SMT. PARVATIBAI CHOWGULE
OF ARTS AND SCIENCE
GOA-403601, GOA(INDIA)
Computer Science and Engineering

Clustered Machine Learning for Predicting Diabetes

Submitted By:
Alexander Roque Rodrigues
SU170331
TYBSC

Submitted To:
Mrs. Ashweta Fondekar
Asst. Professor
Dept. of CSE

June-February
2019-2020
Graduation Dissertation

1 Acknowledgements

Contents

Abstract

Your abstract.

2 Introduction

In recent days, there has been a sharp increase in the cases of diabetes mellitus. Diabetes mellitus is on the rise amongst many people and the rate of contracting this lifestyle disease could be reduced significantly if proper measures and precautions were to be instilled amongst people the number of people can be reduced.

Machine learning is a growing field in computer science. With the development and introduction of many algorithms the prediction and accuracy of the predictions itself has improved substantially. Machine learning and healthcare systems are also becoming increasingly popular in the healthcare sector.

The project encompasses the qualities of Remote Patient Monitoring (RPM) and Clinical Decision Support (CDS). RPM provides medical facilities that have the ability to transmit patient data to healthcare professionals who might very well be halfway around the world. RPM can monitor blood glucose levels and blood pressure. It is particularly helpful for patients with chronic conditions such as type 2 diabetes, hypertension, or cardiac disease. Data collected and transmitted via PRM can be used by a healthcare professional or a healthcare team to detect medical events such as stroke or heart attack that require immediate and aggressive medical intervention. Data collected may be used as part of a research project or health study. RPM is a life-saving system for patients in remote areas who cannot access face-to-face health care. CDS analyzes data from clinical and administrative systems. The aim is to assist healthcare providers in making informed clinical decisions. Data available can provide information to medical professions who are preparing diagnoses or predicting medical conditions like drug interactions and reactions. CDS tools filter information to assist healthcare professionals in caring for individual clients.

The objective of this project is to create a system that is able to use the machine learning algorithms and predict the outcome of the parameters entered into the algorithm and help the patient draw a conclusion whether or not he/she has the same traits exibhited by simillar patients that have diabetes. Also the system should have a UI that is capable of displaying the data of the patients to the doctor and to the patients themselves for further interpretation.

3 Software Requirements

The main function of this project is to enable the doctors to advise their patients with the help of the prediction software. The system should be accessible to the patient as weel as the doctor, therefore it should have a web interface for the two parties to interact with.

3.1 Functionalities for Doctors

As an owner of a doctors account a doctor should be able to:

- Add new patients.
- Add new observations for the machine learning algorithm to predict.
- Analyse the patients previous records.
- Leave notes for patients to act on.

3.2 Functionalities for Patients

As the patient, one should be able to:

- Should be able to see the predicted risk of developing diabetes.
- Should be able to view historic data.
- Should be able to view notes or suggestions left by doctor.

3.3 Functionalities for Master Nodes

As the patient, one should be able to:

- Should be able to see the predicted risk of developing diabetes.
- Should be able to view historic data.
- Should be able to view notes or suggestions left by doctor.

3.4 Functionalities for Slave Nodes

As the patient, one should be able to:

- Should be able to see the predicted risk of developing diabetes.
- Should be able to view historic data.
- Should be able to view notes or suggestions left by doctor.

4 Hardware Requirements

4.1 Raspberry Pi

According to raspberrypi.org, the Raspberry Pi 3 Model B is the earliest model of the third-generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016. Some of the key features of this single board computer or SBC are:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU.
- 1GB RAM.
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board.
- 100 Base Ethernet.
- 40-pin extended GPIO.
- 4 USB 2 ports.
- Full size HDMI.
- Micro SD port for loading the operating system and storing data.

In this project I will be using 3 Raspberry Pi's to implement a cluster and deploy the machine learning algorithm on.

4.2 Switch

A network switch was used in the project to make up for the lack of ethernet ports available on the router. The dumb network switch was able to connect upto 3 Raspberry Pi and one cable back to the network router itself to connect the switch to the main network.

4.3 Router

A router is required to assign internet protocol addresses to the nodes using dynamic host control protocol (DHCP). The router also is responsible for displaying the nodes connected to the network thereby displaying hostnames and making it more easier to capture the addresses of each node.

4.4 Master Node

The master node is assigned the task of managing the the slave nodes connected to the network. The master node and the slave nodes should be connected to the same database to run and execute queries.

5 Technology Stack

5.1 Python

5.1.1 Pandas

5.1.2 Sklearn

5.1.3 Numpy

5.1.4 Itertools

5.2 MYSQL

5.3 Apache Web Server

5.4 PHP

5.5 AJAX

5.6 GitHub

6 Finding the Right Algorithm

6.1 Introduction

For our predictions we need to use algorithms that give us the maximum accuracy and once we find that algorithm we will further need to tune the selected algorithm from the algorithms we selected to optimise the performance of the predictions generated by the machine learning algorithm. For this purpose we turn to the EDA or exploratory data analysis stage of any machine learning project.

6.2 Algorithms

6.2.1 Decision Tree

Decision trees are flowcharts that represent the decision-making process as rules for performing categorization. Decision trees start from a root and contain internal nodes that represent features and branches that represent outcomes. As such, decision trees are a representation of a classification problem. Decision trees can be exploited to make them easier to understand. Each decision tree is a dis-junction of implications (i.e., if–then statements), and the implications are Horn clauses that are useful for logic programming. A Horn clause is a dis-junction of literals. On the basis that there are no errors in the data in the form of inconsistencies, we can always construct a decision tree for training datasets with 100% accuracy. However, this may not roll out in the real world and may indicate overfitting, as we will discuss.

Classifying an example involves subjecting the data to an organized sequence of tests to determine the label. Trees are built and tested from top to bottom as so:

1. Start at the root of the model
2. Wait until all examples are in the same class
3. Test features to determine best split based on a cost function
4. Follow the branch value to the outcome
5. Repeat number 2
6. Leaf node output

The central question in decision tree learning is which nodes should be placed in which positions, including the root node and decision nodes. There are three main decision tree algorithms. The difference in each algorithm is the measure or cost function for which nodes, or features, are selected. The root is the top node. The tree is split into branches; evaluated through a cost function; and a branch that doesn't split is a terminal node, decision, or leaf.

Decision trees are useful in the way that acquired knowledge can be expressed in an easy to read and understandable format (see Figure 4-1). It mimics human decision-making whereby priority—determined by feature importance, relationships, and decisions—is clear. They are simple in the way that outcomes can be expressed as a set of rules. Figure 4-1. Decision tree of n = 2 nodes Decision trees provide benefits in how they can represent big datasets and prioritize the most discriminatory features. If a decision tree depth is not set, it will eventually learn the data presented and overfit.

It is recommended to set a small depth for decision tree modeling. Alternatively, the decision tree can be pruned, typically starting from the least important feature, or the incorporation of dimensionality reduction techniques.

Overfitting is a common machine learning obstacle, and not limited to decision trees. All algorithms are at risk of overfitting, and a variety of techniques exist to overcome this problem. Random forest or jungle decision trees can be extremely useful in this. Pruning reduces the size of a decision tree by removing features that provide the least information. As a result, the final classification rules are less complicated and improve predictive accuracy. The accuracy of a model is calculated as the percentage of examples in the test dataset that is classified correctly.

- True Positive: Where the actual class is yes, and the value of the predicted class is also yes.
- False Positive: Actual class is no, and predicted class is yes.
- True Negative: The value of the actual class is no, and the value of the predicted class is no.
- False Negative: When the actual class value is yes, but predicted class is no.

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

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

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

Classification is a common method used in machine learning; and ID3 (Iterative Dichotomizer 3), C4.5 (Classification 4.5), and CART (Classification And Regression Trees) are common decision tree methods where the resulting tree can be used to classify future samples.

6.2.2 Random Forest

6.2.3 Gradient Boosting

6.2.4 Support Vector Machine

Support Vector Machines is an algorithm that is capable of handling linear as well as data that occurs non-linearly. For example, for a long time, SVMs were the best choice for MNIST dataset classification, thanks to the fact that they can capture very high non-linear dynamics using a mathematical trick, without complex modifications in the algorithm.

6.2.5 Linear Support Vector Machines

Let us consider a dataset of features we want to classify.

$$X = \{x_1, x_2, x_3, ..., x_n\}$$

For the target variable, we will consider the dataset Y, with target outcomes as $\{0, 1\}$ indicating a true or false condition.

$$Y = \{y_1, y_2, y_3, ..., y_n\}$$

6.2.6 Perceptron

6.2.7 Multilayered Perceptron

6.2.8 Linear Regression

Linear regression is a forecasting technique that can be use to predict the future of a number series based on the historic data given. The perks of using a linear regression model are as follows:

- produces decent and easy to interpret results.
- is computationally inexpensive.
- conversion of algorithm into code does not take much effort or time.
- numeric values as well as nominal values support is offered.

However, a major drawback of linear regression is that it **poorly models nonlinear data**.
Considering a dataset that has values ranging from $X = \{x_1 + x_2 + x_3 + ... + x_n\}$ where all the entries of the dataset are real numbers. Each x_i is associated with a corresponding value of y_i from the dataset $Y = \{y_1 + y_2 + y_3 + ... + y_n\}$.

The most basic equation for linear regression can be expressed via this simple equation.

$$y = \beta_0 x + \beta_1 + \epsilon$$

So to minimize the error in the predictions, a way to calculate the error should be formulated. A loss function in machine learning is simply a measure of how different the predicted value is from the actual value. The Quadratic Loss Function to calculate the loss or error in our linear regression model. It can be defined as:

$$L(x) = \sum_{i=1}^{n} (y_i - p_i)^2$$

Therefore using the method of Least Squares, we can find the values of β_0 and β_1.

$$\beta_0 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The linear regression model with an error value close to 1.00 indicates a perfect model and those with values closer to 0.00 indicates a model that delivers poor performance.

6.2.9 Logistic Regression

Logistic Regression is a classification method which is based on the probability for a sample to belong to a class. As our probabilities must be continuous in R and should bounded between the lower limit of 0 and upper limit of 1 it's necessary to introduce a threshold function to filter the term z. The name logistic comes from the decision to use the sigmoid (or logistic) function, defined as shown below.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The sigmoid function in its domain is R has two asymptotes[1] that occur at 0 and 1. So, the probability for a sample to belong to a class from 0 and 1 can be defined as:

$$P(y|\bar{x}) = \sigma(\bar{x}; \bar{w})$$

At this point, finding the optimal parameters is equivalent to maximizing the log-likelihood[2] given the output class:

$$L(\bar{w}; y) = logP(y|\bar{w}) = \sum_i logP(y_i|\bar{x_i}, \bar{w})$$

[1] In analytic geometry, an asymptote of a curve is a line such that the distance between the curve and the line approaches zero as one or both of the x or y coordinates tends to infinity.

[2] The log-likelihood is, as the term suggests, the natural logarithm of the likelihood.

6.2.10 K-Nearest Neighbours

The k-means algorithm is based on the (strong) initial condition to decide the number of clusters through the assignment of k initial centroids or means:
Then the distance between each sample and each centroid is computed and the sample is assigned to the cluster where the distance is minimum. This approach is often called minimizing the inertia of the clusters, which is defined as follows:
The process is iterative—once all the samples have been processed, a new set of centroids K (1) is computed (now considering the actual elements belonging to the cluster), and all the distances are recomputed. The algorithm stops when the desired tolerance is reached, or in other words, when the centroids become stable and, therefore, the inertia is minimized. Of course, this approach is quite sensitive to the initial conditions, and some methods have been studied to improve the convergence speed. One of them is called k-means++ (Karteeka Pavan K., Allam Appa Rao, Dattatreya Rao A. V., and Sridhar G.R., Robust Seed Selection Algorithm for K-Means Type Algorithms, International Journal of Computer Science and Information Technology 3, no. 5, October 30, 2011), which selects the initial centroids so that they are statistically close to the final ones. The mathematical explanation is quite difficult; however, this method is the default choice for scikit-learn, and it's normally the best choice for any clustering problem solvable with this algorithm.

6.2.11 Finding Optimum Number of Clusters

One of the most common disadvantages of k-means is related to the choice of the optimal number of clusters. An excessively small value will determine large groupings that contain heterogeneous elements, while a large number leads to a scenario where it can be difficult to identify the differences among clusters. Therefore, we're going to discuss some methods that can be employed to determine the appropriate number of splits and to evaluate the corresponding performance.

7 Writing Code

```
import itertools
import mysql.connector
from colorama import init
from colorama import Fore, Back, Style
from mysql.connector import Error
import masterenvironment as en
from os import system
from os import chdir
init()

def divide_chunks(l, n):
    for i in range(0, len(l), n):
        yield l[i:i + n]


def readNodes():
    global lineList
    with open(en.fileName) as f:
        lineList = f.readlines()
    lineList = [line.rstrip('\n') for line in open(en.fileName)]


def db_connect():
    curr_node = 0
    curr_chunk = 0
    print("Connecting to DB ", end="")
    try:
        mydb = mysql.connector.connect(
            host=en.host, user=en.user, passwd=en.passwd, database=en.
    database)
        mycursor = mydb.cursor()
        print(Fore.GREEN+"[SUCCESS]"+Style.RESET_ALL)
        pending = []
        query = "select * from {} where PredictedOutcome {}".format("
    diagnosis", "IS NULL")
        mycursor.execute(query)
        myresult = mycursor.fetchall()
        for x in myresult:
            pending.append(x[0])

        x = list(divide_chunks(pending, en.chunkSize))
        executeStatus = 0
        nodeCount = len(lineList)
        nodeCount = nodeCount - 1
        chunkCount = len(x)
        while(executeStatus != 1):
            temp = x[curr_chunk]
            temp = str(temp)
            temp = temp[1:-1]
            temp = temp.replace(" ", "")
            query = "ssh pi@{} python3 hive-ml/slave.py {}".format(
    lineList[curr_node], temp)
            query = str(query)
            system(query)
            curr_chunk = curr_chunk + 1
            if(curr_node == nodeCount):
```

```
                curr_node = 0
            else:
                curr_node = curr_node+1
            if(curr_chunk == chunkCount):
                executeStatus = 1

    except Error as e:
        print(Fore.RED+"[FAILED]"+Style.RESET_ALL)
        print(e)
        exit(0)

if __name__ == "__main__":
    readNodes()
    db_connect()
```

8 Conclusion