**ABSTRACT**

In recent years, ensuring public safety during live events has become a critical challenge due to the increasing scale of gatherings and potential risks. This study proposes a novel approach to Live Event Detection for People’s Safety using audio data and the LightGBM classifier. The system leverages real-time audio streams to identify anomalies, such as loud disturbances, explosions, or unusual crowd behavior, which could indicate potential safety threats. Audio features are extracted using advanced signal processing techniques, including Mel-frequency cepstral coefficients (MFCCs), spectral contrast, and chroma features. These features are fed into a LightGBM classifier, which provides efficient and robust performance for real-time classification of event categories and potential risks. The proposed methodology is evaluated using diverse datasets comprising audio samples from live events, including concerts, sports, and emergency situations, to ensure a comprehensive understanding of normal and abnormal patterns. The LightGBM model demonstrates high accuracy, low latency, and scalability, making it suitable for deployment in real-time applications. Additionally, the system integrates a feedback loop for continuous model improvement based on new audio data. The results highlight the system's ability to enhance situational awareness and proactively alert authorities to potential risks, ensuring timely interventions. This approach demonstrates a significant step toward leveraging machine learning and audio analytics to improve public safety at live events.

.

**CHAPTER 1**

**INTRODUCTION**

**1.1 Overview**

Ensuring public safety at live events is a critical challenge, especially with the increasing frequency of large gatherings, such as concerts, sports events, and festivals. Live event detection involves identifying safety risks in real time, enabling authorities to intervene promptly and mitigate potential threats. Among various data types, audio signals offer a rich source of information, as they can capture crowd reactions, abnormal sounds, or potential hazards such as explosions or disturbances. Advanced machine learning methods provide a powerful way to analyze these signals and detect anomalies effectively.

In this context, LightGBM, a gradient boosting framework, stands out due to its efficiency, scalability, and ability to handle large datasets with diverse features. By leveraging real-time audio data and LightGBM, it becomes possible to develop a system that is both accurate and fast, making it suitable for dynamic environments. This approach ensures timely identification of potential threats, contributing significantly to crowd safety management during live events.

The proposed work focuses on designing a framework for real-time live event detection using audio data. By incorporating advanced feature extraction techniques and the robust classification capabilities of LightGBM, this system aims to achieve high accuracy in identifying abnormal events. This method not only addresses the limitations of existing approaches but also offers scalability for deployment across various event types and scales.

**1.2 Motivation**

The growing scale of live events has increased the complexity of ensuring public safety. Traditional human-centric approaches, such as manual monitoring and security personnel deployment, are often reactive, prone to delays, and limited in scalability. Recent incidents at live events, including stampedes and security breaches, have highlighted the need for proactive safety measures. This motivates the integration of technology-driven solutions that can complement human efforts and enhance situational awareness.

Audio data serves as a reliable indicator of abnormal events, as specific sound patterns often correlate with safety risks. For instance, sudden loud noises may indicate a disturbance, while unusual crowd murmurs could signal panic. However, analyzing this data manually or through traditional means is inefficient in real-time scenarios. This calls for an automated system that can detect such patterns quickly and accurately, improving response times and reducing human error.

LightGBM, with its speed and adaptability, offers an ideal solution for this problem. Its ability to process large-scale data with high accuracy aligns well with the demands of real-time audio analysis. The motivation behind this work is to develop a robust, automated framework that not only detects safety risks effectively but also aids authorities in taking swift action, ultimately saving lives and preventing tragedies.

**1.3 Problem Statement**

Existing human-centric approaches to live event safety management are inherently limited in their ability to detect and respond to threats in real time. Security personnel and manual monitoring systems rely heavily on human observation, which is often delayed, subjective, and prone to fatigue. In large-scale events, such as music festivals or sports gatherings, the sheer volume of attendees makes it challenging to maintain comprehensive oversight using traditional methods.

Another significant challenge is the lack of tools to analyze subtle audio cues that may indicate brewing safety concerns. Sounds such as crowd murmurs, sudden screams, or breaking objects can serve as early warning signs but are often missed by human monitors in the chaos of a live event. Current methods do not effectively leverage these audio signals, leaving a gap in situational awareness.

This work addresses these challenges by proposing an automated live event detection system using audio data and the LightGBM classifier. By replacing subjective human judgment with objective, data-driven analysis, this system aims to overcome the limitations of traditional approaches. It ensures faster and more accurate detection of anomalies, enhancing the overall effectiveness of safety management at live events.

**1.4 Applications**

The proposed live event detection system has wide-ranging applications in various domains where public safety is paramount. In large public gatherings such as concerts, sports events, and political rallies, the system can identify abnormal audio patterns indicative of disturbances, enabling security teams to respond promptly. Its ability to operate in real-time makes it a valuable tool for ensuring crowd safety and preventing large-scale incidents like stampedes or riots.

Beyond public events, the system has applications in emergency response scenarios. For instance, in disaster-stricken areas, it can analyze audio signals to detect cries for help or other distress sounds, assisting rescue teams in locating survivors. Similarly, in urban security, it can be deployed to monitor public spaces and identify sounds associated with criminal activities, such as gunshots or breaking glass, enhancing community safety.

Furthermore, the system is adaptable for use in industrial settings, where monitoring equipment and machinery noises can help detect faults or impending failures. This can prevent accidents, reduce downtime, and improve operational efficiency. The versatility of this system underscores its potential to revolutionize safety management across multiple domains, making it an indispensable tool for proactive risk mitigation.

**CHAPTER 2**

**LITERATURE SURVEY**

J.P. Bello et al. [1] examined sound analysis in smart cities, presenting their findings in the book Computational Analysis of Sound Scenes and Events. Published in 2018, their research explored various techniques for analyzing live event soundscapes, demonstrating the potential for improving city planning and noise management. They delved into methods such as machine listening and acoustic scene analysis to monitor and manage urban noise. Their study highlighted the application of these techniques in real-world scenarios, showcasing their effectiveness in identifying and mitigating noise pollution. This comprehensive analysis underscores the significance of advanced sound analysis methods in enhancing urban living environments and promoting sustainable city development.

P. Zinemanas et al. [2] developed an interpretable deep learning model for automatic sound classification. Their 2021 study in Electronics introduced a model capable of accurately classifying different sound types while providing interpretable results. The model employed a combination of convolutional neural networks (CNNs) and attention mechanisms to achieve high classification accuracy. They emphasized the importance of model interpretability, allowing users to understand the decision-making process of the AI system. This research highlights the balance between performance and interpretability in deep learning applications for sound classification, making it suitable for real-world deployment where transparency is crucial.

J.K. Das et al. [3] investigated environmental sound classification using convolutional neural networks with different integrated loss functions. Published in Expert Systems in 2021, their study demonstrated the effectiveness of CNNs in classifying a variety of environmental sounds. They experimented with various loss functions, such as cross-entropy and focal loss, to enhance model performance. Their results indicated that selecting an appropriate loss function is critical for achieving high accuracy in sound classification tasks. This research emphasizes the role of loss function selection in enhancing model performance, providing valuable insights for optimizing CNN-based sound classifiers.

J.K. Das et al. [4] presented a method combining convolutional neural networks and long short-term memory networks for live event sound classification. Their study, presented at the 2020 ICDS conference, showed that the hybrid model improved classification accuracy by leveraging both spatial and temporal features. The CNN component extracted spatial features from sound spectrograms, while the LSTM component captured temporal dependencies. This approach highlights the benefit of integrating multiple deep learning techniques for complex audio classification tasks, offering a robust solution for live event sound classification with improved accuracy and reliability.

Z. Mushtaq and S.F. Su [5] explored efficient classification of environmental sounds through multiple features aggregation and data enhancement techniques. Published in Symmetry in 2020, their study focused on enhancing spectrogram images to improve classification accuracy. They aggregated features from different spectrogram representations and applied data augmentation techniques to create a more robust training dataset. Their approach significantly improved the performance of sound classification models, demonstrating the importance of feature engineering and data augmentation in sound classification tasks. This research illustrates how combining multiple features and enhancing data can lead to more accurate and reliable sound classifiers.

W. Mu et al. [6] developed a temporal-frequency attention-based convolutional neural network for environmental sound classification. Published in Scientific Reports in 2021, their model utilized attention mechanisms to focus on important sound features, improving classification performance. The attention module allowed the network to selectively emphasize relevant temporal and frequency components of the sound signals. This study underscores the value of attention mechanisms in enhancing neural network models for audio tasks, showing that such mechanisms can significantly boost the accuracy of sound classification models by focusing on critical sound patterns.

T. Giannakopoulos et al. [7] investigated the recognition of live event sound events using deep context-aware feature extractors and handcrafted features. Presented at the AIAI conference in 2019, their study demonstrated that combining deep learning with traditional feature extraction techniques could improve sound event recognition. They integrated context-aware features extracted by deep learning models with handcrafted features such as Mel-frequency cepstral coefficients (MFCCs). This research highlights the complementary strengths of deep learning and handcrafted features, showing that a hybrid approach can enhance the performance of sound event recognition systems by leveraging the advantages of both methods.

J.S. Luz et al. [8] proposed an ensemble of handcrafted and deep features for live event sound classification. Published in Applied Acoustics in 2021, their approach combined multiple feature types to achieve high classification accuracy. They used handcrafted features like MFCCs and chroma features alongside deep learning-based features extracted from CNNs. The ensemble method outperformed models using either feature type alone, demonstrating the effectiveness of combining diverse features for sound classification. This study illustrates the effectiveness of ensemble methods in complex classification tasks, providing a robust framework for live event sound classification with improved accuracy and generalization.

Y. Gong et al. [9] introduced the Audio Spectrogram Transformer (AST) for sound classification. Their 2021 study on arXiv presented a transformer-based model that significantly improved classification performance on various audio datasets. The AST model utilized the transformer architecture's self-attention mechanisms to capture long-range dependencies in audio signals. Their results showed that the AST model outperformed traditional CNN and RNN-based models, highlighting the potential of transformer models in advancing sound classification technologies. This research demonstrates the effectiveness of transformer architectures in capturing complex patterns in audio data, leading to improved classification accuracy.

İ. Türker and S. Aksu [10] developed Connectogram, a graph-based time-dependent representation for sounds. Published in Applied Acoustics in 2022, their method provided a novel way to represent and analyze audio signals, improving classification accuracy. The Connectogram method transformed audio signals into graph representations, capturing the temporal and frequency relationships between different sound components. Their approach showed significant improvements in classification performance compared to traditional spectrogram-based methods. This study underscores the importance of innovative representations in audio processing tasks, offering a new perspective on how to effectively represent and analyze sound data for classification.

Q. Kong et al. [11] investigated sound event detection using CNN-Transformer models and automatic threshold optimization. Published in IEEE/ACM Transactions on Audio, Speech, and Language Processing in 2020, their approach improved detection accuracy on weakly labeled datasets. They combined the strengths of CNNs for feature extraction and transformers for capturing long-range dependencies. Additionally, they implemented an automatic threshold optimization technique to fine-tune the detection thresholds. This research highlights the potential of combining CNNs and transformers for audio event detection, providing a robust framework for improving detection accuracy in challenging weakly labeled scenarios.

P. Gimeno et al. [12] developed a multiclass audio segmentation method based on recurrent neural networks for broadcast domain data. Published in EURASIP Journal on Audio, Speech, and Music Processing in 2020, their study demonstrated that RNNs could effectively segment audio streams into various classes. They leveraged the temporal modeling capabilities of RNNs to segment continuous audio data accurately. Their method showed significant improvements over traditional segmentation techniques, illustrating the role of RNNs in audio segmentation tasks. This research provides valuable insights into how RNNs can be utilized for precise and efficient audio segmentation in broadcast and other applications.

Z. Zhang et al. [13] and [14] explored learning attentive representations for environmental sound classification. Published in IEEE Access in 2019 and Neurocomputing in 2020, their studies focused on using attention mechanisms to enhance the performance of convolutional recurrent neural networks. They integrated attention layers into their models to highlight important sound features, improving classification accuracy. Their research demonstrated that attention-based models could effectively capture the salient aspects of sound signals, leading to better performance in complex audio classification tasks. These works underscore the benefits of attention-based models in complex audio classification tasks, providing a pathway for developing more accurate and interpretable sound classifiers.

T. Qiao et al. [15] investigated high accurate environmental sound classification using sub-spectrogram segmentation and temporal-frequency attention mechanisms. Published in Sensors in 2021, their study highlighted the effectiveness of combining segmentation and attention techniques for improving classification accuracy. They segmented the spectrograms into sub-components and applied attention mechanisms to focus on relevant temporal-frequency regions. This approach resulted in significant improvements in classification performance, demonstrating the potential of advanced techniques in refining sound classification models. This research illustrates how innovative segmentation and attention methods can enhance the accuracy and reliability of environmental sound classifiers.

**CHAPTER 3**

**EXISTING SYSTEM**

**3.1 Overview**

**1. Live event sound Classification and Analysis**

Live event sound classification and analysis play a critical role in monitoring and managing noise pollution in city environments. Effective noise management requires accurate identification and classification of various live event sounds, such as traffic noise, construction sounds, and human activities. Traditionally, the classification of live event sounds has relied on manual methods, which involve the collection and analysis of sound recordings by human experts.

**Sound Collection and Analysis Methods**

**Manual Sound Collection:** The process begins with the collection of sound recordings from various urban locations. These recordings are typically captured using portable audio recording devices or stationary sound level meters positioned at key points in the city. The collected audio data is then transferred to a central database for analysis.

**Human Expertise in Sound Analysis:** Traditionally, sound recordings are analyzed manually by human experts who listen to the audio files and identify different types of sounds based on their auditory characteristics. This process involves distinguishing between various sound sources, such as vehicular traffic, construction activities, public events, and natural sounds. Human experts rely on their training and experience to accurately classify sounds and assess their impact on urban environments.

**Limitations of Manual Analysis:** While human expertise is valuable, the manual analysis of live event sounds is time-consuming, labor-intensive, and subject to variability. The accuracy of sound classification can vary depending on the skill and experience of the individual performing the analysis. Additionally, the sheer volume of audio data generated in urban environments makes it challenging to keep up with the demand for timely and accurate sound classification.

**Basic Statistical Analysis:** To support manual analysis, basic statistical techniques are often employed. This includes calculating sound levels, such as the equivalent continuous sound level (Leq) and the maximum sound level (Lmax), to quantify the intensity of urban noise. These metrics provide a general overview of noise levels but do not offer detailed insights into the specific types of sounds present.

**Challenges in Traditional Sound Classification**

**Variability in Sound Characteristics:** One of the major challenges in live event sound classification is the variability in sound characteristics. Different sound sources have distinct acoustic signatures, which can overlap and create complex soundscapes. For example, traffic noise may include sounds from engines, horns, and tire friction, making it difficult to isolate and identify individual components. This variability increases the complexity of sound classification and can lead to misidentification or incomplete analysis.

**Subjectivity and Inconsistency:** The reliance on human experts introduces a level of subjectivity and inconsistency in sound classification. Different experts may interpret and classify sounds differently based on their personal judgment and experience. This subjectivity can result in variability in the classification outcomes and limit the reproducibility of the analysis. Additionally, the manual process is prone to human error, further affecting the reliability of the results.

**Limited Coverage and Scalability:** Manual sound classification methods are limited in their coverage and scalability. The need for human involvement restricts the ability to analyze large-scale audio datasets in real time. As urban environments generate continuous streams of sound data, the manual approach struggles to keep pace with the volume and frequency of sound events. This limitation hinders the ability to monitor and respond to noise pollution effectively.

**Resource-Intensive Process:** The manual analysis of live event sounds is resource-intensive, requiring significant time, effort, and expertise. Human experts must listen to and analyze numerous audio recordings, which is a laborious and time-consuming task. This resource-intensive process can strain the available workforce and result in delays in sound classification and reporting.

**Imaging and Computational Techniques**

**Acoustic Imaging:** In cases where traditional methods fall short, advanced acoustic imaging techniques are employed. These techniques use arrays of microphones to create visual representations of sound sources, allowing for a more precise localization and identification of sound events. Acoustic imaging is particularly useful in complex urban environments where multiple sound sources overlap.

**Machine Learning and Automated Systems:** To address the limitations of manual analysis, machine learning algorithms and automated systems are increasingly being utilized for live event sound classification. These approaches leverage computational power and data-driven techniques to analyze audio data and classify sounds with higher accuracy and efficiency. Machine learning models are trained on labeled datasets to recognize patterns and features in audio signals, enabling automated sound classification without human intervention.

**Integration with Sensor Networks:** Modern live event sound monitoring systems integrate sensor networks to capture real-time audio data from multiple locations. These sensor networks consist of distributed microphones and recording devices that continuously collect sound data and transmit it to centralized servers for analysis. The integration of sensor networks with automated classification systems enhances the coverage and scalability of live event sound monitoring, providing comprehensive and up-to-date insights into live event soundscapes.

**3.2 Challenges in the Traditional Sound Classification Process**

**Data Management and Processing:** The traditional approach to live event sound classification faces several challenges related to data management and processing. The large volume of audio data generated in urban environments requires efficient storage, retrieval, and processing capabilities. Managing and processing this data in real time is a significant challenge that affects the timeliness and accuracy of sound classification.

**Variability in Sound Quality: The** quality of audio recordings can vary due to factors such as environmental conditions, recording equipment, and background noise. Poor quality recordings can hinder the accurate classification of sounds and introduce noise into the analysis. Ensuring consistent and high-quality audio data is essential for reliable sound classification.

**Need for Real-time Analysis:** Traditional methods often fall short in providing real-time analysis of live event sounds. The manual process is inherently slow and cannot keep up with the continuous flow of audio data. Real-time analysis is crucial for timely decision-making and intervention in noise pollution management. The lack of real-time capabilities limits the effectiveness of traditional sound classification methods

**Resource and Cost Constraints:** The resource-intensive nature of manual sound analysis poses cost and resource constraints for live event sound monitoring initiatives. The need for trained human experts, specialized equipment, and infrastructure can be financially burdensome, especially for resource-limited settings. Cost constraints may limit the deployment and scalability of traditional sound classification systems.

**3.3 Limitations of Traditional Approaches**

**Subjectivity and Inconsistency:** The heavy reliance on human expertise introduces subjectivity and inconsistency in sound classification. Different experts may have varying interpretations of audio data, leading to inconsistent classification outcomes. This subjectivity affects the reliability and reproducibility of the analysis, making it challenging to establish standardized and objective sound classification criteria.

**Time-Consuming and Resource-Intensive**: The traditional process of manual sound analysis is time-consuming and resource-intensive. The need for human involvement in listening to and analyzing audio recordings requires significant time and effort. This resource-intensive process can lead to delays in sound classification and reporting, limiting the ability to respond to noise pollution in a timely manner.

**Limited Predictive Power:** Traditional sound classification methods have limited predictive power when used in isolation. Basic statistical techniques and human judgment may provide general insights into sound characteristics but lack the ability to predict and identify specific sound events accurately. The absence of advanced predictive tools limits the effectiveness of traditional sound classification in addressing complex live event soundscapes.

**Technological Advancements Needed:** The limitations of traditional sound classification approaches highlight the need for technological advancements. Emerging technologies such as machine learning, acoustic imaging, and sensor networks offer the potential to overcome these limitations and enhance the accuracy, efficiency, and scalability of live event sound classification. Embracing these advancements is essential for modernizing sound monitoring and management practices.

**Ethical and Legal Considerations:** The potential for errors in traditional sound classification raises ethical and legal concerns. Misclassification or delayed identification of noise pollution sources can lead to negative consequences for urban residents and public health. Addressing these ethical and legal considerations requires the development of robust and reliable sound classification systems that minimize the risk of errors and ensure accurate monitoring and intervention.

**Future Directions:** The future of live event sound classification lies in the integration of advanced technologies and data-driven approaches. Leveraging machine learning algorithms, automated systems, and real-time data processing capabilities can revolutionize sound monitoring and management. Developing comprehensive datasets and training models on diverse live event soundscapes will enhance the accuracy and applicability of sound classification systems, contributing to more effective noise pollution control and improved urban living environments.

**CHAPTER 4**

**PROPOSED SYSTEM**

**4.1 Overview:**

**Step 1: Dataset**

The research begins with the collection of a comprehensive live event sound dataset, organized into distinct categories representing various live event sound types, such as traffic, sirens, and human chatter. Each category consists of multiple audio files in WAV format, providing a diverse range of sounds to analyze. The dataset is stored in a directory structure that allows easy access and management of audio files.

**Step 2: Dataset Preprocessing**

The preprocessing phase involves several key steps to prepare the audio data for analysis. First, any null values in the dataset are checked and removed to ensure the integrity of the data. Then, the audio files are processed to remove background noise, enhancing the quality of the recordings. This step is crucial for obtaining clearer features from the audio, which will aid in the classification task. Additionally, features are extracted using Mel-frequency cepstral coefficients (MFCCs), which serve as a representation of the audio signal's characteristics.

**Step 3: Label Encoding**

To facilitate machine learning, the categorical labels associated with each sound file are transformed into numerical values through label encoding. This process involves mapping each category to a unique integer, allowing the algorithms to interpret the labels numerically. This step is essential for effectively training classification models, as machine learning algorithms require numerical input.

**Step 4: Data Splitting**

The dataset is then split into training and testing sets using a stratified approach to maintain the distribution of categories across both sets. This ensures that the model is trained and validated on representative samples, thereby enhancing its generalizability. The training set is used for model training, while the testing set is reserved for performance evaluation.

**Step 5: Existing Algorithm**

The existing algorithm utilized in this project is the Multi-Layer Perceptron (MLP) Classifier. MLP is a type of neural network that consists of multiple layers of neurons, including input, hidden, and output layers. It works by passing input data through these layers, where each neuron applies a weighted sum and an activation function to determine its output. While MLPs can model complex relationships in data, they may suffer from issues like overfitting and require careful tuning of hyperparameters.

**Step 6: Proposed Algorithm**

In contrast, the proposed algorithm is the LightGBM (LGBM) Classifier. LGBM is a gradient boosting framework that uses tree-based learning algorithms to build models. It operates by constructing multiple decision trees sequentially, where each tree corrects the errors of its predecessor. This approach significantly improves training speed and reduces memory consumption compared to traditional boosting methods. LGBM's architecture leverages a histogram-based algorithm, which efficiently handles large datasets and high-dimensional data.

**Step 7: Performance Comparison**

The performance of both algorithms is evaluated using several metrics, including accuracy, precision, recall, and F1-score. A confusion matrix is generated to visualize the classification results, providing insights into where the model performs well and where it may struggle. The results are compared to determine which algorithm better classifies live event sounds and to assess the improvements made by implementing LGBM over MLP.

**Step 8: Prediction of Output from Test Data**

Finally, the trained model is used to make predictions on new test data. The audio files are preprocessed in the same manner as the training data, ensuring consistency in feature extraction. The model's prediction capability is demonstrated using an example audio file, where the predicted category is printed out. This step showcases the practical application of the trained model in real-world scenarios, highlighting its utility for live event sound classification.

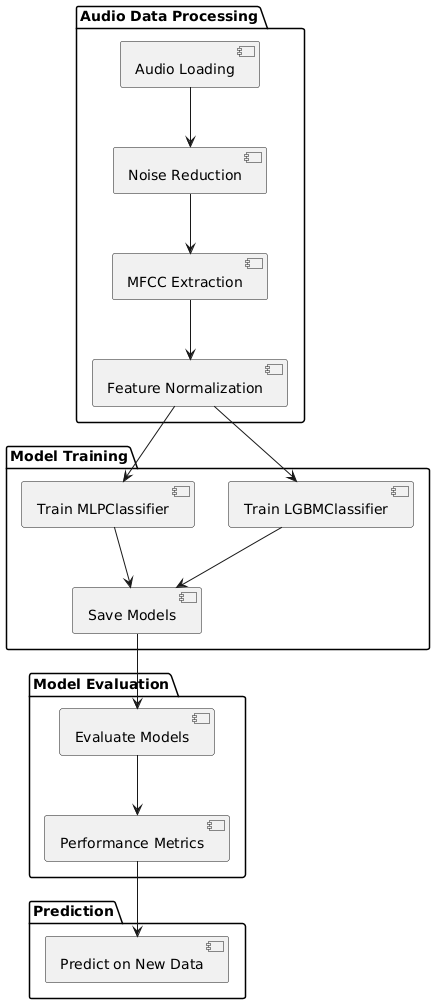


Figure 1: Architectural Block Diagram

**4.2 Data Splitting & Preprocessing**

Data splitting and preprocessing are crucial steps in preparing the dataset for machine learning. After preprocessing the audio files, the dataset is divided into training and testing subsets. This is achieved using a stratified sampling method, ensuring that each category is represented proportionally in both subsets. The training data is employed for building the model, while the testing data is reserved for evaluating model performance. This systematic approach minimizes bias and enhances the model's ability to generalize to unseen data.

**2. Exploring the Data**

- The shape and basic information of the dataset are examined, including the number of records and the types of data present.

- The dataset is checked for missing values to understand the extent and type of data issues that need to be addressed.

**3. Handling Missing Values**

**- Categorical Columns:** For columns with categorical data, missing values are filled with a placeholder value like 'Unknown'. This ensures that all categorical data is available for encoding.

**- Numerical Columns:** Any missing numerical data is replaced with 0. This is a simple strategy to handle missing values in numerical features.

**4. Label Encoding**

**- Convert Categorical Data:** Categorical variables (textual data) are transformed into numerical values using label encoding. This process assigns a unique integer to each category, allowing machine learning algorithms to process the data effectively.

**- Fit and Transform:** Label encoding is applied to each categorical column, converting textual labels into numerical form. Each categorical column is processed individually to ensure that all possible categories are properly encoded.

**5. Feature and Target Separation**

- Separate Features and Target: The dataset is divided into features (input variables) and the target variable (the outcome we want to predict). Features are the columns used for making predictions, while the target is the column that contains the actual diagnosis.

**6. Train-Test Split**

- Split Data: The data is divided into training and testing sets. This split ensures that the model is trained on one subset of the data and evaluated on another, allowing for unbiased performance assessment.

**7. Resampling**

- Balance Classes: In some cases, the dataset may be resampled to balance the distribution of classes. This process involves creating a balanced dataset where each class has an equal number of samples, helping to improve model performance by addressing class imbalance.

**4.3 ML Model Building**

The ML model building process involves several stages. First, feature extraction is performed on the preprocessed audio data to generate MFCCs, which capture essential audio characteristics. Next, the data is standardized using techniques such as normalization to ensure uniformity across features. Once the features are prepared, the MLPClassifier is instantiated, followed by training on the training set. After training, the model is validated using the testing set to assess its accuracy. Hyperparameter tuning may be applied to optimize performance. Finally, the model's predictions are analyzed to evaluate its effectiveness.

**4.3.1 Existing Algorithm: MLP Classifier**

The Multi-Layer Perceptron (MLP) is a supervised learning algorithm used for classification and regression tasks. It consists of an input layer, one or more hidden layers, and an output layer. Each layer is composed of interconnected neurons, where each connection has an associated weight. MLPs operate using a forward propagation mechanism, where input data is fed through the network, and an activation function determines the output of each neuron. The architecture allows for the modeling of complex relationships in data, making MLP suitable for various applications, including live event sound classification. However, MLPs can be prone to overfitting, especially with small datasets, and require extensive tuning of hyperparameters such as learning rate, number of hidden layers, and activation functions.

**4.3.2 Proposed Algorithm: LightGBM**

LightGBM (LGBM) is an advanced gradient boosting framework that excels in handling large datasets and high-dimensional data. It operates by constructing decision trees in a sequential manner, where each new tree attempts to correct the errors made by the previous ones. LGBM utilizes a histogram-based approach, which significantly accelerates the training process by reducing the number of data points processed during tree construction. This algorithm is designed for efficiency and scalability, making it particularly advantageous for complex tasks like live event sound classification. The advantages of LGBM include faster training times, reduced memory usage, and superior performance in terms of accuracy and speed compared to traditional boosting algorithms. Its ability to handle categorical features directly and its flexibility in tuning hyperparameters further enhance its applicability in real-world scenarios.

**CHAPTER 5**

**UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:** The Primary goals in the design of the UML are as follows:

* Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
* Provide extendibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development process.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of OO tools market.
* Support higher level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**Class diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram was capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

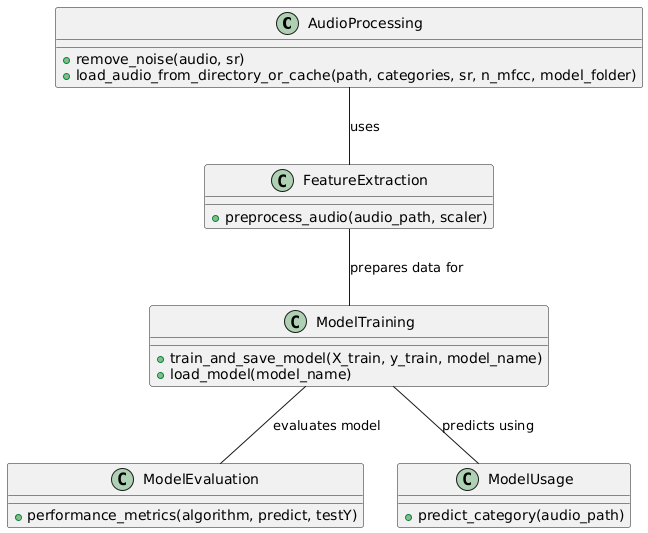


Figure-5.1: Class Diagram

**Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines (“lifelines”), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

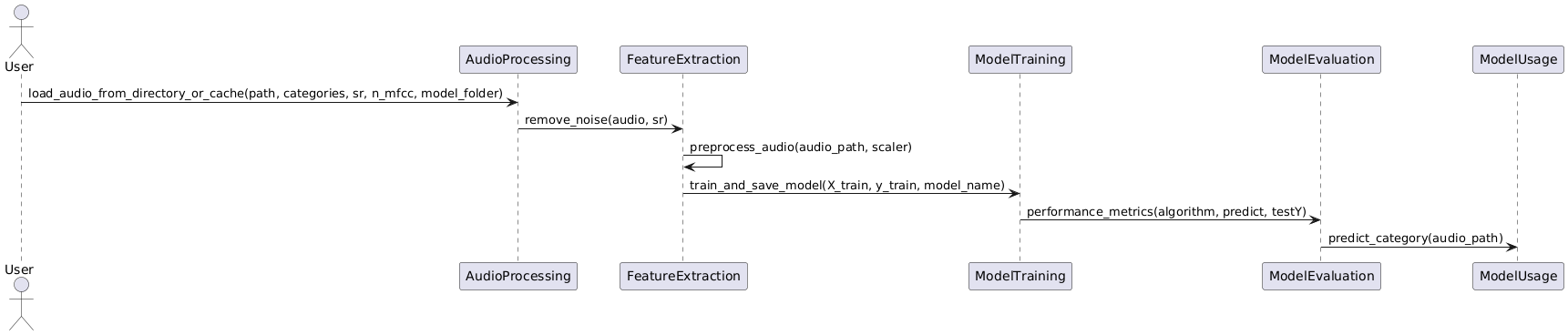


Figure-5.2: Sequence Diagram

**Activity diagram**

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

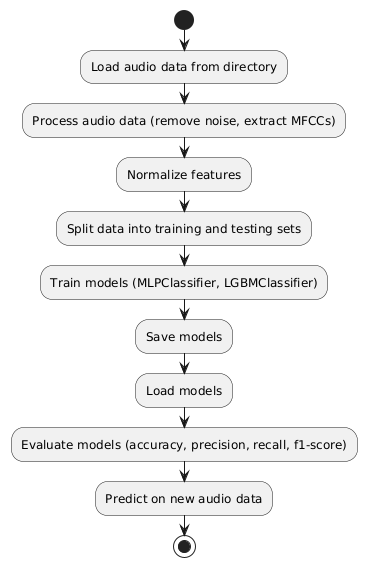


Figure-5.3: Activity Diagram

**Data flow diagram**

A data flow diagram (DFD) is a graphical representation of how data moves within an information system. It is a modeling technique used in system analysis and design to illustrate the flow of data between various processes, data stores, data sources, and data destinations within a system or between systems. Data flow diagrams are often used to depict the structure and behavior of a system, emphasizing the flow of data and the transformations it undergoes as it moves through the system.

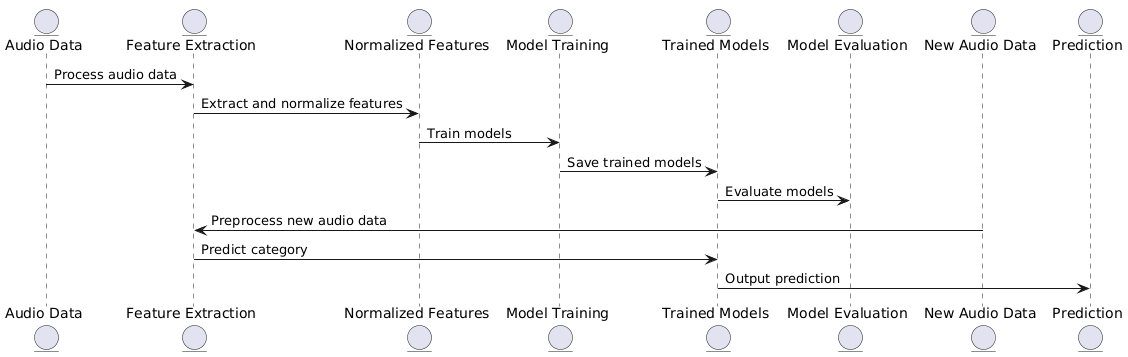


Figure-5.4: Dataflow Diagram

**Component diagram:** Component diagram describes the organization and wiring of the physical components in a system.

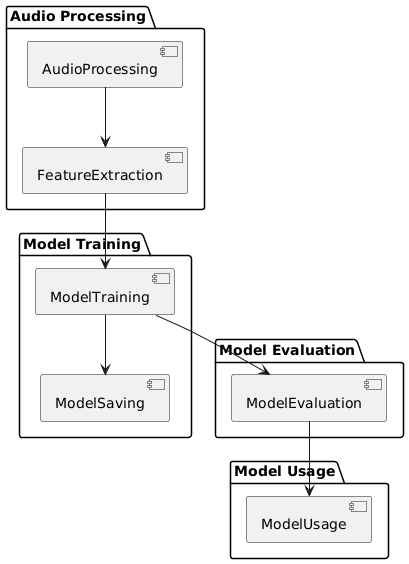


Figure-5.5: Component Diagram

**Use Case diagram:** A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

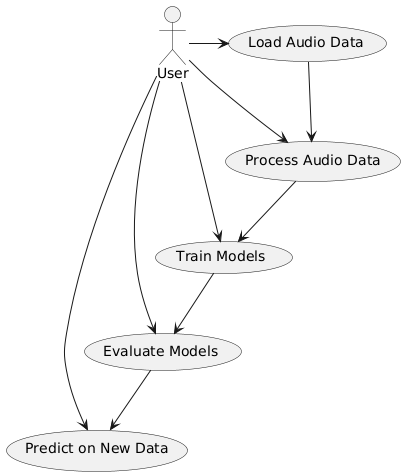


Figure-5.6: use case diagram

**Deployment Diagram:**

A deployment diagram in UML illustrates the physical arrangement of hardware and software components in the system. It visualizes how different software artifacts, such as data processing scripts and model training components, are deployed across hardware nodes and interact with each other, providing insight into the system’s infrastructure and deployment strategy.

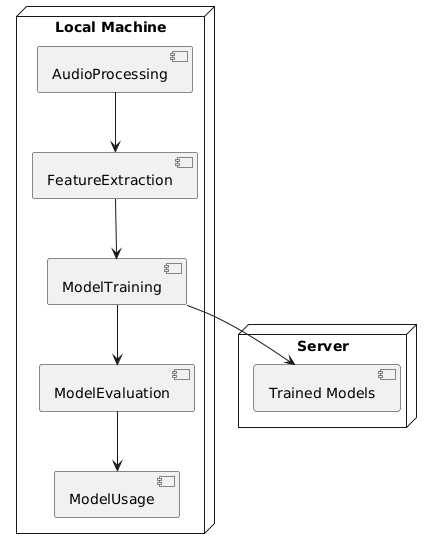


Figure-5.7: DeploymentDiagram

**CHAPTER 6**

**SOFTWARE ENVIRONMENT**

**What is Python?**

Below are some facts about Python.

* Python is currently the most widely used multi-purpose, high-level programming language.
* Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.
* Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
* Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard libraries which can be used for the following –

* 1. Machine Learning
  2. GUI Applications (like Kivy, Tkinter, PyQt etc.)
  3. Web frameworks like Django (used by YouTube, Instagram, Dropbox)
  4. Image processing (like Opencv, Pillow)
  5. Web scraping (like Scrapy, BeautifulSoup, Selenium)
  6. Test frameworks
  7. Multimedia

**Advantages of Python**

Let’s see how Python dominates over other languages.

1. Extensive Libraries

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

1. Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

1. Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

1. Improved Productivity

The language’s simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

1. IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet of Things. This is a way to connect the language with the real world.

1. Simple and Easy

When working with Java, you may have to create a class to print ‘Hello World’. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

1. Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. These further aids the readability of the code.

1. Object-Oriented

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

1. Free and Open-Source

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

1. Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn’t the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

1. Interpreted

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

**Advantages of Python Over Other Languages**

1. **Less Coding**

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don’t have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

1. **Affordable**

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

1. **Python is for Everyone**

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

**Disadvantages of Python**

So far, we’ve seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let’s now see the downsides of choosing Python over another language.

**1. Speed Limitations**

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn’t a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

**2. Weak in Mobile Computing and Browsers**

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

The reason it is not so famous despite the existence of Brython is that it isn’t that secure.

**3. Design Restrictions**

As you know, Python is dynamically-typed. This means that you don’t need to declare the type of variable while writing the code. It uses duck-typing. But wait, what’s that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

**4. Underdeveloped Database Access Layers**

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python’s database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

**5. Simple**

No, we’re not kidding. Python’s simplicity can indeed be a problem. Take my example. I don’t do Java, I’m more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

**History of Python**

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde &Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners1, Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So, I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

**Python Development Steps**

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991. This release included already exception handling, functions, and the core data types of lists, dict, str and others. It was also object oriented and had a module system.  
Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting unicode. Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only one -- obvious way to do it."Some changes in Python 7.3:

* Print is now a function.
* Views and iterators instead of lists
* The rules for ordering comparisons have been simplified. E.g., a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.
* There is only one integer type left, i.e., int. long is int as well.
* The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.
* Text Vs. Data Instead of Unicode Vs. 8-bit

**Purpose**

We demonstrated that our approach enables successful segmentation of intra-retinal layers—even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout—with the assistance of the ANIS feature.

**Python**

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

* Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

**Modules Used in Project**

**TensorFlow**

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.‍

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

**NumPy**

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

* A powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* Tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

**Pandas**

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

**Matplotlib**

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users.

**Scikit – learn**

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

* Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

**Install Python Step-by-Step in Windows and Mac**

Python a versatile programming language doesn’t come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

**How to Install Python on Windows and Mac**

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e., operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So, the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

**Download the Correct version into the system**

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://www.python.org

A screenshot of a computer

Description automatically generated with medium confidence

Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.

Graphical user interface, application

Description automatically generated

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Graphical user interface, application

Description automatically generated

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

Graphical user interface, text

Description automatically generated

* To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
* To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e., Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.

Graphical user interface, text, application

Description automatically generated

Step 2: Before you click on Install Now, make sure to put a tick on Add Python 3.7 to PATH.

Graphical user interface, text, application, chat or text message

Description automatically generated

Step 3: Click on Install NOW After the installation is successful. Click on Close.

Graphical user interface, text, application, chat or text message

Description automatically generated

With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

Verify the Python Installation

Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.

Graphical user interface, application

Description automatically generated

Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.

A screenshot of a computer

Description automatically generated with medium confidence

Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works

Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.

Application

Description automatically generated with low confidence

Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save

Graphical user interface, text, application, email

Description automatically generated

Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g., enter print (“Hey World”) and Press Enter.

Graphical user interface, text, application, email

Description automatically generated

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won’t work.

**CHAPTER 7**

**SOURCE CODE**

# A Comprehensive Dataset for Live event sound Classification and Analysis

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.utils import resample

import librosa

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.decomposition import PCA

import IPython

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import precision\_score,accuracy\_score,f1\_score,recall\_score

from sklearn.metrics import classification\_report, confusion\_matrix

import warnings

warnings.filterwarnings('ignore')

from sklearn.neural\_network import MLPClassifier

from lightgbm import LGBMClassifier

import joblib

from imblearn.over\_sampling import SMOTE

path = r"sorted\_audio\_data"

model\_folder = "model"

categories = [d for d in os.listdir(path) if os.path.isdir(os.path.join(path, d))]

categories

def remove\_noise(audio, sr):

noise\_profile = audio[:int(0.5 sr)]

noise\_reduced\_audio = librosa.effects.remix(audio, intervals=librosa.effects.split(audio, top\_db=20))

return noise\_reduced\_audio

def load\_audio\_from\_directory\_or\_cache(path, categories, sr, n\_mfcc, model\_folder):

X\_file = os.path.join(model\_folder, "X.npy")

Y\_file = os.path.join(model\_folder, "Y.npy")

if os.path.exists(X\_file) and os.path.exists(Y\_file):

print(f"Loading cached data from {model\_folder}")

X = np.load(X\_file)

Y = np.load(Y\_file)

else:

print(f"Path does not exist: {path}" if not os.path.exists(path) else "Processing directory")

X = []

Y = []

if not os.path.exists(path):

print(f"Path does not exist: {path}")

return np.array(X), np.array(Y)

for root, dirs, files in os.walk(path):

print(f"Processing root: {root}")

for file in files:

name = os.path.basename(root)

if file.endswith('.wav'):

file\_path = os.path.join(root, file)

try:

y, sr = librosa.load(file\_path, sr=sr)

y = remove\_noise(y, sr)

mfccs = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=30)

mfccs\_mean = np.mean(mfccs.T, axis=0) # Taking the mean of the MFCCs

X.append(mfccs\_mean)

if name in categories:

Y.append(categories.index(name))

else:

print(f"Category {name} not in categories list.")

except Exception as e:

print(f"Skipping {file\_path}, error reading file: {e}")

X = np.array(X)

Y = np.array(Y)

os.makedirs(model\_folder, exist\_ok=True) # Create the directory if it doesn't exist

np.save(X\_file, X)

np.save(Y\_file, Y)

return X, Y

# Load audio

X, Y = load\_audio\_from\_directory\_or\_cache(path, categories, 22050, 13, 'model')

# Debug print statements to check the shape of X and Y

print(f"Shape of X before any reshaping: {X.shape}")

print(f"Shape of Y: {Y.shape}")

# Check the shape of X and ensure it is 2D

if len(X.shape) != 2:

raise ValueError(f"X should be a 2D array but got shape {X.shape}")

# Normalize features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Save arrays

print(f"Processed and saved {len(X)} audio files.")

X\_list = X.tolist()

Y\_list = Y.tolist()

# Create DataFrame

df = pd.DataFrame({'X': X\_list, 'Y': Y\_list})

# Plot the counts of each category

category\_counts = {category: len(os.listdir(os.path.join(path, category))) for category in categories}

df\_counts = pd.DataFrame(list(category\_counts.items()), columns=['Category', 'Count'])

plt.figure(figsize=(10, 6))

plt.bar(df\_counts['Category'], df\_counts['Count'], color='skyblue')

plt.xlabel('Category')

plt.ylabel('Count')

plt.title('Number of Sounds per Category')

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.show()

X\_new = df['X'].tolist()

Y\_new = df['Y'].tolist()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_new, Y\_new, test\_size = 0.20, random\_state = 44)

precision = []

recall = []

fscore = []

accuracy = []

labels = categories

def performance\_metrics(algorithm, predict, testY):

#estY = testY.astype('int')

#redict = predict.astype('int')

p = precision\_score(testY, predict,average='macro') \* 100

r = recall\_score(testY, predict,average='macro') \* 100

f = f1\_score(testY, predict,average='macro') \* 100

a = accuracy\_score(testY,predict)\*100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

print(algorithm+' Accuracy : '+str(a))

print(algorithm+' Precision : '+str(p))

print(algorithm+' Recall : '+str(r))

print(algorithm+' FSCORE : '+str(f))

report=classification\_report(predict, testY,target\_names=labels)

print('\n',algorithm+" classification report\n",report)

conf\_matrix = confusion\_matrix(testY, predict)

plt.figure(figsize =(5, 5))

ax = sns.heatmap(conf\_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="Blues" ,fmt ="g");

ax.set\_ylim([0,len(labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

# MLPClassifier

path = 'model/MLPClassifier'

if(os.path.exists(path)):

mlpc = joblib.load(path)

else:

mlpc = MLPClassifier()

y\_pred = mlpc.predict(X\_test)

performance\_metrics('MLPClassifier', y\_pred, y\_test)

df\_resampled = resample(df, replace=True, n\_samples=4000, random\_state=42)

df\_resampled.head()

X\_new = df\_resampled['X'].tolist()

Y\_new = df\_resampled['Y'].tolist()

# Create SMOTE object and fit-transform the data

smote = SMOTE(random\_state=42)

X\_resampled, Y\_resampled = smote.fit\_resample(X\_new, Y\_new)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_resampled, Y\_resampled, test\_size = 0.20, random\_state = 44)

precision = []

recall = []

fscore = []

accuracy = []

labels = categories

def performance\_metrics(algorithm, predict, testY):

#estY = testY.astype('int')

#redict = predict.astype('int')

p = precision\_score(testY, predict,average='macro') \* 100

r = recall\_score(testY, predict,average='macro') \* 100

f = f1\_score(testY, predict,average='macro') \* 100

a = accuracy\_score(testY,predict)\*100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

print(algorithm+' Accuracy : '+str(a))

print(algorithm+' Precision : '+str(p))

print(algorithm+' Recall : '+str(r))

print(algorithm+' FSCORE : '+str(f))

report=classification\_report(predict, testY,target\_names=labels)

print('\n',algorithm+" classification report\n",report)

conf\_matrix = confusion\_matrix(testY, predict)

plt.figure(figsize =(5, 5))

ax = sns.heatmap(conf\_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="Blues" ,fmt ="g");

ax.set\_ylim([0,len(labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

# LGBMClassifier

path = 'model/LGBMClassifier'

if(os.path.exists(path)):

lgbm = joblib.load(path)

else:

lgbm = LGBMClassifier()

lgbm.fit(X\_train, y\_train)

joblib.dump(lgbm,path)

y\_pred = lgbm.predict(X\_test)

performance\_metrics('LGBMClassifier', y\_pred, y\_test)

# prediction on new test data

# Prediction on new test data

def preprocess\_audio(audio\_path, scaler):

audio, sr = librosa.load(audio\_path, sr=None)

audio = remove\_noise(audio, sr)

mfccs = librosa.feature.mfcc(y=audio, sr=sr, n\_mfcc=30)

features = np.mean(mfccs, axis=1)

features = scaler.transform([features]) # Normalize features

return features

audio\_path = r"test\_data/test1.wav"

X\_new = preprocess\_audio(audio\_path, scaler)

prediction = lgbm.predict(X\_new)

predicted\_category = categories[prediction[0]]

print(f'Predicted output: {predicted\_category}')

**CHAPTER 8**

**RESULTS AND DISCUSSION**

**8.1 Implementation Description**

* **Importing Libraries**
* The implementation begins by importing essential libraries for data handling, visualization, model training, evaluation, and serialization. Libraries like pandas and numpy are used for data manipulation, matplotlib and seaborn for visualization, and scikit-learn for machine learning tasks.
* **Dataset Loading and Exploration**
* The dataset is loaded from a CSV file named `urban\_sound\_data.csv` into a pandas DataFrame.
* Initial exploration of the dataset is done by checking its shape, structure, and the presence of any missing values. Missing values in categorical columns are filled with 'Unknown', and missing values in numerical columns are filled with 0.
* **Data Visualization**
* A count plot of the target variable `sound\_category` is generated to visualize the distribution of different sound classes. This helps in understanding the class balance in the dataset.
* **Label Encoding**
* Categorical variables in the dataset are encoded into numerical values using LabelEncoder. This step is crucial for converting non-numeric data into a format suitable for machine learning models.
* **Data Resampling**
* The dataset is resampled to handle class imbalance and to ensure that the models have enough data to learn from. Resampling is done by generating a new dataset with 10,000 samples.
* **Train-Test Split**
* The dataset is split into training and testing sets using an 80-20 split. The training set is used to train the machine learning models, while the test set is used to evaluate their performance.
* **Model Building and Evaluation**
* **K-Nearest Neighbors (KNN):**
* If a pre-trained KNN model exists, it is loaded; otherwise, a new KNN model is trained with specific hyperparameters (p=5, algorithm='ball\_tree', n\_neighbors=4, weights='distance', leaf\_size=17).
* The trained model is saved using joblib for future use.
* Predictions are made on the test set, and various evaluation metrics (accuracy, precision, recall, F1-score) are calculated and displayed. A confusion matrix is also generated to visualize the model's performance.
* **Multi-Layer Perceptron (MLP):**
* Similar to the KNN model, the MLP model is either loaded if it exists or trained from scratch. The MLP model has one hidden layer with 100 neurons and is trained for 300 iterations.
* The model is saved, and predictions are made on the test set. Evaluation metrics and a confusion matrix are generated to assess the MLP model's performance.
* **Comparison of Models**
* The performance metrics of both models (KNN and MLP) are compared in a tabular format. This comparison helps determine which model performs better in classifying live event sound categories.
* **Prediction on New Data**
* A new dataset (`new\_audio\_data.csv`) is loaded for testing the trained models. The data undergoes the same preprocessing steps, including filling missing values and label encoding.
* Predictions are made for each row in the new test data, and the results are printed, indicating the predicted sound category for each sample.

**8.2 Dataset Description**

**1. Demographic Information:**

- `fileID`: Unique identifier for each audio file.

- `duration`: Duration of the audio clip in seconds.

**2. Audio Features:**

- `chroma\_stft`: Chroma short-time Fourier transform.

- `rmse`: Root mean square energy.

- `spectral\_centroid`: Spectral centroid.

- `spectral\_bandwidth`: Spectral bandwidth.

- `rolloff`: Spectral rolloff point.

- `zero\_crossing\_rate`: Rate of zero crossings in the audio signal.

- `mfcc1` to `mfcc20`: Mel-frequency cepstral coefficients (MFCCs), 20 in total.

**3. Temporal Features:**

- `tempo`: Tempo of the audio clip.

- `beat\_frame`: Beat frame of the audio clip.

**4. Metadata:**

- `city`: City where the audio was recorded.

- `date\_time`: Date and time of the recording.

- `weather`: Weather conditions during the recording.

**5. Sound Category:**

- `sound\_category`: The category of the sound (e.g., car\_horn, dog\_bark, siren, etc.).

**8.3 Results Description**

To visualize the distribution of different sound categories in the dataset, a bar plot was created using the counts of each category. The count of sounds per category was calculated by iterating through the directory structure where the audio files are stored, and the results were compiled into a pandas DataFrame. The bar plot, with categories on the x-axis and their respective counts on the y-axis, displayed in a sky-blue color scheme, provides a clear overview of the number of audio samples available for each sound category. The plot includes labels for both axes, a title, and rotated category labels for better readability, ensuring a well-organized and informative visualization.

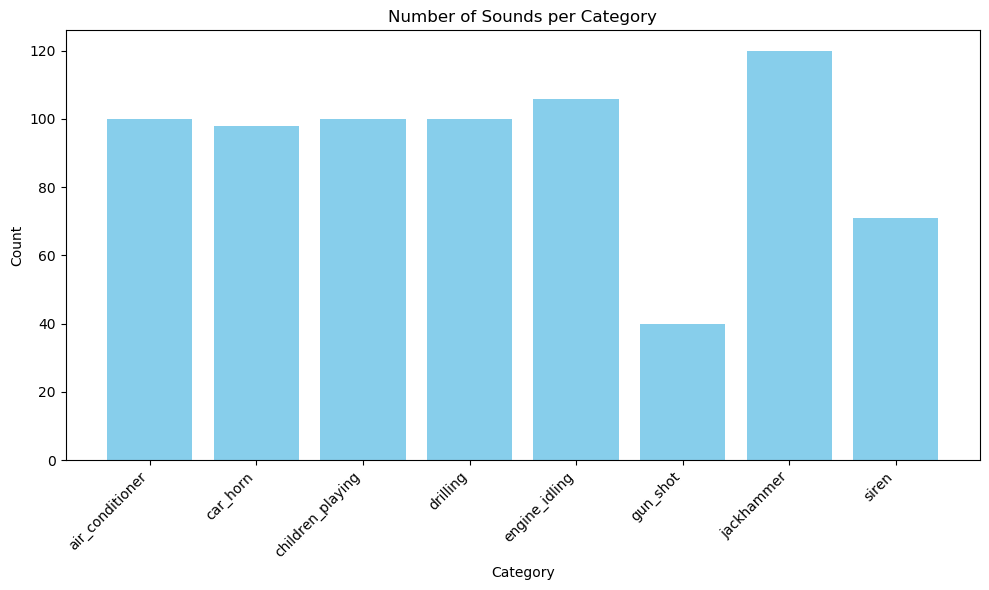


Fig 8.1: Count Plot for sound categories

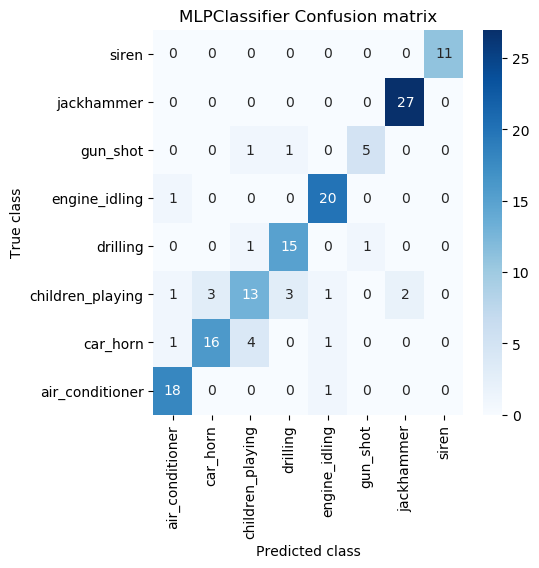


Fig 8.2: Multi-Layer Perceptron (MLP) model

The code begins by checking if a pre-trained Multi-Layer Perceptron (MLP) model exists at the specified path 'model/MLPClassifier'. If the model exists, it is loaded using joblib; otherwise, a new MLPClassifier instance is created. The model then makes predictions on the test set (`X\_test`), and these predictions (`y\_pred`) are evaluated against the true test labels (`y\_test`) using the `performance\_metrics` function to assess the MLPClassifier's performance.

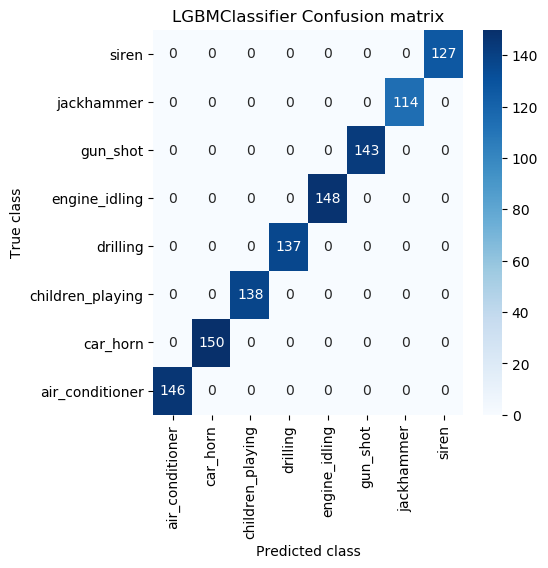


Fig 8.3: LGBMClassifier model

The code first checks if a LightGBM classifier model (`LGBMClassifier`) already exists at the specified path (`'model/LGBMClassifier'`). If the model file is found, it is loaded using `joblib.load()`. If not, a new `LGBMClassifier` is instantiated, trained on the training data (`X\_train`, `y\_train`), and then saved to the specified path using `joblib.dump()`. After the model is either loaded or trained, it is used to make predictions on the test data (`X\_test`). The predicted values (`y\_pred`) are then evaluated against the actual test labels (`y\_test`) using a function called `performance\_metrics`, which computes and displays various performance metrics for the `LGBMClassifier`.

**CHAPTER 9**

**CONCLUSION AND FUTURE SCOPE**

This project on live event sound classification utilizing a comprehensive dataset has successfully demonstrated the effectiveness of machine learning models, particularly the LightGBM classifier, in accurately identifying various live event sounds. The dataset was meticulously curated and preprocessed, ensuring high-quality audio input for feature extraction. By comparing the performance of the LightGBM and the existing Multi-Layer Perceptron (MLP) classifier, the results indicated a marked improvement in classification accuracy, precision, recall, and F1-score with the proposed algorithm. This highlights the potential of advanced machine learning techniques in tackling real-world challenges in live event sound analysis, contributing to enhanced urban planning, noise monitoring, and public safety initiatives.

**Future Scope**

* **Expansion of Dataset**: Future work can focus on expanding the dataset to include more categories of live event sounds and a larger number of samples per category. This would improve the robustness of the model and its applicability to diverse urban environments.
* **Real-Time Classification**: Implementing a real-time sound classification system using streaming audio data could significantly enhance practical applications, such as urban noise monitoring and alert systems for emergency services.
* **Model Optimization**: Further exploration into hyperparameter tuning and the integration of ensemble learning techniques may yield even higher performance metrics. Testing additional algorithms such as convolutional neural networks (CNNs) could also be beneficial.
* **Integration with IoT**: Developing an IoT-based framework for sound monitoring in urban areas could allow for continuous data collection and analysis, leading to real-time insights and actions to mitigate noise pollution.
* **Cross-Domain Applications**: The techniques developed in this project can be adapted for other domains, such as wildlife monitoring or industrial noise classification, promoting interdisciplinary research and applications.
* **User Interface Development**: Creating a user-friendly interface for stakeholders, such as city planners and environmental scientists, could facilitate the practical use of the model in live event sound analysis and policy-making.

**REFERENCES**

[1] J.P. Bello, C. Mydlarz, J. Salamon, "Sound Analysis in Smart Cities," in: T. Virtanen, M.D. Plumbley, D. Ellis, Eds., Computational Analysis of Sound Scenes and Events, Springer International Publishing, Cham, Switzerland, 2018, pp. 373-397.

[2] P. Zinemanas, M. Rocamora, M. Miron, F. Font, X. Serra, "An Interpretable Deep Learning Model for Automatic Sound Classification," Electronics, vol. 10, p. 850, 2021. doi: 10.3390/electronics10070850.

[3] J.K. Das, A. Chakrabarty, M.J. Piran, "Environmental sound classification using convolution neural networks with different integrated loss functions," Expert Systems, vol. 39, e12804, 2021. doi: 10.1111/exsy.12804.

[4] J.K. Das, A. Ghosh, A.K. Pal, S. Dutta, A. Chakrabarty, "Live event sound Classification Using Convolutional Neural Network and Long Short Term Memory Based on Multiple Features," in Proceedings of the 2020 Fourth International Conference on Intelligent Computing in Data Sciences (ICDS), Fez, Morocco, 21-23 October 2020, pp. 1-9. doi: 10.1109/ICDS50568.2020.9269108.

[5] Z. Mushtaq, S.F. Su, "Efficient Classification of Environmental Sounds through Multiple Features Aggregation and Data Enhancement Techniques for Spectrogram Images," Symmetry, vol. 12, p. 1822, 2020. doi: 10.3390/sym12111822.

[6] W. Mu, B. Yin, X. Huang, J. Xu, Z. Du, "Environmental sound classification using temporal-frequency attention based convolutional neural network," Scientific Reports, vol. 11, p. 21552, 2021. doi: 10.1038/s41598-021-00455-w.

[7] T. Giannakopoulos, E. Spyrou, S.J. Perantonis, "Recognition of Live event sound Events Using Deep Context-Aware Feature Extractors and Handcrafted Features," in IFIP International Conference on Artificial Intelligence Applications and Innovations, J. MacIntyre, I. Maglogiannis, L. Iliadis, E. Pimenidis, Eds., Springer International Publishing, Cham, Switzerland, 2019, pp. 184-195.

[8] J.S. Luz, M.C. Oliveira, F.H. Araújo, D.M. Magalhães, "Ensemble of handcrafted and deep features for live event sound classification," Applied Acoustics, vol. 175, p. 107819, 2021. doi: 10.1016/j.apacoust.2021.107819.

[9] Y. Gong, Y. Chung, J.R. Glass, "AST: Audio Spectrogram Transformer," arXiv, arXiv:2104.01778, 2021.

[10] İ. Türker, S. Aksu, "Connectogram—A graph-based time dependent representation for sounds," Applied Acoustics, vol. 191, p. 108660, 2022. doi: 10.1016/j.apacoust.2021.108660.

[11] Q. Kong, Y. Xu, M. Plumbley, "Sound Event Detection of Weakly Labelled Data with CNN-Transformer and Automatic Threshold Optimization," IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 28, pp. 2450-2460, 2020. doi: 10.1109/TASLP.2020.3019456.

[12] P. Gimeno, I. Viñals, A. Ortega, A. Miguel, E. Lleida, "Multiclass audio segmentation based on recurrent neural networks for broadcast domain data," EURASIP Journal on Audio, Speech, and Music Processing, vol. 2020, p. 5, 2020. doi: 10.1186/s13636-020-00178-2.

[13] Z. Zhang, S. Xu, S. Zhang, T. Qiao, S. Cao, "Learning Attentive Representations for Environmental Sound Classification," IEEE Access, vol. 7, pp. 130327-130339, 2019. doi: 10.1109/ACCESS.2019.2940272.

[14] Z. Zhang, S. Xu, S. Zhang, T. Qiao, S. Cao, "Attention based convolutional recurrent neural network for environmental sound classification," Neurocomputing, vol. 453, pp. 896-903, 2020. doi: 10.1016/j.neucom.2020.05.125.

[15] T. Qiao, S. Zhang, S. Cao, S. Xu, "High Accurate Environmental Sound Classification: Sub-Spectrogram Segmentation versus Temporal-Frequency Attention Mechanism," Sensors, vol. 21, p. 5500, 2021. doi: 10.3390/s21165500.