USING EXTREME LEARNING MACHINE TO PREDICT HYPERTENSION FROM A CLINICAL DATASET

A PROJECT REPORT

Submitted by

ABHAY GOYAL [Reg No: RA1511003010397]
DURGESH SINGH [Reg No: RA1511003010179]

Under the guidance of
S. SHARANYA
(Assistant Professor, Department of Computer Science & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2019

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**USING EXTREME LEARNING MACHINE TO PREDICT HYPERTENSION FROM A CLINICAL DATASET**" is the bonafide work of " **ABHAY GOYAL [Reg No: RA1511003010397], DURGESH SINGH [Reg No: RA1511003010179]**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

S. SHARANYA
SUPERVISOR
Assistant Professor
Dept. of Computer Science & Engineering

SIGNATURE

Dr. B. AMUTHA
HEAD OF THE DEPARTMENT
Dept. of Computer Science & Engineering

Signature of the Internal Examiner

Signature of the External Examiner

ABSTRACT

Hypertension is a disease that sometimes leads to life threatening diseases such as coronary artery disease, stroke, heart failure, thickening of the heart muscle and other conditions ,if left untreated. The real cause of hypertension as found by many researchers and clinicians is that due to the overburden of work that people face at their work front and are not able to cope with the amount of stress their work gives to them. There have been many different studies on the ways and methods to actually find in advance what are the symptoms and something of any use could be done. Extreme Learning Machine are emerging as promising solution for many real-life problems. This work uses Extreme Learning Machine to predict the hypertension based on patient medical records. The heterogeneous data includes vital features like sex, body weight, height, heart failure, systolic blood pressure, diastolic blood pressure, max blood pressure to predict the hypertension. The model takes the help of Radial Basis Function (RBF) to accomplish the task. The result show that the model achieves 90% accuracy with the test train split of 80-20.

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. Sandeep Sancheti**, Vice Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. C. Muthamizhchelvan**, Director, Faculty of Engineering and Technology, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. B. Amutha**, Professor & Head, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

We are extremely grateful to our Academic Advisor **Dr. K. Annapurani**, Associate Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her great support at all the stages of project work.

We would like to convey our thanks to our Panel Head, **Dr. Revathi Venkataraman**, Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her inputs during the project reviews.

We register our immeasurable thanks to our Faculty Advisor, **Mrs. R.Radha**, Assistant Professor & **Mrs. G.K.Sandhia**, Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my supervisor, **Mrs. S.Sharanya**, Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for providing me an opportunity to pursue my project under his mentorship. She provided me the freedom and support to explore the research topics of my interest. Her passion for solving the real problems and making a difference in the world has always been inspiring.

We sincerely thank staff and students of the Computer Science and Engineering Department, SRM Institute of Science and Technology, for their help during my research. Finally, we would like to thank my parents, our family members and our friends for their unconditional love, constant support and encouragement.

ABHAY GOYAL
DURGESH SINGH

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

RBF	Radial Basis Function
GRBF	Gaussian Radial Basis Function
ELM	Extreme Learning Machine
MELM	Modified Extreme Learning Machine
WHO	World Health Organisation
BMI	Body Mass Index
ANN	Artificial Neural Network
GGD	Generalized Gaussian Distribution
SLFN	Single Feedforward Neural Network
COPD	Chronic Obstructive Pulmonary Disease
CI	Confidence Intervals
PWV	Pulse Wave Velocity
ABP	Arterial Blood Pressure
ICP	Intracranial Pressure
AUC	Area Under Curve
RF	Random Forest
TPR	True Positive Rate
RBFNN	Radial Basis Function Neural Network
CPCSSN	Canadian Primary Care Sentinel Surveillance Network

SVM	Support Vector Machine
SD	Standard Deviation

LIST OF SYMBOLS

β	Output Weights
x_i	Training data vector
H	Hidden Layer Output Matrix
T	Training data target matrix
H^{-1}	Moore-penrose generalized inverse of matrix H
H^T	Pseudo inverse of matrix H
g_i	Hidden Layer Output Function
β_i	Output weight of the i-th node

CHAPTER 1

INTRODUCTION

1.1 Hypertension

Hypertension is a health condition of a person that leads to severe illness such as that of stroke, heart disease, and renal failure. The study in the hypertension is done through the machine learning and deep learning methods. According to the a major global organization, namely World Health Organisation (WHO) in its recent survey has shown that force by blood causes one out of the nine people who die and hence it is important for us to work on this.There are more than a billion hypertensive sufferers and around four million patients die every year. Hypertension has not only increased in these over the years but also has been a major cause of loss in revenue due to the loss of manpower. The diagnosis of Hypertension is done when systolic and diastolic readings of blood pressure is greater than 130 and 80 respectively. Hypertension can be divided into stages depending upon the reading of systolic and diastolic blood pressure. The common risk factors include age, gender, Body Mass Index (BMI), stress, smoking habits. Due to the complexity of the data, the doctors are prone to make error due noisy or incomplete data.

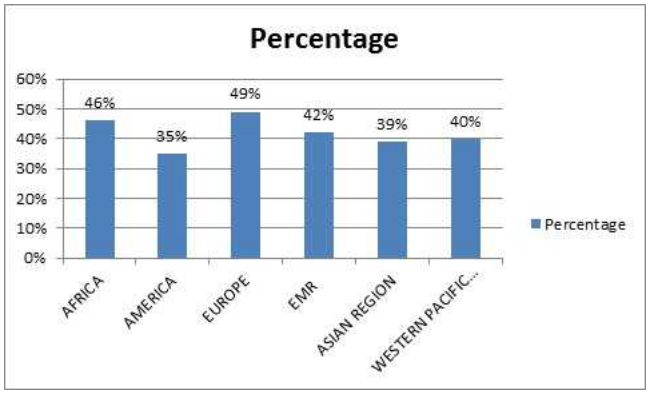


Figure 1.1: Percentage of People Suffering from Hypertension in different regions

1.2 Radial Basis Function & Gaussian Radial Basis Function

Machine Learning is one of the most reasonable options that allows clinicians to identify better pattern in the data set that are sometimes possible are a harder task for humans and improve upon the working of medicine. Feed forward neural network is widely used because of it's obvious benefits but there exists a need for the layers to be tuned based on the parameters which makes feed forward neural network to be time consuming.

Artificial Neural Network (ANN) are mostly utilized in requirements which involves partitions or predictions. It has been show recently that there are several different partitions related to ANN. It is a wide point of research to use it as a universal estimator. The radial basis function neural networks is found among them. This is multi-layered networks and usually shown as connection model. RBFs use hyper-ellipsoids to split the space which has pattern.

RBFs uses the value of distance to get the response value, being function of two arguments , x and c, where $x = (x_1, x_2, \ldots x_k)^T$ is co-ordinate vector of a pattern of the data set and $c = (c_1, c_2, \ldots, c_k)^T$ are parameters to determine kernel positions. The important part of local RBFs is the fact that their reaction magnitude decreases in a linear fashion with distance from the centroids c of the radial function. RBFs are parametrized by a width r. The distance between c and x is small in comparison with the width of the kernel, the value of the kernel is nearly equal to one. The large distance are mapped close to zero. The Gaussian Radial Basis Function (GRBF)s are based on the function called Gaussian Density Function and are denoted by a "width" parameter and a "center" point. The function gives the highest output when the variables are closest to the center position and decreases as the distance from the center increases.This distribution can be parametrized by a real parameter t, this can result in Generalized Gaussian Distribution (GGD). The GGD may denote various forms of function by trying to change the real parameter t.

The GRBF is done by removing the constraints of a probability function. The GRBF is

defined by

$$\phi|x; c, r, \tau| = \exp\left(\frac{-\|x-c\|^t}{r^\tau}\right) \tag{1.1}$$

Training of RBFs is divided into many different types. The important two are: fast learning and total learning. Fast learning has a two-step process. First, the values that govern the basis functions are evaluated using fast, unsupervised clustering methods. Then, By using optimization technique the weights of the basis function parameter is calculated. A full learning scheme optimize all of the parameters in supervised mode. Gradient descent algorithms is very slowly when the learning rate is small. However, if the rate of learning is big, it can be volatile. Hence many learning instructions are based on it are slow as the iterative steps requires lots of time.

1.3 Extreme Learning Machine

A new type of Single Feedforward Neural Network (SLFN) is developed which is known as Extreme Learning Machine (ELM) which was developed to improve the efficiency of SLFN. ELM is auto-implemented without intervention from the user or any iterative tuning. ELM is extremely fast compared to other Neural Networks. In this algorithm the weights and biases that are learning parameters of the hidden nodes can be randomly assigned, and output weights of the network can be determined by using generalized inverse operations. ELM also gives good performance.

In ELM, the input weights are randomly chosen for the hidden nodes, and the SLFNs output weights can be determined using the pseudo-inverse operation of the output matrix. By using this algorithm, many problems in gradient- descent-based learning system can be avoided. There's is another method proposed using the unique features of ELM, named Modified Extreme Learning Machine (MELM)-GRBF proposed by Fernández-Navarro et al. (2011). The main purpose of MELM is the determination of GRBFs. In ELM, the centre and radii of the RBFs are calculated randomly whereas in the MELM-GRBF instruction, the centre are given values by arbitrarily choosing some pattern in the learning data set. Values of the radius is evaluated by solving two equations ensuring the mould fulfills two constraints: coverage and aggregation.

The software and systems made integrate machine learning in order to provide method-

ical and accurate results that a doctor can use to diagnose and cure illnesses. Technologies formed provide timely interventions if made to work with data obtained early in a patients treatment cycle. They provide an insight into the techniques and different methods that have been researched on predicting the possibility of hypertension using the different features. Hypertension has become a priority for health-care providers in the era of high burnout rate and workload. The report provides exhaustive knowledge on how and what changes have taken place over the time.

By using ELM and MELM the speed of the neural network can be boosted and it can be used on the variety of data. Here, Machine learning is used to predict whether a person is hypertensive or not depending upon the digital clinical data-set.

CHAPTER 2

LITERATURE SURVEY

A model intelligent hypertension prediction system was build with the help of machine learning and decision trees. By using the aforementioned technique the researchers were able to solve the well defined goal. The system to use decision trees to predict hypertension was proposed by Sandi et al. (2016). The paper was proposed in 2016. The result illustrated did show the strength of the working model. It produced a result of 93.6%, which is a very good result by using the given data set. In this C4.5 algorithm was used create the decision tree and a total of 1100 data of people was used for the training and testing purpose. The solution presented in the paper to solve this was by developing a smart system of health related to the development of health risk. The prediction for hypertension treatment with machine learning approach and with additional prediction feature. The structure is divided into three parts. Learning process, in this section, the data-set provided and using the approach of decision tree algorithm. This process generates decision trees and produces a classification of the data-set that is inserted into the system, followed by the formation of decision rule. Decision rule is used to predict the diagnosis and prognosis of hypertension. The diagnosis of hypertension section predicts the diagnosis of hypertensive conditions based on the data entered. The information entered is a medical condition parameters obtained from the different measurement results taken during the diagnosis . The information generated in this process is then compared with a decision rule. A similar methods ids used for the prognosis. Similar to diagnosis, the system uses decision trees to predict the different methods in which the decision can be made. The information generated in this process is done by comparing the data with a decision rule that is formed in the previous process. The whole system once finished was installed and it was found to have an accuracy of around 92%.

In 2016, in a paper proposed by LaFreniere et al. (2016), they used ANN for the prediction of hypertension by the use of Canadian Primary Care Sentinel Surveillance Network (CPCSSN) data-set. The model was used for a very large data set(185,371 patients

and 193,656 controls). The result obtained was rather good considering the large data set used. The result obtained was 82% accuracy. There were 11 inputs with 8 hidden nodes and 2 outputs from this model. In this the data set was first pre-processed and then the data related to the 11 inputs are only entered in the training data or testing data. These include age of the patients, their gender, the BMI of the patient, systolic and diastolic blood pressure etc. The previous papers had found that a neural networks with n nodes, n here being 20. In the first concealed stratum there are 5 nodes followed by the second concealed stratum result in the best precision of 92.85%. In LaFreniere et al. (2016), a 3 layer network with 11 intake nodes, 7 concealed nodes and 2 yield points which thus have a total of 91 weight values ((11*7) + (7*2)) was used. It was found that more yield nodes make better linking .The mapping of the input to the output increases accuracy. But it also had its own unpropitious effects of increased processing time and over-fitting of data. In the data-set it was found that the sufferers may be suffering from many different diseases which were important for examination for the researchers. These diseases included hypertension, diabetes, Chronic Obstructive Pulmonary Disease (COPD),etc. In the paper the patients set for control were those who did not go from any of the aforesaid possibilities but did need other treatments for various causes.

In 2018, the paper proposed by Choi et al. (2018), They used waist to height ratio to predict the hypertension in people. The experiment was done in for a period of over 2.8 years in which 1718 participated and it was calculated that if the ratio of the measured waist to measured height was greater than 0.5, then the chances of having hypertension increases by 4.5 times. It was exclusively a result for a small number of people and there might be many disadvantages for this. The baseline characteristics used chi-square test for examination. To approximately find what are the that best forecast hypertension and the different methods of finding accuracy under the AU-ROC of the baseline model made for BMI, WHR and WHtR, and their Confidence Intervals (CI)s, using receiver operating characteristic curves in the group in which hypertension had newly developed. The ratio of waist to height was found to be a better method of calculating the hypertension probability rather than BMI. This is so because BMI was not able to understand the difference between body fat and muscle mass, and hence such people have the same risk of such illnesses as others. The before time discovery of hypertension is

very important for its command and avoidance, in grown ups; but as, blood pressure is not tested very often in adults between 15-21, it is very hard to find. This study has some limitations. First, the sample is low and hence the number of people who took part showed the sign of hypertension after more than 2 years of follow-up were less. Secondly, the population that was studied was only to adults from , and, thus, does not show the population as a whole. Third, there is bias of selection present in the study done.

Koivistoinen et al. (2018) uses Pulse Wave Velocity (PWV) and Blood pressure. The findings from the risk prediction model indicate that PWV forecasts the increasing of blood pressure and gives us a rurals valuable method in hypertension risk forecasting among young adults. The aim of the study was to forecast the increase in force due to blood by reading the PWV and thus predicting hypertension.Pressure due to blood forms the right brachial artery was evaluated in the sitting position after a 5-minute rest with a random-zero sphygmomanometer in 2007 and 2011. Linear regression was used to understand the interdependence among PWV measured in 2007 and force of blood marked which was absent in 2007 ,in 2011. Separate analysis were done on both of the populations. Logistic regression indicated the links of PWV measured in 2007 to the hypertension in 2011 which absent during study in 2007. Logistic regression evaluates approximately 1200. The regression model made also included conventions heart problems causes such as age, sex, high-density lipoprotein cholesterol, low-density lipoprotein cholesterol, triglycerides, glucose, insulin etc. All forecasting inconstants besides age were made upto a certain standard so as to understand the inconstants proportional to each other. Hence, the regression values in the linear regression moulds and the odds ratio in the logistic regression model indicated that there was effect of a 1-SD change in a predictor variable for any given dependent variable. There is no statistical significant interactions between sex, age, or BMI and hypertension in 2011. A important problem of the study was the lack of PWV data in 2011. Hence, the researchers could not work on the effects of pressure due to blood and hypertension on PWV progression.

In a paper proposed by Wadehn et al. (2017), the main hypertension point was Intracranial Hypertension. By the study of Arterial Blood Pressure (ABP) and Cerebral blood flow velocity was used to predict the Intracranial Pressure (ICP). There were 36 patients suffering from traumatic brain injury. The Random Forest (RF) classifier shows

the best results and has an sensitivity of around 69% at the same time the specificity was around 78%. This paper was especially proposed for the patients that are suffering through brain injury to find intracranial Hypertension in the patient's present. The prediction was done using beat detection and signal detection. Linear Support Vector Machine (SVM), NB and RF classifiers were used. Bagging was used to add many different Random Forests and then implicit feature selection was done. 256 decision trees were used with the depth of 8 partitions. The Random Forest classifier was the best achiever when it was about sensitivity to the data and was similar in precision with other classifiers. Both the linear SVM and the NB classifier were surpassed by the RF in terms of sensitivity. Sensitivity is the a very relevant mathematical means of screening sufferers. Hence it was said that the RF is a better method of classification.

Sakr et al. (2018) is another paper which uses a host of machine learning algorithms and tries to compare between them. The data-set used in this paper, includes the information of approximately 7000 patients that had followed the physicians treadmill stress testing. Other important information such as that of the sufferer's prior information, demographics, medicines given, heart related disease risks were found when the examinations were done the staff. The studies done in this paper used the conventional statistical techniques to find the accurate medical outcomes of the said problems. The techniques in the paper were LogitBoost, Artificial Neural Network, Locally weighted Naive Bayes, Bayesian Network, Support Vector Machine, Random Forests. As the date collected is highly imbalanced, SMOTE analysis was done for all the methods given. This paper shows the working of Neural Networks using the gradient descent back-propagation algorithm using concealed points and the momentum thus evaluated using 10-fold cross validation. Bayesian Network classifier using Simulated Annealing gets maximum Area Under Curve (AUC) of around 0.70. The consequences point out that the working of the RTF and SVM models using SMOTE are much better. The RTF and SVM achieve AUC of 0.91 and 0.71 respectively using the dataset with 30% made up examples were made as in contrast with 0.9 and 0.57 and no sampling was done. LogitBoost and Artificial Neural Network moulds have shown a slight improvement using by achieving Area under Curve of 69% and 63% respectively without sampling and AUC of 70% and 67% with 0.3 created synthetic examples. From the various examination functions, it was found that Random Tree Forest mould was found to have achieved the

highest accuracy using the n-fold cross-validation, here n being 10, 93%,with the help of holdout method 70/30 it was found to be 83% and holdout method 80/20 it turned out to be 88%.

The paper by Chen et al. (2018) presents a very natural method of measuring of hypertension namely by tracking the activities done by any person. The lifestyle of the people were studied and it was found that people who had attended 4 or more years of college were better stress managers as this lead to more stress during the early stages of their life when they more mentally fit and hence it let them have better control over themselves. Social support also plays a vital role in stress management and it was found that those who had some kind of social support were better managers of stress. Living situations, financial difficulties were also some of the reasons for these issues. Presently, the results show that the regression mould is mathematically better in forecasting modifications in HRQOL according to lifestyle-based in between changes, problems due to work, and help from friends and family. Educational levels, ethnicity, level of problems at baseline are significant forecasters for forecasting modification in problems; help from family and friends at baseline is the substantial forecasters for forecasting modification in help from social friends.

Echouffo-Tcheugui et al. (2013) provides a comparative study of the present day approaches of the predicting of Hypertension.This paper shows 15 different hypertension prediction models from 11 studies and shows the impact of those studies. Some of the key comparison includes design, predictors, calibration and reclassification ability and impact analysis. The most common factors included are age, sex, BMI.

Srivastava et al. (2013) uses fuzzy computing logic to grade hypertension in five grades namely very low, low, moderate, high, very high. The input used in the given model are BMI, Heart rate, Systolic blood pressure(SBP), Low Density Lipoprotein(LDL), High Density Lipoprotein(HDL), exercise. The approach is unable to learn as other logic do and depends on the definition of the fuzzy utilities.

Ture et al. (2005) try to compare the performance of four algorithms, three decision tree models, and two ANN models, all of which predicts hypertension. Based of sensitive and specific analysis of the model, it infers that the metric used are good variable predictor for analyzing hypertension.

2.1 Inference from the Survey

- Here we see that in Sandi et al. (2016) a system has been built which continuously learns about the different ways in which hypertension is happening and helps practitioners and clinicians as to how they can understand the early signs of hypertension and proper medication can be provided.
- In the LaFreniere et al. (2016) the data has high prediction. It was better than (Sandi et al., 2016) as here ANN was used and provided better accuracy.
- In Choi et al. (2018) a new aspect was introduced of finding the height to waist ratio which was also found significantly important.
- Koivistoinen et al. (2018) use pulse wave velocity to understand how can they me matched with heart rate. Here, they used data from 2007 and 2011 and tried to predict the possibility of hypertension with and without PWV.
- In the Wadehn et al. (2017) , a new method of research was found using the cranial tissue of the brain and how can it be used to predict hypertension. Here they used Random Forest classifiers for the same.
- In Sakr et al. (2018) comparison have been done on the classification can be done using 6 different methods of comparing and contrasting. The report shows that there are many different ways of comparing the different methods of predicting hypertension.

CHAPTER 3

SYSTEM DESIGN

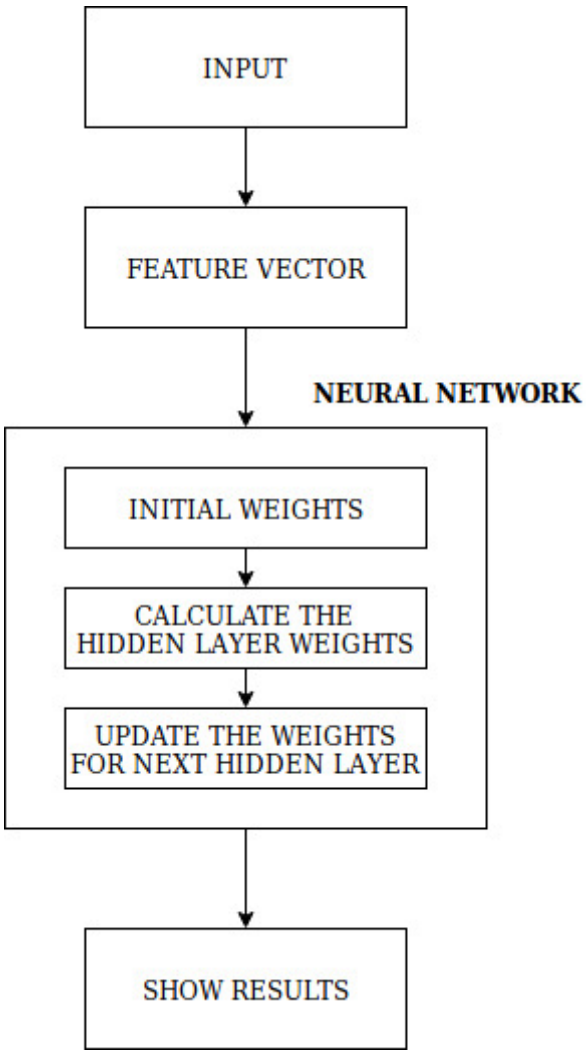


Figure 3.1: Architecture Diagram

As can be seen above, the figure presents the design or flow diagram of how our model looks like. The input data is taken and then the data is cleaned. There is a lot of noise and many different things present in the data. Then extraction the features that are used in the system. By this method that is presented afterwards, It can be seen by experiment that these features are the ones which have most impact on hypertension. After this comes the neural network part.Here radial basis function is used. It is known that, the Radial Basis Function Neural Network (RBFNN) (Manrique et al., 2006) uses

a single hidden layer and for prediction with back-propagation. Instead, it uses the method least square error method to find the best fit and then the requisite weights are given. Further advancements have been made in this. In 2011, a paper presented that instead of giving values randomly, by use of the pattern finding capabilities of RBF to actually give the requisite weights. After the centers are located from the patterns, then the use of k nearest neighbour method to compute the width factor τ. After this, the mapping the smallest distance of distribution to high values of basis function and similarly, the highest distance of distribution to lower values of basis function
After this, the prediction is done and the accuracy is found.

3.1 Hardware Requirements

The following project is done on Dell Inspiron 5558 with the following configuration:

- Processor: Intel® Core™ i5-4210U Processor 1.70GHz 1.70GHz
- RAM: 4.00 GB
- System Type: 64-bit Operating System, x64 based processor
- Hard Disk: 1 TB SATA Drive
- Graphics Card: Nvidia 940M

It is evident that minimal system hardware requirements are required in order to process with the following work. This makes it easier to implement with any-type of machines available.

3.2 Software Requirements

Following software and modules is used in order to perform the exact function of the given project:

- Operating System: Windows 10 Home
- Software: Jet Brains-PyChram
- Language Installed: Python3
- Modules Installed: Numpy, Scipy, Matplolib

CHAPTER 4

PROPOSED SYSTEM

In the proposed system, we try to make another similar prediction system using the technique of extreme machine learning. Using extreme machine learning, we try to make different epochs of the data we have using arbitrary attributes. By using arbitrary attributes, we try to randomize and implement the power of variability to see whether this is better than the other methods researched namely, ANN, NB and RF. Such methods have already been used and may although perform better statistically do not take into account the randomness the society has. By simulating this randomness into our project with the attributes present, we try to understand how would such a prediction work.

4.1 Experimental Design

We have here compared the MELM-GRBF and ELM-GRBF advanced by Huang et al. (2006) (Huang and Chen, 2008) (Lan et al., 2009). In the method advanced by Huang et al. (2006), the centers and radius of the basis functions are chosen arbitrarily and the associations links the concealed strata and the yield strata is solved by the equation(equation number). The important links the ELM-RBF and the Radbas a standard linear combination of intake variants and links between intake and concealed strata whereas in the ELM-RBF method, the length of each mould to its centres weighting the final yield by its radius.

The test mould was made such that the (lambda) value for MELM-GRBF was evaluated by applying values from 0.01,0.02,. . . ,0.10. Similarly, the optimal number of nodes was increased gradually in intervals of 5(5,10) .

4.2 Factors of Interest

Based on the study done by us on the data-set that was found,defined 8 elements of concern for intake to the ELM model. These include BMI, systolic and diastolic blood pressure. Patients which had missing data was not included. We considered that patients for whom the above data was unavailable did not indicate symptoms of hypertension.

4.2.1 Body Mass Index(BMI)

BMI is measured by the ratio between the height and weight and is determined as weight in kilograms divided by height in meters times its own value. The BMI taken is partitioned as follows:

Underweight:	BMI <18.5
Normal :	18.5>=BMI>=24.99
Overweight:	BMI>25
Obese:	BMI>30

4.2.2 Systolic and Diastolic Blood pressure

Systolic blood pressure is said to be the force exerted by the heart muscles when the when the heart contracts.This force is more and diastolic blood pressure is the force exerted admist heart-beats and is usually lesser and the heart muscles contract. Blood pressure is measures as follows:

Normal:	S<120	D<80
Pre-hypertension:	120=<S<139	80=<D<89
Hypertension Stage 1:	140=<S<159	90=<D<99
Hypertension Stage 2:	S>=160	D>=100
Hypertensive crisis:	S>=180	D>=110

4.2.3 Sex

Sex refers to the the gender of the person. This has been taken into account keeping in mind that there are a host of features that when accompanied with this combine to actually impact more than just taking this factor individually.

4.2.4 Smoker

As found in many different papers Dikalov et al. (2019), it has been found that smoking does have a high impact on hypertension and taking this factor into account thus is necessary.

4.2.5 Heart Failure

Heart failure is a big reason why we have taken this factor into account is the fact although this is may seem very related to blood pressure taken into account but it has been shown in many studies that heart failure is actually due to high blood pressure but there are many factors that actually initiate such a response. This is the reason we wanted to take this into account.

4.3 Benefits of the Proposed System

- The proposed system is faster than other neural network system.
- The proposed system removes back-propagation.
- The proposed system effectively reduces randomization for hypertension.
- The proposed system provides higher accuracy.
- No such work has been done in hypertension and hence our is best by far
- Newer methods to find are not being used and hence this percent is best.

CHAPTER 5

METHODOLOGY

The method and algorithms used in this research are described below:

5.1 Algorithms

The ELM algorithm for SLFN network was advanced by Huang et al. (2004) ELM is a kind of adjusting neural networks but with un-adjusted concealed stratum maps. The ELM algorithm used in our project is described below:
The ELM learning instructions are as follows:

1. Get the input feature.
2. Make the ANN using the following steps:
   - Give random weights and bias to all the nodes.
   - Evaluate the concealed yield strata.
   - Evaluate the yield weight using:
     $\hat{\beta} = (HTH)^{-1}H^{T}T$
3. Find the output for second step.
4. Use the classifier layer to classify yes or no for hypertension.
5. Print the output.

While in ELM, the centre and radius of the RBFs randomly calculated. The MELM-GRBF instructs that the centre should be initialized randomly by selecting patterns in the learning data set, For i_{th} concealed point, a arbitrary value k in (1,...,n), where n denotes the number of moulds in the learning sample is selected. After that, the value of center of the i_{th} hidden node is assigned for the value of the k_{th} pattern of the training set, i.e., $c_i \leftarrow x_k$, $k \in U(1, \ldots, n)$ and $U(1, \ldots, n)$ represents a one extent uniformly spread out among the discrete random variables, where n denotes the number of moulds in the learning sample.

When the focus have been selected, the magnitude for the width and the shape parameter of each is calculated. There are majorly 2 alternatives that are taken into account so as to ascertain the magnitude of the width parameter. The foremost is the one which takes the widths r,j which remains constant for all Gaussian function. The algorithm used in the MELM is:

1. Take random centre of the GRBF's from the training set's pattern.
2. Calculate the d_N values by $d_N = \sqrt{(\delta^2) \cdot k}$
3. Determine the d_F values by $d_F = \|c_i - c_j\|$
4. Determine the τ values by $\tau = \frac{\ln\left(\frac{\ln(\lambda)}{\ln(1-\lambda)}\right)}{\ln \frac{d_F}{d_N}}$
5. Determine the r values by $r = \frac{d_N}{(-\ln(1-\lambda))^{\frac{1}{\tau}}} = \frac{d_F}{(-\ln(\lambda))^{\frac{1}{\tau}}}$
6. Compute yield matrix **H** for the hidden layer.
7. Determine the output weights β by $\widehat{\beta} = H^{\dagger}T$ is the pseudo inverse of H.

The most important part in choosing the constant of the method is by selecting the estimation of "far" separation(d_F). The d_F is selected to be the length from the center of the concealed point whose τ is made in such a manner that it is near to the center ,i.e., for the i_{th} concealed node, the d_F is set as $d_F = \|c_i - c_j\|$ where **j** is the nearest hidden node to hidden node i. On investigating different parameters,it was observed that when the m lies in the tail, so the d_N evaluation would be similar in others also. Hence, it was seen that d_N parameter is set as $d_N = \sqrt{(\delta^2) \times k}$, where k is number of intakes and denotes the m as all residual distance in each dimension, which in our case is taken as 0.001.

5.2 Equations

ELM is built with randomly initialized hidden nodes. From learning's point of view, unlike the old learning algorithm, ELM tries to reach the smallest training error but also smallest norm of output weights.

$$Minimize : \|\beta\|_u^{\sigma_1} + \lambda\|H\beta + T\|_v^{\sigma} \tag{5.1}$$

Where, $\sigma_1 > 0, \sigma_2 > 0, \mathrm{u}, \mathrm{v} = 0, (1/2), 1, 2 \ldots, +\infty, \mathrm{x_i}$ is training data vector, H is hidden layer output matrix.

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_n] \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \ldots & h_L(x_1) \\ \vdots & \vdots & \vdots \\ h_1(x_n & \vdots & h_L(x_n) \end{bmatrix} \tag{5.2}$$

T is Training data target matrix, ti represent the target of each matrix.

$$T = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix} = \begin{bmatrix} t_{11} & \ldots & t_{1m} \\ \vdots & \vdots & \vdots \\ t_{N1} & \cdots & t_{Mn} \end{bmatrix} \tag{5.3}$$

ELM learning algorithm is defined as follows:

1. Arbitrarily allocate parameters to the concealed nodes.
2. Compute the concealed output matrix H.
3. Achieve the yield vector.

$$\beta = \left(\frac{1}{\lambda} + HH^T\right)^{-1} H^T \tag{5.4}$$

H^{-1} is the Moore-Penrose universal inverse of matrix H. Orthogonal projection method is a better method of calculating the inverse:

$$H^{-1} = |H|T\ H|^{-1} H^T| \tag{5.5}$$

In order to improve the stability of ELM, we can use:

$$\beta = H^+ T \tag{5.6}$$

$H^\top$is the Pseudo inverse of H. SLFN network methods with n concealed points can be shown as:

$$f_n(x) = \sum_{i=1}^{n} \beta_i g_i|x|, x \in R^d, \beta_i \epsilon R \tag{5.7}$$

Where g_i shows the concealed layer yield method and β_i is the output weight of the i_{th} hidden node, for additive point with activation method g, g_i is elaborated as:

$$g_i = g\left|a_i \cdot x + b_i\right|, a_i \in R^d, b_i \in R| \tag{5.8}$$

and for RBF nodes with operational function g, g_i is defined as:

$$g_i = g\left(\frac{x - a_i}{b_i}\right), a_i \in R^d, b_i \epsilon R^* \tag{5.9}$$

The intrinsic product $< u, v >$ is defined as:

$$\langle u, v\rangle = \int_X u(x)v(x)dx \tag{5.10}$$

The affinity between the network function and the objective function is gauged by:

$$\|f_L - f\| = \left(\int_X |f_n|\, x| - f|x||^2 dx\right)^2 \tag{5.11}$$

CHAPTER 6

CODING AND TESTING

In the algorithm used, namely the MELM-GRBF was tested with clinical data set taken from Apollo Hospitals. The given data set was totally un-tempered and was tried to be made as authentic as possible to mirror the happenings of the real world problems. Initially the first subsection, has the description of the data-set. Its also describes the experimental configuration. After this MELM-GRBF is contrasted with basic ELM model with the accuracy mentioned. The use of (lambda) along with the value of concealed units is then evaluated. After this, the values of (tau) is inspected to find as to what all understanding can be found to use of the model in this problem.

6.1 Coding

6.1.1 Classifier

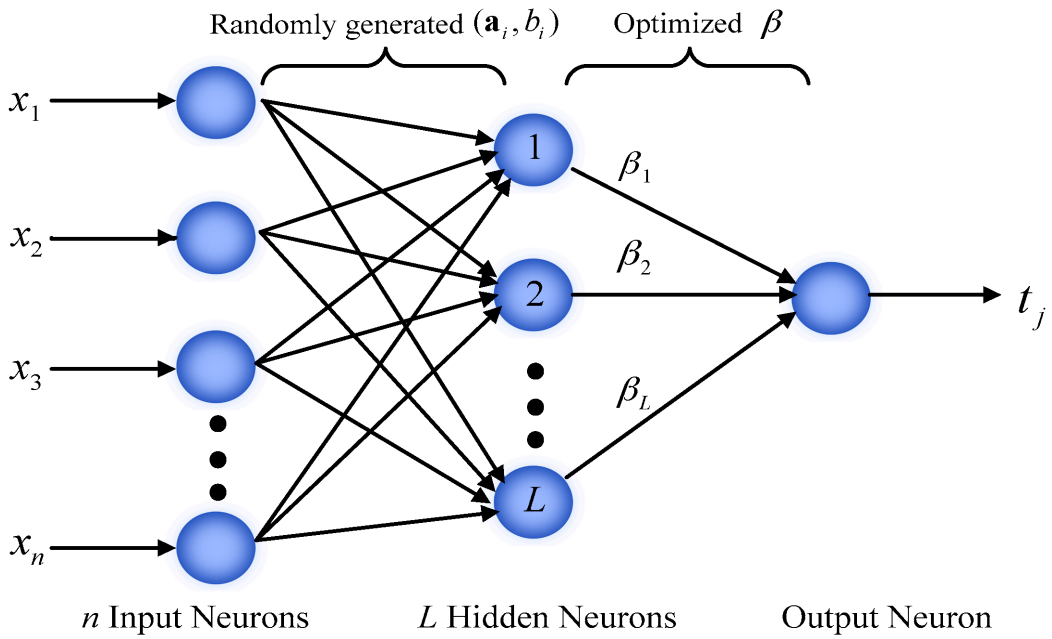


Figure 6.1: Classifiers of ELM

1. **Input Neurons:** These are the 7 input parameters that are being used to predict Hypertension. At a particular instant the node which need to be considered is randomly decided, which is implemented by using binarize function. **Refer Appendix A1**.
2. **Hidden Neurons:** These are the nodes that are present in the concealed layer of the model, for the purpose of scaling the model, maximum of 20 nodes can be used which are decided randomly.
3. **Output Neuron:** This is the final node of the model which actually predicts the value of Hypertension. This node is generated in accordance with input and hidden layer neurons.

By using the Input Neurons and activation function, create the input for hidden layer and by doing the same over all the hidden layer, we create the output neuron which is basically the output for the Hypertension Prediction.

6.1.2 Random Layer Generator

Random Layer works like a generator that maps the feature of the starting layer also called the input layer to the middle or hidden layer units with the components that are arbitrarily triggered.

The generated values are the function that are specified for the input activation that are combination of the dot product and distance.

$in\_ac = \alpha * mlp\_ac + (1 - \alpha) * rbf\_ac$

$mlp\_ac = \cdot(x, weight) + bias$

$rbf\_ac = rbf\_width * \|x - cen\|/radii$

α and the rbf_width should be given by the user.

Biases and weights are generally taken from Normal Distribution of Standard Deviation (SD) 1 and mean 0. **Refer Appendix A2**

6.1.3 Main

- First the data set is loaded.
- The sample of total data items are then split into 2 different parts namely the (i)training data and (ii)test data.
- The classification procedure is carried out.

- The scoring for ELM and M-ELM is calculated.

Refer appendix A3

6.2 Testing

6.2.1 Unit Testing

Unit testing the one wherein each part or module of the whole software is tested separately. This is a method of validation so as to see that each and every part is working well on its own.

6.2.2 Integration Testing

Integration testing is done when we want to see how the whole software works in tandem. It is done when all the parts of the software are joined and then tested. This is used to expose faults of communication between different parts of the software.

6.2.3 Validation Testing

Validation Testing is done to see whether the software has fulfilled the business requirements. It is carried out during and also after the whole process of development is carried.

CHAPTER 7

RESULTS AND DISCUSSION

The model's achievement was calibrated with respect to true positives that the model identified in Python.The more patients is categorized, the better. Victims of the study who did not show any such symptoms but are categorized into an hypertensive state are the ones who are said to be the false positive patients.Another set of sufferers who have or show the conditions of this phenomena and are incorrectly compartmentalized as non-hypertensive are the ones that fall into false negative group. Patients that are placed and were found as hypertensive form the true positive group. Those victims who are said or do not show any sign of hypertension are are flawlessly made into a single group are known as non-hypertensive group. This is the true-negative group.
The True Positive Rate (TPR) is which are also called the sensitivity and recall in machine learning, is defined as the ratio of the accurately classified samples to the total number of samples that were in the first place. Here, we have used the F1 score that combines the recall and the precision score. The experiments done were measured using 2 score. The accuracy and the F1 score. The accuracy of the ELM using the data we found was only about 85% which was increased to around 90% using MELM-GRBF.

7.1 Discussion

The typical performance mentioned in the papers of reference ranges from 80-90% when using the basic ANN network and we show that in the following table how our approach differs in this manner.

Ture et al. has previously progressed the network to achieve a sensitivity of greater than 90% and in contrast to previous work specificity of 66%. The accuracy was lifted to 82%. Samant Rao et al. had achieved 92.85% accuracy with the help of deep learning framework using n hidden layers.In his paper n=2 was used. But the studies shadowed used data-sets from single medical center while our data similar to those tries to mimic

Table 7.1: Comparison of the ANNs

Paper	Input Factors	Hidden Nodes	Output Nodes	Accuracy
LaFreniere et al.	11	8	2	82.3%
Ture et al.	9	Unknown	2	81.48%
Samant et al.	13	20(L-1) 5(L-2)	2	92.85%

the actual scenarios of the real life. LaFrenier et al. collated their data from various different places including different doctors who are involved in the process of ascertaining hypertension. The data thus formed was larger than what the other studies had used for their experiment.

The accuracy achieved by LaFreniere et al. in their paper is high when we take into account the patients they studied had varying factors in the influence of final consequence. The variability of the different medical record systems from which the data was summarised had higher patient records which suggests a greater amount of affirmation on results. It has been mentioned in the literature by LaFeniere at al. that smoking history, stress and family history of hypertension could lead to better results. We included the smoking history of the particular patient and have actually found it a determining factor in our study.

Here, we can see in the graph above, the accuracy increases with the increase in the

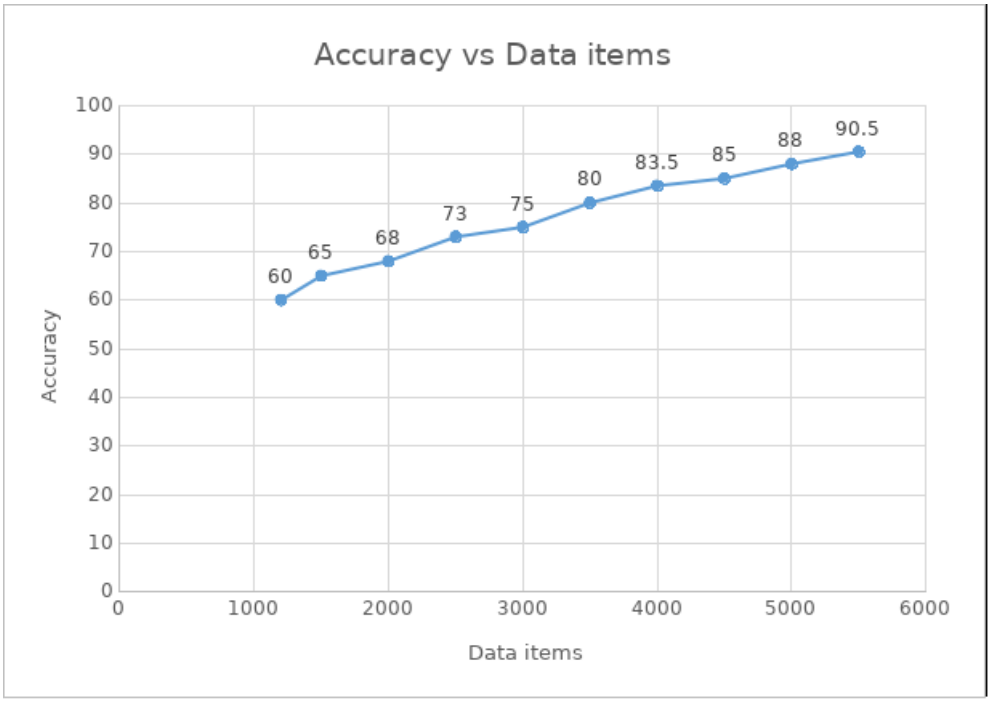


Figure 7.1: Accuracy at different no. of data

size of data . As we first proceeded with small amount of data we saw that the accuracy we achieved than what we had actually theorized. We tried to tweak the parameters but the changes were very less. WE then thought of increasing the amount of data that we had used and found that on increasing the amount of data used, we saw that the accuracy increased. On adding more data, we see that the accuracy reaches a plateau and hence increasing the amount of data beyond that point would not be useful. On observing the trend as shown in figure 1.1 we can see that even after increasing the amount of data, the accuracy does not increase and hence beyond a particular point the amount of data does not equate to percentage of accuracy.

As seen from the figure 1.2, we see that the centers influence the over-fitting and

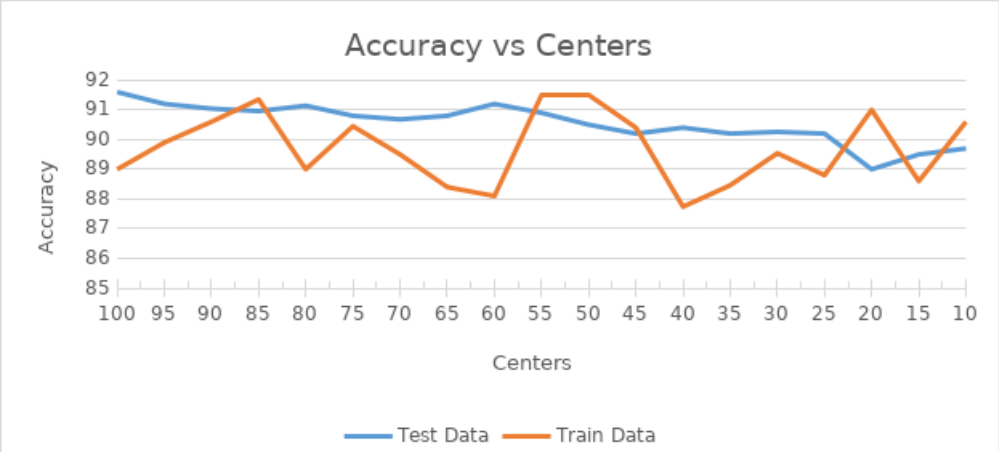


Figure 7.2: Accuracy vs Centers

under-fitting of data and hence we see that the on decreasing the centers, the amount of over fitting decreases but this has nominal effect on the accuracy. The accuracy is not influence a lot by the amount of change in centers and hence we see that there are certain number of centers such as 50,20 where we find that the data does not over-fit and the accuracy is also at its maximum. We can use this amount of centers for optimal values. Here as will be shown afterwards, we have used the 80/20 split for the train/test. This is the best amount of split we can have because after this the accuracy decreases drastically and will be show in figure 1.3.

The figure above shows us that the centers do not have a very large influence on the accuracy but can greatly influence the amount of under-fitting and over-fitting observed in the data.

Fig 1.3 shows the optimal, train/test split we chose for the data we had. As can be seen

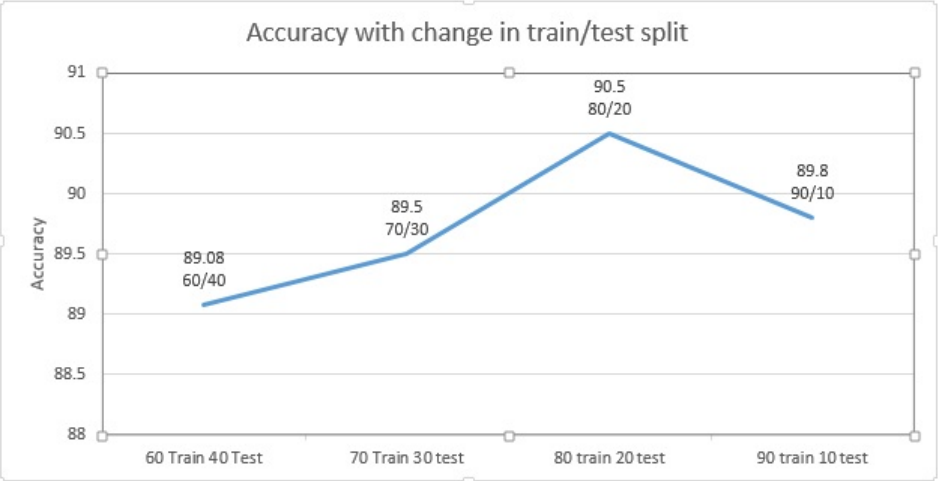


Figure 7.3: Accuracy with change in train/test split

from the data, the 60/40 split gives a very poor accuracy. As we increase the amount of train/test split, we find that the accuracy increases to 89.5 which although is better than 60/40 split does not work that well. We try another change in the split and find that the amount of accuracy increases to around 90%. But, we also observe that the efficiency decreases on further increasing the train/test split. We hypothesise that this maybe due to the problem of over fitting as it will decrease the efficiency and make the prediction bad.

The accuracy can be increased in any of the 2 ways. Firstly, it some key demographic details that are missing in the data we got could be added such as family history of hypertension, alcohol consumption. Second method of increasing the accuracy could also be by making the hyper parameters much more accurate and not a method of hit and trial as currently perceived. The model must be used in life like circumstances to establish the fact of its use as a whole.

CHAPTER 8

CONCLUSION

Hypertension has always been a very deadly problem which has no possible cure of its own. It is not a disease that can be cured by a drug but actually a continuous state of mind and it needs a good care of the self. In this report, we try to predict this problem by taking into account some of the most important and widely known reasons of hypertension.

The data-set collected was from a clinical facility which has been handling such problems from a long time. The method used here has given us the accuracy of 90%. We have used 5400 fields. As shown from the graph, higher amount of data can lead to much better accuracy.

REFERENCES

1. Chen, M.-L., Hu, J., McCoy, T. P., Letvak, S., and Ivanov, L. (2018). "Effect of a lifestyle-based intervention on health-related quality of life in older adults with hypertension." *Journal of aging research*, 2018.

2. Choi, J. R., Koh, S. B., and Choi, E. (2018). "Waist-to-height ratio index for predicting incidences of hypertension: the arirang study." *BMC public health*, 18(1), 767.

3. Dikalov, S., Itani, H., Richmond, B., Arslanbaeva, L., Vergeade, A., Rahman, S. J., Boutaud, O., Blackwell, T., Massion, P. P., Harrison, D. G., et al. (2019). "Tobacco smoking induces cardiovascular mitochondrial oxidative stress, promotes endothelial dysfunction, and enhances hypertension." *American Journal of Physiology-Heart and Circulatory Physiology*, 316(3), H639–H646.

4. Echouffo-Tcheugui, J. B., Batty, G. D., Kivimäki, M., and Kengne, A. P. (2013). "Risk models to predict hypertension: a systematic review." *PloS one*, 8(7), e67370.

5. Fernández-Navarro, F., Hervás-Martínez, C., Sanchez-Monedero, J., and Gutiérrez, P. A. (2011). "Melm-grbf: a modified version of the extreme learning machine for generalized radial basis function neural networks." *Neurocomputing*, 74(16), 2502–2510.

6. Huang, G.-B. and Chen, L. (2008). "Enhanced random search based incremental extreme learning machine." *Neurocomputing*, 71(16-18), 3460–3468.

7. Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2006). "Extreme learning machine: theory and applications." *Neurocomputing*, 70(1-3), 489–501.

8. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K., et al. (2004). "Extreme learning machine: a new learning scheme of feedforward neural networks." *Neural networks*, 2, 985–990.

9. Koivistoinen, T., Lyytikäinen, L.-P., Aatola, H., Luukkaala, T., Juonala, M., Viikari, J., Lehtimäki, T., Raitakari, O. T., Kähönen, M., and Hutri-Kähönen, N. (2018). "Pulse wave velocity predicts the progression of blood pressure and development of hypertension in young adults." *Hypertension*, 71(3), 451–456.

10. LaFreniere, D., Zulkernine, F., Barber, D., and Martin, K. (2016). "Using machine learning to predict hypertension from a clinical dataset." *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 1–7.

11. Lan, Y., Soh, Y. C., and Huang, G.-B. (2009). "Ensemble of online sequential extreme learning machine." *Neurocomputing*, 72(13-15), 3391–3395.

12. Manrique, D., Ríos, J., and Rodríguez-Patón, A. (2006). "Evolutionary system for automatically constructing and adapting radial basis function networks." *Neurocomputing*, 69(16-18), 2268–2283.

13. Sakr, S., Elshawi, R., Ahmed, A., Qureshi, W. T., Brawner, C., Keteyian, S., Blaha, M. J., and Al-Mallah, M. H. (2018). "Using machine learning on cardiorespiratory fitness data for predicting hypertension: The henry ford exercise testing (fit) project." *PloS one*, 13(4), e0195344.

14. Sandi, G., Supangkat, S. H., and Slamet, C. (2016). "Health risk prediction for treatment of hypertension." *2016 4th International Conference on Cyber and IT Service Management*, IEEE, 1–6.

15. Srivastava, P., Srivastava, A., Burande, A., and Khandelwal, A. (2013). "A note on hypertension classification scheme and soft computing decision making system." *ISRN Biomathematics*, 2013.

16. Ture, M., Kurt, I., Kurum, A. T., and Ozdamar, K. (2005). "Comparing classification techniques for predicting essential hypertension." *Expert Systems with Applications*, 29(3), 583–588.

17. Wadehn, F., Walser, D., Bohdanowicz, M., Czosnyka, M., and Heidt, T. (2017). "Non-invasive detection of intracranial hypertension using random forests." *2017 Computing in Cardiology (CinC)*, IEEE, 1–4.

APPENDIX A

CODE

A.1 Classifier

```
from abc import ABCMeta, abstractmethod

import numpy as np
from scipy.linalg import pinv2

from sklearn.utils import as_float_array
from sklearn.utils.extmath import safe_sparse_dot
from sklearn.base import BaseEstimator, ClassifierMixin, RegressorMixin
from sklearn.preprocessing import LabelBinarizer

from random_layer import RandomLayer, MLPRandomLayer

__all__ = ["ELMRegressor",
           "ELMClassifier",
           "GenELMRegressor",
           "GenELMClassifier"]

class BaseELM(BaseEstimator):
    __metaclass__ = ABCMeta

    def __init__(self, hidden_layer, regressor):
        self.regressor = regressor
        self.hidden_layer = hidden_layer
```

```
    @abstractmethod
    def fit(self, X, y):

    @abstractmethod
    def predict(self, X):


class GenELMRegressor(BaseELM, RegressorMixin):
        def __init__(self,
                     hidden_layer=MLPRandomLayer(random_state=0),
                     regressor=None):

            super(GenELMRegressor, self).__init__(hidden_layer, regressor)

            self.coefs_ = None
            self.fitted_ = False
            self.hidden_activations_ = None

    def _fit_regression(self, y):
            if (self.regressor is None):
                self.coefs_ = safe_sparse_dot(pinv2(self.hidden_activations_), y)
            else:
                self.regressor.fit(self.hidden_activations_, y)

            self.fitted_ = True

    def fit(self, X, y):
            # fit random hidden layer and compute the hidden layer
                activations
            self.hidden_activations_ = self.hidden_layer.fit_transform(X)

            # solve the regression from hidden activations to outputs
```

```
            self . _fit_regression ( as_float_array (y, copy=True))

            return self

        def _get_predictions ( self ):
            """get predictions using internal least squares/supplied regressor """
            if ( self . regressor is None):
                preds = safe_sparse_dot ( self . hidden_activations_ , self . coefs_)
            else :
                preds = self . regressor . predict ( self . hidden_activations_ )

            return preds

        def predict ( self , X):
                    if (not self . fitted_ ):
                raise ValueError ("ELMRegressor not fitted ")

            # compute hidden layer activations
            self . hidden_activations_ = self . hidden_layer . transform (X)

            # compute output predictions for new hidden activations
            predictions = self . _get_predictions ()

            return predictions


class GenELMClassifier(BaseELM, ClassifierMixin):
            def __init__ ( self ,
                        hidden_layer=MLPRandomLayer(random_state=0),
                        binarizer =LabelBinarizer (0, 1),
                        regressor =None):
```

```
        super(GenELMClassifier, self).__init__(hidden_layer, regressor)

        self.binarizer = binarizer

        self.classes_ = None
        self.genelm_regressor_ = GenELMRegressor(hidden_layer, regressor)

    def decision_function(self, X):
            return self.genelm_regressor_.predict(X)

    def fit(self, X, y):
            self.classes_ = np.unique(y)

        y_bin = self.binarizer.fit_transform(y)

        self.genelm_regressor_.fit(X, y_bin)
        return self

    def predict(self, X):
            raw_predictions = self.decision_function(X)
        class_predictions = self.binarizer.inverse_transform(raw_predictions)

        return class_predictions


# ELMRegressor with default RandomLayer
class ELMRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, n_hidden=20, alpha=0.5, rbf_width=1.0,
                 activation_func='tanh', activation_args=None,
                 user_components=None, regressor=None, random_state=None):

        self.n_hidden = n_hidden
```

```
        self.alpha = alpha
        self.random_state = random_state
        self.activation_func = activation_func
        self.activation_args = activation_args
        self.user_components = user_components
        self.rbf_width = rbf_width
        self.regressor = regressor

        self._genelm_regressor = None

    def _create_random_layer(self):
        """Pass init params to RandomLayer"""

        return RandomLayer(n_hidden=self.n_hidden,
                           alpha=self.alpha, random_state=self.random_state,
                           activation_func=self.activation_func,
                           activation_args=self.activation_args,
                           user_components=self.user_components,
                           rbf_width=self.rbf_width)

    def fit(self, X, y):
        rhl = self._create_random_layer()
        self._genelm_regressor = GenELMRegressor(hidden_layer=rhl,
                                                 regressor=self.regressor)
        self._genelm_regressor.fit(X, y)
        return self

    def predict(self, X):
        if (self._genelm_regressor is None):
            raise ValueError("SimpleELMRegressor not fitted")

        return self._genelm_regressor.predict(X)
```

```
class ELMClassifier(ELMRegressor):

    def __init__(self, n_hidden=20, alpha=0.5, rbf_width=1.0,
                 activation_func='sigmoid', activation_args=None,
                 user_components=None, regressor=None,
                 binarizer=LabelBinarizer(0, 1),
                 random_state=None):

        super(ELMClassifier, self).__init__(n_hidden=n_hidden,
                                            alpha=alpha,
                                            random_state=random_state,
                                            activation_func=activation_func,
                                            activation_args=activation_args,
                                            user_components=user_components,
                                            rbf_width=rbf_width,
                                            regressor=regressor)

        self.classes_ = None
        self.binarizer = binarizer

    def decision_function(self, X):
        return super(ELMClassifier, self).predict(X)

    def fit(self, X, y):
        self.classes_ = np.unique(y)

        y_bin = self.binarizer.fit_transform(y)

        super(ELMClassifier, self).fit(X, y_bin)
```

```
        return self

    def predict(self, X):
        raw_predictions = self.decision_function(X)
        class_predictions = self.binarizer.inverse_transform(raw_predictions)

        return class_predictions

    def score(self, X, y):
        from sklearn.metrics import accuracy_score
        return accuracy_score(y, self.predict(X))
```

A.2 Layer Generator

```
from abc import ABCMeta, abstractmethod

from math import sqrt

import numpy as np
import scipy.sparse as sp
from scipy.spatial.distance import cdist, pdist, squareform

from sklearn.metrics import pairwise_distances
from sklearn.utils import check_random_state, check_array
from sklearn.utils.extmath import safe_sparse_dot
from sklearn.base import BaseEstimator, TransformerMixin

__all__ = ['RandomLayer',
           'MLPRandomLayer',
           'RBFRandomLayer',
           'GRBFRandomLayer',
```

```
            ]


class  BaseRandomLayer(BaseEstimator, TransformerMixin):
    """ Abstract  Base Class  for  random   layers """
    __metaclass__  = ABCMeta

     _internal_activation_funcs   =  dict ()

    @classmethod
    def  activation_func_names ( cls ) :
        """Get  list   of   internal   activation   function  names"""
        return   cls .  _internal_activation_funcs  .keys ()

    # take  n_hidden and random_state ,   init   components_ and
    #   input_activations_
    def  __init__ ( self ,  n_hidden=20, random_state=0,   activation_func =None,
                    activation_args =None):

        self .n_hidden = n_hidden
        self .random_state  = random_state
        self .  activation_func   =   activation_func
        self .  activation_args   =   activation_args

        self .components_ = dict ()
        self .  input_activations_   = None

       # keyword args  for   internally   defined  funcs
        self . _extra_args  =  dict ()

    @abstractmethod
    def  _generate_components( self ,  X):
```

```
        """Generate components of hidden layer given X"""

    @abstractmethod
    def _compute_input_activations(self, X):
        """Compute input activations given X"""

    # compute input activations and pass them
    # through the hidden layer transfer functions
    # to compute the transform
    def _compute_hidden_activations(self, X):
        """Compute hidden activations given X"""

        self._compute_input_activations(X)

        acts = self.input_activations_

        if (callable(self.activation_func)):
            args_dict = self.activation_args if (self.activation_args) else
                {}
            X_new = self.activation_func(acts, **args_dict)
        else:
            func_name = self.activation_func
            func = self._internal_activation_funcs[func_name]

            X_new = func(acts, **self._extra_args)

        return X_new

    # perform fit by generating random components based
    # on the input array
    def fit(self, X, y=None):
        X = check_array(X)
```

```
            self._generate_components(X)

            return self

        # perform transformation by calling compute_hidden_activations
        # (which will normally call compute_input_activations first)
        def transform(self, X, y=None):
            X = check_array(X)

            if (self.components_ is None):
                raise ValueError('No components initialized')

            return self._compute_hidden_activations(X)


class RandomLayer(BaseRandomLayer):
    # triangular activation function
    _tribas = (lambda x: np.clip(1.0 - np.fabs(x), 0.0, 1.0))

    # inverse triangular activation function
    _inv_tribas = (lambda x: np.clip(np.fabs(x), 0.0, 1.0))

    # sigmoid activation function
    _sigmoid = (lambda x: 1.0 / (1.0 + np.exp(-x)))

    # hard limit activation function
    _hardlim = (lambda x: np.array(x > 0.0, dtype=float))

    _softlim = (lambda x: np.clip(x, 0.0, 1.0))

    # gaussian RBF
```

```
_gaussian = (lambda x: np.exp(-pow(x, 2.0)))

# multiquadric RBF
_multiquadric = (lambda x:
                 np.sqrt(1.0 + pow(x, 2.0)))

# inverse multiquadric RBF
_inv_multiquadric = (lambda x:
                     1.0 / (np.sqrt(1.0 + pow(x, 2.0))))

# internal activation function table
_internal_activation_funcs = {'sine': np.sin,
                              'tanh': np.tanh,
                              'tribas': _tribas,
                              'inv_tribas': _inv_tribas,
                              'sigmoid': _sigmoid,
                              'softlim': _softlim,
                              'hardlim': _hardlim,
                              'gaussian': _gaussian,
                              'multiquadric': _multiquadric,
                              'inv_multiquadric': _inv_multiquadric,
                              }

def __init__(self, n_hidden=20, alpha=0.5, random_state=None,
             activation_func='tanh', activation_args=None,
             user_components=None, rbf_width=1.0):

    super(RandomLayer, self).__init__(n_hidden=n_hidden,
                                      random_state=random_state,
                                      activation_func=activation_func,
                                      activation_args=activation_args)
```

```
        if (isinstance(self.activation_func, str)):
            func_names = self._internal_activation_funcs.keys()
            if (self.activation_func not in func_names):
                msg = "unknown activation function '%s'" % self.
                    activation_func
                raise ValueError(msg)

        self.alpha = alpha
        self.rbf_width = rbf_width
        self.user_components = user_components

        self._use_mlp_input = (self.alpha != 0.0)
        self._use_rbf_input = (self.alpha != 1.0)

    def _get_user_components(self, key):
        """Look for given user component"""
        try:
            return self.user_components[key]
        except (TypeError, KeyError):
            return None

    def _compute_radii(self):
        """Generate RBF radii"""

        # use supplied radii if present
        radii = self._get_user_components('radii')

        # compute radii
        if (radii is None):
            centers = self.components_['centers']

            n_centers = centers.shape[0]
```

```
            max_dist = np.max(pairwise_distances(centers))
            radii = np.ones(n_centers) * max_dist / sqrt(2.0 * n_centers)

        self.components_['radii'] = radii

    def _compute_centers(self, X, sparse, rs):
        """Generate RBF centers"""

        # use supplied centers if present
        centers = self._get_user_components('centers')

        # use points taken uniformly from the bounding
        # hyperrectangle
        if (centers is None):
            n_features = X.shape[1]

            if (sparse):
                fxr = xrange(n_features)
                cols = [X.getcol(i) for i in fxr]

                min_dtype = X.dtype.type(1.0e10)
                sp_min = lambda col: np.minimum(min_dtype, np.min(col.data))
                min_Xs = np.array(map(sp_min, cols))

                max_dtype = X.dtype.type(-1.0e10)
                sp_max = lambda col: np.maximum(max_dtype, np.max(col.data))
                max_Xs = np.array(map(sp_max, cols))
            else:
                min_Xs = X.min(axis=0)
                max_Xs = X.max(axis=0)

            spans = max_Xs - min_Xs
```

```
            ctrs_size = (self.n_hidden, n_features)
            centers = min_Xs + spans * rs.uniform(0.0, 1.0, ctrs_size)

        self.components_['centers'] = centers

    def _compute_biases(self, rs):
        """Generate MLP biases"""

        # use supplied biases if present
        biases = self._get_user_components('biases')
        if (biases is None):
            b_size = self.n_hidden
            biases = rs.normal(size=b_size)

        self.components_['biases'] = biases

    def _compute_weights(self, X, rs):
        """Generate MLP weights"""

        # use supplied weights if present
        weights = self._get_user_components('weights')
        if (weights is None):
            n_features = X.shape[1]
            hw_size = (n_features, self.n_hidden)
            weights = rs.normal(size=hw_size)

        self.components_['weights'] = weights

    def _generate_components(self, X):
        """Generate components of hidden layer given X"""

        rs = check_random_state(self.random_state)
```

```
        if (self._use_mlp_input):
            self._compute_biases(rs)
            self._compute_weights(X, rs)

        if (self._use_rbf_input):
            self._compute_centers(X, sp.issparse(X), rs)
            self._compute_radii()

    def _compute_input_activations(self, X):
        """Compute input activations given X"""

        n_samples = X.shape[0]

        mlp_acts = np.zeros((n_samples, self.n_hidden))
        if (self._use_mlp_input):
            b = self.components_['biases']
            w = self.components_['weights']
            mlp_acts = self.alpha * (safe_sparse_dot(X, w) + b)

        rbf_acts = np.zeros((n_samples, self.n_hidden))
        if (self._use_rbf_input):
            radii = self.components_['radii']
            centers = self.components_['centers']
            scale = self.rbf_width * (1.0 - self.alpha)
            rbf_acts = scale * cdist(X, centers) / radii

        self.input_activations_ = mlp_acts + rbf_acts


class MLPRandomLayer(RandomLayer):
    """Wrapper for RandomLayer with alpha (mixing coefficient) set
       to 1.0 for MLP activations only"""
```

```
    def __init__(self, n_hidden=20, random_state=None,
                 activation_func='tanh', activation_args=None,
                 weights=None, biases=None):
        user_components = {'weights': weights, 'biases': biases}
        super(MLPRandomLayer, self).__init__(n_hidden=n_hidden,
                                             random_state=random_state,
                                             activation_func=activation_func,
                                             activation_args=activation_args,
                                             user_components=
                                                 user_components,
                                             alpha=1.0)


class RBFRandomLayer(RandomLayer):
    """Wrapper for RandomLayer with alpha (mixing coefficient) set
       to 0.0 for RBF activations only"""

    def __init__(self, n_hidden=20, random_state=None,
                 activation_func='sigmoid', activation_args=None,
                 centers=None, radii=None, rbf_width=1.0):
        user_components = {'centers': centers, 'radii': radii}
        super(RBFRandomLayer, self).__init__(n_hidden=n_hidden,
                                             random_state=random_state,
                                             activation_func=activation_func,
                                             activation_args=activation_args,
                                             user_components=
                                                 user_components,
                                             rbf_width=rbf_width,
                                             alpha=0.0)
```

```
class GRBFRandomLayer(RBFRandomLayer):
    # def _grbf(acts, taus):
    #     """GRBF activation function"""

    #     return np.exp(np.exp(-pow(acts, taus)))

    _grbf = (lambda acts, taus: np.exp(np.exp(-pow(acts, taus))))

    _internal_activation_funcs = {'grbf': _grbf}

    def __init__(self, n_hidden=200, grbf_lambda=0.01,
                 centers=None, radii=None, random_state=None):
        super(GRBFRandomLayer, self).__init__(n_hidden=n_hidden,
                                              activation_func='grbf',
                                              centers=centers, radii=radii,
                                              random_state=random_state)

        self.grbf_lambda = grbf_lambda
        self.dN_vals = None
        self.dF_vals = None
        self.tau_vals = None

    # get centers from superclass, then calculate tau_vals
    # according to ref [1]
    def _compute_centers(self, X, sparse, rs):
        """Generate centers, then compute tau, dF and dN vals"""

        super(GRBFRandomLayer, self)._compute_centers(X, sparse, rs)

        centers = self.components_['centers']
        sorted_distances = np.sort(squareform(pdist(centers)))
        self.dF_vals = sorted_distances[:, -1]
```

```
            self.dN_vals = sorted_distances[:, 1] / 100.0
            # self.dN_vals = 0.0002 * np.ones(self.dF_vals.shape)

            tauNum = np.log(np.log(self.grbf_lambda) /
                            np.log(1.0 - self.grbf_lambda))

            tauDenom = np.log(self.dF_vals / self.dN_vals)

            self.tau_vals = tauNum / tauDenom

            self._extra_args['taus'] = self.tau_vals

        # get radii according to ref [1]
        def _compute_radii(self):
            """Generate radii """

            denom = pow(-np.log(self.grbf_lambda), 1.0 / self.tau_vals)
            self.components_['radii'] = self.dF_vals / denom
```

A.3 Main Class

```
from time import time
from matplotlib.pyplot import *
from sklearn.model_selection import train_test_split
from sklearn.cluster import k_means
from sklearn.metrics import f1_score, precision_recall_fscore_support as score
    , accuracy_score
import numpy as np
import pandas as pd
from elmclassify import ELMClassifier, ELMRegressor, GenELMClassifier,
    GenELMRegressor
from random_layer import RandomLayer, MLPRandomLayer, RBFRandomLayer,
```

```
    GRBFRandomLayer
from math import sqrt
import csv
from sklearn.preprocessing import StandardScaler
#from sklearn.datasets import load_iris, load_digits, load_diabetes,
    make_regression

# <codecell>
"""
def make_toy():
    x = np.arange(0.25, 20, 0.1)
    y = x * np.cos(x) + 0.5 * sqrt(x) * np.random.randn(x.shape[0])
    x = x.reshape(-1, 1)
    y = y.reshape(-1, 1)
    return x, y


# <codecell>
"""
pdf = pd.read_csv('hype.csv')
#print(pdf.head())
#x = np.arange(0.25, 20, 0.1)
x = pdf[[' sex ',' body weight ',' height ',' smoker',' systolic blood preassure ','
    diastolic blood preassure ',' max systolic blood preassure ',' heart failure
    ']]
y = pdf[[' HYPERTENSION(1,0)']]
#print(x.head())
#print(y.head())


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =0.15)
```

```
# RBFRandomLayer tests
for af in RandomLayer.activation_func_names():
    print
    af,
    elmc = ELMClassifier( activation_func =af)


# <codecell>

elmc.classes_

# <codecell>

for af in RandomLayer.activation_func_names():
    print
    af
    elmc = ELMClassifier( activation_func =af, random_state=0)


# <codecell>

elmc = ELMClassifier(n_hidden=500, activation_func ='multiquadric')


# <codecell>

elmr = ELMRegressor(random_state=0, activation_func='gaussian', alpha=0.0)
elmr.fit(x, y)
print
elmr.score(x, y), elmr.score(x, y)
plot(x, y, x, elmr.predict(x))
```

```
#rhl = RandomLayer(n_hidden=1000, alpha=1.0)
rhl = RBFRandomLayer(n_hidden=500, rbf_width=0.01)
elmr = GenELMClassifier(hidden_layer=rhl)
elmr.fit(x_train, y_train)
predicted = elmr.predict(x_test)
precision, recall, fscore, support = score(y_test, predicted)

print("RBF Random")
print('precision: {}'.format(precision))
#print('recall: {}'.format(recall))
print('fscore: {}'.format(fscore))
#print('support: {}'.format(support))
#print("RBF different rbf_width",elmr.score(y_test,elmr.predict(x_test)))



# <codecell>

nh = 700
(ctrs, _, _) = k_means(x_train, nh)
unit_rs = np.ones(nh)

#rhl = RBFRandomLayer(n_hidden=nh, activation_func='inv_multiquadric')
#rhl = RBFRandomLayer(n_hidden=nh, activation_func='gaussian')
#rhl = GRBFRandomLayer(n_hidden=nh, centers=ctrs, radii=unit_rs)
rhl = GRBFRandomLayer(n_hidden=nh, grbf_lambda=.01, centers=ctrs)
elmr = GenELMClassifier(hidden_layer=rhl)
elmr.fit(x_train, y_train)
Y_pred = elmr.predict(x_test)
print("GRBF",elmr.score(x_train, y_train), elmr.score(x_test, y_test))
print("F1 GRBF", f1_score(y_test, Y_pred, average="macro"))
#print("accuracy GRBF", accuracy_score(y_test, Y_pred))
```

```
#plot(x, y, x, elmr.predict(xtoy))

# <codecell>

rbf_rhl = RBFRandomLayer(n_hidden=700, random_state=5, rbf_width=0.01)
elmc_rbf = GenELMClassifier(hidden_layer=rbf_rhl)
elmc_rbf.fit(x_train, y_train)
y_pred_rbf = elmr.predict(x_test)

print("F1 RBF rbf", f1_score(y_test, y_pred_rbf, average="macro"))
#print("accuracy RBF", accuracy_score(y_test, y_pred_rbf))
print('RBF GenELM Classifier',elmc_rbf.score(x_train, y_train), elmc_rbf.
    score(x_test, y_test))


def powtanh_xfer(activations, power=1.0):
    return pow(np.tanh(activations), power)


tanh_rhl = MLPRandomLayer(n_hidden=500, activation_func=powtanh_xfer,
    activation_args={'power': 3.0})
elmc_tanh = GenELMClassifier(hidden_layer=tanh_rhl)
elmc_tanh.fit(x_train, y_train)
print("tanh score",elmc_tanh.score(x_train, y_train), elmc_tanh.score(x_test,
    y_test))
```

APPENDIX B

PAPER SUBMISSION

Two Papers have been written for this project. It has been submitted to IET(Institution of Engineering and Technology)-Healthcare Technology Letters and ICICT (International Conference on Inventive Computation Technologies)-2018

4/30/2019 Gmail - Thank you for your manuscript submission - Healthcare Technology Letters

Gmail

Abhay Goyal <goyalabhay97@gmail.com>

Thank you for your manuscript submission - Healthcare Technology Letters

RVT -Review Management System <editorialoffice_1@iet-review.rivervalleytechnologies.com>
Reply-To: htl@theiet.org
To: goyalabhay97@gmail.com

Fri, Apr 12, 2019 at 10:04 PM

Dear Mr Abhay Goyal,

The paper entitled

Using Extreme Learning Machine to Predict Hypertension from a Clinical Dataset

has been submitted by Abhay Goyal and will be given full consideration for publication in Healthcare Technology Letters.

The manuscript details are:

Manuscript ID: HTL-2019-0020

Title:

Using Extreme Learning Machine to Predict Hypertension from a Clinical Dataset

Authors: Abhay Goyal, Durgesh Singh, Sharanya Se

Please quote the manuscript ID in all correspondence with this office.

Healthcare Technology Letters is a fully gold open access journal. This means that you will retain the copyright and the published papers are freely available to everybody that wishes to read and use them. If your paper is accepted you will be asked to select from four Creative Commons Licence types, to find out more about these licence types please see our Author Guide: http://digital-library.theiet.org/journals/oa-author-guide#copyright

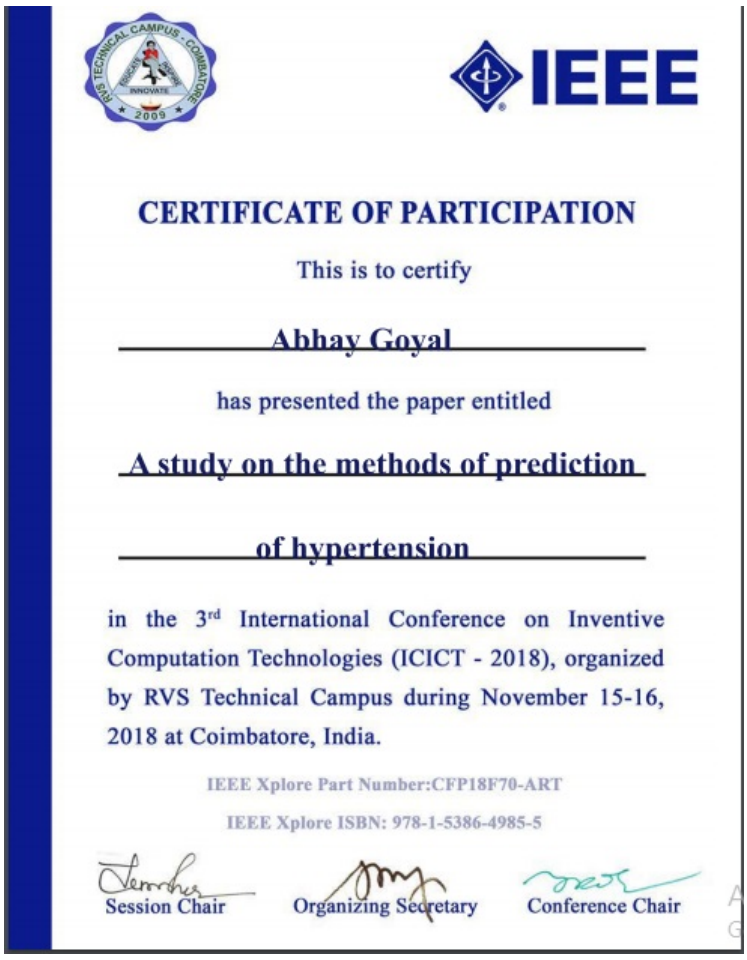
You can view the status of your paper at any time by logging in to https://www.iet-review.rivervalleytechnologies.com/ and going to My Submissions or Co-author Submissions. Please also take the time to make sure your contact details are up-to-date by clicking your name on the top right of the page.

Thank you for your interest in Healthcare Technology Letters.

Jessica Connelly
Healthcare Technology Letters

Disclaimer: This email and any files transmitted with it are confidential and intended solely for the use of the individual or entity to whom they are addressed.This message contains confidential information and is intended only for the individual named. If you are not the named addressee you should not disseminate, distribute or copy this e-mail. Please notify the sender immediately by e-mail if you have received this e-mail by mistake and delete this e-mail from your system. If you are not the intended recipient you are notified that disclosing, copying, distributing or taking any action in reliance on the contents of this information is strictly prohibited.

Figure B.1: IET-Healthcare Technologies Letters Submission



CERTIFICATE OF PARTICIPATION

This is to certify

Abhay Goyal

has presented the paper entitled

A study on the methods of prediction of hypertension

in the 3rd International Conference on Inventive Computation Technologies (ICICT - 2018), organized by RVS Technical Campus during November 15-16, 2018 at Coimbatore, India.

IEEE Xplore Part Number:CFP18F70-ART

IEEE Xplore ISBN: 978-1-5386-4985-5

Session Chair Organizing Secretary Conference Chair

Figure B.2: ICICT Acceptance

CERTIFICATE OF PARTICIPATION

This is to certify

Durgesh Singh

has presented the paper entitled

A study on the methods of prediction of hypertension

in the 3rd International Conference on Inventive Computation Technologies (ICICT - 2018), organized by RVS Technical Campus during November 15-16, 2018 at Coimbatore, India.

IEEE Xplore Part Number:CFP18F70-ART

IEEE Xplore ISBN: 978-1-5386-4985-5

Session Chair Organizing Secretary Conference Chair

Figure B.3: ICICT Acceptance