**MACHINE LEARNING ADOPTED HUMAN ACTIVITY CLASSIFICATIONUSING SHIMMER WEARABLE SENSORS FOR HEALTH MONITORING**

**ABSTRACT**

Traditional approaches for human activity recognition used simple sensors and manual observation, but have developed greatly. Early systems collected data from accelerometers and gyroscopes and classified activities using simple algorithms. With more advanced sensors and processing capability, researchers explored statistical pattern recognition algorithms. Machine learning made categorization systems that could automatically handle complicated datasets more accurate and efficient. This research aims to create a machine learning-based system that accurately classifies human activities using Shimmer wearable sensor data to improve health monitoring by providing real-time insights into physical activity and well-being. The research describes the study's use of machine learning and wearable sensors to monitor and classify human activities for health purposes. Before machine learning and AI, human activity recognition systems used manual data collection and basic algorithms with predefined thresholds and simple statistical methods, limiting their accuracy and adaptability in diverse real-world environments. Due to their fixed algorithms and limited data processing capabilities, traditional systems misclassified human activities and failed to understand human movement, hindering health monitoring. This research was motivated by the need for improved accuracy and efficiency in human activity classification for health monitoring. Traditional methods often fail to capture the complexity of human behaviors, which machine learning algorithms can address through advanced data analysis. Real-time classification of walking, sitting, and standing is possible with Shimmer wearable sensor data and machine learning algorithms. Advanced techniques like XGBoost and Gradient Boosting improve accuracy, health monitoring, and individualized feedback, improving health outcomes. Moving from old systems to machine learning-based ones will increase human activity recognition quality and reliability.

**CHAPTER 1**

**INTRODUCTION**

**1.1 Introduction**

Human activity classification is an essential component of health monitoring systems, enabling the assessment and tracking of daily activities for various health-related applications. With the advent of wearable technology, sensors have become increasingly integrated into everyday life, providing continuous and unobtrusive monitoring of physical activities. One such advanced wearable sensor system is SHIMMER (Sensing Health with Intelligence, Modularity, Mobility, and Experimental Reusability), which has shown significant promise in accurately capturing and classifying human movements. These wearable sensors offer a wealth of data that, when harnessed effectively through machine learning techniques, can lead to substantial improvements in health monitoring and management.

The prevalence of chronic diseases, the aging population, and the need for proactive health management have driven the demand for robust activity classification systems. Traditional methods of activity monitoring often rely on subjective self-reporting or stationary devices that limit mobility and real-time analysis. In contrast, wearable sensors provide a more dynamic and accurate representation of an individual's activities, capturing nuances in movement that are critical for precise health monitoring.

Recent advancements in machine learning have further enhanced the capabilities of these wearable systems, allowing for more sophisticated analysis and classification of human activities. By leveraging the continuous data stream from SHIMMER sensors, machine learning models can identify and categorize various activities such as walking, running, sitting, and sleeping, among others. This technology holds significant potential not only for individual health monitoring but also for broader applications in rehabilitation, elderly care, and chronic disease management.

**1.2 Motivation**

Despite the proliferation of wearable sensors, the effective utilization of the vast amounts of data they generate remains a challenge. The primary motivation for adopting machine learning in human activity classification is to bridge this gap and unlock the full potential of wearable technology for health monitoring. The SHIMMER sensors provide detailed and high-frequency data that, if analyzed correctly, can yield valuable insights into an individual's health status and activity patterns. Traditional data analysis techniques are often inadequate to handle the complexity and volume of this data, necessitating the use of advanced machine learning algorithms.

Moreover, accurate activity classification is critical for developing personalized health interventions. For instance, detecting deviations from normal activity patterns can help in early diagnosis of conditions such as Parkinson's disease or in monitoring the recovery progress of patient’s post-surgery. The ability to continuously and accurately monitor activities also supports elderly individuals in maintaining independence while ensuring timely intervention in case of falls or other health emergencies.

**1.3 Problem Statement**

The challenge in human activity classification lies in the variability and complexity of human movements, which can be influenced by numerous factors such as age, health condition, and environment. Manual classification of these activities is not only labor-intensive but also prone to inaccuracies. The SHIMMER wearable sensors, while capable of collecting comprehensive data, require sophisticated processing to accurately interpret this information.

Existing methods often fall short in dealing with the high dimensionality and noise inherent in sensor data. Additionally, traditional models may not generalize well across different populations or adapt to new activities without significant re-training. Thus, there is a need for a robust machine learning framework that can efficiently process sensor data, adapt to varying activity patterns, and provide accurate classifications in real-time.

**1.4 Applications**

* **Enhanced Health Monitoring:** Machine learning models can process sensor data to provide continuous and detailed monitoring of an individual's activity, helping in early detection of health issues and tracking of chronic conditions.
* **Rehabilitation Support:** Activity classification can aid in designing personalized rehabilitation programs, monitoring patient progress, and adjusting therapy plans based on real-time data.
* **Elderly Care**: Accurate classification of daily activities can ensure the safety and well-being of elderly individuals, detecting falls, and other anomalies that require immediate attention.
* **Fitness and Wellness**: Wearable sensors combined with machine learning can offer detailed insights into physical activity, helping individuals to optimize their fitness routines and achieve health goals.
* **Integration with Electronic Health Records (EHRs):** Seamless integration with EHR systems can provide healthcare professionals with comprehensive activity data, improving the accuracy of diagnoses and the effectiveness of treatment plans.

**CHAPTER 2**

**LITERATURE SURVEY**

J. Hayano et al. [1] examined the use of wearable technology to detect sleep apnea with a watch device. Their study, published inPLoS ONE in 2020, presented a quantitative approach to monitor and analyze sleep patterns through wearable sensors. They employed advanced algorithms to detect apnea events by analyzing physiological signals collected from the device. This approach offers a non-invasive method to improve sleep disorder diagnosis and management. The study highlights the potential of wearable technology in sleep medicine, emphasizing its ability to provide continuous monitoring outside of clinical settings. The research shows how wearable sensors can enhance patient care by offering real-time insights into sleep patterns and apnea events, thus paving the way for better management strategies.

F. Delmastro et al. [2] explored cognitive training and stress detection in frail older individuals using wearable sensors and machine learning. Their 2020 study in IEEE Accessdemonstrated how machine learning algorithms could analyze data from wearable devices to assess cognitive function and stress levels. The study integrated cognitive training with stress monitoring to offer personalized interventions for elderly patients. By utilizing data from wearable sensors, the research provides insights into managing cognitive decline and stress in older populations. This approach could lead to improved quality of life for elderly individuals through tailored cognitive and stress management strategies. The findings highlight the effectiveness of combining technology with healthcare interventions to address age-related challenges.

M.V. Perez et al. [3] conducted a large-scale assessment of smartwatch technology to identify atrial fibrillation. Published in the New England Journal of Medicinein 2019, their research evaluated the efficacy of smartwatches in detecting irregular heart rhythms. The study analyzed data from a large cohort, showing that smartwatches could accurately identify atrial fibrillation. This capability has the potential to reduce the need for invasive diagnostic procedures. The findings emphasize the role of wearable technology in early cardiovascular disease detection and its ability to enhance patient monitoring. By demonstrating the effectiveness of smartwatches in identifying arrhythmias, the research supports the integration of wearable technology into cardiovascular care.

J.S. Chorba et al. [4] developed a deep learning algorithm for automated cardiac murmur detection using a digital stethoscope platform. Their 2021 study in the Journal of American Heart Associationintroduced a novel approach for analyzing heart sounds to identify cardiac murmurs. Utilizing deep learning techniques, the study achieved high accuracy in detecting murmurs, which could improve the diagnostic process in cardiology. This research highlights the potential of artificial intelligence to enhance the accuracy and efficiency of cardiac assessments. The integration of deep learning with medical devices represents a significant advancement in cardiology, offering more precise and timely diagnostics.

S. Seneviratne et al. [5] reviewed wearable devices and their associated challenges in their 2017 study published in IEEE Communications Surveys & Tutorials. The survey provided a comprehensive overview of various wearable technologies, addressing challenges such as battery life, data accuracy, and user acceptance. The review emphasized the advancements in wearable technology and the need for continued innovation to overcome existing limitations. This study offers valuable insights into the state of wearable devices in healthcare and identifies future research directions. By highlighting both the progress and the challenges faced by wearable technology, the research underscores the importance of ongoing development in this field.

M. Chan et al. [6] discussed the current status and future challenges of smart wearable systems in their 2012 article in Artificial Intelligence in Medicine. Their study highlighted advancements in wearable technology, including improvements in sensor accuracy and data analysis capabilities. The paper addressed challenges such as data integration, user privacy, and system reliability. By providing a critical assessment of the progress made in smart wearable systems, the research identifies areas for future research to enhance effectiveness in healthcare applications. The study underscores the need for addressing challenges to maximize the benefits of wearable technology in medical settings.

P. Siirtola et al. [7] investigated the use of sleep time data from wearable sensors for early detection of migraine attacks. Their 2018 study inSensorsdemonstrated that analyzing sleep patterns with wearable devices could help predict the onset of migraines. The research highlighted the potential of wearable technology to offer early warnings and personalized interventions for migraine sufferers. By leveraging sleep data, the study underscores the role of wearable sensors in managing chronic conditions and improving patient outcomes through proactive monitoring. This approach could lead to more effective management strategies for individuals suffering from migraines.

C. Meisel et al. [8] explored machine learning techniques for wearable, noninvasive seizure forecasting using wristband sensor data. Published in Epilepsiain 2020, their study focused on predicting seizures based on data collected from wearable devices. The research demonstrated that machine learning models could effectively forecast seizures, potentially enhancing patient safety and reducing emergency interventions. The study highlights the potential of wearable technology in managing epilepsy and improving the quality of life for patients. By utilizing machine learning for seizure prediction, the research offers a promising approach to epilepsy management and patient care.

A.Y. Hannun et al. [9] investigated cardiovascular arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Their 2019 study in Nature Medicineintroduced a deep learning approach for analyzing ECG data to detect arrhythmias. The research showed that the neural network model could accurately classify various types of arrhythmias, offering a promising tool for remote cardiac monitoring. The study emphasizes the role of artificial intelligence in advancing cardiac care and improving diagnostic accuracy. By integrating deep learning with ECG analysis, the research provides a significant advancement in remote cardiovascular monitoring.

S. Kwon et al. [10] evaluated the use of a ring-type wearable device for detecting atrial fibrillation through deep learning analysis of photoplethysmography signals. Their study, published in J Med Internet Res in 2020, provided proof-of-concept for using wearable technology to monitor heart rhythms. The research demonstrated the potential of combining wearable devices with advanced data analysis techniques to improve the detection of atrial fibrillation. The findings support the use of innovative wearable solutions in cardiovascular health management. By showcasing the effectiveness of ring-type devices, the study contributes to the development of advanced wearable health technologies.

Z. Mei et al. [11] proposed an automatic atrial fibrillation detection method based on heart rate variability and spectral features. Their 2018 study in IEEE Access highlighted a method that leverages heart rate data to identify atrial fibrillation episodes. The study illustrated the effectiveness of combining heart rate variability analysis with spectral features to enhance detection accuracy. This research contributes to the development of reliable methods for monitoring and diagnosing cardiovascular conditions using wearable technology. By integrating advanced analytical techniques, the study offers a valuable approach to improving atrial fibrillation detection.

N. Rashid and M.A. Al Faruque [12] focused on energy-efficient real-time myocardial infarction detection on wearable devices. Their 2020 study presented at the IEEE Engineering in Medicine and Biology Society Conference highlighted methods for optimizing wearable devices to detect myocardial infarction with minimal energy consumption. The research demonstrated advancements in wearable technology aimed at improving real-time monitoring capabilities while addressing power efficiency. This study underscores the importance of integrating energy efficiency in the design of wearable health monitoring devices. By focusing on power consumption, the research contributes to the development of more sustainable and effective wearable technologies.

R. Buettner et al. [13] presented a high-performance detection method for epilepsy in seizure-free EEG recordings at the International Conference on Information Systems in 2019. The study introduced techniques for analyzing EEG data to detect epilepsy-related anomalies without the presence of seizures. The research highlighted the potential of advanced EEG analysis methods to enhance the detection of epilepsy and provide valuable insights into brain activity patterns. By developing high-performance detection methods, the study contributes to improving epilepsy diagnosis and management through sophisticated EEG analysis techniques.

C. Ieracitano et al. [14] developed a multi-modal machine learning approach for automatic classification of EEG recordings in dementia. Published in 2020, their study focused on combining multiple data modalities to improve the classification of EEG recordings in dementia patients. The research demonstrated the effectiveness of multi-modal machine learning techniques in enhancing the accuracy of dementia diagnosis and management. By integrating various data sources, the study offers a comprehensive approach to improving diagnostic accuracy in dementia care.

S. Hwang et al. [15] investigated the use of wearable EEG technology to measure workers’ emotional states during construction tasks. Their 2018 study is explored how wearable EEG devices can monitor and assess emotional responses in real-time. The research highlighted the potential of wearable technology to improve workplace safety and productivity by providing insights into workers' emotional well-being. By leveraging wearable EEG technology, the study contributes to enhancing workplace environments and worker health through real-time emotional monitoring.

**CHAPTER 3**

**EXISTING SYSTEM**

**3.1 Overview:**

**1. Human Activity Classification Using Shimmer Wearable Sensors**

Human activity classification has become increasingly important in health monitoring, enabling accurate tracking and analysis of physical activities for health assessment and intervention. Traditionally, human activity recognition has relied on various sensors and data processing techniques to categorize different types of movements and behaviors.

Wearable sensors, such as those provided by Shimmer devices, have emerged as a key technology in this field. These sensors collect data on various physiological parameters, including accelerometry, gyroscope, and magnetometry, which can be used to monitor and classify human activities. Shimmer sensors offer the advantage of portability and continuous monitoring, allowing for real-time data collection without interrupting daily routines.

The data collected from wearable sensors are typically preprocessed to remove noise and irrelevant information. Features are then extracted from the raw data to capture essential aspects of the activities being performed. These features might include statistical measures, frequency domain characteristics, and temporal patterns that are relevant for activity classification.

Machine learning algorithms play a crucial role in the classification of human activities based on sensor data. Supervised learning methods, such as support vector machines (SVM), decision trees, and neural networks, are commonly employed to train models on labeled datasets. These models learn to differentiate between various activities by recognizing patterns in the sensor data. Advanced techniques like deep learning, particularly convolutional and recurrent neural networks, have shown promise in improving classification accuracy by leveraging complex feature representations and temporal dependencies.

The use of wearable sensors for human activity classification has a wide range of applications. In healthcare, it enables the monitoring of physical activity levels, gait analysis, and fall detection, providing valuable insights into patients' mobility and overall health. In sports and fitness, it helps track performance metrics, detect anomalies, and tailor training programs. Additionally, wearable sensors can be used in research to study activity patterns in different populations and environments.

Despite the advantages of wearable sensors, there are several challenges associated with their use. Data accuracy and reliability can be affected by factors such as sensor placement, movement artifacts, and individual variability. Moreover, the integration of sensor data with machine learning models requires careful consideration of data privacy, computational resources, and real-time processing capabilities. Future research is needed to address these challenges, improve sensor technology, and refine machine learning algorithms to enhance the effectiveness of human activity classification systems.

**3.2 Challenges in Traditional Approaches**

The traditional methods of human activity classification using wearable sensors face several challenges that impact their effectiveness and usability.

One of the main challenges is ensuring the accuracy of the data collected by wearable sensors. Variability in sensor placement and movement artifacts can affect the quality of the data, leading to potential errors in activity classification. Calibration and validation of sensors are crucial to mitigate these issues.

The process of extracting and selecting relevant features from sensor data can be complex and time-consuming. Inadequate feature extraction may lead to suboptimal model performance, while selecting too many features can increase computational complexity and overfitting. Balancing feature richness with computational efficiency is essential for effective activity classification.

Training machine learning models on sensor data requires large and diverse datasets to ensure generalizability. However, collecting such datasets can be challenging, especially for specific activities or populations. Additionally, model validation and evaluation must be rigorous to avoid overfitting and ensure that the models perform well across different scenarios and conditions.

For many applications, real-time activity classification is essential. However, processing sensor data in real-time requires significant computational resources and efficient algorithms. Ensuring that models can make accurate predictions quickly without compromising performance is a key challenge.

The effectiveness of wearable sensors also depends on user acceptance and comfort. Sensors that are uncomfortable or intrusive may not be worn consistently, reducing the reliability of the collected data. Designing sensors that are unobtrusive and comfortable for long-term use is important for maintaining user compliance.

With the continuous monitoring of personal data, privacy and data security concerns must be addressed. Ensuring that data is encrypted and securely stored, and providing users with control over their data, are critical for maintaining trust and compliance with regulations.

**3.3 Limitations of Traditional Approaches**

The limitations of traditional approaches to human activity classification using wearable sensors highlight the need for advancements in technology and methodology.

Traditional systems may be limited in the range of activities they can accurately classify. While many systems perform well with common activities like walking or running, they may struggle with less frequent or more complex activities. Expanding the range of detectable activities requires ongoing research and development.

Many traditional methods rely on supervised learning, which requires labeled data for training. This dependency can limit the applicability of the system to new or unannotated activities. Semi-supervised and unsupervised learning techniques may offer solutions to this limitation, but they also come with their own challenges.

Traditional systems may face difficulties in scaling to larger populations or adapting to different environments. Models trained on specific datasets may not perform well when applied to new contexts or user groups. Ensuring that systems can generalize across diverse settings is an ongoing challenge.

Processing large amounts of sensor data, especially in real-time, can be computationally intensive. Traditional systems may require substantial hardware resources, which can be a barrier to widespread adoption. Developing more efficient algorithms and leveraging edge computing can help address this issue.

Integrating wearable sensors with other health monitoring systems or electronic health records (EHRs) can be complex. Ensuring seamless data integration and interoperability requires standardization and compatibility across different platforms and devices.

The use of wearable sensors raises ethical and legal concerns related to data privacy, informed consent, and potential misuse of personal health information. Addressing these concerns requires careful consideration of ethical guidelines and legal frameworks to protect users' rights and ensure responsible data use.

**CHAPTER 4**

**PROPOSED SYSTEM**

**4.1 Overview**

**Step 1: Dataset Collection**The study begins with the collection of a comprehensive dataset consisting of various human activities captured using Shimmer wearable sensors. This dataset is essential for training machine learning models and ensuring that they can generalize well to different activities such as bending, cycling, sitting, standing, and walking.

**Step 2: Dataset Preprocessing**In this step, the dataset undergoes preprocessing to prepare it for analysis. This involves removing null values, which could skew the results, and handling any inconsistencies in the data. Data normalization techniques may also be applied to ensure uniformity across features, enabling more effective model training.

**Step 3: Label Encoding**To convert categorical variables into a numerical format, label encoding is performed on the activity labels. This transformation is crucial for enabling the machine learning algorithms to interpret the data effectively, as they typically require numerical input.

**Step 4: Training-Testing Split**The preprocessed dataset is then split into training and testing subsets. This division allows for the training of the model on one portion of the data while evaluating its performance on a separate, unseen portion, thereby providing a clear indication of its predictive capability.

**Step 5: Existing Algorithm**The first model implemented is based on the XGBoost (Extreme Gradient Boosting) algorithm. This ensemble technique is renowned for its speed and performance in classification tasks, making it a suitable choice for this project.

**Step 6: Proposed Algorithm**The proposed algorithm utilizes the Gradient Boosting Classifier (GBC) to enhance performance further. This approach builds on the strengths of existing algorithms, allowing for improved accuracy in human activity classification.

**Step 7: Performance Comparison**A comprehensive performance comparison is conducted between the existing and proposed algorithms. Metrics such as accuracy, precision, recall, and F1 score are analyzed to evaluate which model demonstrates superior classification capabilities.

**Step 8: Prediction of Output from Test Data**Finally, predictions are made on the test dataset using the trained models. This step provides insights into how well the algorithms can classify human activities based on new data, ultimately demonstrating the model’s practical applicability in health monitoring.

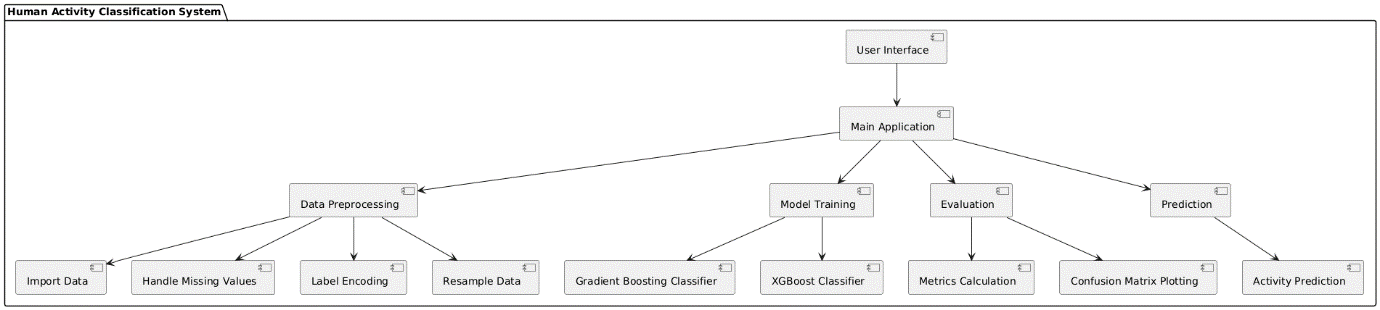


Figure 1: Architectural Block Diagram of Proposed System.

**4.2 Data Splitting & Preprocessing**

In the data splitting and preprocessing phase, the dataset is meticulously prepared for analysis. First, null values are addressed to prevent any detrimental impact on the training process. This is followed by the application of label encoding to convert categorical data into a numerical format. The dataset is then split into training and testing subsets, ensuring that the model can be trained effectively while maintaining a fair evaluation process on unseen data. This structured approach ensures a robust foundation for subsequent machine learning model development.

**4.3 ML Model Building**

Machine learning model building involves several critical steps. Initially, data is collected and preprocessed to ensure quality and consistency. Following preprocessing, the dataset is split into training and testing sets to facilitate model evaluation. Multiple algorithms, such as XGBoost and Gradient Boosting Classifier, are employed to determine which yields the best results in terms of classification accuracy. Each model undergoes training using the training set and is then evaluated on the testing set. Performance metrics such as accuracy, precision, and recall are computed to assess effectiveness. Finally, the model with the highest performance is selected for deployment in real-world applications.

**4.3.1 Existing Algorithm: XGBoost**

**What is XGBoost?**XGBoost, or Extreme Gradient Boosting, is a powerful machine learning algorithm known for its high performance in classification and regression tasks. It operates on the principle of boosting, where weak learners (typically decision trees) are combined to form a strong predictive model.

**How It Works**XGBoost constructs decision trees sequentially, where each new tree attempts to correct the errors made by the previous trees. It employs gradient descent optimization techniques to minimize the loss function, enhancing model accuracy.

**Architecture**  
The architecture of XGBoost consists of several components, including the booster (the core algorithm), objective function (to evaluate model performance), and regularization terms (to prevent overfitting). This structured design allows for efficient computation and scalability.

**Disadvantages**  
Despite its advantages, XGBoost can be complex to tune due to its numerous hyperparameters. Additionally, it may require more computational resources compared to simpler algorithms, making it less suitable for smaller datasets.

**4.3.2 Proposed Algorithm: Gradient Boosting Classifier (GBC)**

**What is GBC?**The Gradient Boosting Classifier (GBC) is another ensemble machine learning technique that builds models in a sequential manner, similar to XGBoost. It combines multiple weak learners to create a robust predictive model.

**How It Works**GBC works by optimizing a loss function through gradient descent, adding trees one at a time, each focused on reducing the residual errors of the existing model. This iterative process continues until a specified number of trees is reached or improvements become negligible.

**Architecture**  
The architecture of GBC comprises components such as decision trees, loss functions, and the boosting framework, which collectively enhance the model's ability to learn from data iteratively.

**Advantages**  
GBC offers several advantages, including its ability to handle different types of data and its robustness against overfitting through regularization techniques. It often outperforms other algorithms in terms of accuracy, particularly in complex datasets with non-linear relationships.

**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.

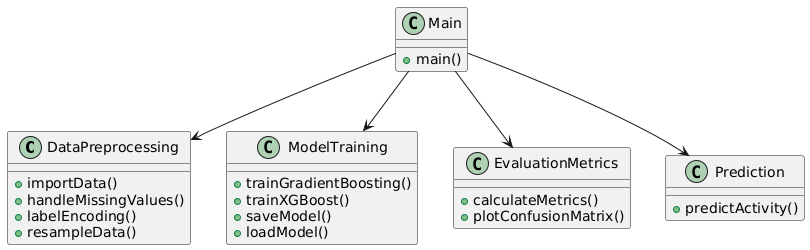


Figure-5.1: Class Diagram

**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.

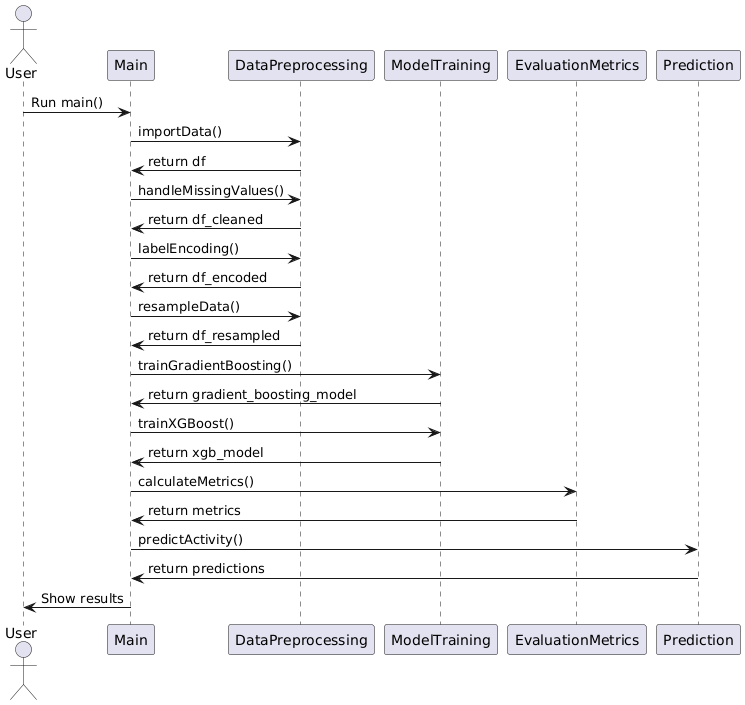


Figure-5.2: Sequence Diagram

**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.

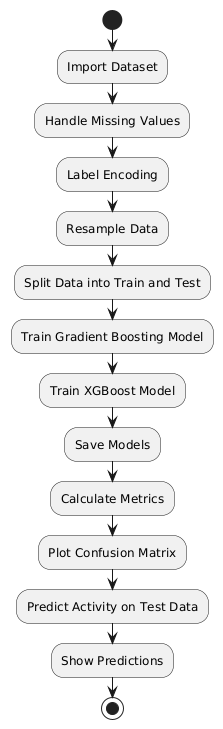


Figure-5.3: Activity Diagram

**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.

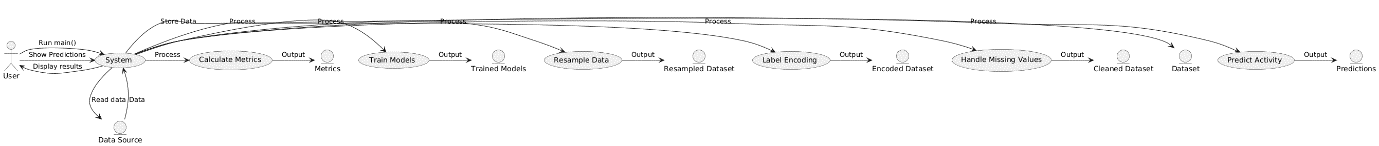


Figure-5.4: Dataflow Diagram

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

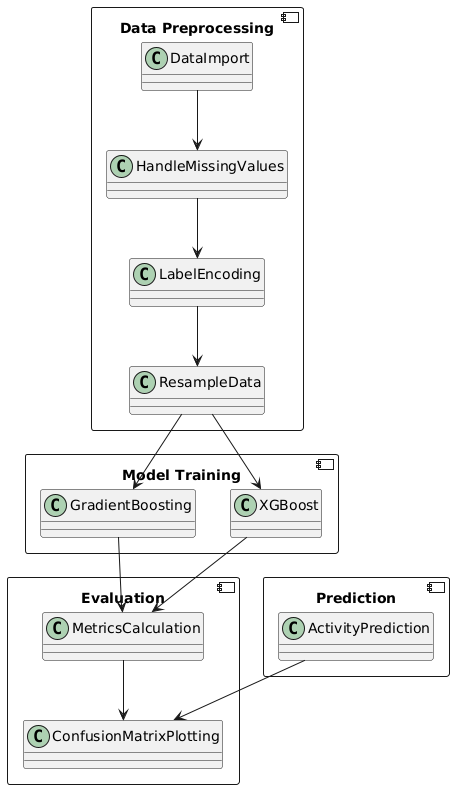


Figure-5.5: Component Diagram

**UseCase 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.

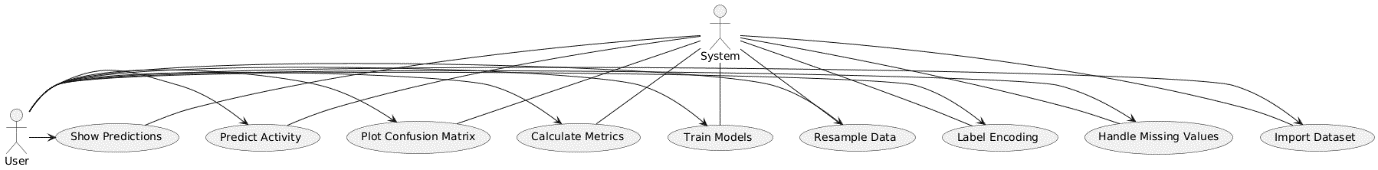


Figure-5.6: use case diagram

**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.

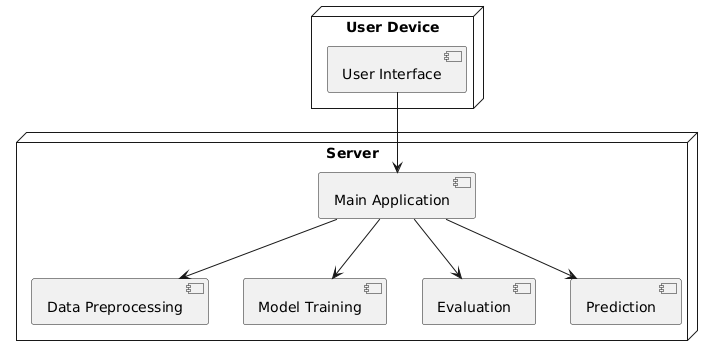


Figure-5.7: DeploymentDiagram

**Architectural Block Diagram**

An architectural block diagram offers a high-level view of a system’s structure, showcasing the main components and their interactions. It represents how major modules, such as data sources, processing units, and evaluation components, are organized and how they communicate with each other to accomplish the system’s objectives. This diagram helps in understanding the overall design and flow of the system.

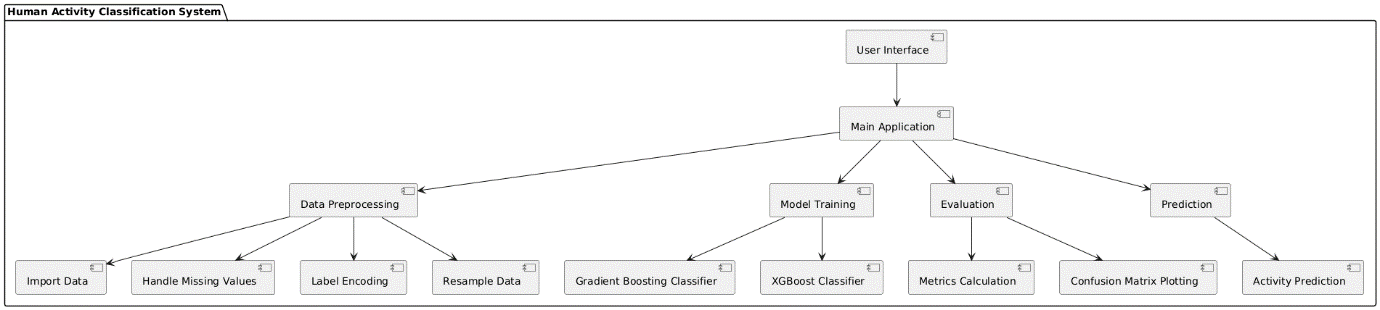


Figure-5.8: architectural block diagram

**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**

**SYSTEM REQUIREMENTS SPECIFICATIONS**

**Software Requirements**

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

* Python IDLE 3.7 version (or)
* Anaconda 3.7 (or)
* Jupiter (or)
* Google colab

**Hardware Requirements**

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

Operating system : Windows, Linux

Processor : minimum intel i3

Ram : minimum 4 GB

Hard disk : minimum 250GB

**CHAPTER 8**

**FUNCTIONAL REQUIREMENTS**

**Output Design**

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provides a permanent copy of the results for later consultation. The various types of outputs in general are:

* External Outputs, whose destination is outside the organization
* Internal Outputs whose destination is within organization and they are the
* User’s main interface with the computer.
* Operational outputs whose use is purely within the computer department.
* Interface outputs, which involve the user in communicating directly.

**Output Definition**

The outputs should be defined in terms of the following points:

* Type of the output
* Content of the output
* Format of the output
* Location of the output
* Frequency of the output
* Volume of the output
* Sequence of the output

It is not always desirable to print or display data as it is held on a computer. It should be decided as which form of the output is the most suitable.

**Input Design**

Input design is a part of overall system design. The main objective during the input design is as given below:

* To produce a cost-effective method of input.
* To achieve the highest possible level of accuracy.
* To ensure that the input is acceptable and understood by the user.

**Input Stages**

The main input stages can be listed as below:

* Data recording
* Data transcription
* Data conversion
* Data verification
* Data control
* Data transmission
* Data validation
* Data correction

**Input Types**

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

* External inputs, which are prime inputs for the system.
* Internal inputs, which are user communications with the system.
* Operational, which are computer department’s communications to the system?
* Interactive, which are inputs entered during a dialogue.

**Input Media**

At this stage choice has to be made about the input media. To conclude about the input media consideration has to be given to;

* Type of input
* Flexibility of format
* Speed
* Accuracy
* Verification methods
* Rejection rates
* Ease of correction
* Storage and handling requirements
* Security
* Easy to use
* Portability

Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As

Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

**Error Avoidance**

At this stage care is to be taken to ensure that input data remains accurate form the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

**Error Detection**

Even though every effort is made to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

**Data Validation**

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

**User Interface Design**

It is essential to consult the system users and discuss their needs while designing the user interface:

**User Interface Systems Can Be Broadly Clasified As:**

* User initiated interface the user is in charge, controlling the progress of the user/computer dialogue. In the computer-initiated interface, the computer selects the next stage in the interaction.
* Computer initiated interfaces

In the computer-initiated interfaces the computer guides the progress of the user/computer dialogue. Information is displayed and the user response of the computer takes action or displays further information.

**User Initiated Intergfaces**

User initiated interfaces fall into two approximate classes:

* Command driven interfaces: In this type of interface the user inputs commands or queries which are interpreted by the computer.
* Forms oriented interface: The user calls up an image of the form to his/her screen and fills in the form. The forms-oriented interface is chosen because it is the best choice.

**Computer-Initiated Interfaces**

The following computer – initiated interfaces were used:

* The menu system for the user is presented with a list of alternatives and the user chooses one; of alternatives.
* Questions – answer type dialog system where the computer asks question and takes action based on the basis of the users reply.

Right from the start the system is going to be menu driven, the opening menu displays the available options. Choosing one option gives another popup menu with more options. In this way every option leads the users to data entry form where the user can key in the data.

**Error Message Design**

The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

This application must be able to produce output at different modules for different inputs.

**Performance Requirements**

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

* The system should be able to interface with the existing system
* The system should be accurate
* The system should be better than the existing system
* The existing system is completely dependent on the user to perform all the duties.

**CHAPTER 9**

**SOURCE CODE**

# Machine Learning Adopted Human Activity Classification using Shimmer Wearable Sensors for Health Monitoring

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import joblib

import os

from sklearn.utils import resample

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

from imblearn.over\_sampling import SMOTE

from sklearn.ensemble import GradientBoostingClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report,precision\_score,recall\_score,f1\_score

from sklearn.preprocessing import LabelEncoder

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_csv('dataset.csv')

df.head()

df.shape

df['activity'].unique()

df.describe()

df.info()

df.isnull().sum()

# Create the count plot

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

sns.countplot(data=df, x='activity')

plt.title('Count Plot for Activity')

plt.xlabel('Activity')

plt.ylabel('Count')

plt.show()

# Resample the DataFrame to have 10,000 samples

df\_resampled = resample(df, replace=True, n\_samples=10000, random\_state=42)

# Create the count plot

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

sns.countplot(data=df\_resampled, x='activity')

plt.title('Count Plot for Activity')

plt.xlabel('Activity')

plt.ylabel('Count')

plt.show()

label\_encoder = LabelEncoder()

# Fit and transform the activity column

df\_resampled['activity'] = label\_encoder.fit\_transform(df\_resampled['activity'])

df\_resampled.head()

df\_resampled.shape

df\_resampled['activity'].unique()

# Specifying dependent and independent variables

# Separate features and target

X = df\_resampled.drop('activity', axis=1)

y = df\_resampled['activity']

print("Length of X:", len(X))

print("Length of y:", len(y))

X

y

# Data Splitting

X\_train,X\_test,y\_train,y\_test= train\_test\_split(X,y,test\_size=0.2,random\_state=77 )

smote = SMOTE(sampling\_strategy='auto', random\_state=42)

X\_train ,y\_train = smote.fit\_resample(X\_train,y\_train)

X\_train.shape

y\_train.shape

labels = ['bending', 'cycling', 'sitting', 'standing', 'walking'] # The corresponding names for the labels

#defining global variables to store accuracy and other metrics

precision = []

recall = []

fscore = []

accuracy = []

#function to calculate various metrics such as accuracy, precision etc

def calculateMetrics(algorithm, testY,predict):

testY = testY.astype('int')

predict = 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()

# GradientBoostingClassifier

if os.path.exists('model/GradientBoostingClassifier.pkl'):

# Load the trained model from the file

gbb = joblib.load('model/GradientBoostingClassifier.pkl')

print("Model loaded successfully.")

predict = gbb.predict(X\_test)

calculateMetrics("GradientBoostingClassifier", predict, y\_test)

else:

# Train the model (assuming X\_train and y\_train are defined)

gbb = GradientBoostingClassifier()

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

# Save the trained model to a file

joblib.dump(gbb, 'model/GradientBoostingClassifier.pkl')

print("Model saved successfully.")

predict = gbb.predict(X\_test)

calculateMetrics("GradientBoostingClassifier", predict, y\_test)

# XGBClassifier

if os.path.exists('model/XGBClassifier.pkl'):

# Load the trained model from the file

xgbc = joblib.load('model/XGBClassifier.pkl')

print("Model loaded successfully.")

predict = xgbc.predict(X\_test)

calculateMetrics("XGBClassifier", predict, y\_test)

else:

# Train the model (assuming X\_train and y\_train are defined)

xgbc = XGBClassifier()

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

# Save the trained model to a file

joblib.dump(xgbc, 'model/XGBClassifier.pkl')

print("Model saved successfully.")

predict = xgbc.predict(X\_test)

test = pd.read\_csv('test.csv')

#dataset = dataset.fillna(0)

object\_cols1 = test.select\_dtypes(include=['object']).columns

test[object\_cols1] = test[object\_cols1].fillna('Unknown')

# Apply Label Encoding to each object column

label\_encoders = {}

for col in object\_cols1:

# Fill null values with a placeholder, e.g., 'Unknown'

# dataset[col] = dataset[col].fillna('Unknown')

test = test.fillna(0)

# Initialize and fit the LabelEncoder

le = LabelEncoder()

test[col] = le.fit\_transform(test[col])

# Store the label encoder for future use

label\_encoders[col] = le

# Display the transformed dataset

print("Transformed Dataset:")

print(test)

# Define your labels

labels = ['bending', 'cycling', 'sitting', 'standing', 'walking']

# Make predictions

predict = xgbc.predict(test)

# Loop through each prediction and print the corresponding row for the first 10 rows

for i, p in enumerate(predict[:10]): # Slice to limit to first 10 rows

label = labels[p] # Map the prediction to the label

print(f"Row {i}: {test.iloc[i]}") # Print the row content

print(f"Row {i}: Prediction: {label}")

test['predict']=predict

test

**CHAPTER 10**

**RESULTS AND CONCLUSION**

**10.1 Implementation Description**

**1. Importing Libraries:**The code 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.

**2. Dataset Loading and Exploration:**The dataset is loaded from a CSV file named dataset.csv into a pandas DataFrame.

- Initial exploration of the dataset is done by checking its shape, structure, and the presence of any missing values.

**3. Data Visualization:**

- A count plot of the target variable activity is generated to visualize the distribution of different activity classes. This helps in understanding the class balance in the dataset.

**4. 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.

**5. 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.

**6. 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. SMOTE is applied to the training data to handle class imbalance.

**7. Model Building and Evaluation:**

**GradientBoostingClassifier:**

- This classifier is trained using the training data, and its performance is evaluated on the test data. Key metrics such as accuracy, precision, recall, and F1-score are calculated.

**Logistic Regression:**

Logistic regression is applied to the training data and evaluated using similar metrics. It serves as a baseline model for comparison with more complex algorithms.

**RandomForestClassifier:** This ensemble method is trained and evaluated to understand its performance in detecting fraudulent activities. It leverages the power of multiple decision trees to improve prediction accuracy.

**StackingClassifier:**A StackingClassifier is used to combine the predictions from multiple models to improve overall performance. It leverages the strengths of different algorithms to produce a robust final prediction.

**8. Model Serialization:**

The trained models are serialized using joblib for future use. This allows for the models to be saved and loaded efficiently, facilitating easy deployment and usage in production environments.

**9. Prediction on New Data:**

The final step involves using the trained models to make predictions on new data. This step demonstrates the practical application of the models in a real-world scenario, ensuring they can accurately identify fraudulent activities.

**10.2Dataset Description**

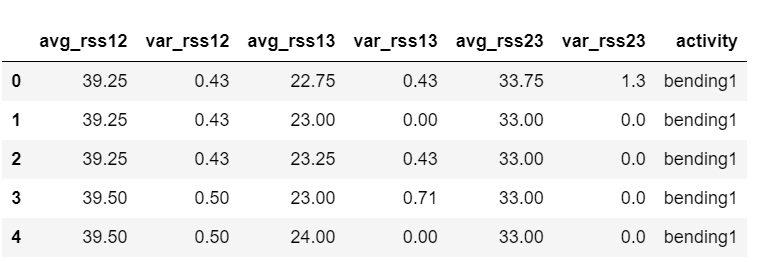


Fig10.1: Dataset description

Human activity classification using SHIMMER wearable sensors involves analyzing detailed sensor data to accurately identify various activities. The dataset provided includes six features: avg\_rss12, var\_rss12, avg\_rss13, var\_rss13, avg\_rss23, var\_rss23, and an activity label. These features represent the average and variance of received signal strength (RSS) between different sensor pairs (12, 13, and 23), which are critical in distinguishing between activities such as 'bending1.' For example, the first few rows of data show that during the 'bending1' activity, avg\_rss12 values are consistently around 39.25 to 39.50, with low variance, while avg\_rss13 and avg\_rss23 values range from 22.75 to 24.00 and 33.00 to 33.75, respectively. These specific signal patterns and variances captured by the SHIMMER sensors are key to the machine learning models' ability to classify and monitor human activities accurately.

**10.3 Results Description**

The count plot for the 'Activity' variable provides a visual representation of the frequency of each activity category within the dataset. This plot, created using seaborn's `countplot` function, displays the number of occurrences for each distinct activity, allowing us to easily identify the most and least common activities recorded by the SHIMMER sensors. The x-axis represents the different activity types, while the y-axis indicates the count of each activity. This visualization is essential for understanding the distribution of activities and can highlight any imbalances in the dataset that may need to be addressed during the machine learning modeling process.

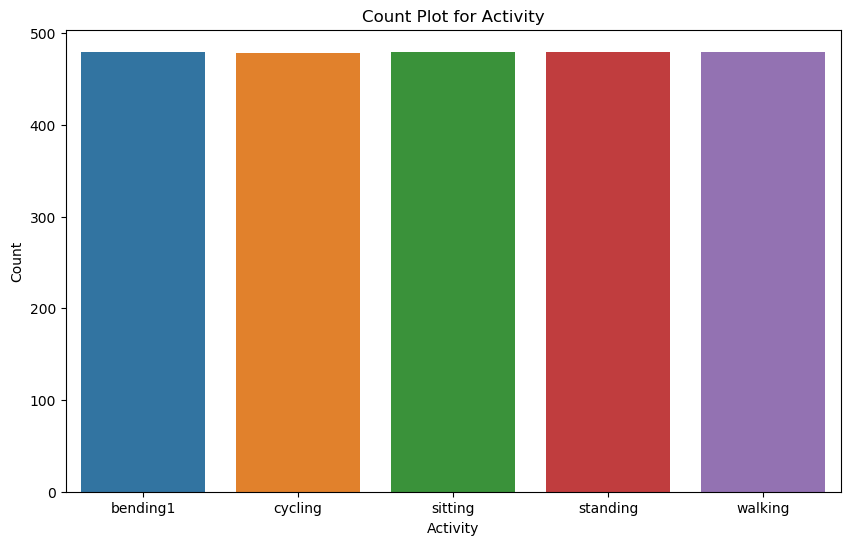


Fig10.2: Count Plot for Different activities

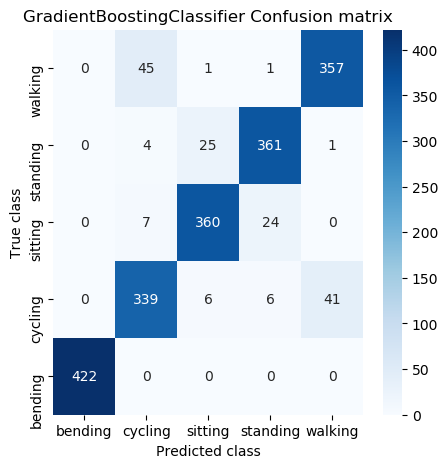


Fig 10.3: Gradient Boosting Classifier of confusion matrix

The code snippet first checks if a pre-trained model for `GradientBoostingClassifier` exists in the file system. If the model file is found, it loads the model using `joblib`, prints a confirmation message, and then uses the model to make predictions on `X\_test`, followed by calculating metrics with the `calculateMetrics` function. If the model file is not found, it trains a new `GradientBoostingClassifier` on the provided training data (`X\_train` and `y\_train`), saves this trained model to a file, and prints a success message. The newly trained model is then used to predict outcomes on `X\_test`, and the metrics are calculated.

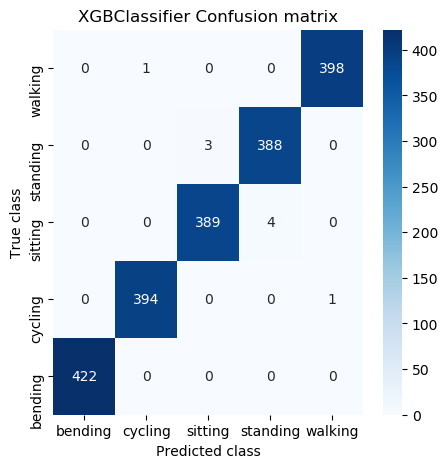


Fig 10.4: XGBClassifier of confusion matrix

The code checks if a pre-trained model file `XGBClassifier.pkl` exists in the `model` directory. If the file is present, it loads the model using `joblib`, prints a success message, and then makes predictions on `X\_test`, followed by calling `calculateMetrics` to evaluate the model's performance. If the file is not found, it trains a new `XGBClassifier` model using `X\_train` and `y\_train`, saves the trained model to `XGBClassifier.pkl` using `joblib`, prints a success message, and then predicts outcomes for `X\_test`.

**CHAPTER 11**

**CONCLUSION AND FUTURE SCOPE**

The integration of machine learning techniques with Shimmer wearable sensors for human activity classification represents a significant advancement in health monitoring systems. The proposed approach not only improves the accuracy and efficiency of activity recognition but also enhances the potential for personalized health insights. By employing sophisticated algorithms such as XGBoost and Gradient Boosting, the system can adapt to the complexities of human behavior, providing real-time feedback that is invaluable for users seeking to monitor and improve their health outcomes.

Looking to the future, there are several avenues for further development. First, expanding the dataset to include a broader range of activities and diverse populations could enhance model robustness and generalizability. Additionally, incorporating real-time data processing capabilities could facilitate immediate feedback for users, improving engagement and adherence to health recommendations. Future research could also explore the integration of multimodal data sources, such as physiological measurements alongside activity data, to provide a more comprehensive understanding of health and well-being. Finally, investigating the application of deep learning techniques may further enhance classification accuracy and open up new possibilities for health monitoring solutions.

**REFERENCES**

[1] J. Hayano, H. Yamamoto, I. Nonaka, M. Komazawa, K. Itao, N. Ueda, H. Tanaka, E. Yuda. Quantitative detection of sleep apnea with wearable watch device.PLoS ONE, 15 (2020), e0237279. [10.1371/journal.pone.0237279](https://doi.org/10.1371/journal.pone.0237279).

[2] F. Delmastro, F.D. Martino, C. Dolciotti. Cognitive training and stress detection in MCI frail older people through wearable sensors and machine learning.IEEE Access, 8 (2020), pp. 65573. [10.1109/access.2020.2985301](https://doi.org/10.1109/access.2020.2985301).

[3] M.V. Perez, K.W. Mahaffey, H. Hedlin, J.S. Rumsfeld, A. Garcia, F. Todd, V. Balasubramanian, A.M. Russo, A. Rajmane, L. Cheung, G. Hung, J. Lee, K. Peter, N. Talati, D. Nag, S.E. Gummidipundi, A. Beatty, M.T. Hills, S. Desai, C.B. Granger, M. Desai, M.P. Turakhia. Large-Scale Assessment of a smartwatch to identify atrial fibrillation.New England Journal of Medicine, 381 (2019), pp. 1909–1917.

[4] J.S. Chorba, A.M. Shapiro, L. Le, J. Maidens, J. Prince, S. Pham, M.M. Kanzawa, D.N. Barbosa, C. Currie, C. Brooks, B.E. White, A. Huskin, J. Paek, J. Geocaris, D. Elnathan, R. Ronquillo, R. Kim, Z.H. Alam, V.S. Mahadevan, S.G. Fuller, G.W. Stalker, S.A. Bravo, D. Jean, J.J. Lee, M. Gjergjindreaj, C.G. Mihos, S.T. Forman, S. Venkatraman, P.M. McCarthy, J.D. Thomas. Deep learning algorithm for automated cardiac murmur detection via a digital stethoscope platform.Journal of American Heart Association, 10 (2021), e019905. [10.1161/JAHA.120.019905](https://doi.org/10.1161/JAHA.120.019905).

[5] S. Seneviratne, Y. Hu, T. Nguyen, G. Lan, S. Khalifa, K. Thilakarathna, M. Hassan, A. Seneviratne. A survey of wearable devices and challenges.IEEE Communications Surveys &Tutorials, 19 (2017), pp. 2573–2620. [10.1109/comst.2017.2731979](https://doi.org/10.1109/comst.2017.2731979).

[6] M. Chan, D. Estève, J.-Y. Fourniols, C. Escriba, E. Campo. Smart wearable systems: current status and future challenges.Artificial Intelligence in Medicine, 56 (2012), pp. 137–156. [10.1016/j.artmed.2012.09.003](https://doi.org/10.1016/j.artmed.2012.09.003).

[7] P. Siirtola, H. Koskimäki, H. Mönttinen, J. Röning. Using sleep time data from wearable sensors for early detection of migraine attacks.Sensors, 18 (2018), no. 5. [10.3390/s18051374](https://doi.org/10.3390/s18051374).

[8] C. Meisel, R. El Atrache, M. Jackson, S. Schubach, C. Ufongene, T. Loddenkemper. Machine learning from wristband sensor data for wearable, noninvasive seizure forecasting.Epilepsia, 61 (2020), no. 12, pp. 2653–2666. [10.1111/epi.16719](https://doi.org/10.1111/epi.16719).

[9] A.Y. Hannun, P. Rajpurkar, M. Haghpanahi. Cardiovascular arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network.Nature Medicine, 25 (2019), pp. 65–69. [10.1038/s41591-018-0268-3](https://doi.org/10.1038/s41591-018-0268-3).

[10] S. Kwon, J. Hong, E. Choi, B. Lee, C. Baik, E. Lee, E. Jeong, B. Koo, S. Oh, Y. Yi. Detection of atrial fibrillation using a ring-type wearable device (CardioTracker) and deep learning analysis of photoplethysmography signals: prospective observational proof-of-concept.J Med Internet Res, 22 (2020), no. 5.

[11] Z. Mei, X. Gu, H. Chen, W. Chen. Automatic atrial fibrillation detection based on heart rate variability and spectral features.IEEE Access, 6 (2018), pp. 53566. [10.1109/access.2018.2871220](https://doi.org/10.1109/access.2018.2871220).

[12] N. Rashid, M.A. Al Faruque. Energy-efficient real-time myocardial infarction detection on wearable devices.Annu Int Conf IEEE Eng Med Biol Soc., 2020 (2020), pp. 4648–4651. [10.1109/EMBC44109.2020.9175232](https://doi.org/10.1109/EMBC44109.2020.9175232).

[13] R. Buettner, J. Frick, T. Rieg. High-performance Detection of Epilepsy in Seizure-free EEG Recordings.Proceedings of the International Conference on Information Systems, December (2019), Munich, Germany.

[14] C. Ieracitano, N. Mammone, A. Hussain, C. Francesco, A. Morabito. A novel multi-modal machine learning based approach for automatic classification of EEG recordings in dementia.Neural Networks, 123 (2020), pp. 176–190. [10.1016/j.neunet.2019.12.006](https://doi.org/10.1016/j.neunet.2019.12.006).

[15] S. Hwang, H. Jebelli, B. Choi, M. Choi, S.H. Lee. Measuring workers’ emotional state during construction tasks using wearable EEG.Journal of Construction Engineering and Management, 144 (2018). [10.1061/(asce)co.1943-7862.0001506](https://doi.org/10.1061/(asce)co.1943-7862.0001506).