

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfilment of the requirements for the award of
Bachelor of Technology Degree in

Information Technology

By

NANDHITHA.B
(Reg.No.37120051)



DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade “A” by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119



SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

(Established under Section 3 of UGC Act, 1956)

Jeppiar Nagar, Rajiv Gandhi Salai

Chennai -600119 www.sathyabama.ac.in



DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **NANDHITHA B** (Reg.no.37120051) who carried out the project entitled “**Real Time Non Invasive Continuous Glucose Monitoring System Using AI**” under our supervision from DEC 2020 to JAN 2021.

Internal Guide

Ms.A.Sivasangari M.E.,(Ph.D.,)

Head of the Department

Dr.R SUBHASHINI M.E,Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I , NANDHITHA.B (Reg.No.37120051) hereby declare that the Project Report entitled “**Real Time Non Invasive Continuous Glucose Monitoring System Using AI**” done by me under the guidance of **Ms.A.SIVASANGARI M.E.,Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Information Technology .

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** or their encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr.T. SASIKALA M.E Ph .D**, Dean, School Of Computing and **Dr.R SUBHASHINI M.E Ph. D** Head of the Department, Dept .of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms.A.SIVASANGARI M.E.,Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non teaching staff members of the Department of Information Technology who were helpful in many ways for the completion of the project.

ABSTRACT

Diabetes mellitus is one of the most common chronic diseases. Blood glucose level elevated for a long period of time can result in vascular, neurological or metabolic complications, to prevent such complications, a strict control over blood glucose level is needed. The goal of our proposal is to determine the future blood glucose value of a diabetic patient using an Artificial Neural Network. Feature extraction procedures were implemented (Diabetic Dynamic Model) on blood glucose time series data, in order to extract knowledge on how patient blood glucose level will change according to the external food intake and other human activities. These changes can be monitored using tear strips and the data collected is compared with the data collected from healthy individuals. A classification algorithm is applied to the data collected to classify the type of the patient and detect if the person has diabetes or is just showing subtle signs of diabetes.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF FIGURES	9
1.	INTRODUCTION	1
	1.1 OVERVIEW	1
	1.1.1 SCOPE OF THE PROJECT	2
	1.2 DOMAIN OVERVIEW	2
	1.2.1 ARTIFICIAL INTELLIGENCE(AI)	2
2.	LITERATURE SURVEY	5
	2.1 SALIVARY ANALYSIS	5
	2.2 BREATH ANALYSIS	5
	2.3 TEAR ANALYSIS	6
3	SYSTEM ANALYSIS	7
	3.1 EXISTING SYSTEM	7
	3.1.1 CONSTRAINTS	8
	3.2 PROPOSED SYSTEM	8
	3.2.1 ADVANTAGES	9
4	REQUIREMENT SPECIFICATION	10

4.1	INTRODUCTION	10
4.1.1	HARDWARE SOFTWARE REQUIREMENT	10
5	SYSTEM DESIGN	18
5.1	INTRODUCTION	18
5.2	ARCHITECTURE DIAGRAM	18
5.3	DATA FLOW DIAGRAM	19
5.4	UML DIAGRAMS	20
5.4.1	USE CASE DIAGRAM	21
5.4.2	CLASS DIAGRAM	21
5.4.3	SEQUENCE DIAGRAM	21
5.4.4	ACTIVITY DIAGRAM	23
5.4.5	COMPONENT DIAGRAM	25
6	SYSTEM IMPLEMENTATION	25
6.1	INTRODUCTION	25
6.2	LIST OF MODULES	26
6.3	MODULE IMPLEMENTATION	26
6.3.1	COLLECTION OF DATASET	26
6.3.2	DATA PREPROCESSING	26
6.3.3	CLASSIFICATION AND TRAINING	27
7	SOFTWARE TESTING	31
7.1	INTRODUCTION	31
7.2	TESTING OBJECTIVES	31
7.3	TYPES OF TESTING	31
7.3.1	WHITEBOX TESTING	32
7.3.2	BLACKBOX TESTING	32
7.3.3	UNIT TESTING	33
7.3.4	INTEGRATION TESTING	33
7.3.5	FUNCTIONAL TESTING	33

7.3.6	VALIDATION TESTING	34
4.1.1	HARDWARE SPECIFICATION	10
1.	HARDWARE DESCRIPTION	10
i.	SOFTWARE SPECIFICATION	13
1.	SOFTWARE SPECIFICATION	13

7.3.7	ACCEPTANCE TESTING	34
7.3.8	PERFORMANCE TESTING	35
7.3.9	END-TO-END TESTING	36
7.3.10	OUTPUT TESTING	36

8	CONCLUSION AND FUTURE ENHANCEMENT	37
8.1	CONCLUSION	37
8.2	FUTURE ENHANCEMENT	37

	APPENDIX I (SCREENSHOTS)	38
--	---------------------------------	-----------

	APPENDIX II (SAMPLE CODING)	40
--	------------------------------------	-----------

	REFERENCES	46
--	-------------------	-----------

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
3.1.1	Existing System Architecture	14
4.2.1.1	Non invasive Self-diagnostic Device for Tear Collection and Glucose Measurement	16
4.2.1.3	Glucose sensor	24
4.2.1.4	Overall Glucose Test Circuit	24
5.2.1	System architecture	25
5.3.1	Data Flow Diagram	26
5.3.2	Use Case Diagram	27
5.3.3	Class Diagram	28
5.3.4	Activity Diagram	30
5.3.5	Component Diagram	31
6.2.1	Layers used to build RNN	35
6.2.2	RNN Architecture	35
6.2.3	Long short-term memory (LSTM)	36
7.3.1	Types of Testing	38

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The blood glucose value of a diabetic patient is determined using an Artificial Neural Network. Feature extraction procedures will be implemented (Diabetic Dynamic Model) on blood glucose time series data, to extract knowledge on how patient blood glucose levels will change according to the external food intake and other human activities. An Artificial Neural Network is then trained using these features to predict the future value of blood glucose level with acceptable accuracy. We will develop an AI-based edge-intelligent hypoglycemia prediction system for the environment with low-periodic blood glucose levels. By using Long-Short-Term Memory (LSTM), a specialized neural network for handling time series data is used that includes alternate learning and inference. The performance of the system is evaluated using the method of the classification problem. The accuracy is improved by performing learning and prediction alternatively. The trained Artificial Neural Network (ANN) is then used to predict the future value of blood glucose concentration every 30 minutes based on the current value. A continuous non-invasive blood glucose monitoring system would provide an alternative to the present discrete methods of glucose determination and is expected to reduce the long term complications of diabetes while minimizing hypoglycemia and inconvenience. The advantage of this type of glycemic control is that measurements with a sensor are continuous but blood glucose testing with finger sticking is intermittent.

1.2 SCOPE OF THE PROJECT

The main contributions of this project therefore are:

- Experimentation on an open dataset having data of patients with diabetic and non-diabetic.
- Detailed analysis of continuous glucose monitoring of patients.
- Detailed evaluation of machine learning algorithms for patient classification.
- Validation and testing of new patients.

1.3 DOMAIN OVERVIEW

1.3.1 ARTIFICIAL INTELLIGENCE

Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think. AI becomes an expert by learning how the human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this learning as a basis for developing intelligent software and systems. Many AI algorithms are capable of learning from data; they can improve themselves by learning new heuristics (strategies, or "rules of thumb", that have worked well in the past), or can themselves write other algorithms. Some of the "learners" described below, including Bayesian networks, decision trees, and nearest-neighbour, could theoretically, if given infinite data, time, and memory, learn to approximate any function, including whatever combination of mathematical functions would best describe the entire world. These learners could therefore, in theory, derive all possible knowledge, by considering every possible hypothesis and matching it against the data. In practice, it is almost never possible to consider every possibility, because of the phenomenon of "combinatorial explosion", where the amount of time needed to solve a problem grows exponentially. Much of AI research involves figuring out how to identify and avoid such problems considering broad swaths of possibilities that are unlikely to be fruitful.

Neural networks, or neural nets, were inspired by the architecture of neurons in the human brain. A simple "neuron" N accepts input from multiple other neurons, each of which, when activated (or "fired"), cast a weighted "vote" for or against whether neuron N should itself activate. Learning requires an algorithm to adjust these weights based on the training data; one simple algorithm (dubbed "fire together, wire together") is to increase the weight between two connected neurons when the activation of one triggers the successful activation of another. The net forms "concepts" that are distributed

among a sub network of shared neurons that tend to fire together; a concept meaning "leg" might be coupled with a sub network meaning "foot" that includes the sound for "foot". Neurons have a continuous spectrum of activation; in addition, neurons can process inputs in a nonlinear way rather than weighing straightforward votes. Modern neural nets can learn both continuous functions and, surprisingly, digital logical operations. Neural networks' early successes included predicting the stock market and (in 1995) a mostly self-driving car.

Artificial intelligence is breaking into the healthcare industry by assisting doctors. According to Bloomberg Technology, Microsoft has developed AI to help doctors find the right treatments for cancer. There is a great amount of research and drugs developed relating to cancer. In detail, there are more than 800 medicines and vaccines to treat cancer. This negatively affects the doctors, because there are too many options to choose from, making it more difficult to choose the right drugs for the patients. Microsoft is working on a project to develop a machine called "Hanover". Its goal is to memorize all the papers necessary to cancer and help predict which combinations of drugs will be most effective for each patient. One project that is being worked on at the moment is fighting myeloid leukemia, a fatal cancer where the treatment has not improved in decades. Another study was reported to have found that artificial intelligence was as good as trained doctors in identifying skin cancers. Another study is using artificial intelligence to try and monitor multiple high-risk patients, and this is done by asking each patient numerous questions based on data acquired from live doctor to patient interactions. One study was done with transfer

learning, the machine performed a diagnosis similarly to a well-trained ophthalmologist, and could generate a decision within 30 seconds on whether or not the patient should be referred for treatment, with more than 95% percent accuracy

Some medical conditions are easy to diagnose. Measles give you a rash. Chest pain might indicate a heart attack. But the symptoms of depression can be harder to pinpoint. Medical professionals currently use the Patient Health Questionnaire (PHQ) designed by the American Psychological Association to diagnose depression. The brief form asks patients about their level of interest in regular activities, appetite and eating habits, ability to concentrate, and other queries designed to detect depression. The questions are based on criteria from the Diagnostic and Statistical Manual of Mental Disorders (DSM).

CHAPTER 2

LITERATURE SURVEY

NON INVASIVE SAMPLE BASED GLUCOSE MONITORING

There are techniques that require a sample, e.g., saliva, tears, urine, breath, etc., to estimate the glucose concentration in the blood and are termed as sample-based techniques.

2.1 SALIVARY ANALYSIS

The biochemical markers found in saliva are believed to represent the human body as well as a mirror. Using saliva for diagnostic or monitoring purposes has a lot of advantages such as convenience, cost effectiveness, accuracy, sensitivity, and portability, but the correlation varies for every individual, the saliva viscosity is affected by the environment, also there is lag in the glucose concentration changes between the blood and the saliva [19]. Though the clinical trials were carried out accurately, the sample acquisition procedure is a bit complex and time consuming. The Overall procedure is costly, energy consuming, and will take about 90 s to measure the glucose concentration.

2.2 BREATH ANALYSIS

The amount of acetone in one's breath is directly proportional to the blood glucose concentration, which means breath analysis can be used to estimate blood glucose levels. The change in the conductivity of a set of sensors sensitive to different components/elements in the breath can be used for signal acquisition. The data are labeled with four levels according to the glucose concentration. The same data signals were processed using support vector machine and sparse-representation-based classification to compare the performance and accuracy of SVOR. SVOR was found to be the most accurate among the three. The sample acquisition method used is fast, simple, and relatively low in cost as compared to the alternate/conventional method but transmitting data first to a computer for further processing is very time

consuming. The processing algorithm is better suited for such classification but it does not work very well with the dataset outliers and is a bit complicated as compared to alternate machine learning algorithms; thus, consume more resources.

2.3 TEAR ANALYSIS

Tears can be used to measure the blood glucose concentration as they have prominent biomarkers same as in the blood. Both saliva and onion induced tears are used to estimate the blood glucose levels by centrifugation process. The results from the tear samples were found to be more accurate as compared to the saliva samples. The proposed method uses a complicated procedure to remove proteins from the collected samples also the volume of the samples were found to be directly proportional to the accuracy of the proposed method. The procedure to get to the final results is time consuming and complex.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In the existing system, exhaled human breath is composed of a multitude of volatile organic compounds (VOC) which act as promising biomarkers to many pathological disorders thereby differentiating diseased individuals from healthy subjects. Among these VOCs, breath acetone ($(\text{CH}_3)_2\text{CO}$) concentration is found to remain elevated in diabetic patients. Researchers have reported that the normal acetone level in exhaled breath in a healthy human subject varies between 0.22 ppm–0.80 ppm. This level rises to 1.76 ppm–3.73 ppm in a subject diagnosed with type-2 diabetes and it soars to as high as 21 ppm in a type-1 diabetic case. It explores the use of a one-dimensional (1-D) modified convolution neural network (CNN) algorithm that combines feature extraction and classification techniques and to apply a modified 1-D CNN on real-time breath signals obtained from an array of gas sensors. The experimentation and the performance of the system is carried out and evaluated.

3.1.1 CONSTRAINTS

- Low effectiveness and less accuracy of the algorithm.
- Minor interference in the response obtained from the MOS sensors due to the effect of moisture present in breath has not been considered.
- It does not work very well with the dataset outliers and is a bit complicated as compared to alternate machine learning algorithms; thus, consume more resources

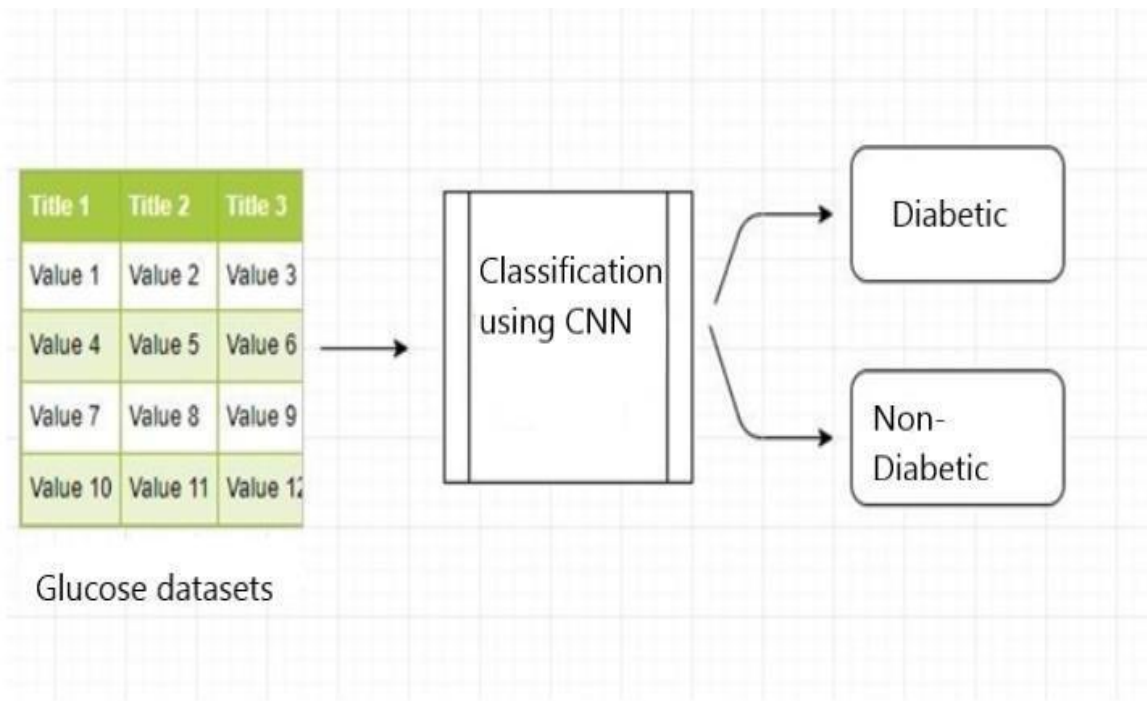


Figure 3.1.1 Existing System Architecture

3.2 PROPOSED SYSTEM

In the proposed system, Neural Network & Models is implemented for best accuracy purpose. The BG data used as the input was time series data. For this reason, LSTM , which is specialized for handling time series data was adopted. The input and output layer was composed of one unit. The hidden layer consisted of 30 units in one layer. LSTM handled the time series data well by referring to the past hidden layer. In this experiment, we referred to hidden layers up to 30 later. The neural network consisted of several interconnected units and nodes. Learning involved updating the values of these weights. In the machine learning process, it is known that performance is improved by normalizing inputs. Therefore, the BG data, used as input to LSTM, was normalized according to the following formula. Normalized data = $(y - \min.) / (\max. - \min.)$. The system continued to normalize the measured BG levels.

3.2.1 ADVANTAGES

- Non invasive, self-diagnostic device that enables safe tear collection and glucose measurement.
- This method is pain-free and customized calibration can be done.
- An AI-based edge-intelligent hypoglycemia prediction system will be developed for the environment experiencing low-periodic blood glucose levels.
- Long-Short-Term Memory (LSTM), a specialized neural network for handling time series data is used to perform alternate learning and inference

REQUIREMENT SPECIFICATION

4.1 INTRODUCTION

The requirement specification is a technical specification of requirements for the software products. It lays out functional and nonfunctional requirements. It provides the requirements of a particular software system including functional, performance and security requirements. This describes the project's target audience and the user interface, hardware and software requirements.

4.2 HARDWARE AND SOFTWARE SPECIFICATION

4.2.1 HARDWARE SPECIFICATIONS

Tear Strips

Sensors: Glucose sensor

Hard Disk: 500GB

RAM: 4GB

Processor: Intel i5

4.2.1.1 HARDWARE DESCRIPTION 4.2.1.2 TEAR STRIPS

We propose a non-invasive, self-diagnostic device that enables safe tear collection and glucose measurement. The device described herein was manufactured by tight assembly of a lid for tear collection in conjunction with a strip-type glucose sensor. The lid was designed to be in contact with the inferior palpebral conjunctiva for tear collection and was thus designed to possess a proper contact area and rounded boundaries to avoid eye tissue damage. For the strip-type glucose sensor, we employed a commercially available electrochemical sensor (Accu-Chek test strips), which was modified to reduce the volume of the reaction chamber (0.4 μ l) for a small amount of collected tear fluid. When tested with *in vivo* animal models, the device was able to collect tear fluid in a relatively short time (<2 s) without causing eye tissue damage, and the device allowed the collected tear fluid to be delivered to the sensor for measurement of tear glucose concentrations. The blood glucose concentrations estimated with the tear glucose

concentrations obtained with the device exhibited a high correlation with those actually measured with a clinically available glucometer ($R^2 = 0.9617$).

(a) Assembly of the lid and-modified strip-type glucose sensor.

(b) Tear-glucose device connected to the potentiostat.

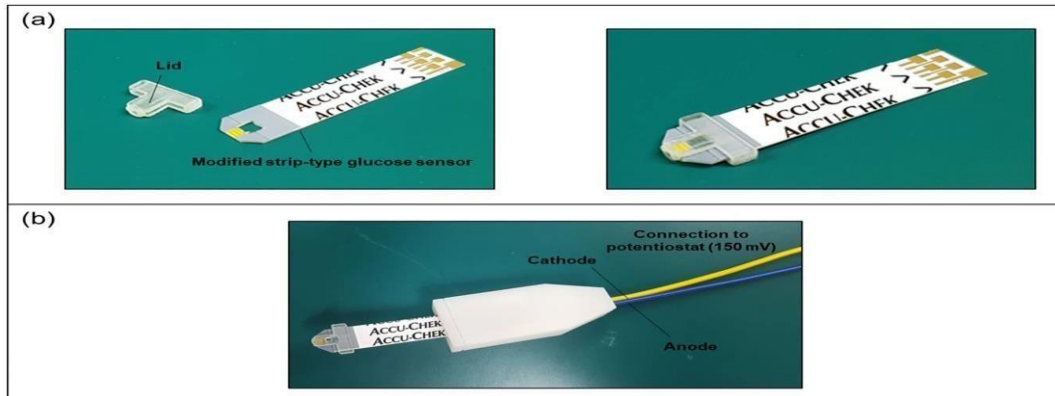


Figure 4.2.1.2 Noninvasive Self-diagnostic Device for Tear Collection and Glucose Measurement

4.2.1.3 GLUCOSE SENSOR

Glucose sensors are used to measure the blood glucose concentration of a patient and are an important part of managing diabetes mellitus. The majority of blood glucose sensors, or glucose meters, are categorized as aerometric sensors. In aerometric glucose sensors, reducing property of glucose is measured as a current. Sensors contain electrodes to measure the current generated by an enzymatic reaction usually between glucose, an enzyme, and a mediator.

Glucose sensor applies the working principle of first-generation glucose sensors by detecting the concentration of glucose oxidase. However, instead of measuring it through electrochemical methods, the device uses capacitive sensing to detect changes in the dielectric constant of the material. The capacitive sensor is formed using interdigitated composite gold and oxide electrodes placed on a silicon substrate. The glucose oxidase enzyme converts glucose and oxygen into gluconolactone and hydrogen peroxide. Oxygen is derived from water surrounding

the sensor. When glucose is oxidized, the concentration of water decreases and the concentration of hydrogen peroxide increases, resulting in a change in the dielectric constant of the area between the sensing electrodes. The dielectric constant of water and hydrogen peroxide is 78 and 60, respectively. Measurement of the fluctuation of dielectric constants can be detected by observing the change in capacitance of the sensor. The sensing capacitors are connected to a series of oddnumbered inverters such that they form an oscillator circuit. The oscillator circuit generates a frequency output ranging from 17 to 25 MHz, depending on the sensor's capacitance. Using this simple circuit, the change in capacitance can now be easily monitored by observing the oscillator's output since the frequency change is proportional to the change in capacitance and thus the change in glucose concentration. The sensitivity of the glucose sensor was about 1.48 MHz mM⁻¹.

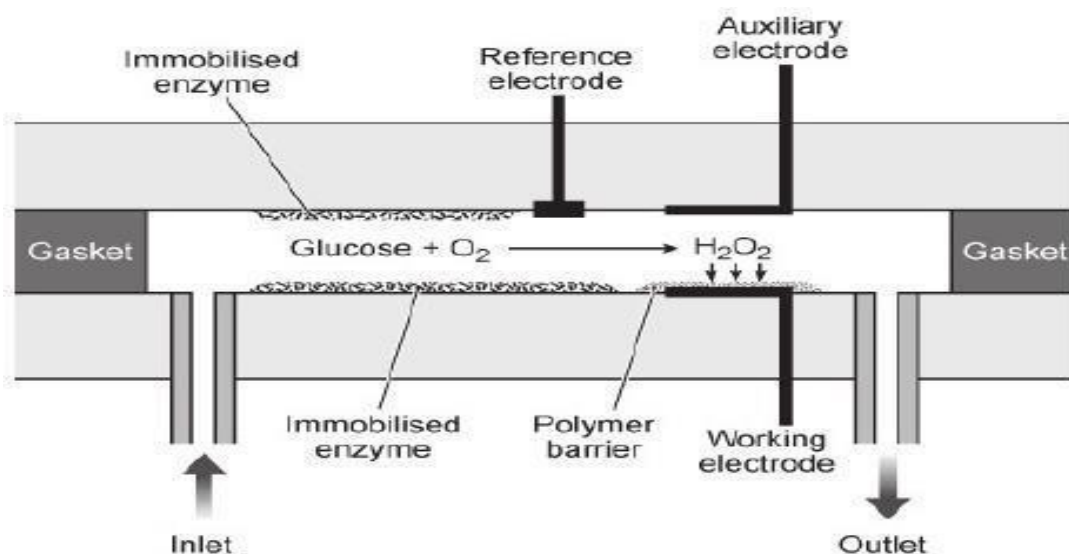


FIGURE 4.2.1.3Glucose sensor

- The above diagram shows the electrode configuration, the polymer barrier deposited onto the working electrode and the surface where the enzyme (glucose oxidase) is immobilized.

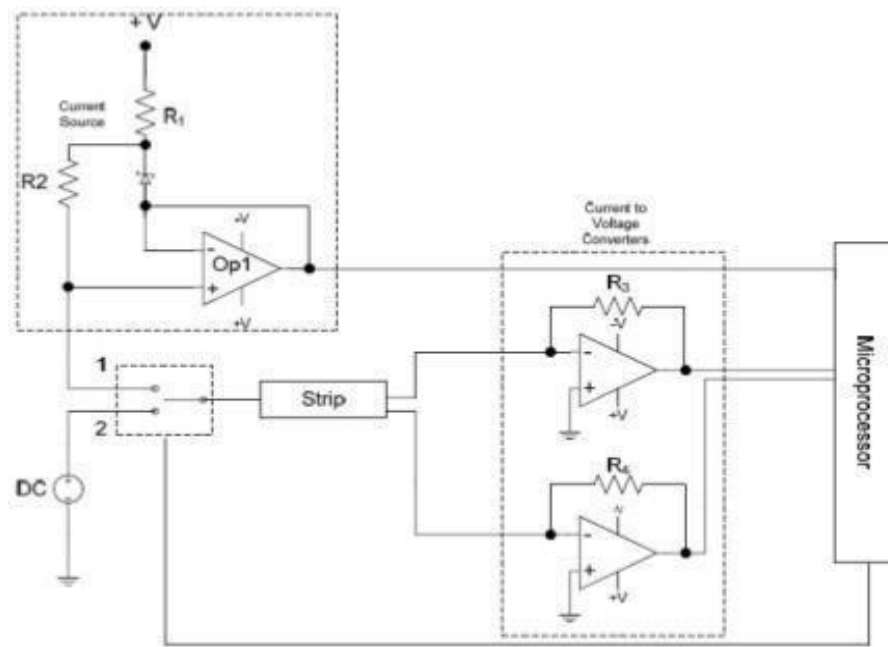


FIGURE 4.1.2.4 Overall Glucose Test Circuit

4.2.2 SOFTWARE SPECIFICATIONS

Operating System: window 10 (64 bit) Tool:

Anaconda Navigator

Language: Python

4.2.2.1 SOFTWARE DESCRIPTION

PYTHON

Python is an interpreter, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018. 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 interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software¹ and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation. Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming. Many other paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has `filter()`, `map()` and `reduce()` functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML. Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule. Python's large

standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation `wsgiref` follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

Most Python implementations (including CPython) include a read-eval-print loop (REPL), permitting them to function as a command line interpreter for which the user enters statements sequentially and receives results immediately. Other shells, including IDLE and IPython, add further abilities such as auto-completion, session state retention and syntax highlighting. As well as standard desktop integrated development environments, there are Web browser-based IDEs; SageMath (intended for developing science and math-related Python programs); Python Anywhere, browser-based IDE and hosting environment; and Canopy IDE, a commercial Python IDE emphasizing scientific computing.

Python IDLE

IDLE (short for integrated development environment or integrated development and learning environment) is an integrated development environment for Python, which has been bundled with the default implementation of the language since 1.5.2b1. It is packaged as an optional part of the Python packaging with many

Linux distributions. It is completely written in Python and the Tkinter GUI toolkit (wrapper functions for Tcl/Tk).

IDLE is intended to be a simple IDE and suitable for beginners, especially in an educational environment. To that end, it is cross-platform, and avoids feature clutter.

According to the included README, its main features are:

- Multi-window text editor with syntax highlighting, auto completion, smart indent and other.
- Python shell with syntax highlighting.
- Integrated debugger with stepping, persistent breakpoints, and call stack visibility.

IDLE has been criticized for various usability issues, including losing focus, lack of copying to clipboard feature, lack of line numbering options, and general user interface design; it has been called a "disposable" IDE, because users frequently move on to a more advanced IDE as they gain experience.

Author Guido van Rossum says IDLE stands for "Integrated Development Environment", and since van Rossum named the language Python partly to honor British comedy group Monty Python, the name IDLE was probably also chosen partly to honor Eric Idle, one of Monty Python's founding members.

ANACONDA NAVIGATOR

Anaconda is an open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS.

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator :

- JupyterLab
- Jupyter Notebook
- QtConsole
- Orange
- Rstudio
- Visual Studio Code

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The Conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

CHAPTER 5

SYSTEM DESIGN

5.1 INTRODUCTION

Design Engineering deals with the various UML (Unified Modelling language) diagrams for the implementation of project. Design is a meaningful engineering representation of a thing that is to be built. Software design is a process through which the requirements are translated into representation of the software. Design is the place where quality is rendered in software engineering. Design is the means to accurately translate customer requirements into finished product.

5.2 ARCHITECTURE DIAGRAM

A system architecture or systems architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

The architecture diagram of a system in which the principal parts of functions are represented by blocks connected by the lines that show the relationships of the blocks. The architecture diagram is typically used for a higher level, less detailed description aimed more at understanding the overall concepts and less at understanding the details of implementation. It shows the relationship between different components of a system. System architecture can comprise system components, the externally visible properties of those components, the relationships between them.

SYSTEM ARCHITECTURE

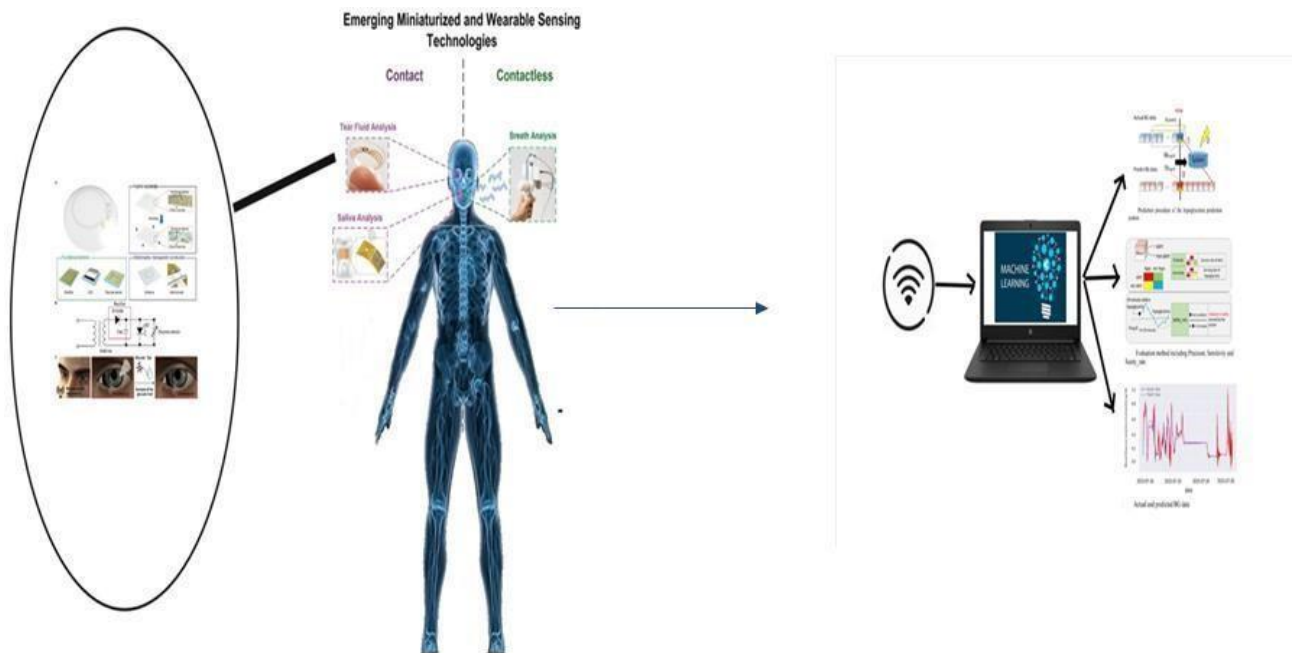


Fig no. 5.2.1 System architecture

A system's architecture can provide a plan from which products can be procured, and systems developed, that will work together to implement the overall system.

We present a system that identifies users at the risk of depression from their social media posts. The data have been divided into two parts: train and test. The part train for the implementation of neural network models used And the part test used to calculate the model's error. After calculating the error of each models, the comparison of these errors is discussed, then selecting the optimization model for diagnosis bipolar disorder

5.3 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the “flow” of data through an information system, modelling its process aspects. It differs from the flowchart as it shows the data flow instead of the control flow of the program. The DFD is designed to show how a system is divided into smaller portions and to highlight the flow of data between those parts. It concerns things like where the

data will come from and go to as well as where it will be stored. The graphical depiction identifies each source of data and how it interacts with other data sources to reach a common output. This type of diagram helps business development and design teams visualize how data is processed and identify or improve certain aspects.

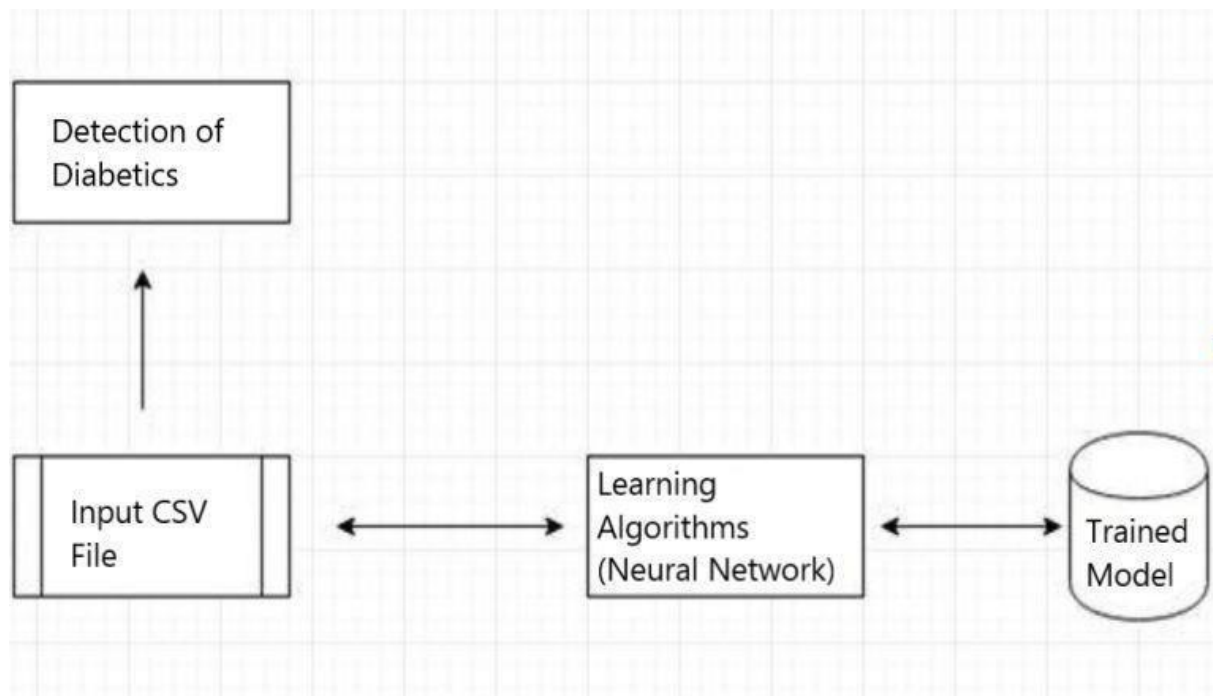


Figure.5.3.1 Data flow diagram

5.4 UML DIAGRAM

It can be described as a general purpose visual modeling language to visualize, specify, construct and document software system. Although UML is generally used to model software systems but it is not limited within this boundary. It is also used to model non software systems as well like process flow in a manufacturing unit etc.

5.4.1 USE CASE DIAGRAM

A use case diagram is a graph of actors, a set of use cases enclosed by

a system boundary, communication (participation) association between the actors and the use cases, and generalization among the use cases. A use case diagram shows the relationship among the actors and use cases within a system. A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The purpose of use case diagram is to capture the dynamic aspect of a system. When a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

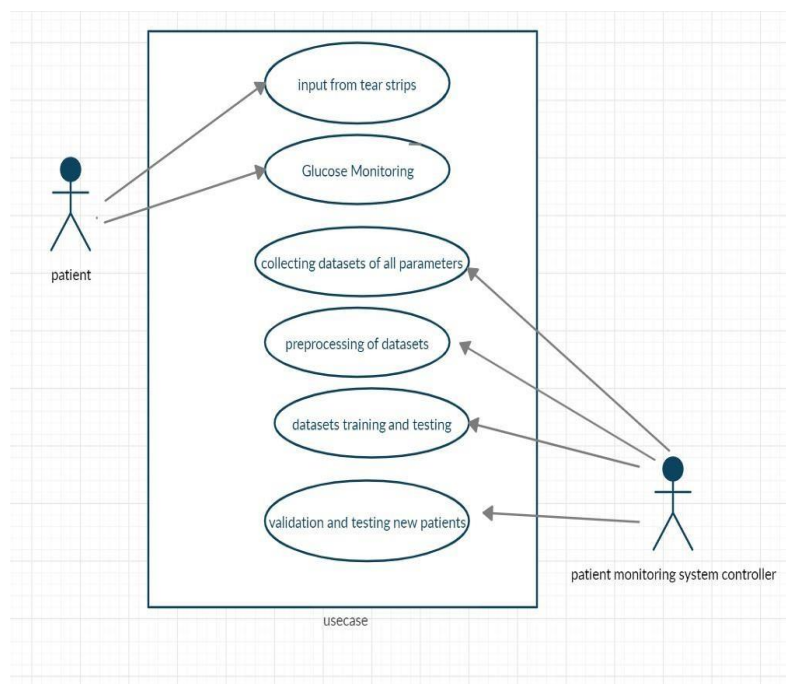


Figure.5.3.2 Use case diagram

5.4.2 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships between the classes and among objects. Private visibility hides information from anything outside the class partition. Public visibility allows all other classes to view the marked information. Protected visibility allows child classes to access information they inherited from a parent class.

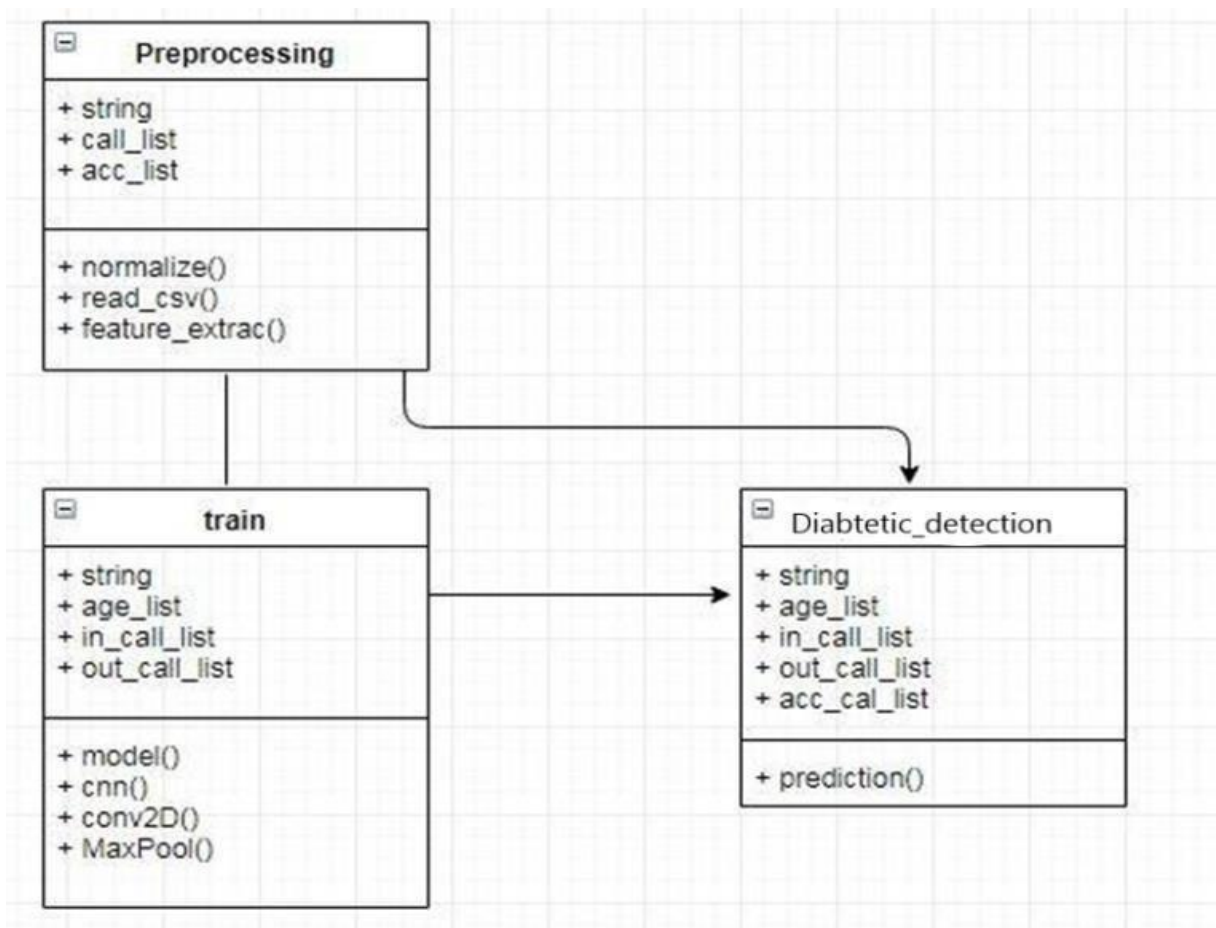


Figure. 5.3.3 Class diagram

5.4.4 ACTIVITY DIAGRAM

Activity diagram are a loosely defined diagram and are graphical representations to show workflows of stepwise activities and actions, with support for choice, iteration and concurrent computational and organizational processes. UML, activity diagrams can be used to describe the business and operational step bystep workflows of components in a system. UML activity diagrams could potentially model the internal logic of a complex operation. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structural development. Activity diagrams show the overall flow of control. Activity diagrams are constructed from a limited number of shapes, connected with arrows.

The most important shape types:

- Rounded rectangles represent actions;
- Diamonds represent decisions;
- Bars represent the start (split) or end (join) of concurrent activities;
- A black circle represents the start (initial state) of the workflow;
- An encircled black circle represents the end (final state).

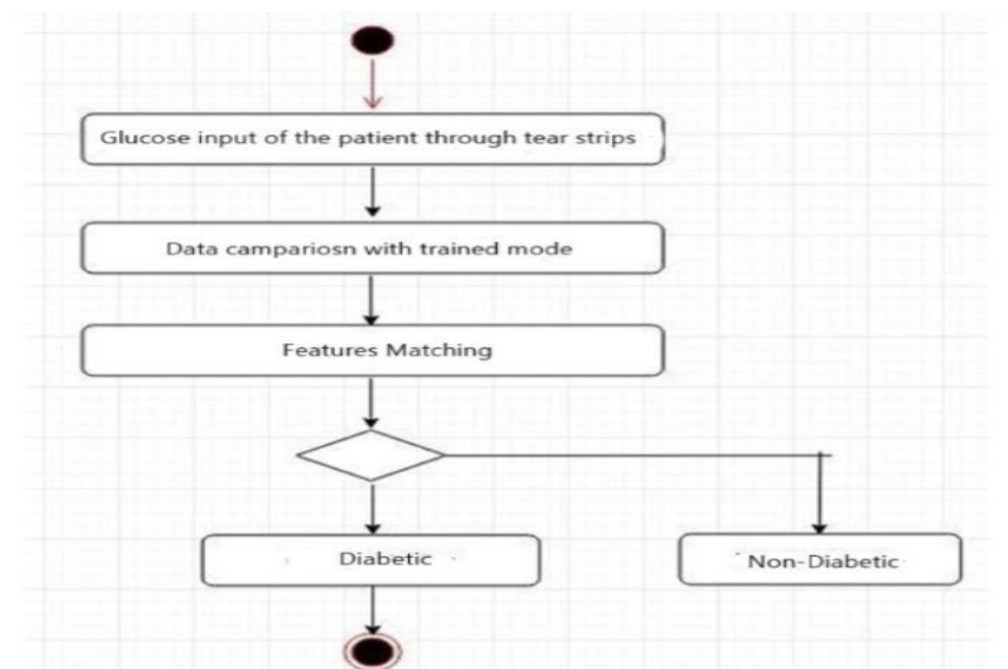


FIGURE 5.4.4 ACTIVITY DIAGRAM

5.4.5 COMPONENT DIAGRAM

Component-based development (CBD) and object-oriented development go hand-in-hand, and it is generally recognized that object technology is the preferred foundation from which to build components. I typically use UML 2 component diagrams as an architecture-level artifact, either to model the business software architecture, the technical software architecture, or more often than not both of these architectural aspects. Physical architecture issues, in particular hardware issues, are 30 better addressed via UML deployment

diagrams or network diagrams. In fact I'll often iterate back and forth between these diagrams.

Component diagrams are particularly useful with larger teams. Your initial architectural modeling efforts during cycle 0 should focus on identifying the initial architectural landscape for your system. UML component diagrams are great for doing this as they enable you to model the high-level software components, and more importantly the interfaces to those components. Once the interfaces are defined, and agreed to by your team, it makes it much easier to organize the development effort between sub teams. You will discover the need to evolve the interfaces to reflect new requirements or changes to your design as your project progresses, changes that need to be negotiated between the sub teams and then implemented appropriately.

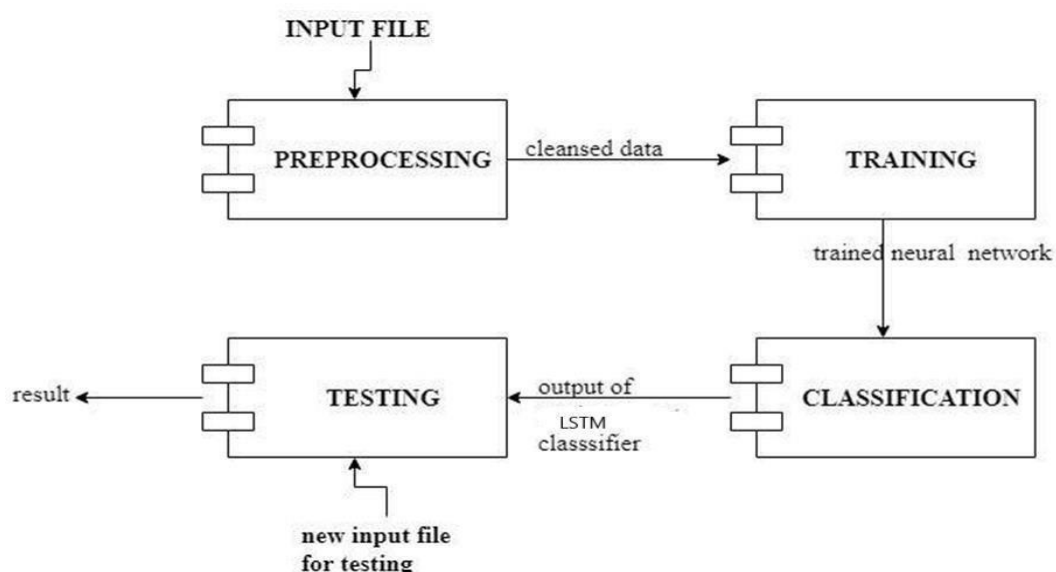


Figure. 5.3.5 component diagram

SYSTEM IMPLEMENTATION

6.1 INTRODUCTION

A software application in general is implemented after navigating the complete life cycle method of a project. Various life cycle processes such as requirement analysis, design phase, verification, testing and finally followed by the implementation phase result in a successful project management. System implementation is an important stage of theoretical design is turned into practical system. Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and it's constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods. Each program is tested individually at the time of development using the data and has verified that this program linked together in the way specified in the programs specification, the computer system and its environment is tested to the satisfaction of the user. The system that has been developed is accepted and proved to be satisfactory for the user and so the system is going to be implemented very soon. A simple operating procedure is included so that the user can understand the different functions clearly and quickly. The final stage is to document the entire system which provides components and the operating procedures of the system.

6.2 LIST OF MODULES

- Collection of Dataset

- Data Pre-processing
- Classification and training of datasets

6.3 MODULE IMPLEMENTATION

6.2.1 COLLECTION OF DATASET

The datasets are collected from various trusted online resources. These data sets include three parameters which are related body movement of the patient, social behavior of the patient and the questionnaire. Body movements of the patient is measured by using the pedometer sensor. Social behavior includes the call log details of the patient. Incoming calls, outgoing calls and number of attended calls are being monitored and stored in a database. Various questions related to the age, family health history and job of the patient are collected and asked to the patient.) The answers collected from the patient are stored in the database. The datasets are collected for both diabetic and non-diabetic patients. So that it is easy to compare and give somewhat exact results. By combining these parameters, a new dataset is created and sent to the training neural network. Existing datasets are collected and stored. An application is created to collect these datasets from a new patient who may have diabetes. This application collects the data and store it in the firebase database. This datasets can be converted into a suitable processing file which is required for training the network. Most suitable form of file used in this project is comma separated files.

6.2.2 DATA PRE-PROCESSING

As the research is focused mainly on identifying users susceptible to diabetes, we selected a test dataset consisting 769 users labeled as either Diabetic or NonDiabetic. The test dataset can be considered as random and not following the same distribution as the training dataset. The training data contained an average length of 6,921 words are used by a user in the test set. Unlike the training dataset, the test dataset is not

extracted considering the age and gender attributes, and it does not have a similar age and gender distribution between the diabetic and nondiabetic groups.

To data we have achieved two major milestones. First, have completed preprocessing of the data. The pre-processing step involves exporting the data from its native CSV file location into the Python workspace. From, there we structured the data such that each input vector element contained one word of the article text. The output for each corresponding gee rated over a normalized period.

In order to train machine learning classifiers, each participant's day was characterized by a feature vector which was computed by extracting a set of features on a per day basis from the activity level. The extracted features were the mean outcome, the corresponding standard deviation and the percentage of events with no outcome i.e, outcome= 0. The features were then normalized between 0 and 1.

6.2.3 CLASSIFICATION AND TRAINING OF DATASETS

The main idea of machine learning Warren McCulloch and Walter Pitts in 1940 by taking the model of the human brain neurons were raised. Set the input weights of each neuron makes learning neural network is an artificial neural network can be a single layer or multiple layers , resistance and damage tolerance can be learned, being (i.e., the ability to adjust the weight LAN), generalize, because of the high speed parallel processing, coping with change.

RECURRENT NEURAL NETWORK(RNN)

A **recurrent neural network (RNN)** is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feed forward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs.^[1] This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

The term “recurrent neural network” is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feed forward neural network, while an infinite impulse recurrent network is a directed cyclic graph that cannot be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored states, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks (LSTMs) and gated recurrent units. This is also called Feedback Neural Network.

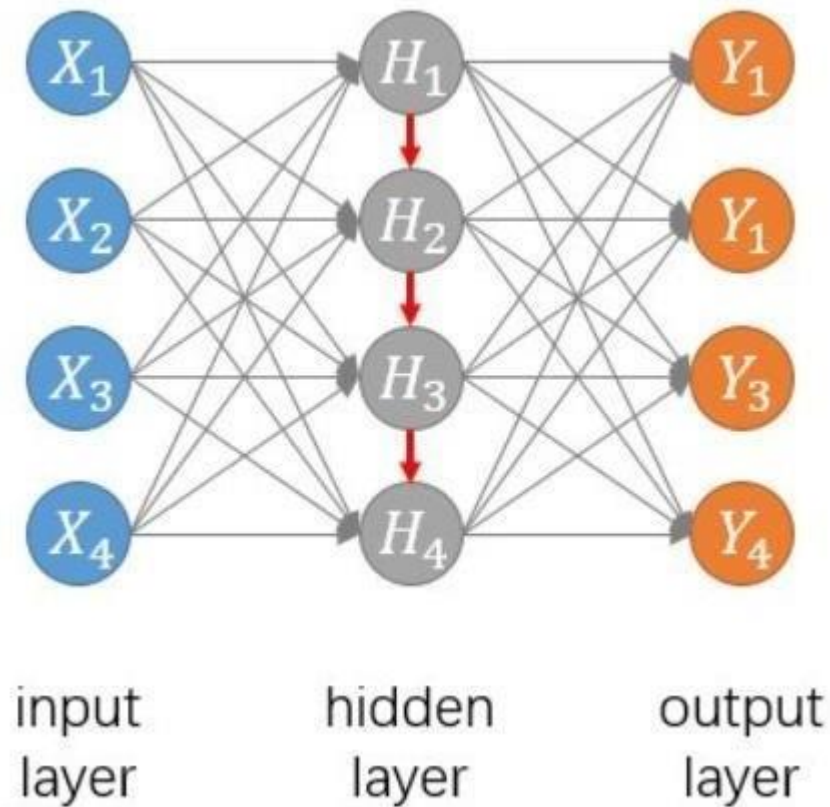


Figure.6.2.1 Layers used to build RNN

We use three main types of layers to build RNN architectures: Input Layer, Hidden Layer, and Output Layer (exactly as seen in regular Neural Networks). We will stack these layers to form a full RNN architecture.

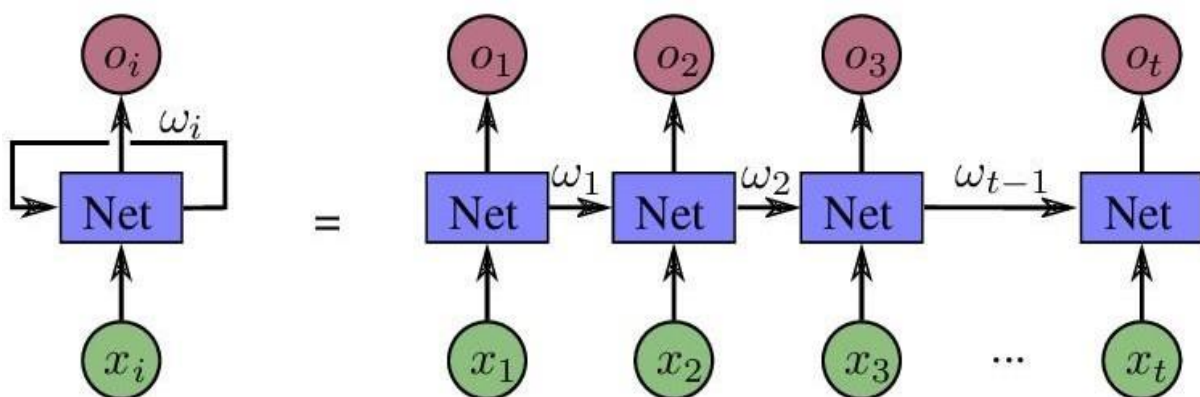


Figure.6.2.2 RNN Architecture

LONG SHORT TERM MEMORY (LSTM)

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture^[1] used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition^{[3][4]} and anomaly detection in network traffic or IDS's (intrusion detection systems).

A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications

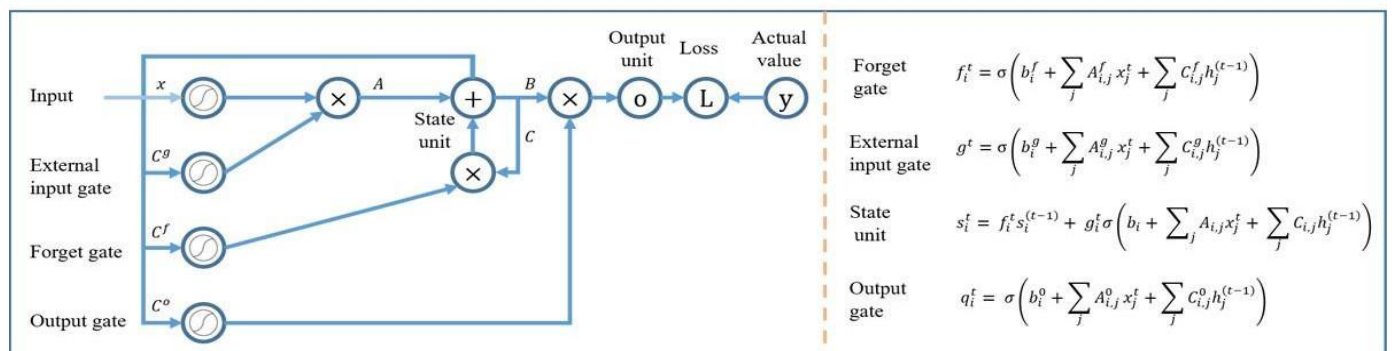


Figure.6.2.3 Long short-term memory (LSTM)

CHAPTER 7

SYSTEM TESTING

7.1 INTRODUCTION

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item. The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

7.2 TESTING OBJECTIVES

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time.

7.3 TYPES OF TESTING

Testing is a process of executing a program or application with the intent of finding the bugs. It can also be stated as the process of validating and verifying that a program or application or product: Meets the business and technical requirements that guided its design and development. System testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully achieved.

Types of Software Testing:

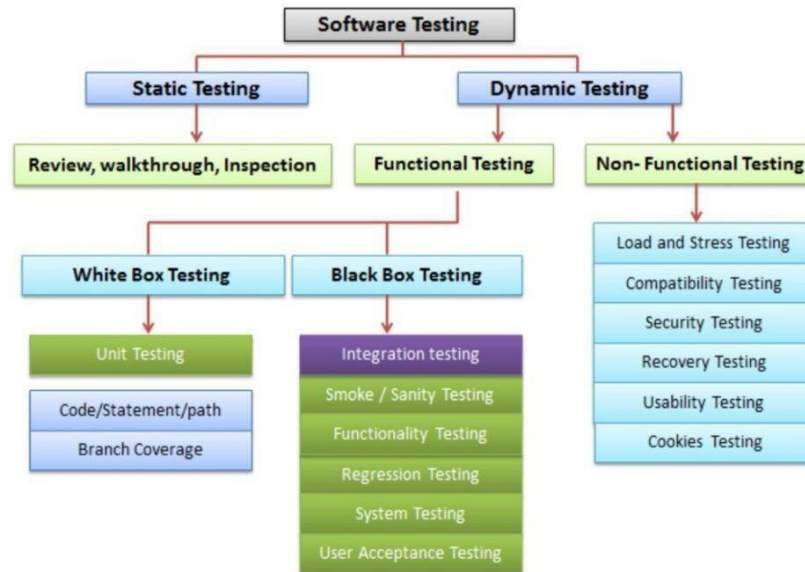


Figure 7.3.1 Types of Testing

7.3.1 WHITE BOX TESTING

White Box Testing (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester.

The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system. This method is named so because the software program, in the eyes of the tester, is like a white/transparent box inside which one clearly sees.

7.3.2 BLACK-BOX TESTING

Black-box testing, also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being Tested is not known to the tester. These tests can be functional or non- functional, though usually functional.

7.3.3 UNIT TESTING

Usually done at a low level, close to the source of your application, these types of software tests examine individual methods and functions of the classes, components or modules used by the software. Usually performed by a programmer, it ensures the unit they have implemented is producing expected output based on any given input. Generally inexpensive to automate, they can be run quickly and consistently by a continuous integration server.

7.3.4 INTEGRATION TESTING

It is quite obvious that the data can be lost across the interface; one module can have an adverse effect on others. Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems. It conducts tests to uncover errors associated within the interface conducted. The objective is to take unit tested modules and combine them test it as a whole. Thus, in the integration-testing step all the errors uncovered are corrected for the next testing steps. The various integration testing techniques are Big Bang integration testing, Top down integration testing and Bottom up integration testing.

7.3.5 FUNCTIONAL TESTING

These specifically verify the output of an action without checking the intermediate states of the system while performing said actions – Functional testing focuses on the business requirements of an application. There is, oftentimes, confusion between functional and integration testing because they both require multiple

parts of an application to be running in orders to interact with each other. The main difference is that integration tests simply assure. User can query the database while functional tests would expect to get a specific /value from the database, as defined by the product requirements.

7.3.6 VALIDATION TESTING

Validation test succeeds when the software functions in a manner that can be reasonably expected by the client. Software validation is achieved through a series of black box testing which confirms to the requirements. Black box testing is conducted at the software interface. The test is designed to uncover interface errors, is also used to demonstrate that software functions are operational, input is properly accepted, output are produced and that the integrity of external information is maintained. The outputs that come out of the system are as a result of the inputs that go into the system. In order to get the expected inputs that go into the system should be correct and proper. So this testing is done to check if the inputs are correct and they are validated before it goes to system for processing.

7.3.7 ACCEPTANCE TESTING

It is a welcomed fact that user acceptance of a system is the key factor for the success of any system. Acceptance Testing is a level of the software testing where a system is tested for acceptability .The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery. This is done in regard to the following point:

- Input screen design
- Output screen design

An acceptance test has the objective of selling the user on the validity and reliability of the system. It verifies that the system's procedures operate to system specifications and that the integrity of important data is maintained.

Performance of an acceptance test is actually the user's show. User motivation is very important for the successful performance of the system. After that a comprehensive report is prepared.

7.3.8 PERFORMANCE TESTING

Performance testing is a testing practice which mainly aims at testing the responsiveness and stability of a system under various workloads. Performance testing measures the quality attributes of the system, such as scalability, reliability and resource usage. The various types of performance testing include Load testing - It is the simplest form of testing conducted to understand the behavior of the system under a specific load. Load testing will result in measuring important business critical transactions and load on the database, application server, etc., are also monitored.

Stress testing - It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.

Soak testing - Soak Testing also known as endurance testing, is performed to determine the system parameters under continuous expected load. During soak tests the parameters such as memory utilization is monitored to detect memory leaks or other performance issues. The main aim is to discover the system's performance under sustained use.

Spike testing - Spike testing is performed by increasing the number of users suddenly by a very large amount and measuring the performance of the system. The main aim is to determine whether the system will be able to sustain the workload.

7.3.9 END-TO-END TESTING

End-to-end testing validates that various user workflows perform as expected as it imitates user behavior with the software in a complete application environment. This can be both simple, from opening a webpage or logging in, to very complex, like verifying email notifications, online payments, etc. These tests are, needless to say, very useful. However, they can be expensive and hard to maintain when they are automated. We recommend having a few end-to-end tests and relying more on lower-level (unit and integration) types of testing to have the ability to quickly and easily identify negative/breaking changes.

7.3.10 OUTPUT TESTING

Output of test cases compared with the expected results created during design of test cases. Asking the user about the format required by them tests the output generated or displayed by the system under consideration. Here, the output format is considered into two was, one is on screen and another one is printed format. The output on the screen is found to be correct as the format was designed in the system design phase according to user needs. The output comes out as the specified requirements as the user's hard copy.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

We have used the LSTM algorithm to classify the patient because in terms of medical related application, to handle time series data, it gives more accuracy than any other algorithms. Collection of datasets from two different applications and integrating them to get a single dataset needs manual work which increases the complexity. The training is done for previously available patient datasets and analyzed for the new patients.

8.2 FUTURE ENHANCEMENTS

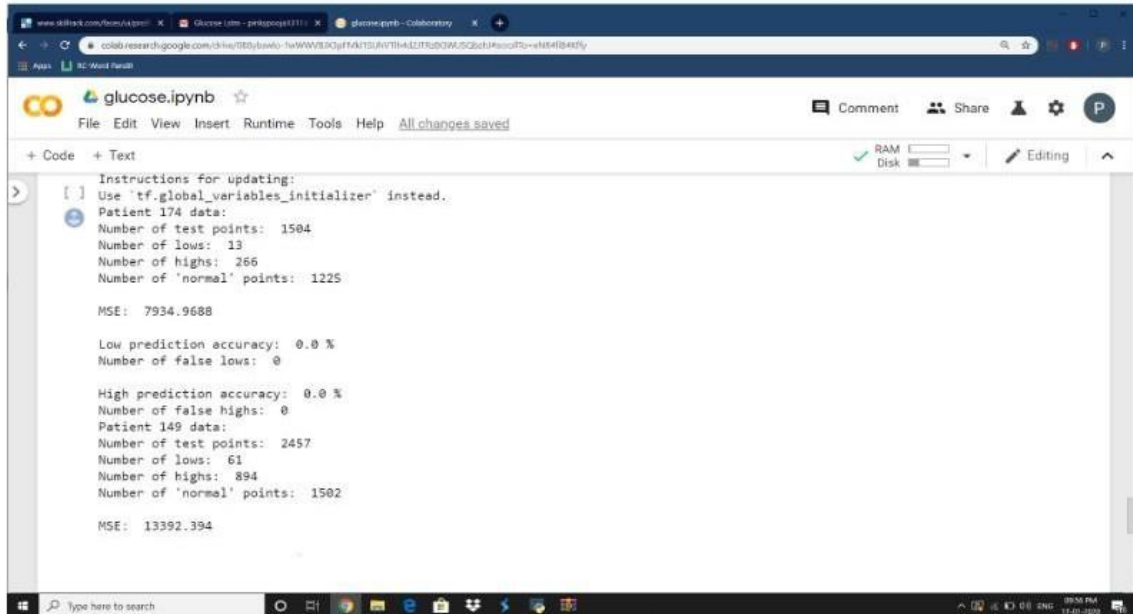
LSTM algorithm can be replaced by GRU algorithm(Gated Recurrent Unit) since it has two gates (reset and update gates) whereas an LSTM has three gates (namely input, output and forget gates). GRU is related to LSTM as both utilizes different ways of gating information to prevent vanishing gradient problems. The GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. It just exposes the full hidden content without any control. GRU is relatively new, the performance is on par with LSTM, but computationally more efficient due to its less complex structure. GRUs train faster and perform better than LSTMs on less training data. GRUs are simpler and thus easier to modify, for example adding new gates in case of additional input to the network. LSTM should remember longer sequences than GRUs and outperform them in tasks requiring modeling long-distance relations. The algorithm can be designed in such a way that it will accept the input from two different files and process them as a single file to avoid manual integration process of datasets.

APPENDIX - I (SCREENSHOTS)

The figure shows the number of patients being tested and classified based on the following metrics:

Number of test points (total number of patients tested)

1. Number of lows (patients who are hypoglycemic)
2. Number of highs (patients who are hyperglycemic)
3. Number of 'normal' points (non-diabetic patients)



The screenshot shows a Jupyter Notebook window titled 'glucose.ipynb'. The code cell contains the following output:

```
Instructions for updating:
Use 'tf.global_variables_initializer' instead.
Patient 174 data:
Number of test points: 1504
Number of lows: 13
Number of highs: 266
Number of 'normal' points: 1225

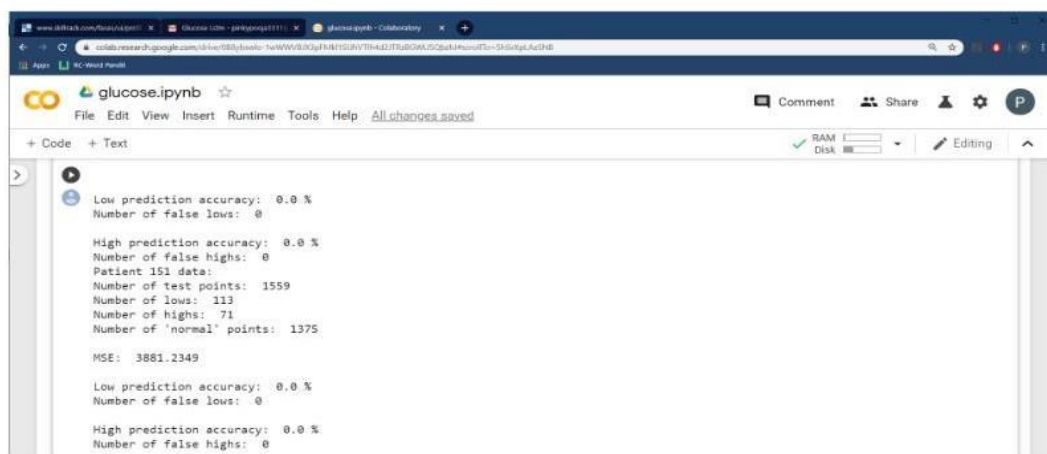
MSE: 7934.9688

Low prediction accuracy: 0.0 %
Number of false lows: 0

High prediction accuracy: 0.0 %
Number of false highs: 0
Patient 149 data:
Number of test points: 2457
Number of lows: 61
Number of highs: 894
Number of 'normal' points: 1502

MSE: 13392.394
```

The figure calculates prediction accuracy and Mean Square Error (MSE) and evaluates the performance of the algorithm for the tested data.



The screenshot shows a Jupyter Notebook window titled 'glucose.ipynb'. The code cell contains the following output:

```
Low prediction accuracy: 0.0 %
Number of false lows: 0

High prediction accuracy: 0.0 %
Number of false highs: 0
Patient 151 data:
Number of test points: 1559
Number of lows: 113
Number of highs: 71
Number of 'normal' points: 1375

MSE: 3881.2349

Low prediction accuracy: 0.0 %
Number of false lows: 0

High prediction accuracy: 0.0 %
Number of false highs: 0
```

The graph is plotted against the number of Epochs in the X-axis and MSE(Mean Square Error) in the Y-axis, which denotes the accuracy of the algorithm that is, it is depicted by the decreasing nature of the graph.

The screenshot shows a Google Colaboratory notebook interface. The code cell contains a function call `train_neural_network(x)`. The output displays a warning from TensorFlow, followed by data statistics for two patients (174 and 149), their respective MSE values, and prediction accuracies for low and high values.

```
[9] #End train_neural_network(x)

train_neural_network(x)
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/util/tf_should_use.py:170: initialize_all_variables (from tensorflow.python.op
Instructions for updating:
Use `tf.global_variables_initializer` instead.
Patient 174 data:
Number of test points: 1504
Number of lows: 13
Number of highs: 266
Number of 'normal' points: 1225

MSE: 6102.814

Low prediction accuracy: 53.84615384615385 %
Number of false lows: 1

High prediction accuracy: 0.0 %
Number of false highs: 0
Patient 149 data:
Number of test points: 2457
Number of lows: 61
Number of highs: 894
Number of 'normal' points: 1502

MSE: 11004.933
```

This block shows the continuation of the output from the previous code cell, specifically the data statistics and MSE for Patient 151, along with prediction accuracies.

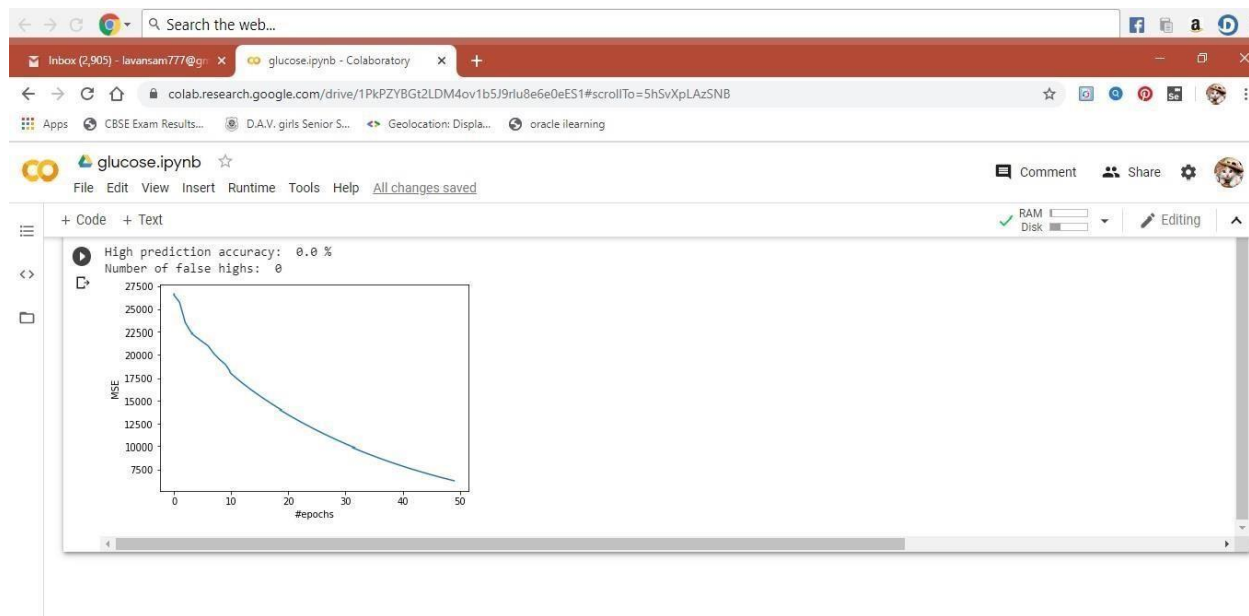
```
Low prediction accuracy: 47.540983606557376 %
Number of false lows: 3

High prediction accuracy: 0.0 %
Number of false highs: 0
Patient 151 data:
Number of test points: 1559
Number of lows: 113
Number of highs: 71
Number of 'normal' points: 1375

MSE: 2810.3987

Low prediction accuracy: 42.47787610619469 %
Number of false lows: 1

High prediction accuracy: 0.0 %
Number of false highs: 0
```



APPENDIX - II (SAMPLE CODING)

```
import tensorflow as tf #
%tensorflow_version 2.x
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
#from tensorflow.python.ops import rnn, rnn_cell from
tensorflow.contrib import rnn import matplotlib.pyplot as
plt
from google.colab import drive

print(tf.__version__)

drive.mount('/content/gdrive') path =
'/content/gdrive/My Drive/Test/'

NUM_EPOCHS = 500
batch_size = 128
chunk_size = 1 n_chunks
= 7
rnn_size = 25

# readData reads data from the specified pre-processed input data file.
# The function returns an array of input data points and an array of the
```

```

# corresponding desired outputs. def
readData(filePath) :
    x_data = []    y_data = []
    with open(filePath, 'r') as f:
    for line in f:        values =
    line.split(',')        time1 =
    float(values[0])        time2 =
    float(values[1])        time3 =
    float(values[2])        time4 =
    float(values[3])        time5 =
    float(values[4])        time6 =
    float(values[5])        time7 =
    float(values[6])        time8 =
    float(values[7])
        newPointx = [time1, time2, time3, time4, time5, time6, time7] # Input
    newPointy = [time8] # Desired Output        x_data.append(newPointx)
    y_data.append(newPointy)    data = [x_data, y_data]
    return data;

# evaluateNetwork runs the trained network on the the provided network and #
reports the following evaluation metrics:
# - mean squared prediction error
# - percentage of lows that were correctly identified
# - percentage of highs that were corretly identified
# - number of falsely reported lows
# - number of falsely reported highs
#
# These metrics are defined as follows:
# - MSE:
#     -> Average of (y_desired - y_actual)^2 for each test point #
- Low prediction accuracy:
#     -> 100 * (Number of correct lows) / (Number of lows) #
-> Lows are any blood glucose level less than 70 mg/dL # -
High prediction accuracy:
#     -> 100 * (Number of correct highs) / (Number of highs)
#     -> Highs are any blood glucose level greater than 200
# - Number of false lows:
#     -> Number of false lows where (y_desired - y_actual) > 6

```

```

# -> Note: false alarms are not counted if the prediction error is small #
- Number of false highs:
# -> Number of false highs where (y_actual - y_desired) > 6
# -> Note: false alarms are not counted if the prediction error is small def
evaluateNetwork(session, inData, outData, prediction) :
    # Compute mse: mse = session.run(tf.reduce_mean(tf.square(prediction
- y)), feed_dict={x: inData, y: outData}) numTestPoints = len(inData)
numPredictedLows = 0 numLows = 0 numFalseLows = 0
numPredictedHighs = 0 numHighs = 0 numFalseHighs = 0 for i,
inputPoint in enumerate(inData) :
    # Apply network on current point: predicted =
session.run(prediction, feed_dict={x: [inputPoint]}) desired =
outData[i][0]

    # Update numLows, numHighs:
if(desired < 70) : numLows
+= 1 elif(desired > 200) :
numHighs += 1

    # Update prediction counts:
if(predicted < 70) : # If predicted low
    if(desired < 70) : # If low prediction was correct
numPredictedLows += 1
    elif((desired - predicted) > 8) : # If low prediction was incorrect and error
was 'large'
numFalseLows += 1
    elif(predicted > 200) : # If predicted high
if(desired > 200) : # If high prediction was correct
numPredictedHighs += 1
    elif((predicted - desired) > 8) : # If high prediction was incorrect and error
was 'large'
numFalseHighs += 1

# Print results:
print('Number of test points: ', numTestPoints)
print('Number of lows: ', numLows) print('Number
of highs: ', numHighs)
print("Number of 'normal' points: ", numTestPoints - numLows - numHighs)
print("") # New line

```

```

    print('MSE: ', mse)
print("")
    print('Low prediction accuracy: ', 100 * numPredictedLows / numLows, '%')
print('Number of false lows: ', numFalseLows)    print("")
    print('High prediction accuracy: ', 100 * numPredictedHighs / numHighs, '%')
print('Number of false highs: ', numFalseHighs)
# End evaluateNetwork(...)

x = tf.placeholder('float', [None, n_chunks, chunk_size]) y
= tf.placeholder('float')

# recurrent_neural_network() defines the RNN model. The network architecture
# used consists of a single LSTM cell followed by an output layer. def
recurrent_neural_network(x):
    layer      =      {'weights':tf.Variable(tf.random_normal([rnn_size,      1])),
'biases':tf.Variable(tf.random_normal([1]))}

    # Reshape x to the format desired by the LSTM:
    x = tf.transpose(x, [1,0,2])    x =
tf.reshape(x, [-1, chunk_size])
##changed x as 0 and 0 as x
    x = tf.split(x, n_chunks, 0)

    ##replaced rnn_cell as rnn
    lstm_cell    =      rnn.BasicLSTMCell(rnn_size)##,      state_is_tuple=True,
activation=tf.nn.relu)
    ##replaced rnn.rnn as rnn.static_rnn and
    outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)

    output = tf.matmul(outputs[-1], layer['weights']) + layer['biases']

    return output

# The RNN is trained by feeding in sequences of glucose measurements separated
# by 10 minute intervals, and the desired output at a 20 minute prediction horizon.

def      train_neural_network(x):
trainData_in, trainData_out =
readData(path+'tblADDataRTCGM_Unblinded_ControlGroup_1_output_RNN_20/1
74_train.csv')

```

```

testData_in, testData_out =
readData(path+'tblADataRTCGM_Unblinded_ControlGroup_1_output_RNN_20/1
74_test.csv')  trainData_in = np.reshape(trainData_in, [-
1,n_chunks,chunk_size])  testData_in = np.reshape(testData_in, [-
1,n_chunks,chunk_size])  prediction = recurrent_neural_network(x)

# Use MSE as cost function to be minimized
cost = tf.reduce_mean(tf.square(prediction - y))

# AdamOptimizer produced better results than simple
GradientDescentOptimizer
optimizer = tf.train.AdamOptimizer(0.01).minimize(cost)

errors = []  with tf.Session() as sess:
sess.run(tf.initialize_all_variables())

# Per-epoch training:      for i
in range(NUM_EPOCHS):
    sess.run(optimizer, feed_dict={x: trainData_in, y: trainData_out})
# if i % 10 == 0 :
    #  mse = sess.run(tf.reduce_mean(tf.square(prediction - y)), feed_dict={x:
testData_in, y: testData_out})
    #  errors.append(mse)
    #  print(mse)

print('Patient 174 data:')
evaluateNetwork(sess, testData_in, testData_out, prediction)
print('Patient 149 data:')  testData_in, testData_out =
readData(path+'tblADataRTCGM_Unblinded_ControlGroup_1_output_RNN_20/1
49_test.csv')  testData_in = np.reshape(testData_in, [-
1,n_chunks,chunk_size])  evaluateNetwork(sess, testData_in,
testData_out, prediction)  print('Patient 151 data:')
testData_in, testData_out =
readData(path+'tblADataRTCGM_Unblinded_ControlGroup_1_output_RNN_20/1
51_test.csv')  testData_in = np.reshape(testData_in, [-
1,n_chunks,chunk_size])  evaluateNetwork(sess, testData_in,
testData_out, prediction)  #evaluate the current network on a
different patient:  testData_in, testData_out =

```

```

readData('tblADDataRTCGM_Blind_Baseline_Split_output/78_test.csv')
evaluateNetwork(sess, testData_in, testData_out, prediciton)

# Plot the MSE throughout training
plt.plot(errors)
plt.xlabel('#epochs')
plt.ylabel('MSE')
plt.show()
End train_neural_network(x)

train_neural_network(x)

```

REFERENCES:

- [1] E. A. M. Gale, "The Rise of Childhood Type 1 Diabetes in the 20th Century," Diabetes, vol. 51, no. 12, pp. 3353-3361, Dec. 2002. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, pp.68–73, 1892.
- [2] K. Niitsu, A. Kobayashi, K. Hayashi, Y. Nishio, K. Ikeda, T. Ando, Y. Ogawa, H. Kai, M. Nishizawa, and K. Nakazato, "A Self-Powered SupplySensing Biosensor Platform Using BioFuel Cell and LowVoltage, Low-Cost CMOS Supply-Controlled Ring Oscillator with Inductive-Coupling Transmitter for Healthcare IoT," accepted to IEEE Transactions on Circuits and Systems I (TCAS-I), vol. 65, no. 9, pp.2784 - 2796, 2018.
- [3] A. Kobayashi, K. Ikeda, K. Nakazato, and K. Niitsu, "Energy efficient and low-voltage design methodology for a supply-sensing CMOS biosensor using biofuel cells for energy-autonomous healthcare applications," Japanese Journal of Applied Physics, vol. 56, no. 1S, pp. 01AH03, 2017.
- [4] K. Niitsu, A. Kobayashi, K. Yoshida, and K. Nakazato, "Design and experimental verification of CMOS Magnetic-Based Microbeads Detection Using an

- Asynchronous Intra-chip Inductive-Coupling Transceiver," Japanese Journal of Applied Physics, vol. 56, no. 1S, pp. 01AH05, 2017.
- [5] K. Niitsu, K. Ikeda, K. Muto, and K. Nakazato, "Design, Experimental Verification, and Analysis of a 1.8-V-Input-range Voltage-to-Current Converter Using Source Degeneration for LowNoise Multimodal CMOS Biosensor Array," Japanese Journal of Applied Physics, vol. 56, no. 1S, pp. 01AH06, 2017.
- [6] [6] K. Ikeda, A. Kobayashi, K. Nakazato and K. Niitsu, "A current-mode analog-to-time converter with short-pulse output capability using local intra-cell activation for high-speed time-domain biosensor array," in Proc. IEEE Nordic Circuits and Systems Conference (NORCAS), Copenhagen Nov. 2016, pp. 1–6.
- [7] K. Hayashi, S. Arata, S. Murakami, Y. Nishio, A. Kobayashi and K. Niitsu, "A 6.1-nA Fully Integrated CMOS Supply Modulated OOK Transmitter in 55nm DDC CMOS for Glasses-Free, Self-Powered, and Fuel-Cell-Embedded Continuous Glucose Monitoring Contact Lens," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 65, no. 10, pp. 1360-1364, 2018.
- [8] Maya Matsunaga, Atsuki Kobayashi, Kazuo Nakazato, and Kiichi Niitsu, "Design Trade-Off between Spatial Resolution and Power Consumption in CMOS Biosensor Circuit Based on Millimeter-Wave LC-Oscillator Array," Japanese Journal of Applied Physics, vol. 57, no. 3S2, pp. 03EC02, 2018.
- [9] T. Nakanishi, M. Matsunaga, A. Kobayashi, K. Nakazato, and K. Niitsu, "A 40-GHz fully integrated circulating tumor cell analysis vector network analyzer in 65-nm CMOS technology with coplanarline-based detection area," Japanese Journal of Applied Physics, vol. 57, no. 3S2, pp. 03EC01, Jan. 2018.
- [10] K. Itakura, K. Kayano, K. Nakazato and K. Niitsu, "Theoretical Analysis and Simulation Study of Low-Power CMOS Electrochemical Impedance Spectroscopy Biosensor in 55nm DDC Technology for Cell-State Monitoring," Japanese Journal of Applied Physics, vol. 57, no. 1S, pp. 01AG02, 2018.

