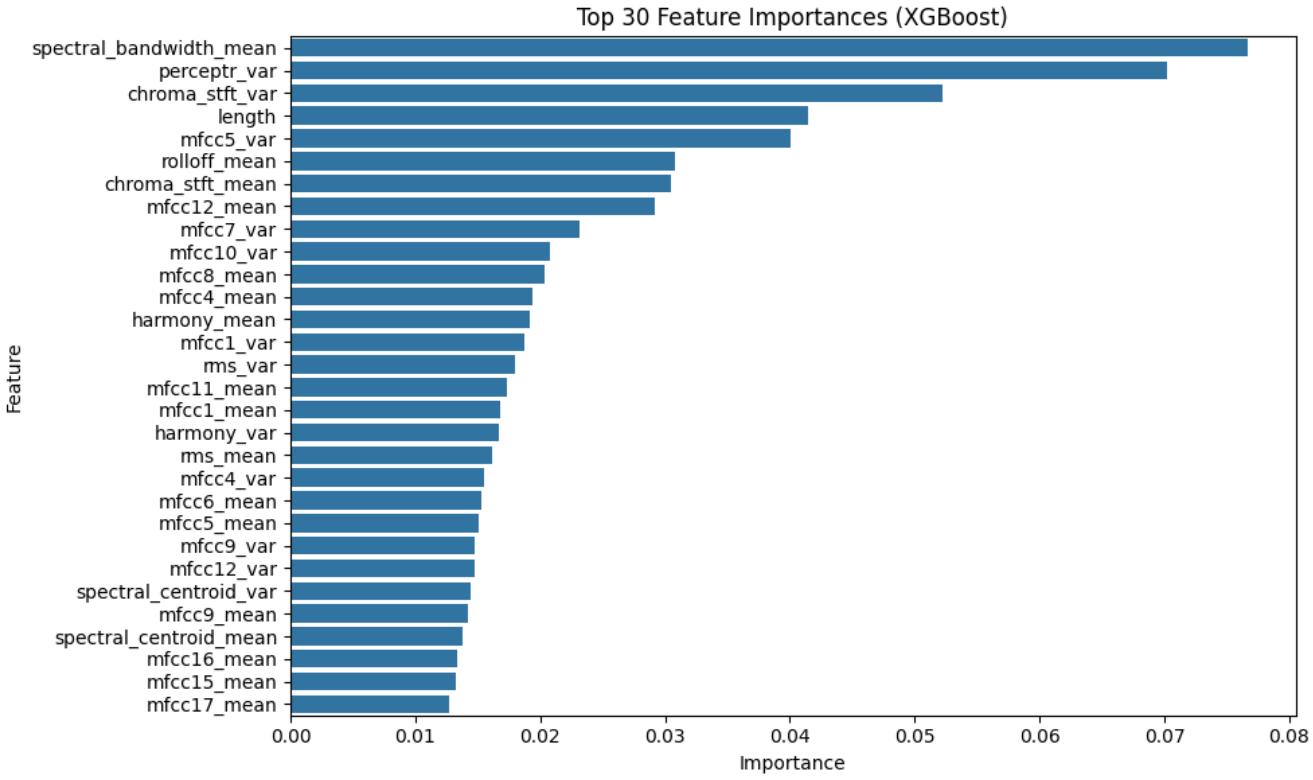
5. Random Forest Limitations

- While Random Forests are strong on tabular data, they:
    - Do not exploit the temporal structure of audio (sequence of features).
    - Treat features as independent, ignoring correlations between time-varying aspects.
    - Can become less interpretable with hundreds of features and many trees.

NEXT APPROACH

We now thought of implementing boosted trees. Starting with the most common Extreme Gradient Boosting and use the already provided csv file “features_30_seconds.csv”. Having the following random variables along with their PCA (XGBoost feature importance) :

filename, length, chroma_stft_mean, chroma_stft_var, rms_mean, rms_var, spectral_centroid_mean, spectral_centroid_var, spectral_bandwidth_mean, spectral_bandwidth_var,rolloff_mean, rolloff_var, zero_crossing_rate_mean, zero_crossing_rate_var, harmony_mean, harmony_var, perceptr_mean, perceptr_var, tempo, mfcc1_mean, mfcc1_var, mfcc2_mean, mfcc2_var, ……., mfcc19_mean, mfcc19_var, mfcc20_mean, mfcc20_var, label.



After applying XGBoost the accuracy went up to 79% but training with top 30 data and then 50 data the accuracy dropped down to 76% and 77% respectively.

OVERVIEW

The pipeline:

1. Load a CSV features_30_sec.csv containing precomputed audio features (one row per audio file).
2. Build X (features) and y (genre labels).
3. Encode labels and standardize features.
4. Split into train/test (stratified).
5. Train an XGBoost XGBClassifier (multiclass).
6. Evaluate with accuracy + classification report.
7. Plot top feature importances, then retrain using only the top-30 features and re-evaluate.

WHAT IS XGBCLASSIFIER ?

A scikit learn compatible wrapper for XGBoost based on gradient boosted decision trees. Builds trees in sequence each learning to correct previous errors. It also supports multi class classification and GPU acceleration. It builds decision trees attempts to reduce the errors of the previous trees.

Extreme gradient boosting is applied on data set like

$D = (x_i, y_i)|_{i=1}^{n}$

where ,

$x_i \epsilon \mathbb{R}^d are feature vectors$

$y_i \epsilon Label\ class/Index$

The model is an additive function
$\hat{y} = \sum_{t=1}^{T} f_t(x_i)\ , f_t \epsilon F$

where,

$\hat{y}_i \epsilon predicted label$

$x_i \epsilon input feature vectors$

$f_t = decision tree$

$F = Space of regression tree$

Extreme Boosting uses Taylor expansion to boost training

Let $\hat{y}^{(t-1)}$ be the current prediction at iteration t. We add a new tree $f_t(n)$ such that

$\hat{y}^t = \hat{y}^{(t-1)} + f_t(x_i)$

The objective function for multi class classification
$L(\theta) = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{\,t}\right) + \sum_k \Omega\left(f_k\right)$

becomes

$$L^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

The above problem is an convex optimization.

where,

$l\left(y_i, \hat{y}^{(t-1)}\right)$ is the multi class log loss function = $\frac{-1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \,.\, log(\hat{y_{ik}})$

where ,

N = total number of samples

K = number of classes

$y_{ik} = 1$ if sample i belongs to class k, else 0

$\hat{y_{ik}}$ = predicted probzbility that sample i belongs to class k

$\Omega(k) = \gamma * T + \frac{1}{2}\lambda * \sum w_j^2$ is the L2 Regularization (Ridge) which adds penalty to large weights / leaf scores.
T = number of leaves

w_j = score of each leaf
λ = L2 regularization parameter

$$w_j = \begin{matrix} \frac{-(\sum g_i)}{(\sum h_i + \lambda)} if |\sum g_i| > \alpha \\ 0 otherwise \end{matrix}$$

$\alpha = L1 regularization parameter$

$g_i = gradient/first\ order differential of predicted value$

$h_i = hessian/second\ order differential of predicted value$

HOW THE NODES ARE SPLITTED

XGBoost relies on Gain formula
$Gain = \frac{1}{2} * \frac{(G_l)^2}{(H_l+\lambda)} + \frac{(G_r)^2}{(H_r+\lambda)} - \frac{(G)^2}{(H+\lambda)} - s$

where,

G_l, H_l = sum of gradients and hessians for left child

G_r, H_r = sum of gradients and hessians for right child

G, H = total gradient and hessian of the parent

s = minimum split loss branching condition or prunning condition

DATA INPUT AND SPLITTING

Reads the dataset :

X : feature vectors which are only numeric type

y : labels which contains genre's

filenames : Used to validate results

So overall the data gets splitted into two tuple's

$(X_{train}, y_{train}), (X_{test}, y_{test})$

We used stratification which preserves class properties ensuring balanced genre distribution 80% for training 20% for testing.

DATA PREPROCESSING

Label encoder converts object labels into integer values. This creates y_encoded which the classifier can process.

Standard Scalar standardizes features to zero mean and unit varience

$x' = \frac{x-\mu}{\sigma}$ $\mu = meanoffeature, \sigma = standarddeviation$

MODEL PARAMETERS

Default model label encoder is disabled, evaluation metric is set to 'mlogloss' log loss for multi class classification, tree generation method is set to GPU and verbosity is set to 0 to suppress output.

MODEL HYPER PARAMETERS

n_estimators : Number of boosting trees, more trees results in higher capacity but might lead to overfitting.

max_depth : Maximum depth of individual trees, controlling model complexity resulting in shallower trees resulting in simpler models.

learning_rate : Shrinks each trees contribution to next tree.

subsample : Adds randomness to controll overfitting.

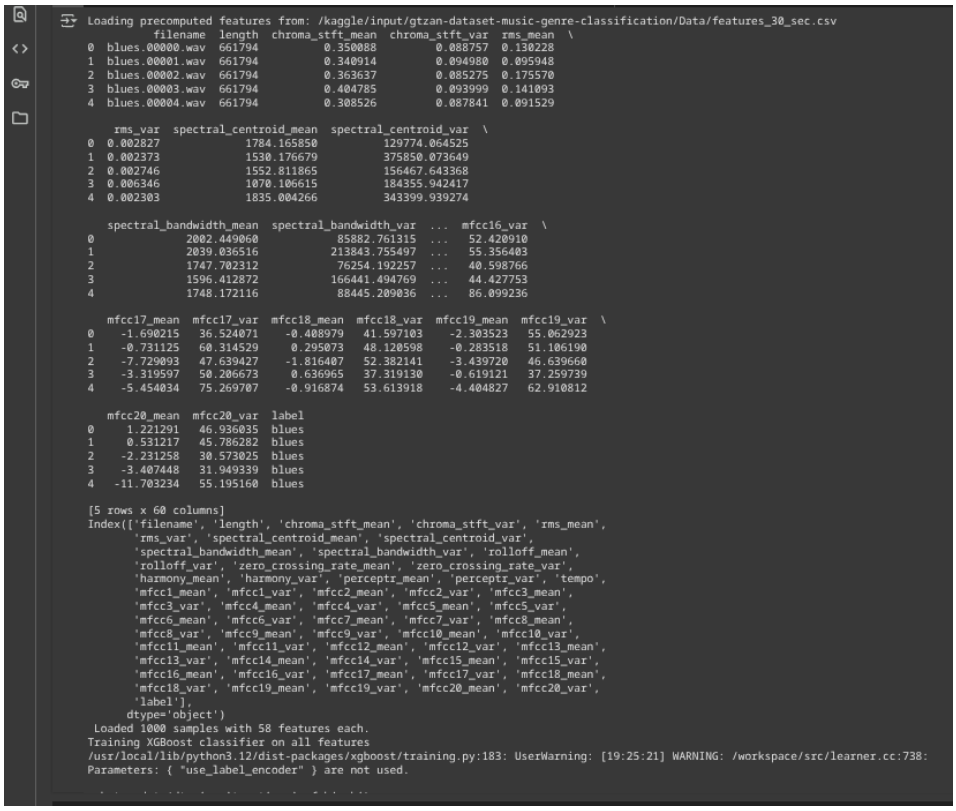
colsample_bytree : Feature bagging like random forest.

gamma : Minimum loss reduction required for split (gain)

MODEL

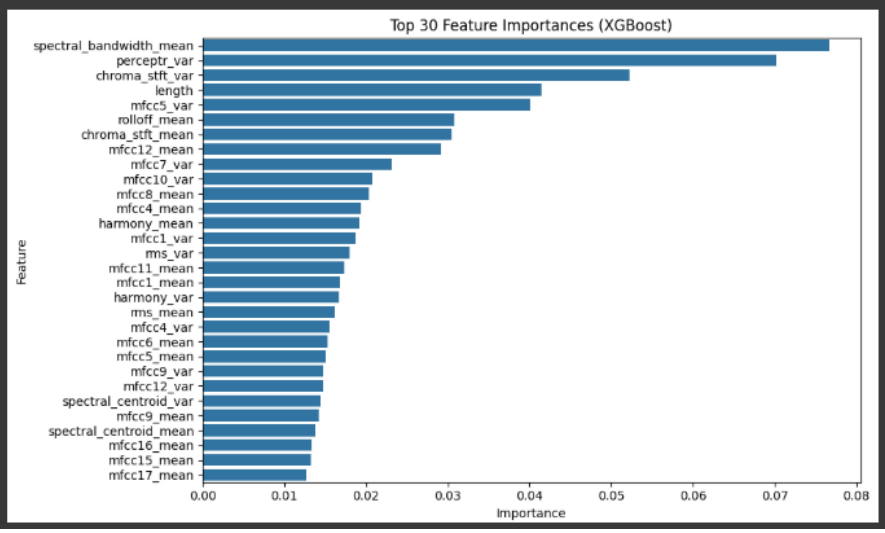
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier
import kagglehub
# Download dataset
path = kagglehub.dataset_download("andradaolteanu/gtzan-dataset-music-genre-classification")
csv_path = os.path.join(path, "Data", "features_30_sec.csv")
print(f"Loading precomputed features from: {csv_path}")
df = pd.read_csv(csv_path)
print(df.head())
print(df.columns)
X = df.drop(columns=["filename", "label"]).values
y = df["label"].values
feature_names = df.drop(columns=["filename", "label"]).columns
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
print(f" Loaded {X.shape[0]} samples with {X.shape[1]} features each.")
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
X_scaled, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)
print("Training XGBoost classifier on all features")
clf = XGBClassifier(n_estimators=300, max_depth=8, learning_rate=0.05,
use_label_encoder=False, eval_metric='mlogloss')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(f"\nAccuracy: {accuracy_score(y_test, y_pred):.2f}")
print("\n Classification Report:")
print(classification_report(label_encoder.inverse_transform(y_test),
label_encoder.inverse_transform(y_pred)))
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
top_n = 30
plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices[:top_n]], y=feature_names[indices[:top_n]])
plt.title("Top 30 Feature Importances (XGBoost)")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
print(f"Retraining using top {top_n} features based on importance...")
```

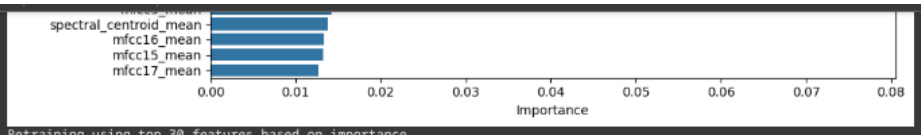
```
X_train_top = X_train[:, indices[:top_n]]
X_test_top = X_test[:, indices[:top_n]]
clf_top = XGBClassifier(n_estimators=300, max_depth=8, learning_rate=0.05,
use_label_encoder=False, eval_metric='mlogloss')
clf_top.fit(X_train_top, y_train)
y_pred_top = clf_top.predict(X_test_top)
print(f"\nAccuracy with top {top_n} features: {accuracy_score(y_test, y_pred_top):.2f}")
print("\nClassification Report (Top Features):")
print(classification_report(label_encoder.inverse_transform(y_test),
label_encoder.inverse_transform(y_pred_top)))
```



```
Loading precomputed features from: /kaggle/input/gtzan-dataset-music-genre-classification/Data/features_30_sec.csv
          filename  length  chroma_stft_mean  chroma_stft_var  rms_mean  \
0  blues.00000.wav  661794          0.350088         0.088757  0.130228
1  blues.00001.wav  661794          0.340914         0.094980  0.095948
2  blues.00002.wav  661794          0.363637         0.085275  0.175570
3  blues.00003.wav  661794          0.404785         0.093999  0.141093
4  blues.00004.wav  661794          0.308526         0.087841  0.091529

    rms_var  spectral_centroid_mean  spectral_centroid_var  \
0  0.002827             1784.165850          129774.064525
1  0.002373             1530.176679          375850.073649
2  0.002746             1552.811865          156467.643368
3  0.006346             1070.106615          184355.942417
4  0.002303             1835.004266          343399.939274

   spectral_bandwidth_mean  spectral_bandwidth_var  ...  mfcc16_var  \
0              2002.449060            85882.761315  ...   52.420910
1              2039.036516           213843.755497  ...   55.356403
2              1747.702312            76254.192257  ...   40.598766
3              1596.412872           166441.494769  ...   44.427753
4              1748.172116            88445.209036  ...   86.099236

   mfcc17_mean  mfcc17_var  mfcc18_mean  mfcc18_var  mfcc19_mean  mfcc19_var  \
0    -1.690215   36.524071    -0.408979   41.597103    -2.303523   55.062923
1    -0.731125   60.314529     0.295073   48.120598    -0.283518   51.106190
2    -7.729093   47.639427    -1.816407   52.382141    -3.439720   46.639660
3    -3.319597   50.206673     0.636965   37.319130    -0.619121   37.259739
4    -5.454034   75.269707    -0.916874   53.613918    -4.404827   62.910812

   mfcc20_mean  mfcc20_var  label
0     1.221291   46.936035  blues
1     0.531217   45.786282  blues
2    -2.231258   30.573025  blues
3    -3.407448   31.949339  blues
4   -11.703234   55.195160  blues

[5 rows x 60 columns]
Index(['filename', 'length', 'chroma_stft_mean', 'chroma_stft_var', 'rms_mean',
       'rms_var', 'spectral_centroid_mean', 'spectral_centroid_var',
       'spectral_bandwidth_mean', 'spectral_bandwidth_var', 'rolloff_mean',
       'rolloff_var', 'zero_crossing_rate_mean', 'zero_crossing_rate_var',
       'harmony_mean', 'harmony_var', 'perceptr_mean', 'perceptr_var', 'tempo',
       'mfcc1_mean', 'mfcc1_var', 'mfcc2_mean', 'mfcc2_var', 'mfcc3_mean',
       'mfcc3_var', 'mfcc4_mean', 'mfcc4_var', 'mfcc5_mean', 'mfcc5_var',
       'mfcc6_mean', 'mfcc6_var', 'mfcc7_mean', 'mfcc7_var', 'mfcc8_mean',
       'mfcc8_var', 'mfcc9_mean', 'mfcc9_var', 'mfcc10_mean', 'mfcc10_var',
       'mfcc11_mean', 'mfcc11_var', 'mfcc12_mean', 'mfcc12_var', 'mfcc13_mean',
       'mfcc13_var', 'mfcc14_mean', 'mfcc14_var', 'mfcc15_mean', 'mfcc15_var',
       'mfcc16_mean', 'mfcc16_var', 'mfcc17_mean', 'mfcc17_var', 'mfcc18_mean',
       'mfcc18_var', 'mfcc19_mean', 'mfcc19_var', 'mfcc20_mean', 'mfcc20_var',
       'label'],
      dtype='object')
 Loaded 1000 samples with 58 features each.
Training XGBoost classifier on all features
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [19:25:21] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
```

```
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

Accuracy: 0.79

Classification Report:
              precision    recall  f1-score   support

       blues       0.75      0.75      0.75        20
   classical       0.95      0.95      0.95        20
     country       0.71      0.75      0.73        20
       disco       0.67      0.60      0.63        20
      hiphop       0.74      0.70      0.72        20
        jazz       0.74      0.85      0.79        20
       metal       0.89      0.80      0.84        20
         pop       0.82      0.90      0.86        20
      reggae       0.81      0.85      0.83        20
        rock       0.78      0.70      0.74        20

    accuracy                           0.79       200
   macro avg       0.79      0.78      0.78       200
weighted avg       0.79      0.79      0.78       200
```






```
Retraining using top 30 features based on importance...
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [19:25:36] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

Accuracy with top 30 features: 0.76

Classification Report (Top Features):
              precision    recall  f1-score   support

       blues       0.74      0.70      0.72        20
   classical       0.91      1.00      0.95        20
     country       0.61      0.70      0.65        20
       disco       0.75      0.60      0.67        20
      hiphop       0.75      0.75      0.75        20
        jazz       0.75      0.90      0.82        20
       metal       0.89      0.85      0.87        20
         pop       0.79      0.95      0.86        20
      reggae       0.74      0.70      0.72        20
        rock       0.64      0.45      0.53        20

    accuracy                           0.76       200
   macro avg       0.76      0.76      0.75       200
weighted avg       0.76      0.76      0.75       200
```

PROPER APPLICATION OF XGBOOST USING RANDOM SEARCH TO IMPROVE MODEL ACCURACY FOR SMALLER DATASET.

Randomized Search is a method to tune hyperparameters of a machine learning model efficiently. Unlike Grid Search, which exhaustively tries all combinations, Randomized Search samples a fixed number of parameter combinations randomly from a defined distribution or set.

This is particularly useful when:

- The hyperparameter space is large.
- Training each model is computationally expensive.

Mathematically:

Let the hyperparameter space be:

$\Theta = (\theta_1, \theta_2, \ldots, \theta_m)$

where each $\theta_i [i \epsilon \mathbb{N}]$ represents a hyperparameter

We then define the objective function as

$f(\theta) = CV - score$

Randomized Search selects n_iter random points $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_{iter})}$ from Θ and evaluates $f(\theta^{(i)})$ using cross-validation .

Finally the best hyperparameters are $\theta^x = \underset{\theta^{(i)}}{argmax} f(\theta^{(i)})$

param_dist defines the search space for each hyperparameter. Each hyperparameter can take a set of discrete values.

For each combination θ(i):
a. Perform 5-fold stratified cross-validation (skf).
b. Compute macro F1-score for each fold:

$F1_{macro} = \frac{1}{K} \sum_{k=1}^{K} F\,1_k$

where K is the number of class

estimator=xgb : The XGBoost model we want to tune.

param_distributions=param_dist : Defines the hyperparameter space to sample from.

n_iter=20: Number of random combinations to try.

scoring='f1_macro' : Objective function to optimize (macro F1-score across all classes).

cv=skf: Stratified K-Fold cross-validation for evaluation.

n_jobs=1: Number of parallel jobs.

random_state=42: For reproducibility.

Select the combination θ^x with highest average score.

random_search.fit(X_train_full, y_train_full, sample_weight=sample_weights)

sample_weight ensures that imbalanced classes (like 'rock') are weighted more heavily during evaluation.

CODE AND MODEL EVALUATION

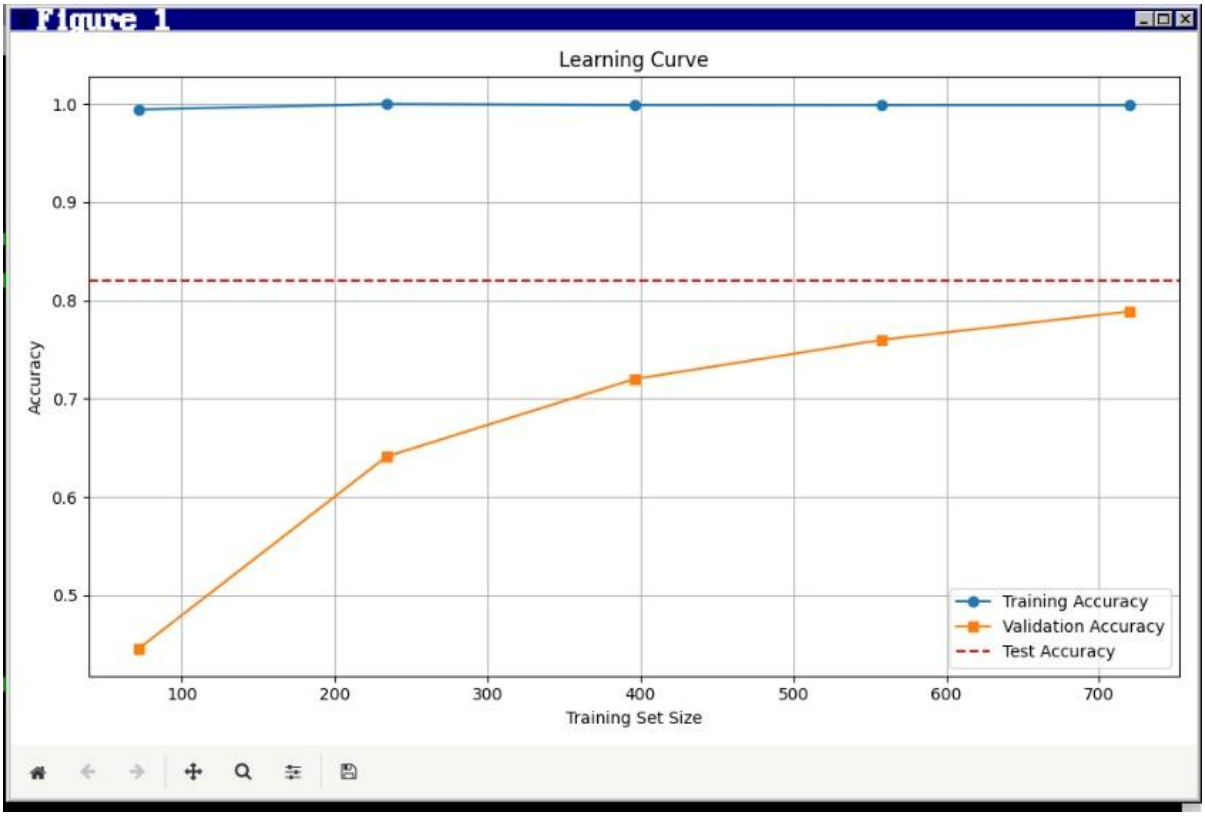
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold, RandomizedSearchCV,
train_test_split, cross_val_predict
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
from sklearn.utils.class_weight import compute_class_weight

def main():
    df = pd.read_csv("/home/parijat/machine_learning/Data/features_30_sec.csv")
    y = df['label']
    X = df.drop(columns=['label', 'filename'])
    X = X.select_dtypes(include=[np.number])

    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    X_train_full, X_test, y_train_full, y_test = train_test_split(
        X_scaled, y_encoded, test_size=0.1, stratify=y_encoded, random_state=42
    )
```

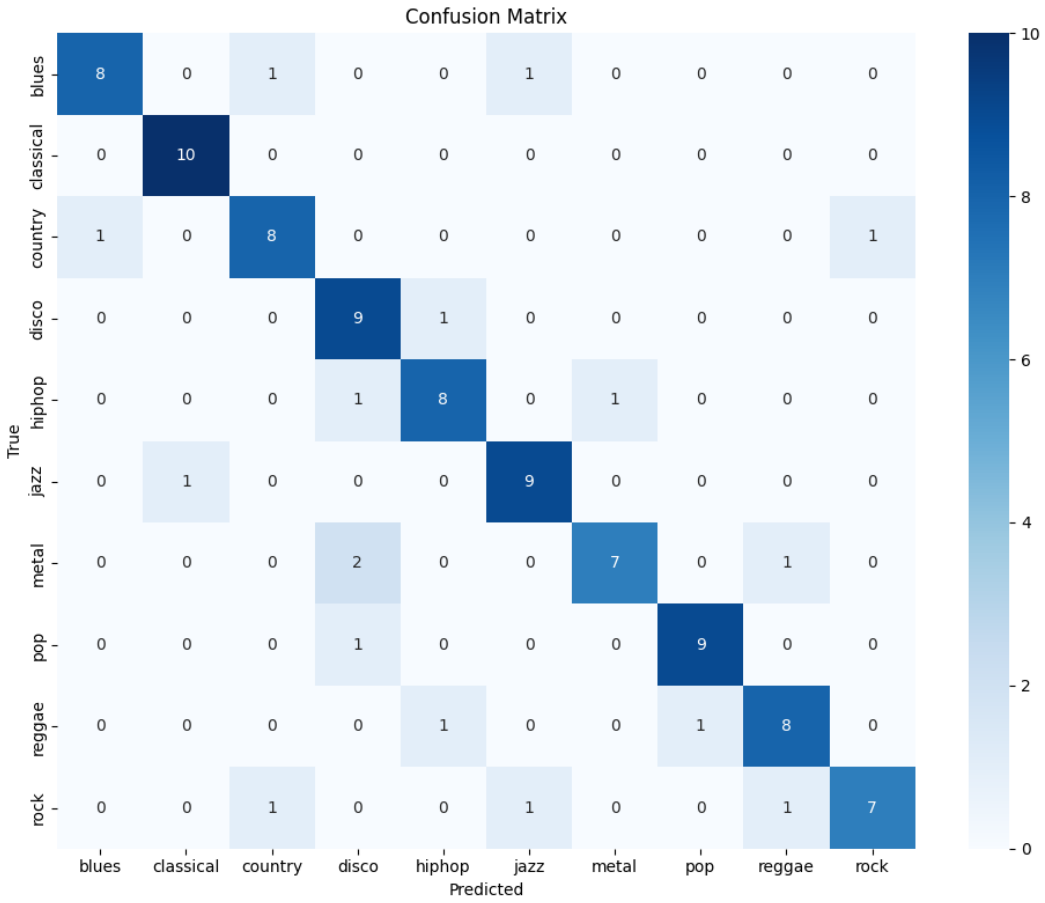
```
    classes = np.unique(y_train_full)
    class_weights = compute_class_weight('balanced', classes=classes, y=y_train_full)
    scale_pos_weight_array = {i: w for i, w in zip(classes, class_weights)}
    rock_idx = np.where(label_encoder.classes_ == 'rock')[0][0]
    scale_pos_weight_array[rock_idx] *= 2.0

    sample_weights = np.array([scale_pos_weight_array[label] for label in y_train_full])

    xgb = XGBClassifier(
        use_label_encoder=False,
        tree_method='gpu_hist',
        predictor='gpu_predictor',
        verbosity=0
    )

    param_dist = {
        'n_estimators': [150, 180, 200, 220],
        'max_depth': [3, 4],
        'learning_rate': [0.05, 0.08],
        'subsample': [0.8, 0.85],
        'colsample_bytree': [0.7, 0.8],
        'gamma': [0, 0.5],
        'reg_lambda': [10, 15],
        'reg_alpha': [0, 1],
        'min_child_weight': [1, 3]
    }

    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
    random_search = RandomizedSearchCV(
        estimator=xgb,
        param_distributions=param_dist,
        n_iter=20,
        scoring='f1_macro',
        n_jobs=1,
        cv=skf,
        verbose=2,
        random_state=42
    )
    random_search.fit(X_train_full, y_train_full, sample_weight=sample_weights)

    print(f"Best parameters: {random_search.best_params_}")
    print(f"Best CV f1_macro: {random_search.best_score_:.4f}")

    best_model = XGBClassifier(
        use_label_encoder=False,
        tree_method='gpu_hist',
        predictor='gpu_predictor',
        verbosity=0,
        **random_search.best_params_
    )
    best_model.fit(X_train_full, y_train_full, sample_weight=sample_weights)
    best_model.set_params(predictor='cpu_predictor')

    y_test_pred = best_model.predict(X_test)
    test_acc = accuracy_score(y_test, y_test_pred)
```

```
    print(f"Test Accuracy: {test_acc:.4f}")
    print("\nClassification Report:\n", classification_report(y_test, y_test_pred,
target_names=label_encoder.classes_))

    cm = confusion_matrix(y_test, y_test_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=label_encoder.classes_,
                yticklabels=label_encoder.classes_)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.savefig("confusion_matrix.png")
    plt.show()

    with open("xgb_genre_classifier.pkl", "wb") as f:
        pickle.dump(best_model, f)
    with open("label_encoder.pkl", "wb") as f:
        pickle.dump(label_encoder, f)
    with open("scaler.pkl", "wb") as f:
        pickle.dump(scaler, f)

    y_train_cv_pred = cross_val_predict(best_model, X_train_full, y_train_full, cv=skf)
    train_cv_acc = accuracy_score(y_train_full, y_train_cv_pred)

    plt.figure(figsize=(8,5))
    plt.bar(['CV Training Accuracy', 'Test Accuracy'], [train_cv_acc, test_acc],
color=['skyblue', 'salmon'])
```

```
    plt.ylim(0, 1)
    plt.ylabel('Accuracy')
    plt.title('Cross-validated Training vs Test Accuracy')
    plt.tight_layout()
    plt.show()

    print(f"Cross-validated Train Accuracy: {train_cv_acc:.4f}")

if __name__ == "__main__":
    main()
```






```
ha=1, reg_lambda=10, subsample=0.85; total time=   1.7s
[CV] END colsample_bytree=0.8, gamma=0, learning_rate=0.08, max_depth=4, min_child_weight=3, n_estimators=180, reg_alp
ha=1, reg_lambda=10, subsample=0.85; total time=   1.7s
Best parameters: {'subsample': 0.8, 'reg_lambda': 15, 'reg_alpha': 1, 'n_estimators': 220, 'min_child_weight': 1, 'max
_depth': 4, 'learning_rate': 0.08, 'gamma': 0, 'colsample_bytree': 0.8}
Best CV f1_macro: 0.7986
Test Accuracy: 0.8300

Classification Report:
              precision    recall  f1-score   support

       blues       0.89      0.80      0.84        10
   classical       0.91      1.00      0.95        10
     country       0.80      0.80      0.80        10
       disco       0.69      0.90      0.78        10
      hiphop       0.80      0.80      0.80        10
        jazz       0.82      0.90      0.86        10
       metal       0.88      0.70      0.78        10
         pop       0.90      0.90      0.90        10
      reggae       0.80      0.80      0.80        10
        rock       0.88      0.70      0.78        10

    accuracy                           0.83       100
   macro avg       0.84      0.83      0.83       100
weighted avg       0.84      0.83      0.83       100

Cross-validated Train Accuracy: 0.7967
```

DISADVANTAGES

1) The graph clearly shows overfitting.

2) XGBoost is not very compatible with small datase.

3) The parameters sets needs to be proper not suitable for all devices.

4) Grid Search might be better but size of dataset is small.

5) Limited hyperparameters exploration

6) No early stopping parameter to control overfitting.

7) Single evaluation metric.

FINAL MODEL, CODE AND IMPLEMENTATION OF WEB APP .

We decided to switch to CatBoost i.e. categorical boosting (we didn't go with KNN as CatBoost is better with dataset like these). CatBoost minimizes the following regularized loss at each iteration:

$$L = \sum_{i=1}^{N} w_i \, . \, l(y_i, \hat{y}_i) + \Omega(f)$$

Handles class imbalance by assigning larger weights to underrepresented genres.

Further increases weights for rock and reggae classes (ad-hoc adjustment).

Each sample now has an associated weight for loss scaling during training.

CatBoost Pool object allows passing sample weights, categorical features, and evaluation sets.

Gradient boosting tries to minimize a loss function L(y,$\hat{y}$) by building an additive model:

$F_M(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$

$F_M(x) = Finalmodelafter Mtrees$

$h_m(x) = istheweaklearnermthdecesiontree$

$\gamma_m = weight/score\ of the tree$

$L(y, \hat{y})$ = Log loss

Algorithm:

Initialize model with a constant prediction:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L\,(y_i, \gamma)$$

For m=1 to M: Compute pseudo-residuals (gradient of the loss w.r.t current predictions):

$$r_{\Im} = -\left[\frac{\partial L(y_i, \hat{y}_i)}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

Fit a new tree hm(x) to these residuals.

Find optimal weight γm for the tree:

$$\gamma_m = \underset{\gamma}{argmin} \sum_{i=1}^{n} L\left(y_i, y_{i-1}^{\wedge} + \gamma h_m(x_i)\right)$$

Update model:

$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$

Final Prediction:

$\hat{y} = F_m(x)$

CatBoost Enhancements Over Classic Gradient Boosting

CatBoost improves traditional gradient boosting in three key ways:

Handling Categorical Features

Instead of one-hot encoding (which blows up feature dimension), CatBoost uses Ordered Target Statistics:

$Categoryvalueencoding = \frac{\sum_{j=1}^{i-1} y_j + a.prior}{i-1+a}$

where:

y_j are labels of previous rows with the same category.

a is a smoothing parameter to avoid overfitting.

This is computed sequentially to prevent target leakage.

Ordered Boosting

Classic gradient boosting can overfit when using target statistics on the same data. CatBoost uses ordered boosting:

Split data into permutations.

For each row, only use previous rows in the permutation to compute residuals and statistics.

This prevents information leakage.

Symmetric Trees (Oblivious Trees)

Instead of arbitrary decision trees, CatBoost builds symmetric trees:

All splits at a given depth use the same feature and threshold.

Benefits:

Faster inference.Regularization against overfitting.Easier vectorized implementation.

Mathematically, each leaf has a sum of gradients and Hessians, and the split that minimizes the loss function is chosen:

For a node split :

$Gain = Loss_{parent} - (Loss_{leftchild} + Loss_{rightchild})$

where,

$Loss_{node} = \sum_{i \epsilon node} L(y_i, \hat{y_i})$

Regularization and Overfitting Control

CatBoost uses:

- **L2 leaf regularization** (λ∑w_j^2): reduces overfitting of leaf values.
- **Bagging temperature** : introduces randomness like stochastic gradient boosting.
- **Random strength:** randomly perturbs splits for robustness.

Value Encoding Uses ParameterGrid to enumerate all hyperparameter combinations.

Iterates manually and selects the best validation accuracy.

depth : Max depth of trees (controls model complexity)

learning_rate: Shrinks each tree's contribution:

iterations: Number of boosting rounds

l2_leaf_reg: L2 regularization for leaf weights ()

bagging_temperature: Controls randomness for Ordered Boosting

random_strength: Randomness added in tree splits to prevent overfitting

CODE AND MODEL EVALUATION

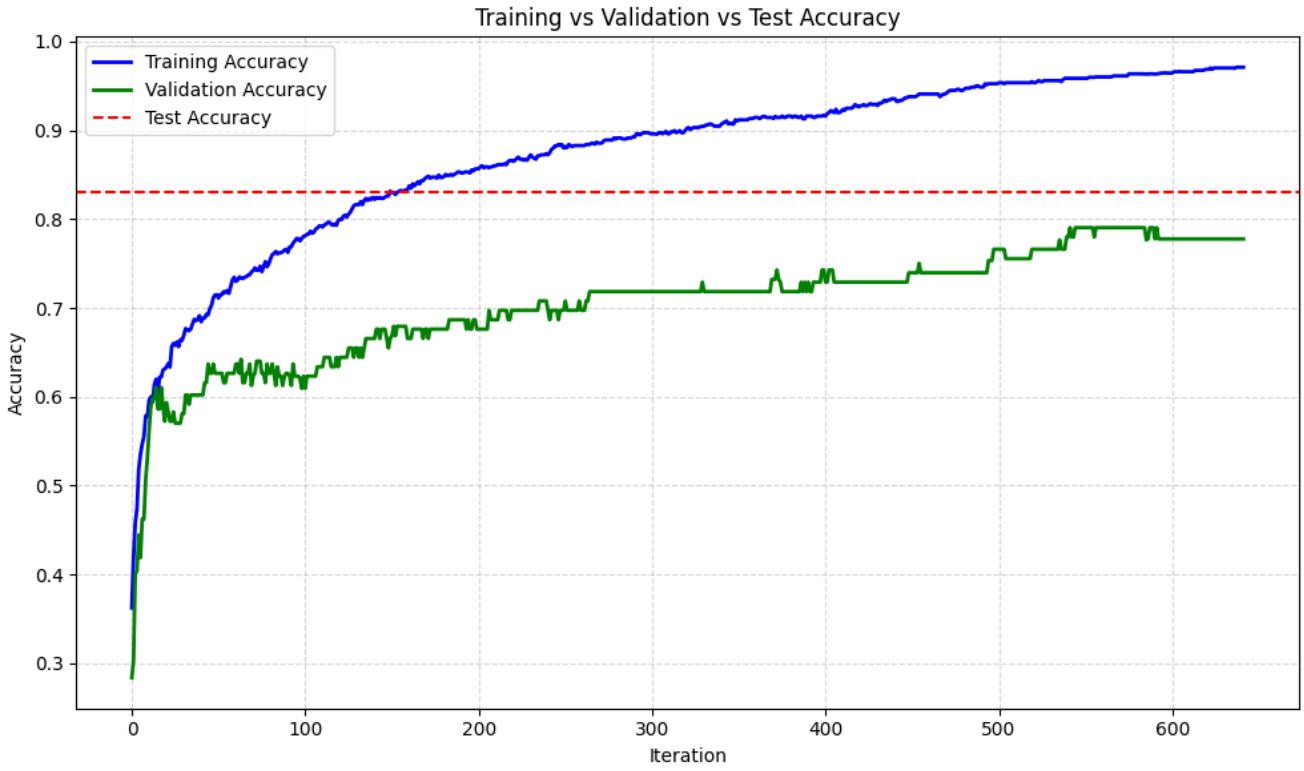
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, ParameterGrid, StratifiedKFold
from catboost import CatBoostClassifier, Pool
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.utils.class_weight import compute_class_weight
import seaborn as sns

def main():
    df = pd.read_csv("/home/parijat/machine_learning/Data/features_30_sec.csv")
    y = df['label']
    X = df.drop(columns=['label', 'filename'])
    X = X.select_dtypes(include=[np.number])

    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    X_train_full, X_test, y_train_full, y_test = train_test_split(
        X_scaled, y_encoded, test_size=0.1, stratify=y_encoded, random_state=42
    )

    classes = np.unique(y_train_full)
```

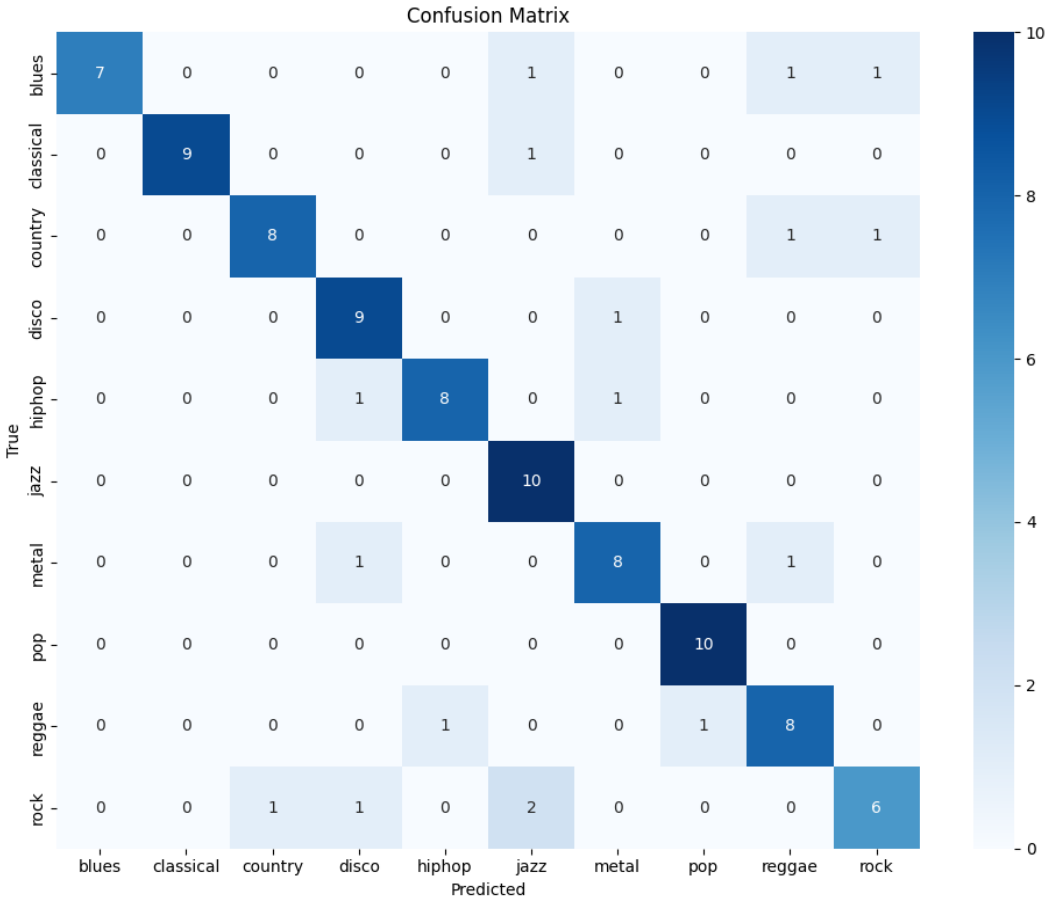
```
    class_weights = compute_class_weight('balanced', classes=classes, y=y_train_full)
    scale_pos_weight_array = {i: w for i, w in zip(classes, class_weights)}

    rock_idx = np.where(label_encoder.classes_ == 'rock')[0][0]
    reggae_idx = np.where(label_encoder.classes_ == 'reggae')[0][0]

    scale_pos_weight_array[rock_idx] *= 1.3
    scale_pos_weight_array[reggae_idx] *= 1.2

    sample_weights = np.array([scale_pos_weight_array[label] for label in y_train_full])

    X_train, X_val, y_train, y_val, sw_train, sw_val = train_test_split(
        X_train_full, y_train_full, sample_weights, test_size=0.1, stratify=y_train_full,
random_state=42
    )

    train_pool = Pool(X_train, y_train, weight=sw_train)
    val_pool = Pool(X_val, y_val, weight=sw_val)

    param_grid = {
        'depth': [4, 5],
        'learning_rate': [0.03, 0.05],
        'iterations': [1000],
        'l2_leaf_reg': [7, 10],
        'bagging_temperature': [0.5, 1.0],
        'random_strength': [1, 2]
    }
```

```
    best_acc = 0
    best_params = None
    best_model = None

    for params in ParameterGrid(param_grid):
        model = CatBoostClassifier(
            **params,
            boosting_type='Ordered',
            eval_metric='Accuracy',
            early_stopping_rounds=100,
            task_type='CPU',
            devices='0',
            random_seed=42,
            verbose=False
        )
        model.fit(train_pool, eval_set=val_pool, use_best_model=True)
        val_pred = model.predict(X_val)
        val_acc = accuracy_score(y_val, val_pred)
        if val_acc > best_acc:
            best_acc = val_acc
            best_params = params
            best_model = model

    print(f"Best Parameters: {best_params}")
    y_test_pred = best_model.predict(X_test)
    test_acc = accuracy_score(y_test, y_test_pred)
    print(f"Test Accuracy: {test_acc:.4f}")
```

```
    print("\nClassification Report:\n", classification_report(y_test, y_test_pred,
target_names=label_encoder.classes_))

    cm = confusion_matrix(y_test, y_test_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=label_encoder.classes_,
                yticklabels=label_encoder.classes_)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.show()

    with open("catboost_genre_classifier.pkl", "wb") as f:
        pickle.dump(best_model, f)
    with open("label_encoder.pkl", "wb") as f:
        pickle.dump(label_encoder, f)
    with open("scaler.pkl", "wb") as f:
        pickle.dump(scaler, f)

    evals_result = best_model.get_evals_result()
    train_acc_line = evals_result['learn']['Accuracy']
    val_acc_line = evals_result['validation']['Accuracy']

    plt.figure(figsize=(10,6))
    plt.plot(train_acc_line, label='Training Accuracy', color='blue', linewidth=2)
    plt.plot(val_acc_line, label='Validation Accuracy', color='green', linewidth=2)
```

```
    plt.axhline(y=test_acc, color='red', linestyle='--', label='Test Accuracy')
    plt.xlabel('Iteration')
    plt.ylabel('Accuracy')
    plt.title('Training vs Validation vs Test Accuracy')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```






```
[parijat@Parijatarch ~]$ cd machine_learning/Fin
[parijat@Parijatarch Fin]$ python Music.py
Best Parameters: {'bagging_temperature': 0.5, 'depth': 4, 'iterations': 1000, 'l2_leaf_reg': 7, 'learning_rate': 0.05,
 'random_strength': 2}
Test Accuracy: 0.8300

Classification Report:
              precision    recall  f1-score   support

       blues       1.00      0.70      0.82        10
   classical       1.00      0.90      0.95        10
     country       0.89      0.80      0.84        10
       disco       0.75      0.90      0.82        10
      hiphop       0.89      0.80      0.84        10
        jazz       0.71      1.00      0.83        10
       metal       0.80      0.80      0.80        10
         pop       0.91      1.00      0.95        10
      reggae       0.73      0.80      0.76        10
        rock       0.75      0.60      0.67        10

    accuracy                           0.83       100
   macro avg       0.84      0.83      0.83       100
weighted avg       0.84      0.83      0.83       100

[parijat@Parijatarch Fin]$
```

1. Purpose of the Script

The script is a Streamlit web application that allows users to:

- Sign up or log in with credentials (simple in-memory user management).
- Upload a music file (.mp3 or .wav).
- Visualize audio features such as waveform, MFCCs, and spectrogram.
- Extract relevant audio features using Librosa.
- Use a pre-trained ML model (CatBoost classifier) with scaler and label encoder to predict the genre of the uploaded music.

2. Libraries Used

- **Streamlit (**st**)** = Web app UI framework.
- **Pandas (**pd**)** = Data handling (feature storage).
- **NumPy (**np**)** = Numerical computations.
- **Librosa** = Audio feature extraction and visualization.
- **Matplotlib (**plt**)** = Plotting audio visualizations.
- **Pickle** = Loading pre-trained ML models, scaler, and label encoder.
- **Re (Regex)** = Email validation during signup.

3. Script Structure

The script is modular and organized into functional sections:

A. Page Configuration

```
st.set_page_config(
   page_title="Music Genre Classification",
   page_icon=":musical_note:",
   layout="wide"
)
```

- Sets app title, icon, and layout (wide for better visualization).

B. User Data Management

```
if 'users' not in st.session_state:
   st.session_state['users'] = { "admin@example.com": {"name": "Admin User","password": "password123"} }
```

- Implements a simple in-memory user database.
- Uses st.session_state to persist users across sessions.

- Includes a default admin account.

C. Authentication Pages

1. **Login Page (**`login_page`**)**
    - Collects email + password.
    - Validates credentials against `st.session_state['users']`.
    - On success, sets `logged_in = True` and stores `user_info`.
    - On failure, shows error messages.
2. **Signup Page (**`signup_page`**)**
    - Collects user info: full name, email, password, confirm password.
    - Performs checks:
        - All fields filled.
        - Email format valid (via regex).
        - Email not already in use.
        - Passwords match.
    - If valid → stores user in session.
    - Provides feedback with `st.success`, `st.warning`, or `st.error`.

D. Main App (main_app)

Accessible only when logged in.

1. **Sidebar**
    - Displays user name.
    - Provides logout button (clears session info).
2. **Title & Instructions**
    - Introduces the app.
    - Guides the user to upload music files.
3. **Model Loading**
    - Loads:
        - Pre-trained classifier (`catboost_genre_classifier.pkl`).
        - Scaler (`scaler.pkl`).

        - Label Encoder (label_encoder.pkl).
    - Uses @st.cache_data for performance (prevents reloading on each interaction).

4. **Feature Extraction (**extract_features**)**
    - Extracts numerical features from audio:
        - **Chroma STFT**
        - **RMS Energy**
        - **Spectral Centroid**
        - **Spectral Bandwidth**
        - **Spectral Rolloff**
        - **Zero Crossing Rate**
        - **Harmonic components**
        - **Perceptual weighting**
        - **Tempo**
        - **MFCCs (20 coefficients, mean + variance each)**
    - Returns a DataFrame suitable for model input.

5. **File Upload & Processing**
    - Accepts .mp3 or .wav.
    - Plays audio in browser (st.audio).
    - Shows audio visualizations:
        - Waveform
        - MFCC heatmap
        - Spectrogram
    - Extracts features → scales → predicts genre → displays result.

E. Router / Main Script Logic

Controls app navigation:

- If logged_in == True: run main_app().
- Else: show login/signup options via sidebar.

4. Flow of Execution

1. User opens app → sees login/signup.
2. Logs in or signs up.
3. Upon login:
    - Sidebar shows welcome + logout.
    - Main section shows file uploader.
    - User uploads audio file.
    - Audio processed → features extracted → model predicts genre.
    - Prediction + visualizations shown.

CODE FOR WEB APP

```
import streamlit as st
import pandas as pd
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt
import pickle
import re

st.set_page_config(
    page_title="Music Genre Classification",
    page_icon=":musical_note:",
    layout="wide"
)

if 'users' not in st.session_state:
    st.session_state['users'] = {
        "admin@example.com": {
            "name": "Admin User",
            "password": "password123"
        }
    }

def login_page():
    st.header("Login to Your Account")
    with st.form("login_form"):
        email = st.text_input("Email")
```

```
        password = st.text_input("Password", type="password")
        submitted = st.form_submit_button("Login")

        if submitted:
            users = st.session_state.get('users', {})
            if email in users and users[email]['password'] == password:
                st.session_state['logged_in'] = True
                st.session_state['user_info'] = users[email]
                st.success("Logged in successfully!")
                st.rerun()
            else:
                st.error("Incorrect email or password.")

def signup_page():
    st.header("Create a New Account")
    with st.form("signup_form"):
        name = st.text_input("Full Name")
        email = st.text_input("Email Address")
        password = st.text_input("Password", type="password")
        confirm_password = st.text_input("Confirm Password", type="password")
        submitted = st.form_submit_button("Sign Up")

        if submitted:
            users = st.session_state.get('users', {})
            if not (name and email and password and confirm_password):
                st.warning("Please fill out all fields.")
            elif not re.match(r"[^@]+@[^@]+\.[^@]+", email):
                st.warning("Please enter a valid email address.")
```

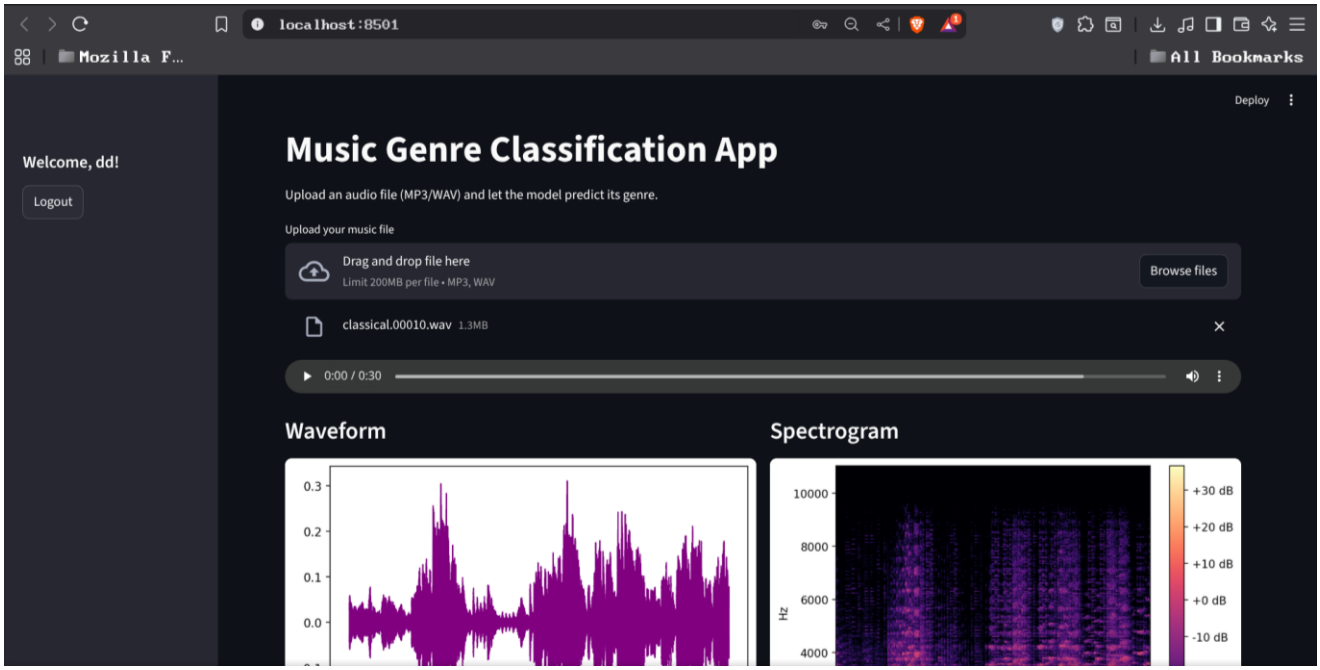
```
        elif email in users:
            st.error("An account with this email already exists.")
        elif password != confirm_password:
            st.error("Passwords do not match.")
        else:
            users[email] = {"name": name, "password": password}
            st.session_state['users'] = users
            st.success("Account created successfully! Please go to the Login page to log in.")

def main_app():
    st.sidebar.header(f"Welcome, {st.session_state['user_info']['name']}!")
    if st.sidebar.button("Logout"):
        del st.session_state['logged_in']
        del st.session_state['user_info']
        st.rerun()

    st.title("Music Genre Classification App")
    st.write("Upload an audio file (MP3/WAV) and let the model predict its genre.")

    @st.cache_data
    def load_assets():
        with open("catboost_genre_classifier.pkl", "rb") as f: model = pickle.load(f)
        with open("scaler.pkl", "rb") as f: scaler = pickle.load(f)
        with open("label_encoder.pkl", "rb") as f: label_encoder = pickle.load(f)
        return model, scaler, label_encoder

    try:
```

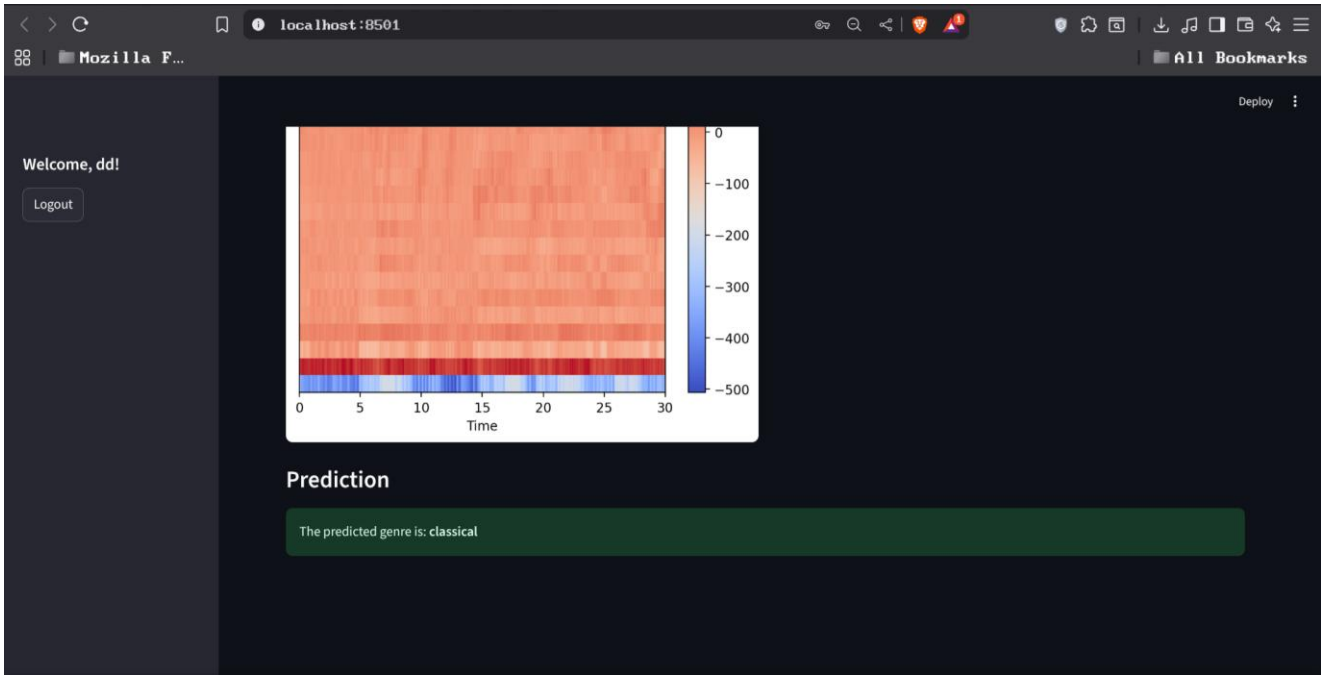
```
        model, scaler, label_encoder = load_assets()
    except FileNotFoundError:
        st.error("Model files not found. Please ensure 'catboost_genre_classifier.pkl', 'scaler.pkl', and 'label_encoder.pkl' are in the same directory.")
        return

    def extract_features(y, sr):
        features = {}
        features['length'] = len(y)/sr
        chroma = librosa.feature.chroma_stft(y=y, sr=sr)
        features['chroma_stft_mean'] = np.mean(chroma)
        features['chroma_stft_var'] = np.var(chroma)
        rms = librosa.feature.rms(y=y)
        features['rms_mean'] = np.mean(rms)
        features['rms_var'] = np.var(rms)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        features['spectral_centroid_mean'] = np.mean(spec_cent)
        features['spectral_centroid_var'] = np.var(spec_cent)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        features['spectral_bandwidth_mean'] = np.mean(spec_bw)
        features['spectral_bandwidth_var'] = np.var(spec_bw)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        features['rolloff_mean'] = np.mean(rolloff)
        features['rolloff_var'] = np.var(rolloff)
        zcr = librosa.feature.zero_crossing_rate(y)
        features['zero_crossing_rate_mean'] = np.mean(zcr)
        features['zero_crossing_rate_var'] = np.var(zcr)
        y_harmonic = librosa.effects.harmonic(y)
```

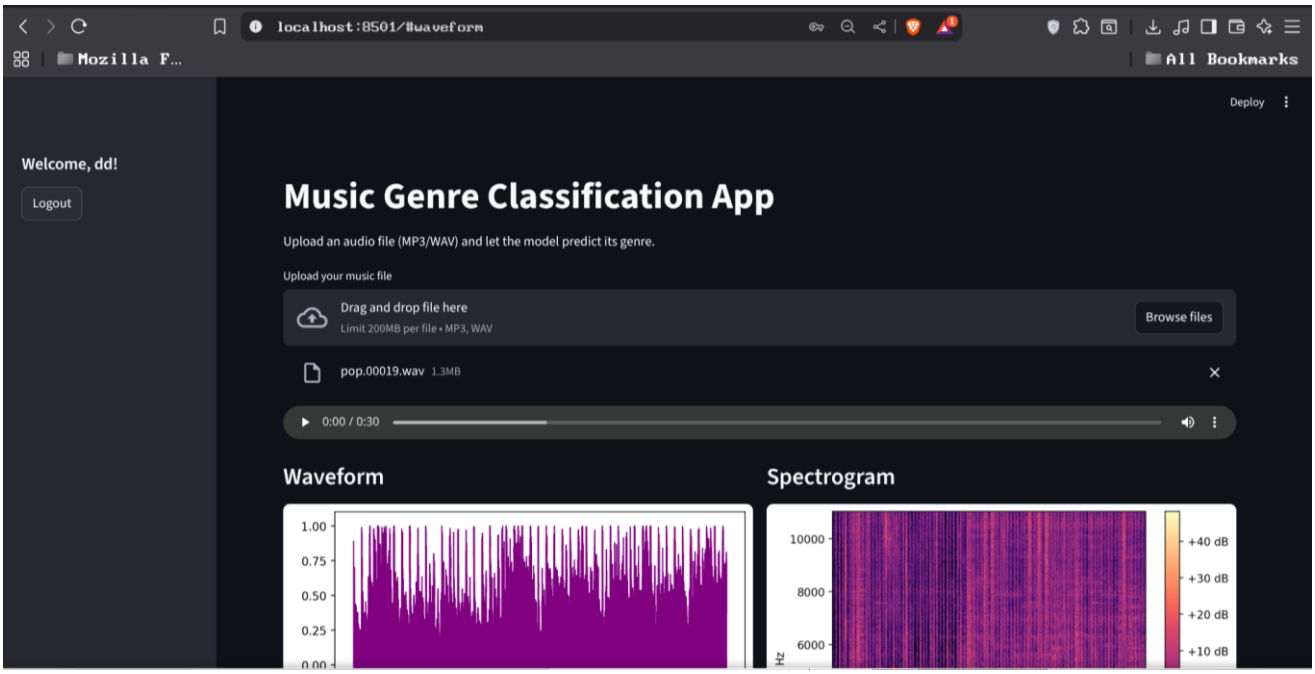
```
        features['harmony_mean'] = np.mean(y_harmonic)
        features['harmony_var'] = np.var(y_harmonic)
        X = librosa.stft(y)
        frequencies = librosa.fft_frequencies(sr=sr)
        perceptual = librosa.perceptual_weighting(np.abs(X)**2, frequencies)
        features['perceptr_mean'] = np.mean(perceptual)
        features['perceptr_var'] = np.var(perceptual)
        onset_env = librosa.onset.onset_strength(y=y, sr=sr)
        tempo, _ = librosa.beat.beat_track(onset_envelope=onset_env, sr=sr)
        features['tempo'] = tempo[0] if isinstance(tempo, np.ndarray) and tempo.size > 0
else tempo
        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)
        for i in range(20):
            features[f'mfcc{i+1}_mean'] = np.mean(mfcc[i])
            features[f'mfcc{i+1}_var'] = np.var(mfcc[i])
        return pd.DataFrame([features])

    uploaded_file = st.file_uploader("Upload your music file", type=["mp3", "wav"])

    if uploaded_file is not None:
        y, sr = librosa.load(uploaded_file, sr=None)
        st.audio(uploaded_file, format="audio/wav")

        with st.spinner("Analyzing and Visualizing..."):
            col1, col2 = st.columns(2)

            with col1:
                st.subheader("Waveform")
```

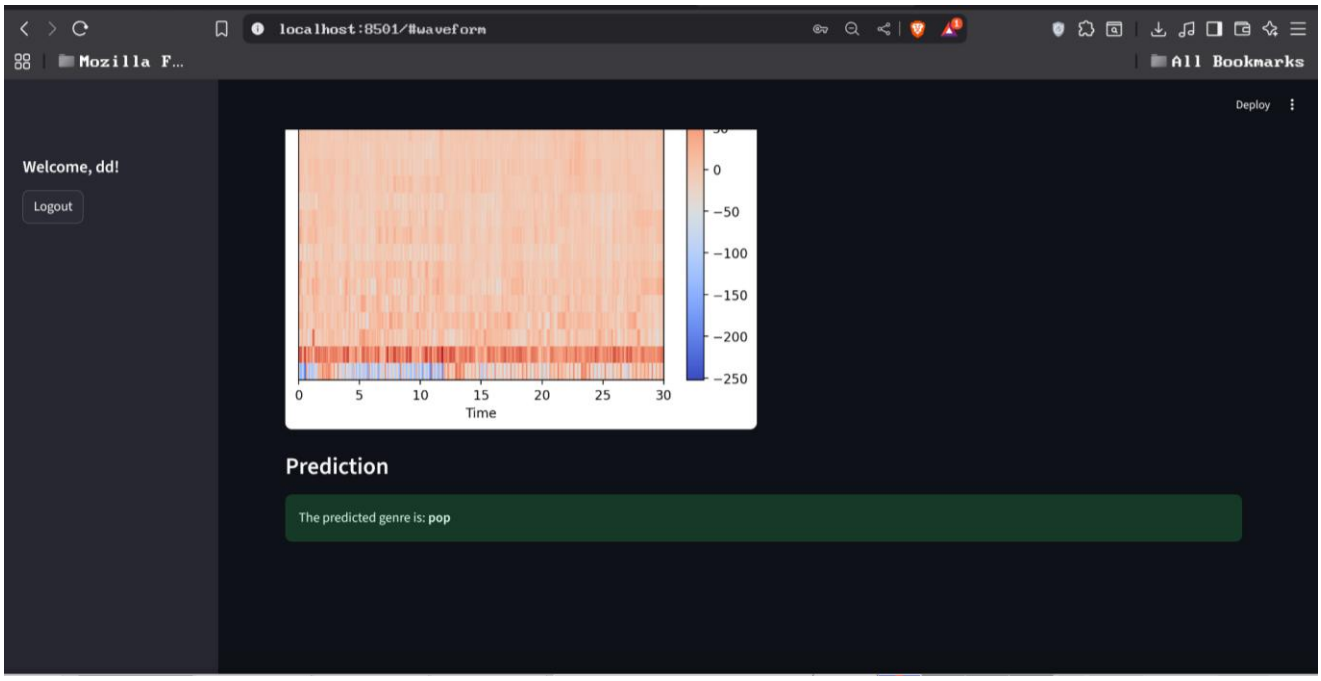
```
            fig, ax = plt.subplots()
            librosa.display.waveshow(y, sr=sr, ax=ax, color="purple")
            st.pyplot(fig)

            st.subheader("MFCCs")
            mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)
            fig, ax = plt.subplots()
            img = librosa.display.specshow(mfccs, x_axis="time", sr=sr, ax=ax,
cmap="coolwarm")
            fig.colorbar(img, ax=ax)
            st.pyplot(fig)

        with col2:
            st.subheader("Spectrogram")
            X_stft = librosa.stft(y)
            Xdb = librosa.amplitude_to_db(abs(X_stft))
            fig, ax = plt.subplots()
            img = librosa.display.specshow(Xdb, sr=sr, x_axis="time", y_axis="hz", ax=ax,
cmap="magma")
            fig.colorbar(img, ax=ax, format="%+2.0f dB")
            st.pyplot(fig)

        st.subheader("Prediction")
        with st.spinner("Extracting features and predicting genre..."):
            features_df = extract_features(y, sr)
            X_scaled = scaler.transform(features_df)
            y_pred = model.predict(X_scaled)
            predicted_genre = label_encoder.inverse_transform(y_pred)[0]
            st.success(f"The predicted genre is: **{predicted_genre}**")
```

```
    else:
        st.info("Upload a music file to get started")

if 'logged_in' not in st.session_state:
    st.session_state['logged_in'] = False

if st.session_state['logged_in']:
    main_app()
else:
    st.title("Welcome to the Music Genre Classifier")
    st.sidebar.title("Navigation")
    page = st.sidebar.radio("Choose an action", ["Login", "Sign Up"])

    if page == "Login":
        login_page()
    elif page == "Sign Up":
        signup_page()
```


localhost:8501
Mozilla F...
All Bookmarks
Deploy
Welcome, dd!
Logout
Music Genre Classification App
Upload an audio file (MP3/WAV) and let the model predict its genre.
Upload your music file
Drag and drop file here
Limit 200MB per file • MP3, WAV
Browse files
classical.00010.wav 1.3MB
0:00 / 0:30
Waveform
Spectrogram
0.3
0.2
0.1
0.0
10000
8000
6000
4000
Hz
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB


localhost:8501
Mozilla F...
All Bookmarks
Deploy
Welcome, dd!
Logout
0
−100
−200
−300
−400
−500
0
5
10
15
20
25
30
Time
Prediction
The predicted genre is: classical


localhost:8501/#waveform
Mozilla F...
All Bookmarks
Deploy
Welcome, dd!
Logout
Music Genre Classification App
Upload an audio file (MP3/WAV) and let the model predict its genre.
Upload your music file
Drag and drop file here
Limit 200MB per file • MP3, WAV
Browse files
pop.00019.wav 1.3MB
0:00 / 0:30
Waveform
Spectrogram
1.00
0.75
0.50
0.25
10000
8000
6000
Hz
+40 dB
+30 dB
+20 dB
+10 dB


localhost:8501/#waveform
Mozilla F...
All Bookmarks
Deploy
Welcome, dd!
Logout
0
−50
−100
−150
−200
−250
0
5
10
15
20
25
30
Time
Prediction
The predicted genre is: pop

ADVANTAGES

1) Overfitting has been completely removed.

2) Cross validation increased.

3) Controls introduced for overfitting.

DISADVANTAGES

1) Labels blue and rock have low recall value, when biased results in overfitting. This proves that the raw data is dirty.