A

Project Report

on

Heart Disease Detection Using Apache Spark and Kafka

In Partial Fulfillment of the Requirements
for the Award of the Degree of

BACHELOR OF ENGINEERING

In

INFORMATION TECHNOLOGY

Submitted by

ROHAN TALAKA 160118737106

SK SAMEER 160118737110

SUPERVISOR

Dr. M. Venu Gopalachari,

Associate Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)

(Affiliated to Osmania University; Accredited by NBA, NAAC, ISO)

KOKAPET(V), GANDIPET(M), HYDERABAD-500075

Website: www.cbit.ac.in

2021-2022

CERTIFICATE

This is to certify that the project work entitled "**Heart Disease Detection Using Apache Spark and Kafka**" is submitted by **Rohan Talaka (160118737106), Sk Sameer (160118737110)** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Engineering** in **Information Technology** to **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY(A)** affiliated to **OSMANIA UNIVERSITY**, Hyderabad is a record of bonafide work carried out by them under my supervision and guidance. The results embodied in this report have not been submitted to any other University or Institute for the award of any other Degree or Diploma.

Signature of the Supervisor
Dr. M Venu Gopalachari
Associate Professor,

Signature of the HoD
Dr. K. Radhika,
Professor and Head, IT

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We wish to express our deep sense of gratitude to **Dr. M. Venu Gopalachari**, Associate Professor and Project supervisor, Department of Information Technology, Chaitanya Bharathi Institute of Technology, for his able guidance and useful suggestions, which helped us in completing the project in time.

We are particularly thankful to **Dr. K. Radhika**, the Head of the Department, Department of Information Technology, her guidance, intense support and encouragement, which helped us to mould our project into a successful one.

We show gratitude to our honorable Principal **Dr.P. Ravinder Reddy**, for providing all facilities and support.

We avail this opportunity to express our deep sense of gratitude and heartful thanks to **Mr. Subash Garu**, President, CBIT for providing a congenial atmosphere to complete this project successfully.

We also thank all the staff members of Information Technology department for their valuable support and generous advice. Finally thanks to all our friends and family members for their continuous support and enthusiastic help.

Rohan Talaka
Sk Sameer

ABSTRACT

According to the World Health Organization (WHO), stroke and heart attack are the most common cause (80%) of global death. Hence, the availability of data and data mining techniques especially machine learning can help in early detection of heart disease to alert patients to react in advance to a probable disease. In healthcare field, due to the advancement of technologies, huge amount of data (Big Data) generated from multiple areas and multiple sources such as streaming machines, advanced healthcare systems, high throughput instruments, sensor networks, internet of things, mobile application; data collection and processing is carried out on a regular basis continuously. The combination of the stream of Big Data Analytics and machine learning is a breakthrough technology that can have a vital impact in the healthcare field. This technology has better precision and is less expensive.

The proposed system is based on Apache Spark, an open source, distributed processing and computing platform/system, which is used to take the medical parameters from the UI, extract the attributes from the data. The proposed Machine Learning Algorithm is then applied and the prediction is shown on the user side. The system consists of two main sub-parts, namely stream processing and data storage and visualization. The first uses Spark MLlib with Spark streaming and applies a classification model on data events to predict heart disease. The second uses Kafka to combine, messaging, storage and stream processing to allow storage and analysis of both historical and real-time data. This prediction system could offer high accuracy in comparison with literature work with the predictability of 89.6 for heart disease.

Keywords: Big Data Analytics, Apache Spark, Machine Learning, Kafka

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 OVERVIEW

Diagnosing cardiac disease is a difficult task that can provide an automated prognosis of a patient's cardiac health, allowing for more successful therapy. The signs, symptoms, and physical examination of the patient are commonly used to diagnose heart disease. Smoking, high cholesterol, a family history of heart disease, obesity, high blood pressure and physical inactivity are factors that increase the risk of heart disease. Several researchers are currently developing machine learning algorithms for early detection of chronic diseases. Wearable technologies provide lightweight, reliable, cost-effective, wearable healthcare health monitoring solutions. Continuously monitoring changes in the body with smart sensors has become a way of life as a result of various medical awareness projects. Most health education efforts require prevention and early detection of disease. Big data analytics in healthcare is a very simple and useful way to use technology to generate medical data using machine learning algorithms and Spark to predict health problems. It will allow consumers to receive health-related information and warnings sooner. It can also help doctors track patients using smartphone apps. It also facilitates therapies for human ailments based on advanced diagnostics by providing recommendation systems based on machine learning methods.

The most popular frameworks for working with big data are Hadoop Map-Reduce with the next-generation big data processing engine and Apache Spark. Hadoop Map-Reduce has a number of disadvantages, including the fact that it only allows for batch processing and is not ideal for real-time stream processing or in-memory calculations. Apache Spark is an extension of Map-Reduce that allows for more complex calculations. The core concept of Spark is Resilient Distributed Datasets (RDD), which was developed to facilitate in-memory data storage and distributed computing. The Spark stack currently consists of Spark Core and four libraries, namely MLlib for machine learning and Spark Stream for streaming data processing. The algorithms in this library are optimized to run on a distributed data-set, which is better suited for real-time predictions.

1.2 APPLICATIONS

The main application of this project is to predict the likelihood of a person getting heart disease based on certain health conditions or attributes using big data technologies like Apache Spark and Kafka, which help the process to be done efficiently, i.e., accurate and fast prediction. The data processing and monitoring tasks can be carried out much more methodically and efficiently. Other applications include storing data in a more distributed way which aids in visualizing, reporting, and real-time monitoring. This project can not only be used for heart disease prediction but also to predict various medical issues and conditions of patients.

1.3 PROBLEM STATEMENT

To develop an efficient heart disease prediction system that combines big data technologies such as Spark and Kafka along with machine learning algorithms to make the prediction faster and increase the accuracy rate of the predicted result.

1.4 AIM OF THE PROJECT

The purpose of this project is to develop a data processing and classification application. The system consists of two main sub-parts, namely streaming processing data storage and visualization. The first one uses Spark MLlib with Spark streaming and applies a machine learning model on health data events to predict heart disease. And the second uses Kafka which streams data from user to the model and vice versa. This system focuses on applying real-time classification model on heart disease attributes for predicting the presence of heart disease in patients

2. LITERATURE SURVEY

2.1 Heart Disease Prediction Using Machine Learning Algorithms

Authors: Archana Singh, Rakesh Kumar

The heart plays an important role in living organisms. The diagnosis and prediction of heart disease requires more precision, perfection and correction, because a small error can lead to a fatigue problem or the death of the person. There are numerous heart-related deaths and their number is increasing exponentially every day. In order to cope with the problem, there is an essential need for a disease awareness prediction system, which can be achieved using Machine Learning technologies. Machine learning is the branch of artificial intelligence (AI) which offers prestigious support to predict any type of event trained by natural events. In this work, the accuracy of machine learning algorithms is estimated for predicting heart disease. The algorithms used are K-Nearest Decision Tree, Support Vector Machine (SVM), Neighbor (KNN) and Linear Regression. A data-set from UCI repository for training and testing. For Python programming implementation, Anaconda (Jupyter) Notebook is used, which includes many kinds of libraries, header files and that make work more accurate and precise.

Pro(s):

This system uses Four algorithms which works on same data and the one with highest accuracy is used to predict/detect for the new data provided.

Con(s):

It cannot handle the large data and it is not real time.

Proposed system

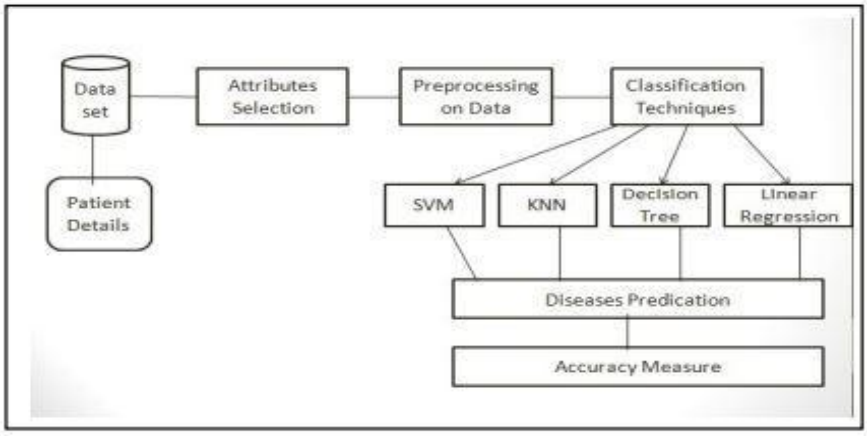

Fig 2.1 Machine Learning Based Heart Disease Prediction System

2.2 Real-time machine learning for early detection of heart disease using big data approach.

Authors: Abderrahmane Ed-daoudy, Khalil Maalmi

This is a scalable system for heart disease monitoring using on Spark and Cassandra frameworks. This system focuses on applying real-time classification model on heart disease attributes for continuous monitoring of the patient's health.The system consists of 2 main sub parts, specifically streaming process and information storage and visualization. the primary uses Spark MLlib with Spark streaming during which the classification model is performed by applying random forest to data events to predict heart disease. The seconds uses Apache Cassandra for storing the big volume of generated data. Once the data is streamed from all data sources, the proposed heart disease monitoring system is based on Spark framework and uses the random forest algorithm with MLlib for developing the prediction model to predict heart disease. Sensitivity, Specificity and Accuracy are calculated for evaluating of the prediction model. On the other hand system performance is performed using TCP messages of heart disease attributes. Integrating other big data technologies to our approach will be more efficient. Developing a distributed and real-time healthcare analytics system using traditional analytical tools is extremely complex, while exploiting open source big data technologies can do it in a simpler and more effective way.

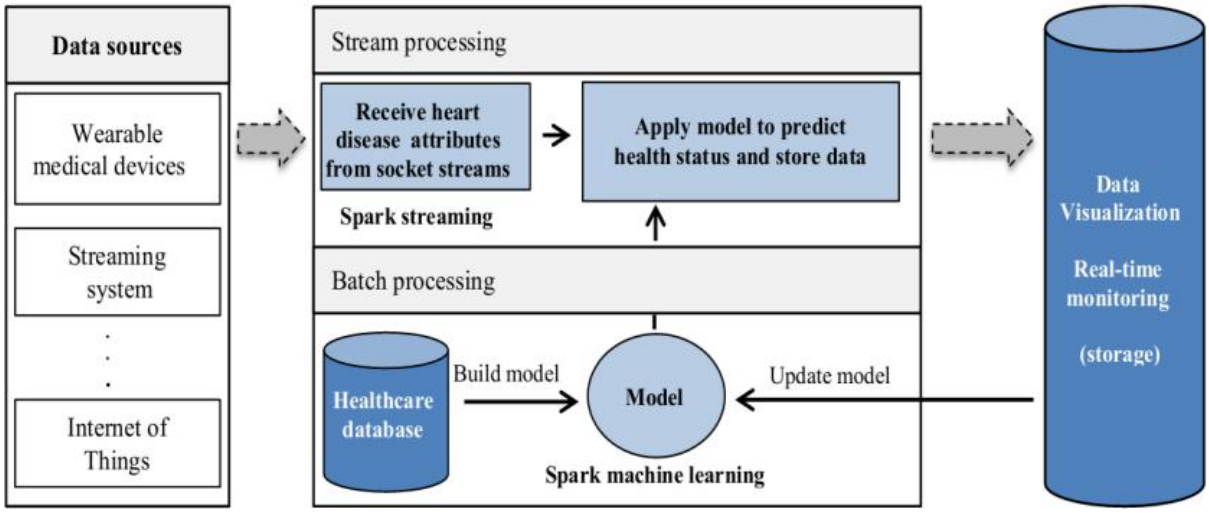

Fig 2.2 Real-time Heart Disease Prediction Overview

2.3 Machine Learning Based Method for Prediction of Heart Disease in Big Data Environment

Authors: Sharmila Rengasamy, Chellammal Surianarayanan, Pethuru Raj Chellaih

Prediction of diseases is one of the challenging tasks in healthcare domain. Conventionally the heart diseases were diagnosed by experienced medical professional and cardiologist with the help of medical and clinical tests. With conventional method even experienced medical professional struggled to predict the disease with sufficient accuracy. In addition, manually analyzing and extracting useful knowledge from the archived disease data becomes time consuming as well as infeasible. The advent of machine learning techniques enables the prediction of various diseases in healthcare domain. Machine learning algorithms are trained to learn from the existing historical data and prediction models are being created to predict the unknown raw data. In this paper, an approach is proposed for prediction of heart diseases using Support Vector Algorithm in Spark environment. Spark is a distributed big data processing platform which has a unique feature of keeping and processing a huge data in memory. The proposed approach is tested with a benchmark dataset from UCI repository and results are discussed.

An approach for prediction of heart diseases using SVM from Spark MLLib has been proposed in order to apply in-memory concept of Spark. When data grows, Spark is capable of keeping huge data in memory and process the data efficiently. The proposed approach has been tested with 297 records. In the future, using in-memory concept of Spark, it is expected to create a huge a huge collection of around one million records. In addition, it is planned to split the data into different nodes and analyse impact of big data tools in the accuracy of prediction.

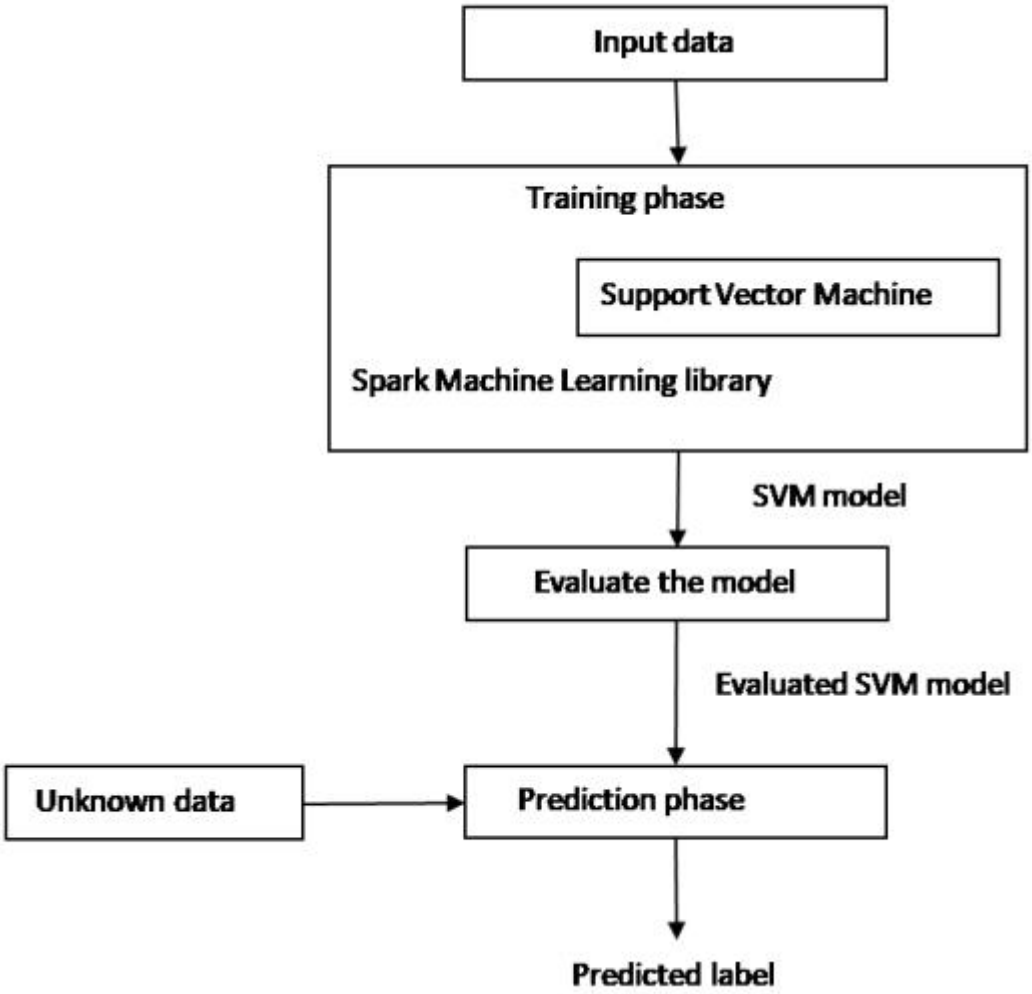

Fig 2.3 Block Diagram of the proposed approach

2.4 Big Data Analysis For Heart Disease Detection System Using Map Reduce Technique

Authors: G.Vaishali1, V. Kalaivani

In today's world, the vast amount of information in healthcare needs to be processed to identify, diagnose, detect and prevent various diseases. It is proposed to develop a centralized patient monitoring system using big data. In the proposed system, a large set of medical records is used as input. From this set of medical data, the necessary information is to be extracted from the register of heart patients using the map reduction technique. Heart disease is a major public health problem and the leading causes of death worldwide. The early detection of heart disease has become an important topic in medical research. First, the data set is pre-processed and features are analyzed. Then some features are analyzed like RR interval, QRS interval and QT interval. The classification process determines whether the patient is normal or abnormal and uses the map reduction technique in the detection step to detect the disease and reduce the data set. Therefore, the proposed system helps classify a large and complex medical data set and detect heart disease.

Pros: This algorithm uses Storm for real time processing and mahoot for machine learning and it uses hive for processing the structured data.

Cons: This system uses map-reduce technique to process the data which takes a lot of time to execute.

Proposed system

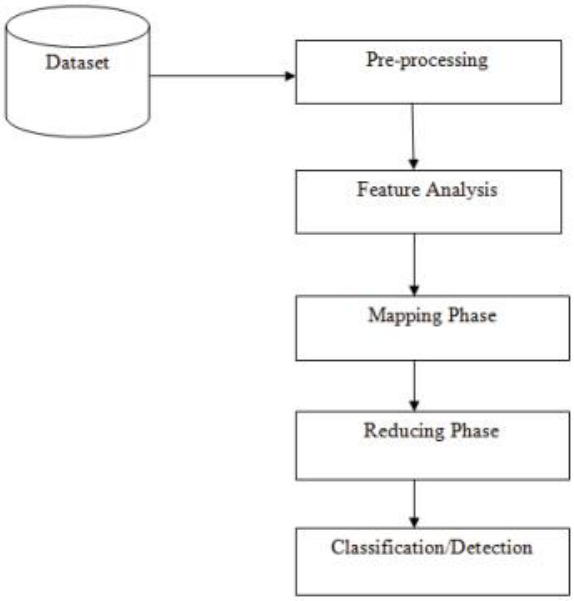

Fig 2.4 Flowchart of the proposed system

3. SYSTEM REQUIREMENT SPECIFICATION

3.1 Functional Requirements

A Functional Requirement (FR) is a description of the service that the software must offer. It describes a package of software system or its components. A function is nothing but inputs to the software system, its behavior, and outputs. This project uses Ubuntu, which is a Linux distribution based on Debian and composed mostly of free and open-source software.

3.2 Non-Functional Requirements

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Some of the non-functional requirements include:

3.2.1 Software Requirements:

1. Ubuntu 16.04.1 LTS: Ubuntu is a free desktop operating system. It is based on Linux, a massive project that enables millions of people around the world to run machines powered by free and open software on all kinds of devices. Linux comes in many shapes and sizes, with Ubuntu being the most popular iteration on desktops and laptops.
2. Apache Zookeeper: Zookeeper is an open-source Apache project that provides a centralized service for providing configuration information, naming, synchronization and group services over large clusters in distributed systems. The goal is to make these systems easier to manage with improved, more reliable propagation of changes.
3. Kafka: Kafka is used to build real-time streaming data pipelines and real-time streaming applications. A data pipeline reliably processes and moves data from one system to another, and a streaming application is an application that consumes streams of data.
4. Spark: Apache Spark is an open-source, distributed processing system used for **big data workloads**. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size.
5. Hadoop: Apache Hadoop is an open source framework that is used to efficiently store and process large datasets ranging in size from gigabytes to petabytes of data. Instead of using one large computer to store and process the data, Hadoop allows clustering multiple computers to analyze massive datasets in parallel more quickly.

3.2.2 Hardware Requirements:

1. Processor: i5 or above
2. RAM: 8 GB
3. Hard disk: 16 GB

4. PROPOSED SYSTEM

4.1 Existing System:

Upto now, several studies have been reported that have focused on heart disease diagnosis. They have used the traditional way of storing and computing of data. Huge amounts of data are stored and are processed later using traditional methods. Even though the overall accuracy achieved by the system is around 85 - 90%, the time taken to process the stored data, is a lot. Dealing with the storage and processing of data simultaneously and continuously requires a lot of time and resources. The traditional or existing systems include use of Machine Language Algorithms such as SVM, Logistic Regression, Decision Tree, etc, which might give accuracy but the time taken and the data being generated is huge and is difficult to handle/deal with.

4.2 Proposed System Design:

This detection system is designed for heart disease data in the big data environment and the framework for proposed prediction system is shown in 4.1

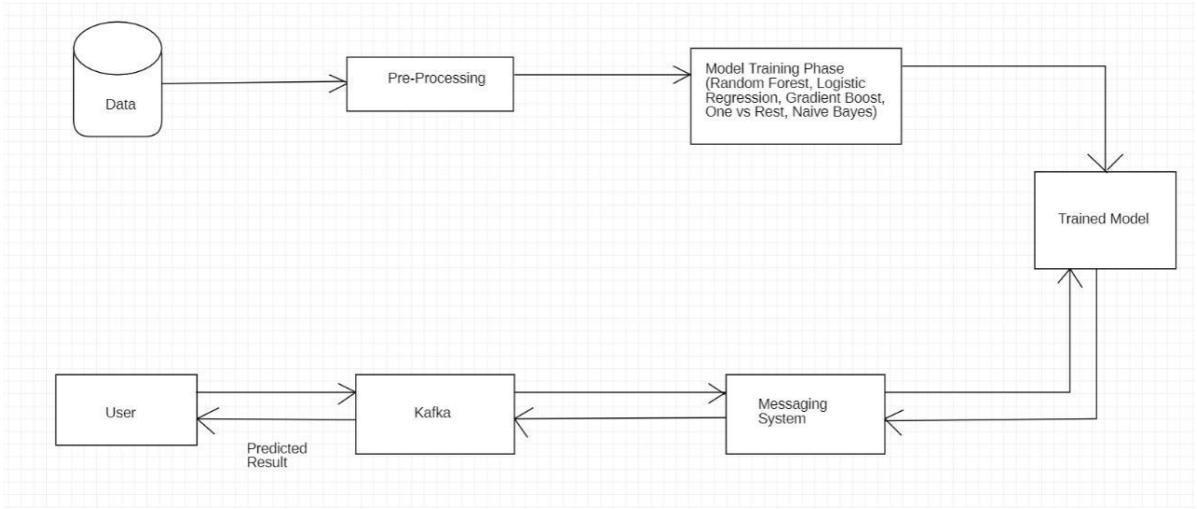

Fig 4.1 Kafka framework for the proposed heart disease prediction system

The initial step in the framework is collecting data from UCI repository. The second step is pre-processing where the null values is removed from the data-set and need to convert some category values to fake values in the form of "0" and "1". The third step is training the machine learning models like Random Forest, logistic regression, decision tree, linear support vector machine, gradient boost. The fourth step is creating a ui where user can enter

the attributes of the heart disease and creating a kafka topic to store the user data in the message system. The data is streamed by spark by using topic name and it is tested against the stored data using the trained model. Then the predicted result is streamed back and displayed to the user.

Since Kafka combines messaging, storage and stream processing to allow storage and analysis of both historical and real-time data, it can help do the processing faster and more accurately.

4.3 Methodology:

Here, random forest algorithm is implemented on UCI dataset by using spark mllib library.

4.3.1 Dataset

In this system, the data-set is freely available and used in the majority of research papers is the heart disease data-set obtained from the UCI (University of California, Irvine C.A) has been used. There are 303 records in this database. This database is divided in the ratio 70-30 for training and testing purposes respectively. Each record in the database has fourteen attributes. The fourteen attributes are detailed in table below:

Table 4.1 Attributes based on which the prediction is done

	Attributes	Description
1	Age	Age in years
2	Sex	Sex (1 = male, 0 = female)
3	Cp	Chest pain type (1= typical angina, 2= atypical angina, 3= non-anginal pain, 4= asymptomatic)
4	Trestbpss	Resting blood pressure (in mm Hg on admission to the hospital)
5	Chol	Serum cholestoral in mg/dl
6	Fbs	fasting blood sugar > 120 mg/dl (1 = true, 0 = false)
7	Restecg	Resting electrocardiographic results
8	Thalach	Maximum heart rate achieved
9	Oldpeak	ST depression induced by exercise relative to rest
10	Exang	Exercise induced angina
11	Slope	The slope of the peak exercise ST segment
12	Ca	Number of major vessels (0-3)
13	Thal	3 = normal; 6 = fixed defect; 7 = reversable defect
14	Num	Class (presence or absence of heart disease)

In the sample data, 139 people (45.87%) have heart disease, while 164 people (54.13%) don't. The existence of heart disease and absence of heart disease are indicated by the class label. The data is loaded into an RDD and DataFrame from a CSV file in this section.

4.3.2 Pre-processing:

Pre-processing is required for the machine learning algorithms to produce prestigious results. For example, the Random Forest technique does not handle datasets with null values, so we must manage null values from the original raw data. We need to convert some category values to fake values in the form of "0" and "1" for our project.The raw data obtained is parsed and a .csv file is created. The null values are removed and the data is manually classified and a data-set is created.

4.3.3 Model Training Phase:

The model is implemented through Apache Spark's MLlib. The algorithms being used for training and testing purpose are:

1. Random Forest
2. Decision Tree
3. Logistic Regression
4. Linear Support
5. Vector Machine
6. Gradient Boost

1. Random Forest

Random forest is a supervised machine learning algorithm that constructs several decision trees. The final decision is made based on the majority of decision tree. Decision tree suffer from low bias and high variance. Random forest converts high variance to low variance.

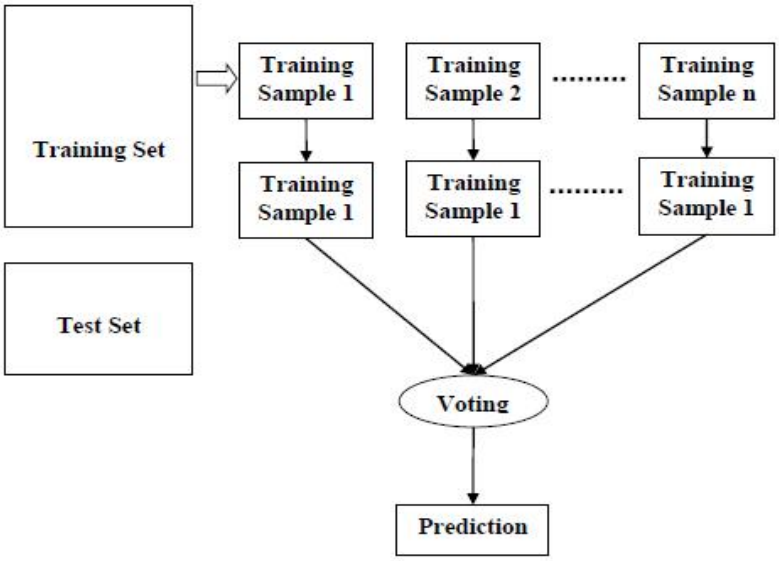

Fig 4.2 Working of Random Forest Model

Random forests are decision tree ensembles that are one of the most successful and efficient supervised classification algorithms for both classification and regression. Random Forest generates the forest from numerous decision trees, as the name implies; in general, the more trees in the forest, the more reliable the prediction and thus the higher reliability. It was chosen to do the prediction in the suggested system based on this. The prediction of each tree is regarded a vote for one class when classifying a new object based on attributes. The most popular class should be chosen as the label. The good news is that random forest combines decision tree simplicity with flexibility, resulting in a significant improvement in precision. The data events are handled by Spark streaming library, and the Random Forest implementation is handled by Spark MLlib.

2. Decision Tree:

Decision tree is the graphical representation of the data, and it is also the kind of supervised machine learning algorithm. The tree-like structure underpins this classification methodology. This is classified as supervised learning because the desired outcome is already known. The Decision tree algorithm can be used with both category and numerical data. The root node, branches, and leaf nodes make up a decision tree. The traversal path from the root to a leaf node is used to evaluate the data. We use the entropy of the data attributes to build the tree, and we draw the root and other nodes based on the attribute root. When the number

of nodes in a tree is unbalanced, the tree develops an overfitting problem, which is bad for calculations and one of the reasons why decision trees are less accurate than Random Forest.

3. Logistic Regression:

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. A logistic regression model predicts a dependent data variable by analysing the relationship between one or more existing independent variables. It allows algorithms used in machine learning applications to classify incoming data based on historical data. As additional relevant data comes in, the algorithms get better at predicting classifications within data sets.

4. Linear Support Vector Machine:

Support Vector Machine is a machine learning classification approach that is used in classification and regression analysis to examine data and uncover patterns. When data is described as a two-class problem, SVM is usually considered. Data is described using this technique by determining the optimum hyper plane that isolates all data points from one class from the other. The better the model is considered, the greater the gap or edge between the two classes. Support vectors are the data points that are closest to the margin's limit.

5. Spark streaming and Kafka:

Spark Streaming is used to process real-time data from various sources including like Kafka, Flume, and Amazon Kinesis. Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time. Kafka uses producer and consumer model. At first we need to create a topic name so that the Kafka broker allows consumers to fetch messages by topic. Kafka producer push the user entered ui data to the messaging system in a particular topic name and in kafka consumer spark stream is used to retrieve the data from the kafka. It is tested against the stored data using the trained model. Then the predicted result is streamed back and displayed to the user.

All these algorithms are implemented in big data environment.

Big data is a collection of huge amounts of data that is difficult for any traditional database management tools to store it or process it efficiently. The characteristics of big data are volume: volume is the huge amount of data. If the volume of data is very large then it is actually considered as a ‘Big Data’.

Velocity: velocity is the continues flow of data from sources like machines, networks, social media, mobile phones etc.

Variety: variety is the nature/type of data it can be structured, semi-structured and unstructured data.

Veracity: Veracity is the inconsistencies and uncertainty in data, that is data which is available can sometimes get messy and quality and accuracy are difficult to control.

Value: Data in itself is of no use or importance but it needs to be converted into something valuable to extract Information.

6. Spark :

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Spark architecture:

The Apache Spark framework employs a master-slave design, with a driver serving as the master node and a large number of executors serving as the cluster's worker nodes. Apache Spark may be used for both batch and real-time data processing. The Spark architecture depends upon two abstractions:

- Resilient Distributed Dataset (RDD)
- Directed Acyclic Graph (DAG)

Resilient Distributed Datasets (RDD)

The Resilient Distributed Datasets are the group of data items that can be stored in-memory on worker nodes. Here,

Resilient: Restore the data on failure.

Distributed: Data is distributed among different nodes.

Dataset: Group of data.

Directed Acyclic Graph (DAG)

Directed Acyclic Graph is a finite direct graph that performs a sequence of computations on data. Each node is an RDD partition, and the edge is a transformation on top of data. Here, the graph refers the navigation whereas directed and acyclic refers to how it is done.

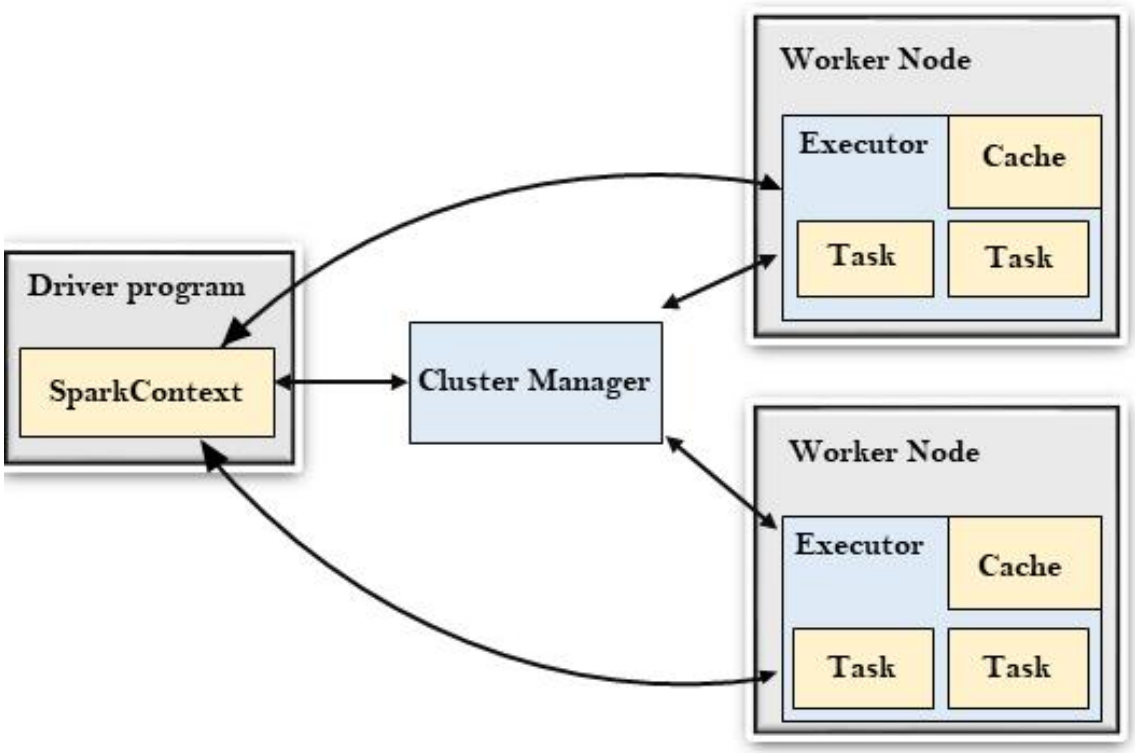

Fig. 4.3 Apache Spark Architecture

Driver Program

The Driver Program is a process that runs the main() function of the application and creates the **SparkContext** object. The purpose of **SparkContext** is to coordinate the spark applications, running as independent sets of processes on a cluster.

Cluster Manager

The role of the cluster manager is to allocate resources across applications. The Spark is capable enough of running on a large number of clusters. It consists of various types of cluster managers such as Hadoop YARN, Apache Mesos and Standalone Scheduler.

Here, the Standalone Scheduler is a standalone spark cluster manager that facilitates to install Spark on an empty set of machines.

Worker Node

Worker node are the slave nodes whose job is to basically execute the tasks. These tasks are then executed on the partitioned RDDs in the worker node and hence returns back the result to the Spark Context.

Executor

An executor is a process launched for an application on a worker node. It runs tasks and keeps data in memory or disk storage across them. It read and write data to the external sources. Every application contains its executor.

Task:

The Spark executors complete the tasks assigned by the Spark driver. A Spark executor's primary role is to take specified tasks, run them, and report back on their success or failure status and results. Each Spark application runs on its own set of executors.

5. IMPLEMENTATION

The proposed work is implemented in three modules. The first module is data collection and pre-processing, the second module is training machine learning models and the third module is spark streaming and Kafka.

Hadoop has been installed on Ubuntu and on top of Hadoop, Spark version 3.2.1 has been installed.

5.1.1 Installation of Spark

Step 1: Pre-requisite setup

Scala needs to be installed for the installation of Spark

$ sudo apt install scala

update-alternatives: using /usr/share/scala-2.12/bin/scala to provide /usr/bin/scala (scala) in auto mode

$ scala -version

Scala code runner version 2.12.15 -- Copyright 2002-2017, LAMP/EPFL

Step 2: Download and Install Spark

On the machine install Spark using the following commands.

$ wget http://apache.claz.org/spark/spark-3.2.1/spark-3.2.1-bin-hadoop3.2.tgz

Extract the files and move them to /usr/local/spark and add the spark/bin into PATH variable.

$ tar xvf spark-3.2.1-bin-hadoop3.2.tgz

$ sudo mv spark-3.2.1-bin-hadoop3.2/ /usr/local/spark

$ vi ~/.bash_profileexport PATH=/usr/local/spark/bin:$PATH$ source ~/.bash_profile

Step 3: Configuration

Now that Spark is installed, a standalone cluster manager will be used for demonstration purposes. Modification of /usr/local/spark/conf/spark-env.sh file needs to be done by providing information about Java location and the node's IP. This will be added to the environment when running Spark jobs.

 # contents of conf/spark-env.sh

export SPARK_MASTER_HOST=<master-private-ip>

export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64/

To check the proper installation of spark the command sh /opt/spark/sbin/start-all.sh should be run. If the command is executed without any error then the setup is done.

Then we have installed Kafka, Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time. Streaming data is data that is continuously generated by thousands of data sources, which typically send the data records in simultaneously. A streaming platform needs to handle this constant influx of data and process the data sequentially and incrementally.

Kafka provides three main functions to its users:

- Publish and subscribe to streams of records
- Effectively store streams of records in the order in which records were generated
- Process streams of records in real time

Kafka is primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.

5.1.2 Installation of Zookeeper

Step 1: Download and Install Zookeeper

cd /home/user

wget http://www-eu.apache.org/dist/zookeeper/zookeeper-3.4.9/zookeeper-3.4.9.tar.gz tar -xvf zookeeper-3.4.9.tar.gz

mv zookeeper-3.4.9 zookeeper

rm -rf zookeeper-3.4.9.tar.gz

Step 2: Configure Zookeeper

cd zookeeper

cp conf/zoo_sample.cfg conf/zoo.cfg

cd /opt

sudo mkdir zookeeper

cd zookeeper/

sudo vi myid --> set a different number for each server cd ~/zookeeper

vi conf/zoo.cfg

Add the below parameters in the file dataDir=/opt/zookeeper

maxClientCnxns=100

server.1=0.0.0.0:2888:3888 (For the current server)

server.2=<Private IP of server 2>:2888:3888

server.3=<Private IP of server 3>:2888:3888

sudo bin/zkServer.sh start

sudo bin/zkServer.sh status

5.1.3 Installation of Kafka

Step 1: Download the Kafka latest build.

$ wget http://mirrors.estointernet.in/apache/kafka/2.1.0/kafka_2.11-2.1.0.tgz

Step 2: Extract the Kafka tar file.

$ tar xzf kafka_2.11-2.1.0.tgz

After the extraction, we need to configure the server. First, the zookeeper should be run in the server. Once the zookeeper is up and running, Kafka needs to be configured. Switch to the Kafka folder.

$ vi config/server.properties

Add this configuration.

 # The id of the broker. This must be set to a unique integer for each broker. For server

broker.id=10

 # A comma separated list of directories under which to store log files log.dirs=/tmp/kafka-logs

advertised.host.name=<ip address of host>

log.dirs=/tmp/kafka-logs

 # add the zookeeper instances ip here zookeeper.connect=<i>:2181,<ip>:2181,<ip>:2181

zookeeper.connection.timeout.ms=6000

To check the proper installation of kafka, the command *$ bin/kafka-topics.sh* should be run. If the command is executed without any error then the setup is done.

Installation of other libraries:

Tkinter installation

pip install tk

kafka-python installation

pip install kafka-python

pyspark

pip install pyspark

The dataset is collected from the UCI repository and loading it into our program. After loading the data pre-processing is done where the null values are removed from the dataset and need to convert some category values to fake values in the form of "0" and "1" which is shown in fig.5.1

```
1
2 from pyspark import SparkContext, SparkConf
3 import pandas as pd
4 import numpy as np
5 from pyspark.mllib.regression import LabeledPoint
6 from pyspark.mllib.tree import DecisionTree
7 conf = SparkConf().setMaster("local[*]").setAppName("heart-disease-prediction-descision-tree")
8 sc   = SparkContext(conf=conf)
9
10
11 heartdf = pd.read_csv("/home/hdoop/Downloads/project1/dataset.csv",header=None)
12
13
14 print(heartdf.shape)
15 print (heartdf.loc[:,13].unique())
16 newheartdf = pd.concat([heartdf.loc[:,13], heartdf.loc[:,0:12]],axis=1)
17 newheartdf = newheartdf.reindex(heartdf.index)
18 newheartdf.replace('?', np.nan, inplace=True)
19 ndf2 = newheartdf.dropna()
20
21 ndf2.to_csv("/home/hdoop/Downloads/project1/heart1.txt",sep=",",index=False,header=None,na_rep=np.nan)
22
23 print(ndf2.shape)
24
25
26
27
28
```

Fig. 5.1 Loading dataset and pre-processing

5.2 Algorithms Implemented

5.2.1 Random Forest

Random Forest is a classifier that contains a number of decision trees on various subsets of the given data-set and takes the average to improve the predictive accuracy of that data-set. A random forest eradicates the limitations of a decision tree algorithm. The greater number of trees in the forest leads to higher accuracy and prevents the problem of over-fitting. It generates predictions without requiring many configurations in packages.

```
#random forest

from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'target',)
rfModel = rf.fit(train)
predictionrf = rfModel.transform(test)
tpr = predictionrf[(predictionrf.target == 1) & (predictionrf.prediction == 1)].count()
tnr = predictionrf[(predictionrf.target == 0) & (predictionrf.prediction == 0)].count()
fpr = predictionrf[(predictionrf.target == 0) & (predictionrf.prediction == 1)].count()
fnr = predictionrf[(predictionrf.target == 1) & (predictionrf.prediction == 0)].count()
accuracyr = float((tpr+tnr)/(predictionrf.count()))
print(accuracyr)
```

Fig. 5.2 Importing Random Forest Classifier from Spark MLlib

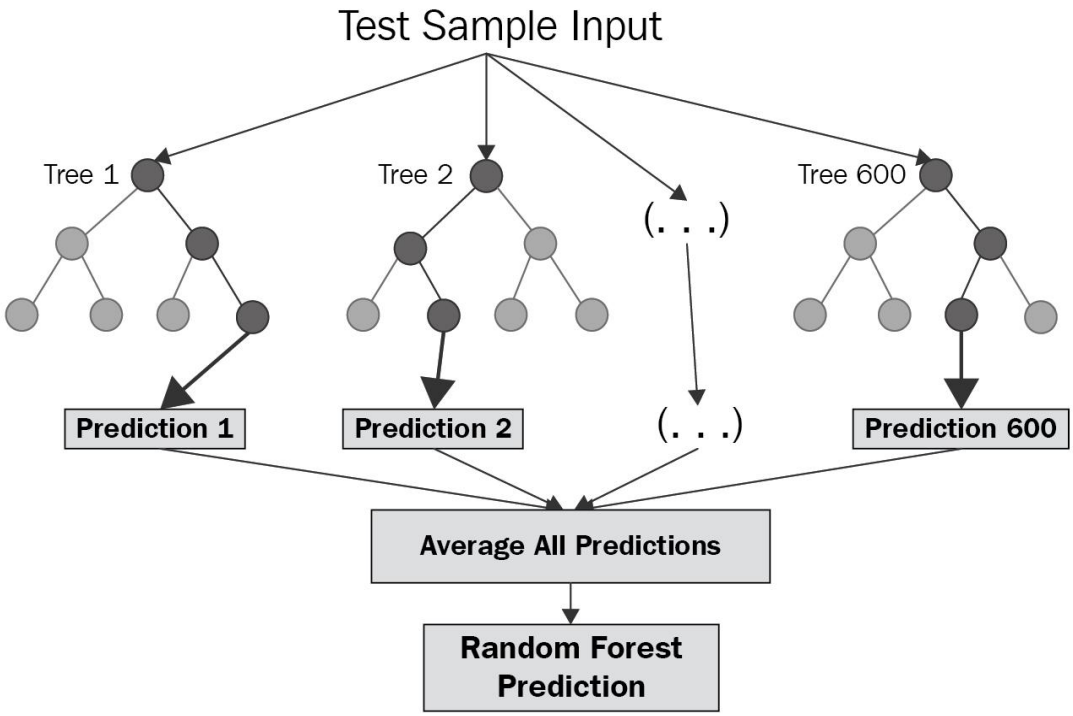

Fig 5.3 Working of Random Forest Algorithm

We have data into vector assemble form we used that data for our random forest model in randomforestclassifier we pass the parameters like featuresCol and labelCol which we get from the vector assembler and it take some default parameters like no of tress =32, impurity = Gini, maxdepth=5, maxbins = 32. Then we train our model and test our model by test data and then calculated the accuracy.

5.2.2 Logistic regression:

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
training = spark.read.csv("/opt/spark/spark-3.1.2-bin-hadoop3.2/hd/heart.csv", inferSchema=True, header=True)
assembler = VectorAssembler(inputCols=['age',
 'sex',
 'cp',
 'trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalach',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal'
 ],outputCol="features")

output = assembler.transform(training)
df_final = output.select("features","target")
print(df_final)
train, test = df_final.randomSplit([0.7,0.3])
lr = LogisticRegression(labelCol="target").fit(train)
tr = lr.evaluate(train).predictions
results = lr.evaluate(test).predictions
tp = results[(results.target == 1) & (results.prediction == 1)].count()
tn = results[(results.target == 0) & (results.prediction == 0)].count()
fp = results[(results.target == 0) & (results.prediction == 1)].count()
fn = results[(results.target == 1) & (results.prediction == 0)].count()
accuracylogistic = float((tp+tn)/(results.count()))
print(accuracylogistic)
```

Fig 5.4 Creating Logistic Regression Model

Vector Assembler is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees. Vector Assembler accepts the following input column types: all numeric types, Boolean type, and vector type. In each row, the values of the input columns will be concatenated into a vector in the specified order.

First, we read the heart disease csv file, then we combine age, sex ,cp ,trestbps ,chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal, into a single feature vector called features and use it to predict heart disease present or not. We set Vector Assembler's input columns as features and output column as target. Vector Assembler work on numeric data so if we have any other datatype, we need to convert it by using hotencoder. Then we split our data into training and testing with 70% of data as training and 30% data as testing

after that we use logistic regression to train our model on training data then we test our model by using testing dataset then we calculated the accuracy, which is shown in the fig. 5.4

5.2.3 Linear Support Vector machine:

Support Vector Machine is a machine learning classification approach that is used in classification and regression analysis to examine data and uncover patterns the implementation of linear support vector machine in shown in fig. 5.5

```
#linear svm

from pyspark.ml.classification import LinearSVC
lsvc = LinearSVC(featuresCol = 'features', labelCol='target', maxIter=10, regParam=0.1)
lsvcModel = lsvc.fit(train)
predictionlsv = lsvcModel.transform(test)
tplsv = predictionlsv[(predictionlsv.target == 1) & (predictionlsv.prediction == 1)].count()
tnlsv = predictionlsv[(predictionlsv.target == 0) & (predictionlsv.prediction == 0)].count()
fplsv = predictionlsv[(predictionlsv.target == 0) & (predictionlsv.prediction == 1)].count()
fnlsv = predictionlsv[(predictionlsv.target == 1) & (predictionlsv.prediction == 0)].count()
accuracylsv = float((tplsv+tnlsv)/(predictionlsv.count()))
print(accuracylsv)
```

Fig. 5.5 Creating Linear Support Vector Machine Model

We have data into vector assemble form we used that data for our linear support vector model in linear svm we pass the parameters like featuresCol and labelCol which we get from the vector assembler and we specifed some parameters like maxIter = 10, regparam = 0.1, regparam is used to reduce the chance of over-fitting. Then we train our model and test our model by test data and then calculated the accuracy.

5.2.4 Gradient Boost:

The Gradient Boost is the machine learning technique used in regression and classification tasks. It gives a prediction model in the form of ensemble of week prediction model, which are typically decision trees. The implementation of gradient boost is shown in fig. 5.6

```
#gbt

from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel
from pyspark.mllib.util import MLUtils
from pyspark.mllib.regression import LabeledPoint
data = sc.textFile("/opt/spark/spark-3.1.2-bin-hadoop3.2/hd/heart1.txt")
def parsePoint(line):
    values = [float(s) for s in line.strip().split(',')]
    if values[0] == -1:
        values[0] = 0
    elif values[0] > 0:
        values[0] = 1
    return LabeledPoint(values[0], values[1:])

parsed_data = data.map(parsePoint)
(trainingData, testData) = parsed_data.randomSplit([0.7, 0.3])
model = GradientBoostedTrees.trainClassifier(trainingData,
                                             categoricalFeaturesInfo={}, numIterations=10)

predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(
    lambda lp: lp[0] != lp[1]).count() / float(testData.count())
print(1-testErr)
print('Test Error = ' + str(testErr))

print('Learned classification GBT model:')
print(model.toDebugString())
```

Fig 5.6 Creating Gradient Boost Model using RDD

We are using RDD to implement gradient boost. Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. The --master option specifies the master URL for a distributed cluster, or local to run locally with one thread, or local[N] to run locally with N threads.setAppName is used to set the name for our Application. we create a rdd by loading our heart disease dataset. A labeled point is a local vector, either dense or sparse, associated with a label/response. In MLlib, labeled points are used in supervised learning algorithms. We use a double to store a label, so we can use labeled points in both regression and classification.so we convert out dataset into labeled points, then we split our dataset into training and testing with 70% of data as training and 30% data as testing after that we use gradient boost to train our model on training data then we test our model by using testing dataset then we calculated the accuracy.

5.2.5 Decision Tree:

```
#decision tree
from pyspark.mllib.tree import DecisionTree
modeldt = DecisionTree.trainClassifier(trainingData, numClasses=5, categoricalFeaturesInfo={}, impurity='gini', maxDepth=3, maxBins=32)

predictionsdt = modeldt.predict(testData.map(lambda x: x.features))
labelsAndPredictionsdt = testData.map(lambda lp: lp.label).zip(predictionsdt)
testErrdt = labelsAndPredictionsdt.filter(lambda x: x[0]!=x[1]).count() / float(testData.count())
print('Test Error = ' + str(testErrdt))

print(modeldt.toDebugString())


modeldtr = DecisionTree.trainRegressor(trainingData, categoricalFeaturesInfo={}, impurity='variance', maxDepth=3, maxBins=32)


predictionsdtr = modeldtr.predict(testData.map(lambda x: x.features))
labelsAndPredictionsdtr = testData.map(lambda lp: lp.label).zip(predictionsdtr)
testMSEdtr = labelsAndPredictionsdtr.map(lambda x : (x[0] - x[1]) * (x[0] - x[1])).sum() / float(testData.count())
print('Test Mean Squared Error = ' + str(testMSEdtr))

print(modeldtr.toDebugString())
```

Fig 5.7 Decision tree implementation

The decision tree classification and regression is implemented by specifying some parameters like training data impurity as Gini for the classification and maxdepth of a tree as 3 and maxbins as 32. Then we have tested our model on training data and calculated the test error. We also performed the regression with parameters like training data, impurity as variance for regression maxdepth and maxbins and tested the model by using test data and calculated the test error.

5.3 Spark Streaming and Kafka:

Spark Streaming is used to process real-time data from various sources including like Kafka, Flume, and Amazon Kinesis. Kafka is used to build real-time streaming data pipelines and real-time streaming. Creating a topic and streaming the data is shown in Fig. 5.8

```
 GNU nano 4.8                                                    kafkaconsumer.py
from pyspark.sql.functions import *
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
sc = SparkContext.getOrCreate()
spark = SparkSession(sc)
import os

spark.sparkContext.setLogLevel("ERROR")
df = spark \
  .readStream \
  .format("kafka") \
  .option("failOnDataLoss", "false") \
  .option("kafka.bootstrap.servers", "localhost:9092") \
  .option("subscribe", "weather") \
  .option("startingOffsets", "latest") \
  .load()


df.printSchema()
df=df.selectExpr("CAST(value as STRING)")

datasett = "age INT,sex INT,cp INT,trestbps INT,chol INT,fbs INT,restecg INT,thalach INT,exang INT,oldpeak FLOAT,slope INT,ca INT,thal INT,target INT"
df2 = df \
        .select(from_csv(col("value"), datasett) \
                .alias("heart"))
df3 = df2.select("heart.*")
df3.printSchema()
print("working")
```

Fig. 5.8 Creating a Kafka topic and streaming the data

Apache Spark is a popular open-source framework that ensures data processing with lightning speed and supports various languages like Scala, Python, Java, and R.

Pyspark is a Python API for Spark that lets you harness the simplicity of Python and the power of Apache Spark in order to tame Big Data. So we are importing the pyspark API and also Pyspark sql, which is a higher-level abstraction module over the PySpark Core. It is majorly used for processing structured and semi-structured datasets. It also provides an optimized API that can read the data from the various data source containing different files formats.

SparkContext is the entry point to any spark functionality. When we run any Spark application, a driver program starts, which has the main function and your SparkContext gets initiated here. The driver program then runs the operations inside the executors on worker nodes. SparkContext uses Py4J to launch a JVM and creates a JavaSparkContext. By default, PySpark has SparkContext available as **'sc'.**

Since Spark 2.0 SparkSession has become an entry point to PySpark to work with RDD, DataFrame. getOrCreate() is useful when applications may wish to share a SparkContext. We can use it to share a SparkContext object across Applications. And We can re-use broadcast variables and temp tables across. To Stop Info messages displaying on the spark console we use setLogLevel = 'ERROR' then we create a readstream to read data from the kafka and we specify some options like kafka.bootstrap.server from which Ip address we are getting data and to which port then we subscribe to the topic which we are interested in and we have startingoffset value as latest where we get the latest data, earlier where we get data from the beginning to the end. But we need latest data, so we specify the value as 'latest'.

The data which we receive from the kafka is in byte format, so we convert byte format to string but in our .csv file we have only integer value, so we typecast the convert string data to integer data and we are printing the schema to see the datatypes.

```
a = assembler.transform(df3)
b=a.select("features","target")
predictionrf = rfModel.transform(b)
query1 = predictionrf.select("prediction")
query = query1 \
  .selectExpr("to_json(struct(*)) AS value") \
  .writeStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "localhost:9092") \
  .option("topic", "testing") \
  .option("checkpointLocation", "/opt/spark/spark-3.1.2-bin-hadoop3.2/hd/") \
  .start()


query.awaitTermination()
```

Fig 5.9 Writing the predicted data to Kafka

After receiving the data from the Kafka using read stream, we predict whether the user has heart disease or not by using random forest and we are storing the result back to kafka by creating a new topic called testing and wait for the new result to come.

```
from kafka import KafkaProducer
import tkinter
from kafka import KafkaConsumer
from json import loads
def takeInput():
    producer = KafkaProducer(bootstrap_servers='localhost:9092')
    age1 = int(age.get())
    sex1=int(sex.get())
    chestpain1=int(chestPain.get())
    trestbps1 = int(rbp.get())
    chol1 = int (serumChol.get())
    fbs1=int(FBS.get())
    ecg1=int(ECG.get())
    thalach1 = int(thalach.get())
    exang1 = int(exang.get())
    oldpeak1 = float(oldpeak.get())
    slope1=int(slope.get())
    ca1=int(ca.get())
    thal1=int(thal.get())
    msg = f'{age1},{sex1},{chestpain1},{trestbps1},{chol1},{fbs1},{ecg1},{thalach1},{exang1},{oldpeak1},{slope1},{ca1},{thal1},{1000}'
    producer.send('weather', bytes(msg, encoding='utf8'))
    print(f'sending data to kafka')
    consumer = KafkaConsumer('testing', bootstrap_servers=['localhost:9092'],
                        auto_offset_reset='latest', enable_auto_commit=True,
                        auto_commit_interval_ms=1000)

    for message in consumer:
        message = message.value.decode('utf-8')
        message = message[14:17]

        substituteWindow = tkinter.Tk()
        substituteWindow.geometry('640x480-8-200')
        substituteWindow.title("RESULT PREDICTION")

        substituteWindow.columnconfigure(0, weight=2)
        substituteWindow.columnconfigure(1, weight=1)
        substituteWindow.columnconfigure(2, weight=2)
        substituteWindow.columnconfigure(3, weight=2)
        substituteWindow.rowconfigure(0, weight=1)
        substituteWindow.rowconfigure(1, weight=10)
        substituteWindow.rowconfigure(2, weight=10)
        substituteWindow.rowconfigure(3, weight=1)
        substituteWindow.rowconfigure(4, weight=1)
        substituteWindow.rowconfigure(5, weight=1)
```

Fig. 5.10 Writing the data to the message stream using Kafka

We are importing the kafkaproducer and kafkaconsumer classes and collecting the data from the user and creating a topic and sending this data to the messaging system and receiving the result from the messaging system and displaying it to the user.

```
FBSFrame = tkinter.LabelFrame(mainWindow, text="Fasting BP(0-4)")
FBSFrame.grid(row=3, column=2)
FBSFrame.config(font=("Courier", -15))
FBS= tkinter.Entry(FBSFrame)
FBS.grid(row=2, column=2, sticky='nw')

ECGFrame = tkinter.LabelFrame(mainWindow, text="ECG (0,1,2)")
ECGFrame.grid(row=4, column=0)
ECGFrame.config(font=("Courier", -15))
ECG = tkinter.Entry(ECGFrame)
ECG.grid(row=2, column=2, sticky='nw')


thalachFrame = tkinter.LabelFrame(mainWindow, text="thalach(71-202)")
thalachFrame.grid(row=4, column=1)
thalachFrame.config(font=("Courier", -15))
thalach = tkinter.Entry(thalachFrame)
thalach.grid(row=2, column=2, sticky='nw')

exangFrame = tkinter.LabelFrame(mainWindow, text="exAngina(0/1)")
exangFrame.grid(row=4, column=2)
exangFrame.config(font=("Courier", -15))
exang = tkinter.Entry(exangFrame)
exang.grid(row=2, column=2, sticky='nw')

oldpeakFrame = tkinter.LabelFrame(mainWindow, text="Old Peak(0-6.2)")
oldpeakFrame.grid(row=5, column=0)
oldpeakFrame.config(font=("Courier", -15))
oldpeak = tkinter.Entry(oldpeakFrame)
oldpeak.grid(row=2, column=2, sticky='nw')

slopeFrame = tkinter.LabelFrame(mainWindow, text="Slope(0,1,2)")
slopeFrame.grid(row=5, column=1)
slopeFrame.config(font=("Courier", -15))
slope = tkinter.Entry(slopeFrame)
slope.grid(row=2, column=2, sticky='nw')

caFrame = tkinter.LabelFrame(mainWindow, text=" C. A (0-3)")
caFrame.grid(row=5, column=2)
caFrame.config(font=("Courier", -15))
ca = tkinter.Entry(caFrame)
ca.grid(row=2, column=2, sticky='nw')

```

Fig 5.11 Designing the UI using tkinter model

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit The implementation of UI is shown in fig 5.11

Execute the system

Step 1: Start Zookeeper

> cd zookeeper/

> sudo bin/zkServer.sh start

2: Start Kafka

>cd kafka/

> bin/kafka-server-start.sh config/server.properties &

Step 3: Create topic for Kafka (this should be done only once)

> bin/kafka-topics.sh --create --zookeeper --partitions 1 --topic whether localhost:9092 --replication-factor 1

> bin/kafka-topics.sh --create --zookeeper --partitions 1 --topic testing localhost:9092 --replication-factor 1

Step 4: Execute streaming code

> Downloads/project/kafkac.p

Step 5: Job submission on spark for prediction

> ./bin/spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2 ./hd/kafkaconsumer.py

6. RESULTS AND DISCUSSIONS

The proposed work is carried out in a single node cluster with core i5 processor having 8GB RAM in Linux platform through spark which integrates machine learning models with two stages, first involves analysis on healthcare dataset to build the machine learning model. The second uses the model in production to make predictions on health data streams. The performance of the models is evaluated by using accuracy, precision and recall.

Performance evaluation of machine learning algorithms: As stated in the previous discussion, machine learning model is tested on heart disease data set in the ratio 70-30. The diagnosis accuracy, precision and recall is calculated as:

$$\text{ACCURACY} = \frac{TP + TN}{TP + TN + FP + FN} \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

$$\text{PRECISION} = \frac{TP}{TP + FP} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

$$\text{RECALL} = \frac{TP}{TP + FN} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Confusion matrix:

Confusion matrix describes performance of a random forest. It contains information about actual and predicted classifications performed by a classifier. At the best accuracy of 87.30% which minimizes maxBins, maxDepth, numTrees parameters with Entropy impurity, the total test simple is equal to 88 and

- True positive (TP) = 39 , True negative (TN) = 38
- False Positive (FP) = 5 , False negative (FN) = 6
- Sensitivity = $100 * TP/ (TP+FN) = 100*39/45 = 86.66\%$
- Specificity = $100 * TN/ (FP+TN) = 100 * 38/43 = 88.37\%$

Table 6.1 Confusion Matrix of a classifier used for classification of heart disease

Predicted	Actual	
	Heart Disease	No Heart Disease
Heart Disease	39	5
No Heart Disease	6	38

A GUI is created for the users to enter the heart disease attributes values as shown in fig. 6.1

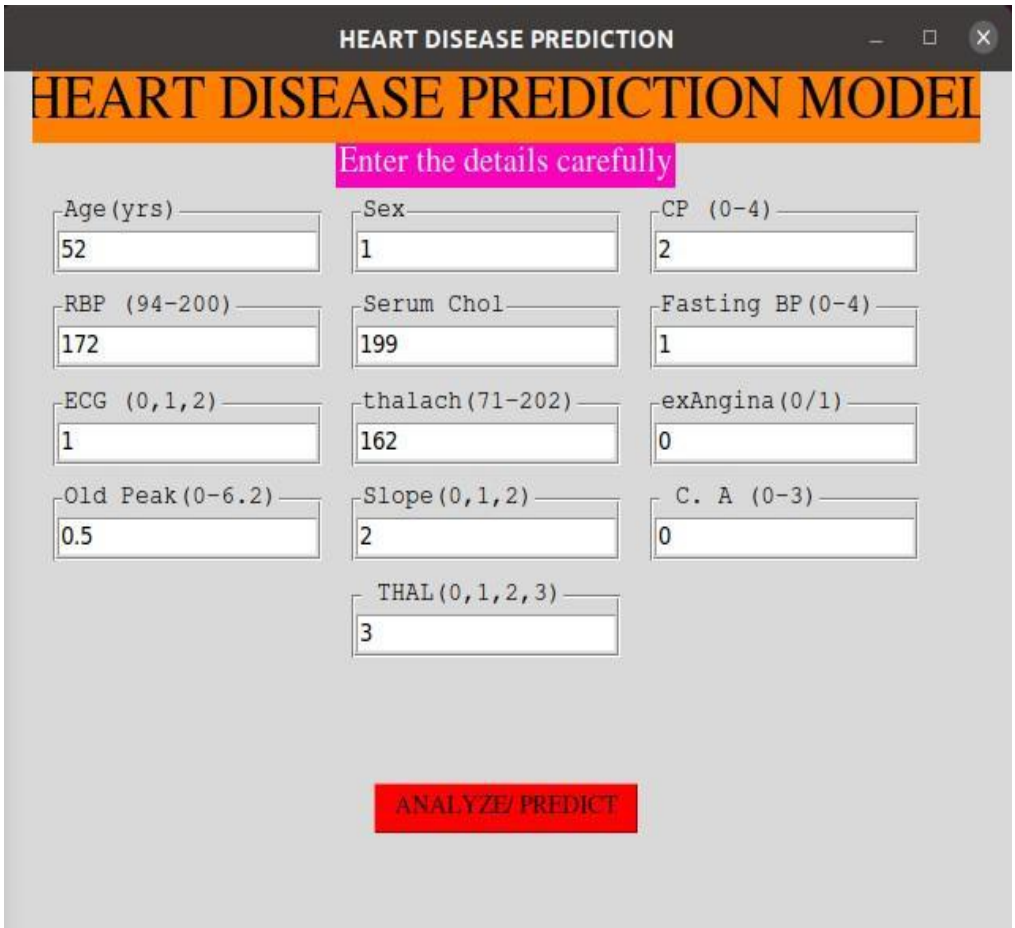

Fig 6.1 The User Interface.

Fig 6.2 The output displaying the presence of heart disease based on the details provided by the user

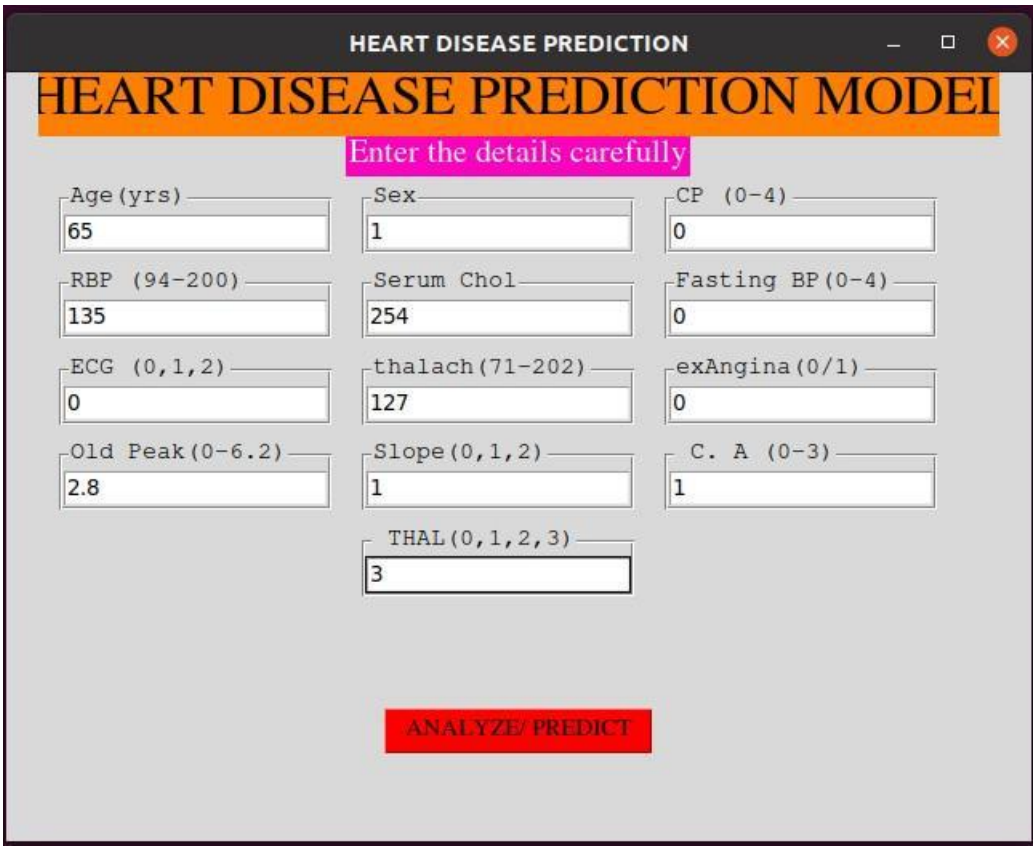

Fig 6.3 The User Interface.

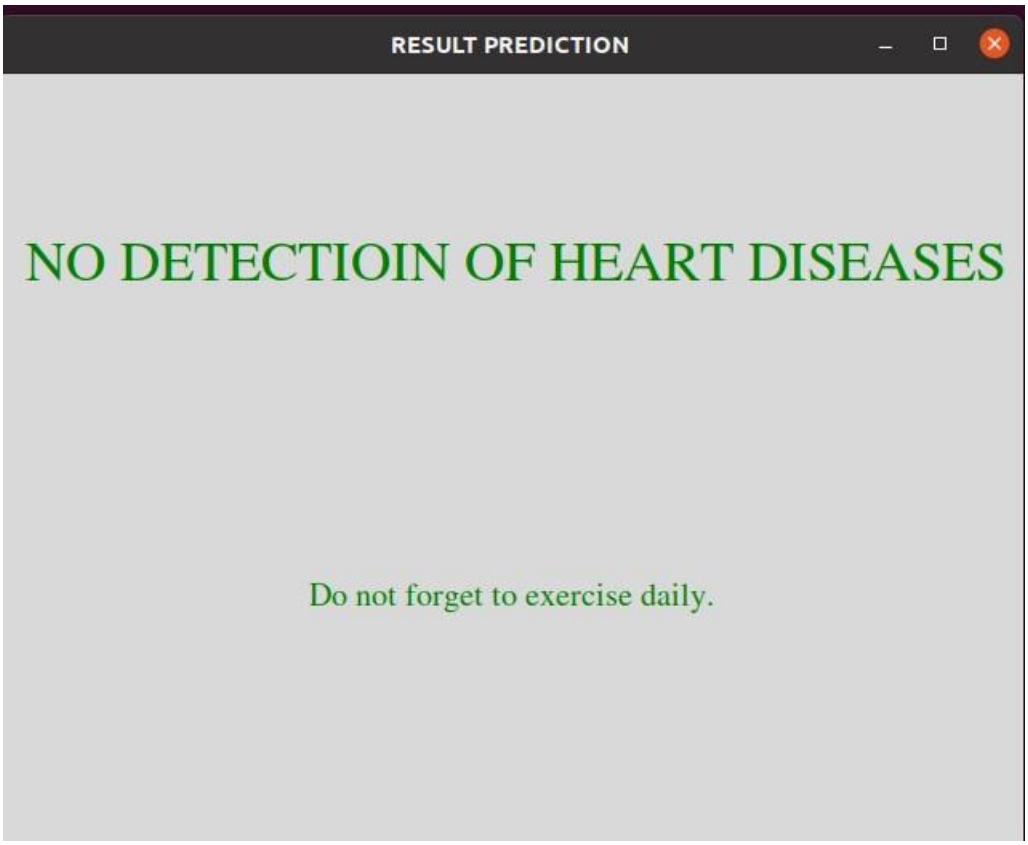

Fig. 6.4 Output displaying that there is no detection of heart disease

The recorded accuracy, precision and recall values for all the models in spark environment is shown in table 6.2

Table 6.2 Table showing the Accuracy, Precision and Recall of the models used

Method	Accuracy in %	Precision in %	Recall in %
Logistic Regression	86	82	94
Random Forest	87	84	95
Gradient Boost	85	93	79
Decision Tree	85	92	78
Linear support vector machine	86	77	95

The visualization bar graph for the table is shown is fig 6.5

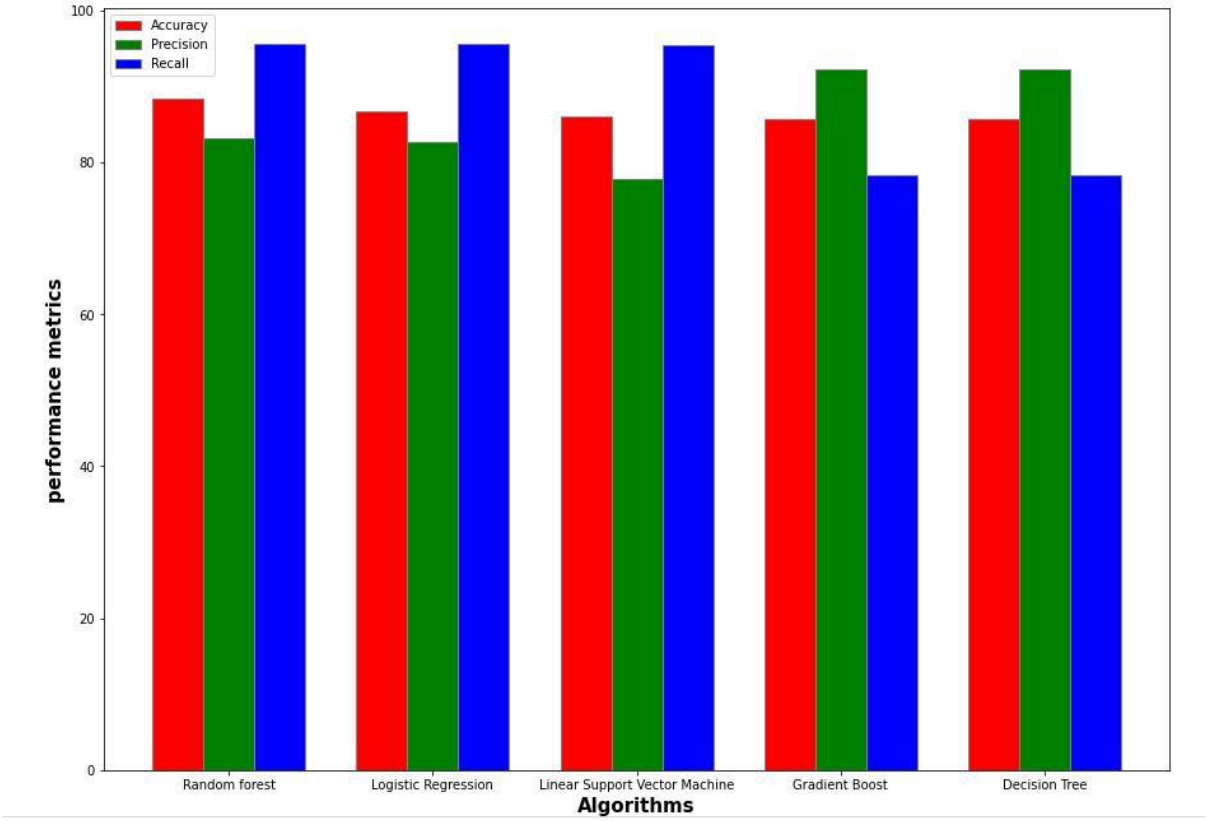

Fig 6.5 Graph representation of the performance metrics of the algorithms

It is observed that all of the algorithms worked similarly well, but random forest is performing somewhat better than the other algorithms and it is also observed that when we are loading data into data frames and performing the experiment we are getting more recall values and less precision values and when we are loading the data into rdd and performing the experiment we are getting more precision values and less recall values.

7. CONCLUSION AND FUTURE SCOPE

Developing a distributed healthcare analytics system using traditional analytical tools is extremely complex, while exploiting open source big data technologies can do it in a simpler and more effective way. This project focuses on applying real-time classification model on heart disease attributes for continuous monitoring of the patient's health. Once the data is streamed from the data source(s), the proposed heart disease monitoring system which is based on Spark framework, uses the random forest algorithm with MLlib for developing the prediction model to predict heart disease. Sensitivity, Specificity and Accuracy are calculated for evaluating of the prediction model. Integrating other big data technologies to our approach will be more efficient.

The dataset used in the current process is historical data. So, in order to increase the accuracy of the model, the data from IoT Devices, Medical Devices, Streaming Systems, etc, can be streamed in real-time through Kafka and Apache Spark. This will help get faster and accurate results with more efficiency. The current proposed system has only one node. More data can be collected and used through connection with multiple clusters. Having multiple clusters can help improve scalability and performance.

REFERENCES

[1] Archana Singh and Rakesh Kumar, "Heart Disease Prediction Using Machine Learning Algorithms," 2020 International Conference on Electrical and Electronics Engineering (ICE3), 2020, pp. 452-457, doi: 10.1109/ICE348803.2020.9122958.

[2] G. Vaishali and V. Kalaivani, "Big data analysis for heart disease detection system using map reduce technique," 2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16), 2016, pp. 1-6, doi: 10.1109/ICCTIDE.2016.7725360.

[3] Ahmed Ismail, Samir Abdlerazek, I.M.El-Henawy, Big Data Analytics in Heart Disease Prediction, JATIT, JUNE 2020

[4] A. Ed-Daoudy and K. Maalmi, "Real-time machine learning for early detection of heart disease using big data approach," 2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS), 2019, pp. 1-5, doi: 10.1109/WITS.2019.8723839.

[5] Sharmila Rengaswamy, Chellammal Surianarayanan, Pethuru Raj Chellaih, "Machine Learning Based Method for Prediction of Heart Disease in Big Data Environment, IJITEE, APRIL 2020.

[6] A. K. Shukla, "Prediction of Heart Disease through Machine Learning Algorithms and Techniques," 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2021, pp. 1-6, doi: 10.1109/ICRITO51393.2021.9596319.

[7] Rallapalli S., Gondkar R.R., Madhava Rao G.V, "Cloud Based KMeans Clustering Running as a MapReduce Job for Big Data Healthcare Analytics Using Apache Mahout," In: Advances in Intelligent Systems and Computing, 2016, vol 433. Springer, New Delhi.

[8] Kompella Sri Charan, Kolluru S S N S Mahendranath, "Heart Disease Prediction Using Random Forest Algorithm", International Research Journal of Engineering and Technology (IRJET), March 2022

[9] L. Nair, S. Shetty, "Applying spark-based machine learning model on streaming big data for health status prediction", Computers & Electrical Engineering, vol 65, pp. 393- 399, 2018.

[10] T.Nagamani, S.Logeswari and B. Gomathy, "Heart Disease Prediction using Data Mining with Mapreduce Algorithm", International Journal of Innovative Technology and Exploring Engineering, Volume 8, Issure 3, 2019

[11] Senthil Kumar Mohan, Chandrasegar Thirumalai and Gautam Srivastava, “Effective Heart Disease Prediction using Hybrid Machine Learning Techniques”, IEEEAccess, Volume 7, pp 81542- 81554, 2019

[12] Prema Jain and Amandeep Kaur, “Big Data Analysis for Prediction of Coronary Artery Disease”, IEEE, pp 188-193, 2018

[13] Vinitha s, Sweetlin s, Vinusha H and sajini s, “Disease Prediction using Machine Learning over Big Data”, Computer Science & Engineering: An International Journal, Volume 8, No 1, pp 1-8, 2018

[14] R. Venkatesh, C. Balasubramanian and M. Kaliappan, “ Development of Big Data Predictive Analytics Model fior Disease Prediction using Machine Learning Technique”, Journal of Medical Systems, 2019

[15] M. Nikhil Kumar, K. V. S. Koushik, K. Deepak, “Prediction of Heart Diseases Using Data Mining and Machine Learning Algorithms and Tools” International Journal of Scientific Research in Computer Science, Engineering and Information Technology ,IJSRCSEIT 2019.