THYROID DISEASE CLASSIFIER USING TWO-TIER ENSEMBLE METHODS

Mini Project submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

221710310021 - G. Sai Naga Rahul

221710310026 - K. Anirudha Raghava Sarma

221710310002 – Alluri Nikhil Yadav

221710310059 – Rohith Reddy Ganga

Under the esteemed guidance of

Mrs. G. Hima Bindu

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

HYDERABAD

DECEMBER 2020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

DECLARATION

We hereby declare that the mini project entitled "Thyroid Disease Classifier Using Two-Tier Ensemble Methods" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

221710310021 - G. Sai Naga Rahul

221710310026 - K. Anirudha Raghava Sarma

221710310002 – Alluri Nikhil Yadav

221710310059 – Rohith Reddy Ganga

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that Mini Project entitled "Thyroid Disease Classifier Using Two-Tier Ensemble Methods" is submitted by G Sai Naga Rahul (221710310021), K Anirudha Raghava Sarma (221710310026), Alluri Nikhil Yadav (221710310002), Rohith Reddy Ganga (221710310059) in partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering. The Mini Project has been approved as it satisfies the academic requirements.

Mrs. G. Hima Bindu
Assistant Professor
Department of CSE

ACKNOWLEDGMENT

Our Mini Project would not have been successful without the help of several people. we would like to thank the personalities who were part of our Mini Project in numerous ways, those who gave us outstanding support from the birth of the Mini Project.

We are extremely thankful to our honourable Pro-Vice Chancellor, **Prof. N. Siva Prasad** for providing necessary infrastructure and resources for the accomplishment of our Mini Project.

We are highly indebted to **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the Mini Project.

We are very much obliged to our beloved **Prof. S. Phani Kumar**, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this mini project and encouragement in completion of this prjoect.

We hereby wish to express our deep sense of gratitude to **Dr. S Aparna**, Assistant Professor, Department of Computer Science and Engineering, School of Technology and to **Mrs. G. Hima Bindu**, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the mini project.

We are also thankful to all the staff members of Computer Science and Engineering department who have cooperated in making our mini project a success. We would like to thank all our parents and friends who extended their help, encouragement and moral support either directly or indirectly in our project work.

Sincerely,

G. Sai Naga Rahul (221710310021)

K. Anirudha Raghava Sarma

(221710310026)

Alluri Nikhil Yadav (221710310002)

Rohith Reddy Ganga (221710310059)

CONTENTS

ABSTRACT

Nowadays hormone disturbance is an essential issue in humans. The fundamental issue that troubles behind thyroid diseases is hormonal disturbance. Thyroid disease is the severe endocrine disease that adversely affects the wellbeing of the human. In such case, disease prediction model can play an important role in disease prognosis.

The work deploys wrapper method for feature selection. We have identified top machine learning algorithms for thyroid prediction and these five classifiers are consolidated in the gathering of three with heterogeneous permutations to further improve the performance of deployed classifiers. Majority voting scheme is deployed on the ensemble methods for getting results. The proposed model utilizes 70:30 proportions of training and test set. The outcome uncovered that the accuracy of proposed 2-tier ensemble model is increased as compared with single machine learning classifiers.

LIST OF FIGURES

1. INTRODUCTION

In today's world, the thyroid disease is widely spread and it causes harsh damage to the human body and life. It damages the normal functioning of thyroid gland. Presently in excess of 25,000 emergency clinics over the globe gather information on patients in different configuration. In the conventional technique, the clinical and medical studies are accomplished utilizing traditional examination and measurable tests. The avoidance of this sickness requires fundamental information on the event of this illness as it is very complicated to fix this malady once it arrives at its last stages.

The advancement of computational biology is used in the healthcare industry. It allowed collecting the stored patient data for the medical disease prediction. There are different intelligent prediction algorithms are available for the diagnosis of the disease at early stages. The Medical information system is rich of data sets, but there are no intelligent systems that can easily analysis the disease. Over the course of time, machine learning algorithms play a crucial role in solving the complex and nonlinear problems in developing a prediction model. In any disease prediction models are required to paramount the features that can be selected from the different datasets which can easily be used as a classification in healthy patient as precisely as possible. Otherwise, misclassification may result in a healthy patient that endures unnecessary treatment. Hence, the factuality of predicting any disease in conjunction with the thyroid disease is of supreme cardinality. The thyroid gland is an endocrine gland in the neck.

It erects in the lessened part of the human neck, beneath the Adam's apple which aids in the secretion of thyroid hormones and that basically influences the rate of metabolism and protein synthesis. To control the metabolism in the body, thyroid hormones are useful in many ways, counting how briskly the heart beats and how quickly the calories are burnt. The composition of thyroid hormones by the thyroid gland helps in the domination of the body's metabolism. The thyroid glands are composed of two active thyroid hormones, levothyroxine and triiodothyronine. To regulate the temperature of the body these hormones are imperative in the fabrication and also in the comprehensive construction and supervision. Specifically, thyroxin (T4) and triiodothyronine (T3) are the two types of active hormones that are customarily composed by the thyroid glands. These hormones are decisive in protein management;

dissemination in the body temperature, along with the energy-bearing and transmission in every part of the body. For these two thyroid hormones i.e. (T3 and T4), iodine is considered as the main building chunk of the thyroid glands and are prostrated in a few specific problems, some of which are exceptionally prevalent. Insufficiency of thyroid hormones elements to hypothyroidism as well as an excessive thyroid hormones element to Hyperthyroidism. There are many origins related to hyperthyroidism and underactive thyroids. There are various kinds of medications. Thyroid surgery is liable to ionizing radiation, continual tenderness of the thyroid, deficiency of iodine and lack of enzyme to make thyroid hormones.

The applications of AI in medicine are developing quickly. In 2016, AI projects coupled with medicine drew in more speculation from the global economy than other projects. In medicine, AI refers to the utilization of automated diagnosis processes and the treatment of patients who require care. Increased AI utilization in prescription will allow a considerable amount of the role to be automated, opening up medicinal experts' time to be used in performing different obligations, ones that cannot be automated. As such, this technology promises progressively significant utilization in the field of human resources (HR).

In general, ML is categorized as supervised (i.e., consists of output variables that are predicted from input variables) or unsupervised (i.e., deals with clustering of different groups for a particular intervention). ML is used to determine complex models, and extract medical knowledge, exposing novel ideas to practitioners, and specialists.

The significance of the disease is classified on the basis of many methods like Decision Tree, Naïve Bayes (NB), SVM, Random Forests and others.Since the nature of the thyroid disease is complex and therefore, it is necessary to handle the disease very carefully. If not handled carefully the effect of the thyroid disease may cause several health problems or affect the normal functioning of thyroid gland. During previous years, numerous studies have been performed on what are the various pathological and serological parameters of TD and why these parameters are necessary for building a classification model. Studies on whether it is possible to predict the TD of the patient based on pathological parameters only, i.e. the patient doesn"t need to undergo for painful laboratory needle tests. Similarly, is it also possible to diagnose the TD based on serological parameters only, i.e. serological parameters comprise the values obtained from blood samples. Since most of the researchers have conducted a

research on UCI dataset. After properly analysing the UCI dataset, is it possible to design a new dataset with proper attributes related to TD.

In clinical practice, ML predictive models can highlight enhanced rules in the decision-making regarding individual patient care. These are also capable of autonomous diagnosis of different diseases under clinical regulations. In, the incorporation of these models in drug prescription can save doctors and offer new medical opportunities in pathology identification. With ML models, it can also be possible to improve quality of medical data, reduce fluctuations in patient rates, and save in medical costs. Therefore, these models are frequently used to investigate diagnostic analysis when compared with other conventional methods. To reduce the death rates caused by chronic diseases (CDs), early detection and effective treatments are the only solutions. Therefore, most medical scientists are attracted to the new technologies of predictive models in disease forecasting. These new advancements in medical care have been expanding the accessibility of electronic data and opening new doors for decision support and productivity improvements. ML methods have been effectively utilized in the computerized interpretation of pneumonic capacity tests for the differential analysis of CDs. It is expected that the models with the highest accuracies could gain large importance in medical diagnosis. Due to the low-progress nature of CDs, it is important to make an early prediction and provide effective medication. Therefore, it is essential to propose a decision model which can help to diagnose chronic diseases and predict future patient outcomes. While there are many ways to approach this in the field of AI, the present study focuses distinctly on ML predictive models used in the diagnosis of CDs, which highlights the importance of this study.

2. LITERATURE SURVEY

We have seen that data mining techniques are used in the prediction of accuracy related to TD. Numerous methods have been used in the past for knowledge abstraction by using recognized techniques of data mining for TD prediction.

In one research, data mining techniques were used for thyroid disease prediction. These methods use dataset from UCI repository, where features were extracted for disease prediction. The dataset with support vector machine (SVM), Decision Tree is used for classification, where data set was chopped for training and testing purpose. The highest accuracy was achieved by SVM with 99.63% accuracy.

In another a research came to conclusion that multiclass classifier algorithm achieved the highest accuracy of 99.5%. The dataset was taken from UCI machine learning repository, which is free, public accessible for research purposes. Ataide et al. proposed soft computing techniques for thyroid prediction. The results on the UCI data set showed that multilayer perceptron (MLP) yielded an accuracy of 97.4%. However, after feature extraction the same classifier shown an accuracy of 91.7% which is less than previous results.

Yadav et al. generated ensemble methods (bagging+boosting) for thyroid prediction after comparing bagging, boosting and stacking methods. The dataset was downloaded from UCI machine learning repository. During experimentation the author found the performance measure of ROC=98.5, MAE=0.49, RMSE=0.07 and RAE=37.83 and RRSE=51.93. Sidiq et al. implemented Decision Tree, Naïve Bayes, SVM and K nearest neighbour (KNN) in anaconda. 10-fold cross validation method was used to guarantee results. After experimentation on UCI dataset, it was concluded that Decision Tree obtained the highest accuracy of 98.89 than other classification techniques.

Razia et al.employed SVM, Multiple Linear Regression, Naïve Bayes and Decision Tree on dataset collected from UCI. The results of these classifiers were compared and it was found that Decision Tree performed well and showed an accuracy of 99.23%.

Deepita et al concluded that the use Decision Tree showed better results during the prediction of various diseases. The decision trees showed an accuracy of 95% on

thyroid data set downloaded from UCI, however both the SVM and artificial neural network (ANN) performed well and obtained an accuracy of 98.6%.

Gurram et al. compared logistic regression and SVM for thyroid disease prediction on the data set taken from UCI. The results showed that former performed well that latter and showed the accuracy of 98.82%.

Shrivas et al. used ensemble approach with forward and backward feature selection method for thyroid prediction on the thyroid dataset taken from UCI. The experiment was carried out in rapid minor tool with 70% of data for training and remaining 30% for testing purposes. The proposed ensemble model of Random Forest, Naïve Bayes and KNN achieved the accuracy of 97.61%.

Ammulu K et al. took hypothyroid data set from UCI machine learning repository and applied random forest classifier for thyroid prediction. The results generated in WEKA tool showed an accuracy of 70.51%.

Agarwal et al. proposed auto associative neutral network (AANN) on thyroid dataset. The data set collected from UCI was partitioned into 60-40% split for training and testing purpose. The resulting AANN approach yielded the accuracy of 95.1%.

Mazin Abdul Rasool Hameed used multilayer forward feed neural network trained by back propagation algorithm for prediction of thyroid disease. The neural network contained only one hidden layer with five neurons which showed the classification rate of 99.2%. The proposed neural network was designed and tested in MATLAB. The dataset was collected from real patients containing three attributes as T3, T4 and TSH.

Mahurkar et al. devised improvised k means algorithm for normalization of raw data. The normalised data set was fed to feed forward neural network, which achieved an accuracy of 98.21%. The data set collected from UCI contained 215 instances.

Dewangan et al. developed classification and regression tree (CART) on the UCI thyroid data set. Initially info gain and gain ratio feature selection techniques were used with CART as base classifier. After comparison of feature selection techniques with CART as classification model, the best classifier (CART-info gain) achieved an accuracy of 99.47%.

Bekar et al. compared the performance of various decision algorithms to find out best algorithm for thyroid prediction. The data set was collected from a general surgeon working at hospital (not mentioned). After experimentation it was concluded

that Naïve Bayes tree showed the top accuracy of 75%. Ionitaet et al compared radial basis function, Naïve Bayes, multi-layer perception and decision tree classifiers to find out the best classifier for thyroid prediction. The data set used to test and validate the classifiers was taken from the website containing Romanian data and UCI machine learning repository. During experimentation it was shown that decision tree showed the best accuracy of 97.35% with removal of three attributes in data set.

Dash et al. proposed Naïve Bayes classifier by using ranker search as feature optimization technique for thyroid disease prediction. The dataset obtained from UCI repository was trained and tested by 10 fold cross validation. The results achieved an accuracy of 95.38%. As shown above, ensemble learning techniques are the most popular and effective machine learning methods used to diagnose TD. However, few studies have been conducted for using ensemble methods to diagnose TD. Ensemble learning methods are effective in diagnosing TD. Ensemble methods are effective in diagnosing TD as they combine the results of several classifiers into one prediction model. Ensemble methods are also known as meta-algorithms that reduce variance and bias to improve the results. It has been also seen that most of the researchers have taken data set from UCI machine learning repository in the field of thyroid disease prediction. The main objective of our research is to predict the thyroid disease of the real world patients.

3. PROBLEM ANALYSIS

Thyroid Disease (TD) has become one of the most common endocrine disorders worldwide. Although the cause of TD is still unknown, however the symptoms of TD can be reduced if the illness is identified at an early stage. Recent survey conducted in India on TD reveals that approximately 42 million people are suffering from TD. It is not easy to identify TD because of a variety of threatening factors such as high cholesterol, high blood pressure, unusual pulse rate and various other factors. Numerous data mining techniques have been applied in order to find out the seriousness of thyroid disease among Homosapiens.

They have been various attempts and success in predicting the TD with help of data mining.

3.1 Clear definition of the problem

The major problem with Thyroid disease prediction using data mining algorithms is its difficult to get the required accuracy in every case tested. They way be cases were accuracy could be achieved and were its accuracy is below average.

3.2 Boundaries and Constraints

Many of the applications require high level to moderate level specifications of the system to be present and also in some cases, internet of strong connectivity to ensemble and run the programs.

Currently, our project aims at working towards a more practical, efficient and a moderately powerful method for getting required accuracy with help of data mining algorithms.

3.3 Feasibility Analysis

As the name implies, a feasibility study is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. It tells us whether a project is worth the investment—in some cases, a project may not be doable. There can be many reasons for this, including requiring too many resources, which not only prevents those resources from performing

other tasks but also may cost more than an organization would earn back by taking on a project that isn't profitable.

3.3.1 Economical Feasibility

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility—helping decision makers determine the positive economic benefits to the organization that the proposed project will provide. Our project is economically feasible because in this we have used Windows OS, loaded with applications like Anaconda Navigator and Python3 installed. Jupyter Notebooks will be used as the execution environment while also providing a GUI with the help of ipywidgets library which are all available as an open source.

3.3.2 Technical Feasibility

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity. Technical feasibility also involves evaluation of the hardware, software, and other technology requirements of the proposed system. The requirements include a basic desktop computer or other form of computation machine loaded with Windows/ Linux/ MacOS operating softwares with atleast 4GB of RAM and 256GB Hard Disk space to run the execution environments without any hassle. An Intel i3 processor with atleast 2 cores or similar processors will be essential to carry out the essential functions. A prototype of the tool was developed to verify the technical feasibility. The prototype is working successfully and hence the project is feasible.

4. DESIGN

4.1 Model Stages

A novel blend of the best machine learning models are utilized to build up a ensemble model for deciding the presence of thyroid in the thyroidal patient. The system creates the proposed model in two stages-

- 1st stage- Six distinct models are trained with 70% dataset and 30% dataset for thyroid disease diagnosis. (Tier 1 models).
- 2nd Stage - Top models are consolidated and ensembled, making unique combinations to possibly further improve the performance of thyroid prediction model (Tier 2 models).

4.2 Data Description

The dataset deployed in this research is the primary dataset that was collected from the UCI repository.

The data was collected by using pre-designed questionnaire. The attributes in our dataset were finalized with the help of thyroid medical expert. The records were created in Microsoft excel 2019 in comma delimited file (CSV) format. The complete details of the records collected are shown in Fig. 1. The dataset contains 7200 instances with 21 attributes and a multi-class attribute. The dataset comprises the combination of pathological and serological attributes. The multi-class attribute is having three possible values (1 for Normal, 2 for Hyperthyroidism, 3 for Hypothyroidism). The complete description of the dataset is given in Fig. 2.

Thyroid Disease (thyroid0387) data set			
Type	Classification	Origin	Real world
Features	21	(Real / Integer / Nominal)	(6 / 15 / 0)
Instances	7200	Classes	3
Missing values?			No

Fig.1: Dataset description

Attribute	Domain	Attribute	Domain	Attribute	Domain
Age	[0.01, 0.97]	Thyroid_surgery	[0, 1]	Hypopituitary	[0, 1]
Sex	[0, 1]	I131_treatment	[0, 1]	Psych	[0, 1]
On_thyroxine	[0, 1]	Query_hypothyroid	[0, 1]	TSH	[0.0, 0.53]
Query_on_thyroxine	[0, 1]	Query_hyperthyroid	[0, 1]	T3	[0.0005, 0.18]
On_antithyroid_medication	[0, 1]	Lithium	[0, 1]	TT4	[0.0020, 0.6]
Sick	[0, 1]	Goitre	[0, 1]	T4U	[0.017, 0.233]
Pregnant	[0, 1]	Tumor	[0, 1]	FTI	[0.0020, 0.642]
Class	{1,2,3}				

Fig.2: Attributes description

Attributes	Description
Age	In years
Sex	Male or female
TSH	Thyroid-Stimulating Hormone
T3	Triiodothyronine
TBG	Thyroid binding globulin
T4U	Thyroxin utilization rate
TT4	Total Thyroxin
FTI	Free Thyroxin Index

Fig .3 : Attributes for feature selection

4.3 Software Requirements

Below are the minimum software requirements the system must be satisfying before loading and running the project.

The system must be running on Windows, Linux or MacOS operating systems. Incase of Windows OS, Anaconda Navigator must be installed to work with jupyter notebooks. Python3 version 3.7 or above is necessary to carry out the operations performed in the project. Microsoft Excel or any similar spreadsheets application will be required to convert the required dataset into a comma delimited file (CSV) format.

Essential packages required to be installed in the anaconda navigator are:

Scikit-learn,a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Pandas, a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

Matplotlib, a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

Seaborn, a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Ipywidgets which are interactive widgets for Jupyter Notebooks, JupyterLab and Ipython Kernel. Notebooks come alive when interactive widgets are used. The objective is for users to gain control of their data and to visualize changes in data. Learning becomes immersive, fun experience. Researchers can easily see how changing inputs to a model impact the results.

4.4 Algorithms Used

4.4.1 Logistic Regression

Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical.

Due to the nature of probability, the prediction will fall in [0, 1]. Decisions are made based on whether the probability is greater than or less than 0.5

However, the range of linear regression is from negative infinite to positive infinite, not in [0, 1]. Thus, sigmoid function is introduced to solve this problem.

Multiclass classification with logistic regression can be done either through the one-vs-rest scheme in which for each class a binary classification problem of data belonging or not to that class is done

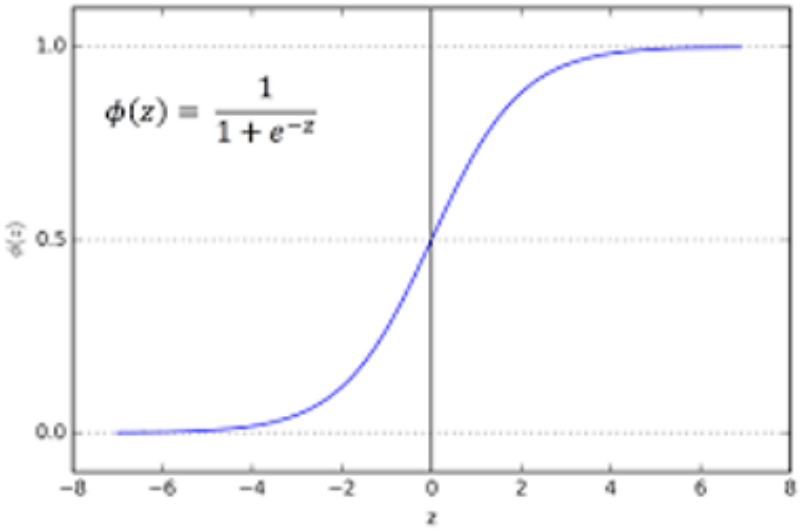


Fig.4: Logistic Regression

4.4.2 Support Vector Machines

Support Vector Machines were developed for binary classification problems, although extensions to the technique have been made to support multi-class classification and regression problems. The algorithm is often referred to as SVM for short. SVM was developed for numerical input variables, although will automatically convert nominal values to numerical values. Input data is also normalized before being used.

SVM work by finding a line that best separates the data into the two groups. This is done using an optimization process that only considers those data instances in the training dataset that are closest to the line that best separates the classes.

The instances are called support vectors, hence the name of the technique. In almost all problems of interest, a line cannot be drawn to neatly separate the classes, therefore a margin is added around the line to relax the constraint, allowing some instances to be misclassified but allowing a better result overall.

Finally, few datasets can be separated with just a straight line. Sometimes a line with curves or even polygonal regions need to be marked out. This is achieved with SVM by projecting the data into a higher dimensional space in order to draw the lines and make predictions. Different kernels can be used to control the projection and the amount of flexibility in separating the classes.

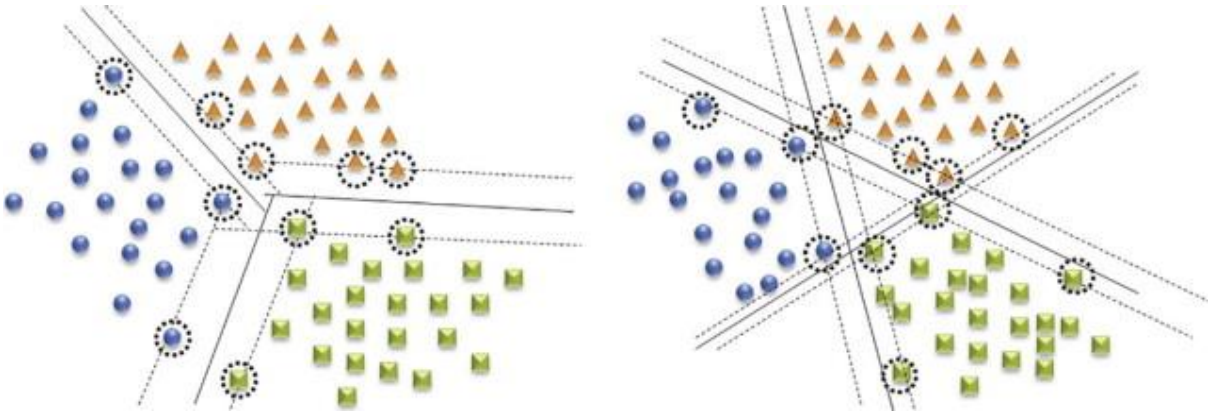


Fig.5: Support Vector Machines

4.4.3 K Nearest Neighbors

The k-nearest neighbors algorithm supports both classification and regression. It is also called KNN for short. It works by storing the entire training dataset and querying it to locate the k most similar training patterns when making a prediction. As such, there is no model other than the raw training dataset and the only computation performed is the querying of the training dataset when a prediction is requested. It is a simple algorithm, but one that does not assume very much about the problem other than that the distance between data instances is meaningful in making predictions. As such, it often achieves very good performance.

KNN or k-nearest neighbours is the simplest classification algorithm. This classification algorithm does not depend on the structure of the data.

Whenever a new example is encountered, its k nearest neighbours from the training data are examined. Distance between two examples can be the euclidean distance between their feature vectors.

The majority class among the k nearest neighbours is taken to be the class for the encountered example

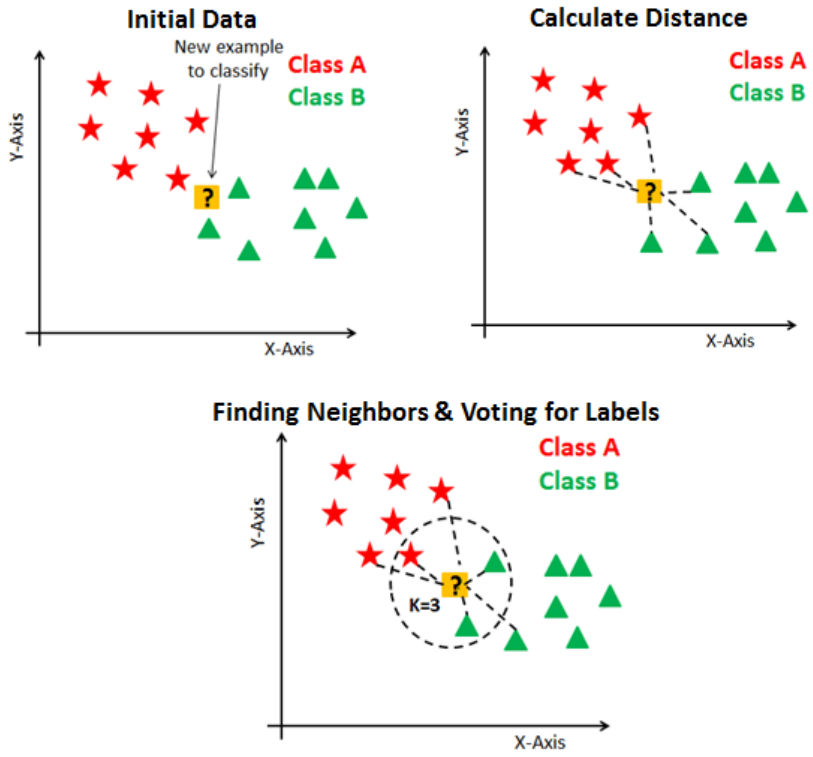


Fig.6: K-Nearest Neighbors

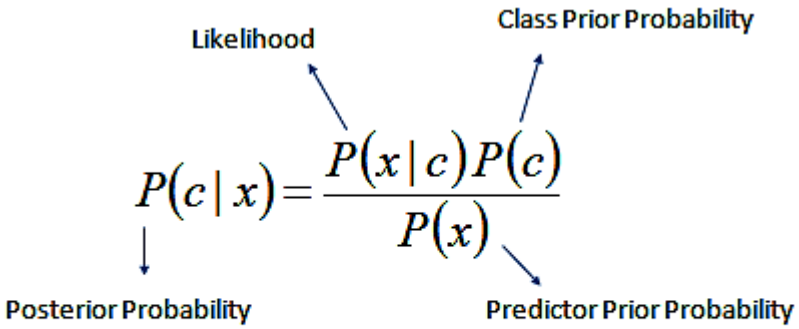
4.4.4 I Bayes Algorithm

Naïve Bayes is a classification algorithm. Traditionally it assumes that the input values are nominal, although it numerical inputs are supported by assuming a distribution. Naïve Bayes uses a simple implementation of Bayes Theorem (hence naïve) where the prior probability for each class is calculated from the training data and assumed to be independent of each other (technically called conditionally independent).

This is an unrealistic assumption because we expect the variables to interact and be dependent, although this assumption makes the probabilities fast and easy to calculate. Even under this unrealistic assumption, Naïve Bayes has been shown to be a very effective classification algorithm. Naïve Bayes calculates the posterior probability for each class and makes a prediction for the class with the highest probability. As such, it supports both binary classification and multi-class classification problems.

The algorithm used in the project is the Multinomial I Bayes Classifier I Bayes classification method is based on Bayes' theorem. It is termed as 'I' because it assumes independence between every pair of feature in the data.

Multinomial I Bayes considers a feature vector where a given term represents the number of times it appears or very often i.e. frequency.



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Fig.7. Naïve bayes formula

4.4.5 Decision Tree Algorithm

Decision trees can support classification and regression problems. Decision trees are more recently referred to as Classification And Regression Trees (CART). They work by creating a tree to evaluate an instance of data, start at the root of the tree and moving town to the leaves (roots) until a prediction can be made.

The process of creating a decision tree works by greedily selecting the best split point in order to make predictions and repeating the process until the tree is a fixed depth. After the tree is constructed, it is pruned in order to improve the model's ability to generalize to new data.

Decision tree classifier is a systematic approach for multiclass classification. It poses a set of questions to the dataset (related to its attributes/features).

The decision tree classification algorithm can be visualized on a binary tree. On the root and each of the internal nodes, a question is posed and the data on that node is further split into separate records that have different characteristics.

The leaves of the tree refer to the classes in which the dataset is split.

We perform breaking down the data by making decisions based on multiple questions at each level.

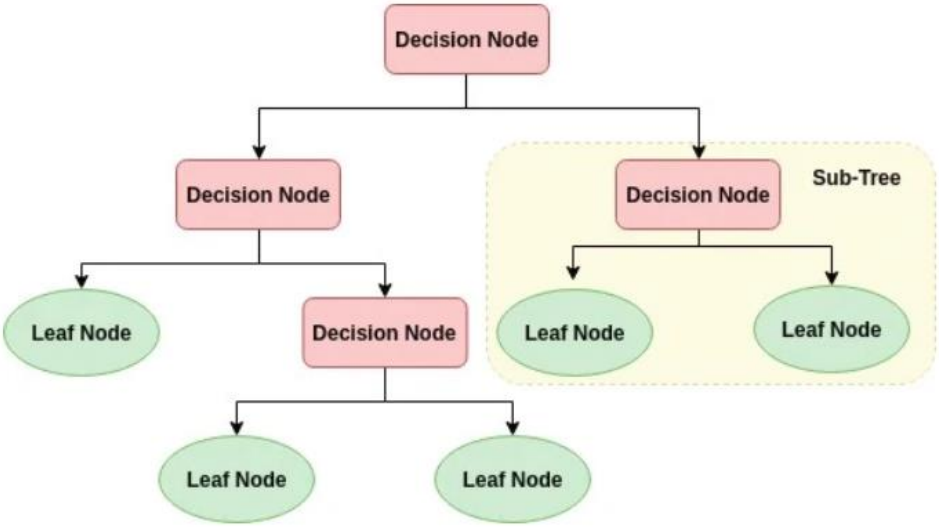


Fig.8: Decision tree model

4.4.6 Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is: A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

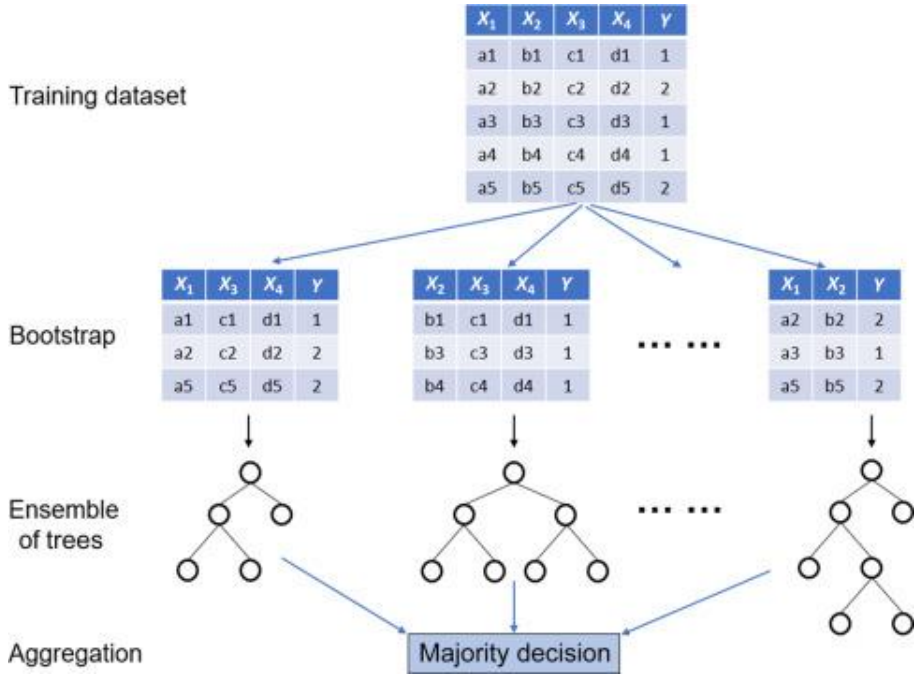


Fig.9: Random forest model

4.4.7 Boosting

Boosting is one of the Meta learning algorithms which focuses on reducing bias. It has the capability of turning weak learners into strong ones. In boosting, the resulting models built, depend on the performance of past built models. During the process of boosting, the machine learning algorithm looks for to find misclassified instances, applies extra weights on each of the misclassified instances and then builds the fresh training data set for new model. During this process the new model built on fresh training data set becomes expert for misclassified instances.

AdaBoost: In AdaBoost, the output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

4.5 UML Diagrams

4.5.1 Use Case Diagram

In this scenario, the developer will be responsible for producing a model with the best performance for the User to make use of it to make calculated predictions based on the input.

The use case model for any system consists of a set of "usecases". The purpose of a use case is to define a piece of coherent behavior without revealing the internal structure of the system. The use cases do not mention any specific algorithm to be used or the internal data representation, internal structure of the software,etc.

A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipse.

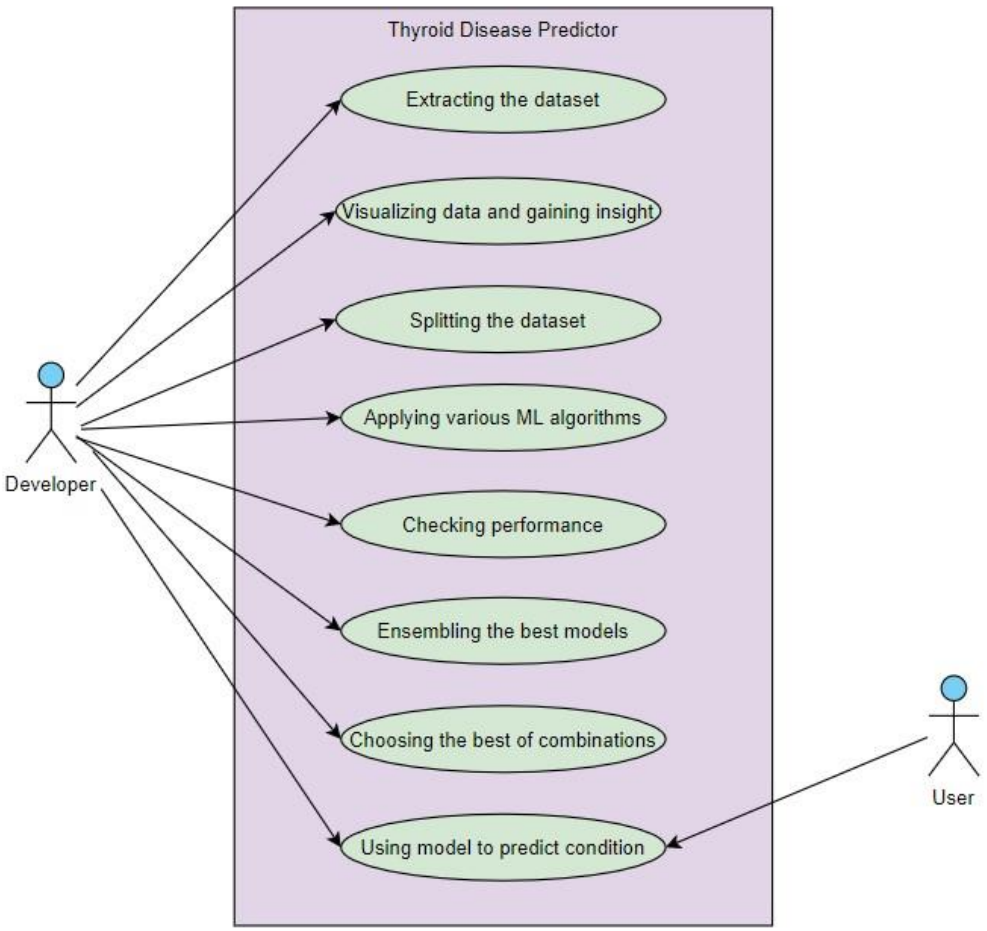


Fig.10: Use case diagram

4.5.2 Sequence Diagram

The below sequence diagram shows a sequence of steps in the development process and the operations being performed based on time.

Sequence Diagrams represent the objects participating the interaction horizontally and time vertically. Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration.

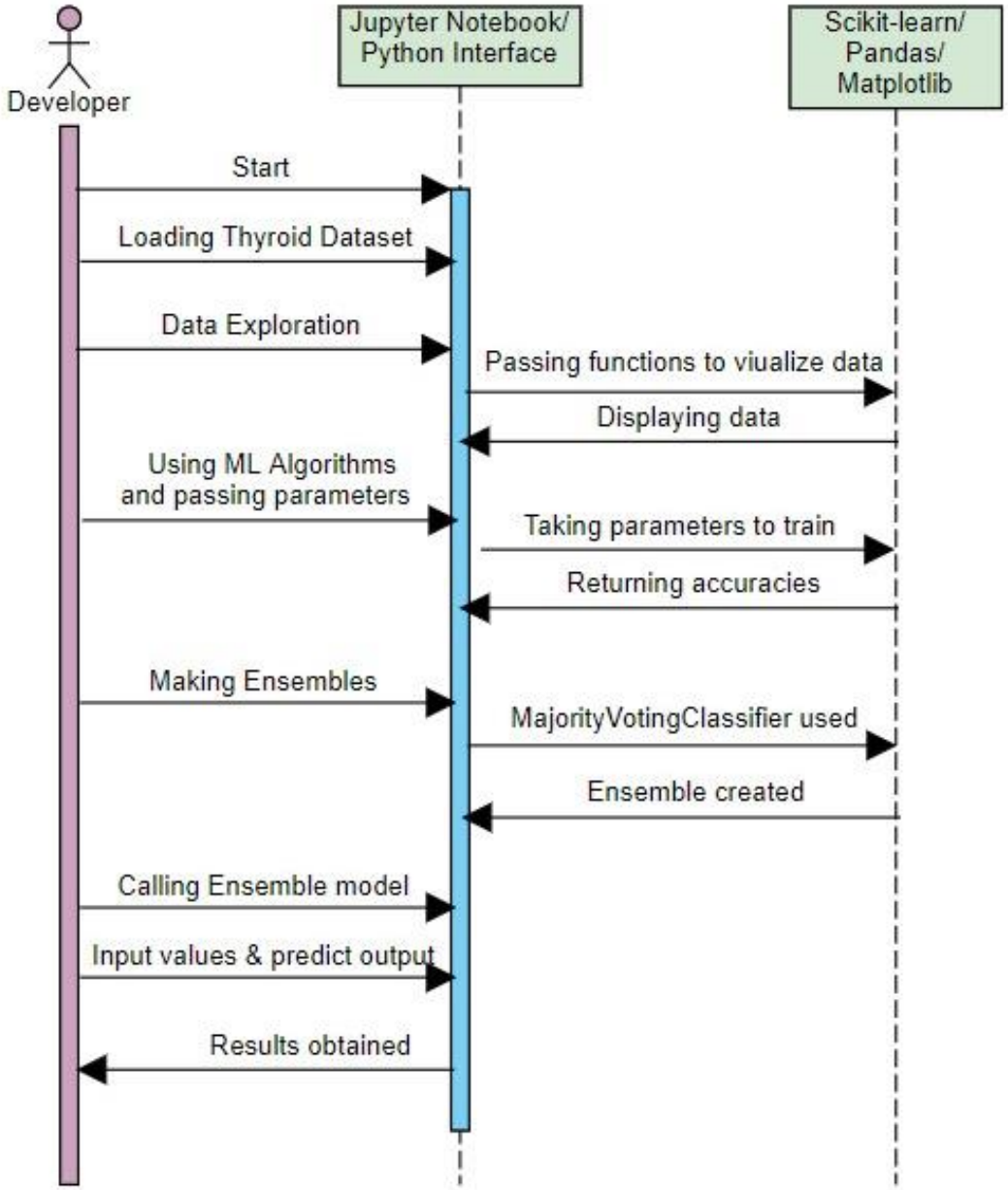


Fig.11: Sequence diagram

4.5.3 Class Diagram

The class diagram shows the possible options of operations that can be performed by every class and their relation to each other as a whole.

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling.

Statechart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

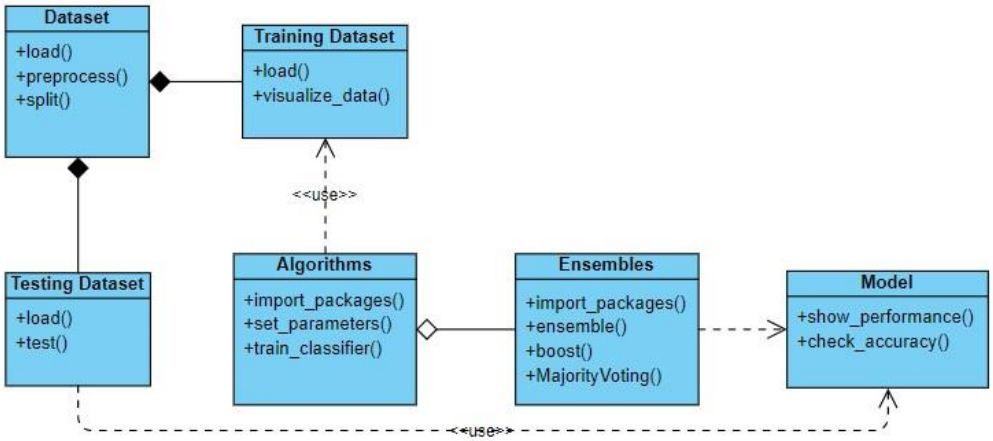


Fig.12: Class diagram

4.5.4 Statechart Diagram

The following statechart diagram describes different states of a component in a system. The states are specific to a component/object of a system.

A Statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

They define different states of an object during its lifetime and these states are changed by events. Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

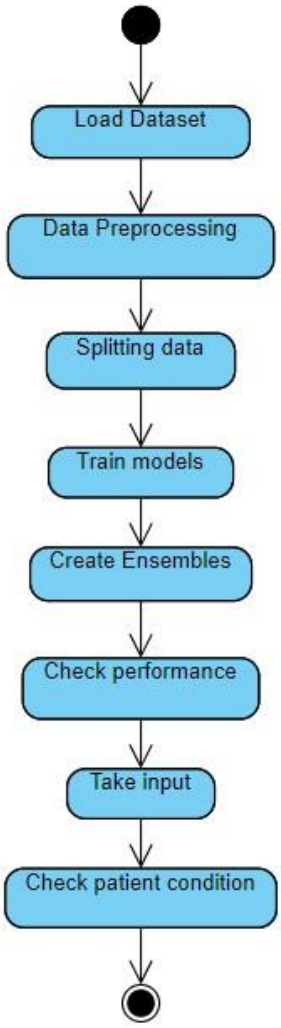


Fig.13: State chart diagram

5. IMPLEMENTATION

5.1 Methodology

Six distinct models are trained with 70% dataset and the rest 30% dataset is used to check the rightness of the models after training for thyroid disease prediction.

Top models are consolidated and ensembled, making unique combinations to possibly further improve the performance of thyroid prediction model.

Figure below shows the approach of the proposed plan for efficient prediction of thyroid disease. The data splitter segments the whole dataset into the proportion of (70:30) of (Train:Test) subsets. The system creates the proposed model in two stages namely tier-1 and tier-2 as denoted in the flowchart.

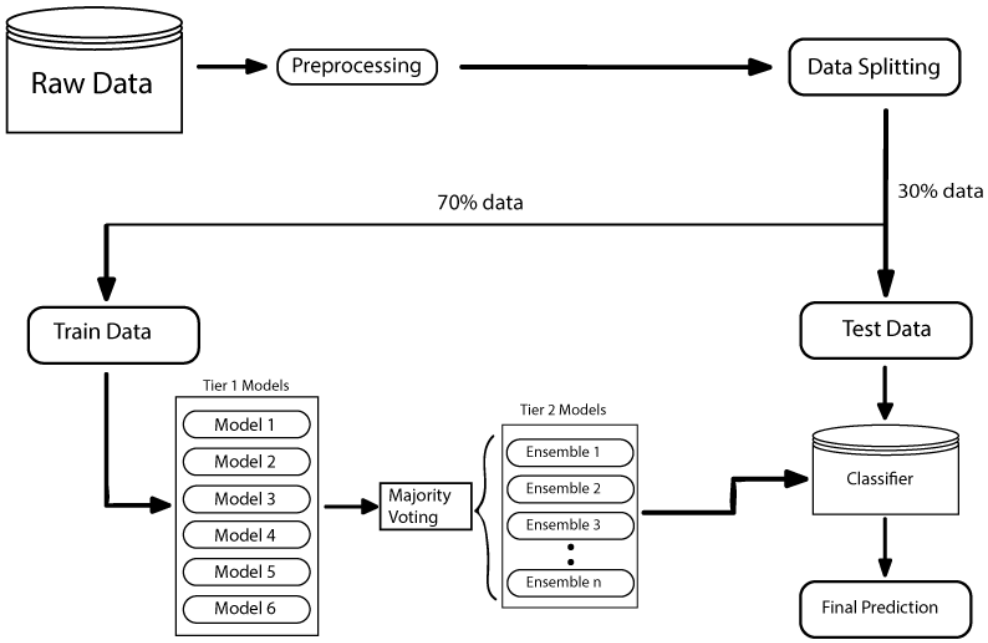


Fig.14: Methodology

In scikit-learn a random split into training and test sets can be quickly computed with the train_test_split helper function. The dataset is split into 70% dataset for training data and 30% dataset for test data.

Scikit-learn machine learning models like Logistic Regression, Support Vector Machines, KNN Classifier, Multinomial Naive Bayes classifier are implemented by importing the essential packages from the sklearn library.

Based on the performance of the models, the models are ensembled using the Voting Classifier in which the ensemble model makes the prediction by majority vote. The models are appended into an array called estimators to pass into the Voting Classifier function. For example, if we use three models and they predict [1, 0, 1] for the target variable, the final prediction that the ensemble model would make would be 1, since two out of the three models predicted 1.

The best of the ensembled model will be utilized to make predictions based on the inputs given through the GUI by the user. The GUI is built using ipywidgets library. Widgets are eventful python objects that have a representation in the browser, often as a control like a slider, textbox, etc.

5.2 Source code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Reading excel file using pandas
df = pd.read_excel("C:/Users/Rahul/Desktop/Stuff/thyroid-new.xls")

# Splitting the dataset into Training and Test sets
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size = 0.30,random_state=0)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='auto', n_jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm_start=False)
logisticRegr.fit(train_x, train_y)

# Using score method to get accuracy of model
score = logisticRegr.score(test_x, test_y)
```

Support Vector Machines

```
from sklearn.svm import SVC
svm = SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
    max_iter=-1, probability=False, random_state=10, shrinking=True, tol=0.001,
```

```
    verbose=False)
svm.fit(train_x, train_y)

# Using score method to get accuracy of model
score = svm.score(test_x, test_y)
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                           weights='uniform')
knn.fit(train_x, train_y)

# Using score method to get accuracy of model
score = knn.score(test_x, test_y)
```

Naive Bayes Classifier

```
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
mnb.fit(train_x, train_y)

# Using score method to get accuracy of model
score = mnb.score(test_x, test_y)
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
# Checking performance with default values
train_accuracy = []test_accuracy = []
for depth in range(1,15):
    decTree = DecisionTreeClassifier(max_depth=depth, random_state=10)
    decTree.fit(train_x, train_y)
    train_accuracy.append(decTree.score(train_x, train_y))
    test_accuracy.append(decTree.score(test_x, test_y))
    frame = pd.DataFrame({'max_depth':range(1,15), 'train_acc':train_accuracy, 'test_
acc':test_accuracy})
```

```
# Checking the depth manually and passing as parameter
decTree = DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=6,
                max_features=None, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, presort=False,
                 random_state=10, splitter='best')
decTree.fit(train_x, train_y)

# Using score method to get accuracy of model
score = decTree.score(test_x, test_y)
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
# Checking performance with default values
train_accuracy = []test_accuracy = []
for depth in range(1,15):
    randForest = RandomForestClassifier(max_depth=depth, random_state=10, n_est
imators=10)
    randForest.fit(train_x, train_y)
    train_accuracy.append(randForest.score(train_x, train_y))
    test_accuracy.append(randForest.score(test_x, test_y))
frame = pd.DataFrame({'max_depth':range(1,15), 'train_acc':train_accuracy, 'test_acc
':test_accuracy})

randForest = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='
gini',
                max_depth=13, max_features='auto', max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, n_estimators=10,
                n_jobs=None, oob_score=False, random_state=10, verbose=0,
                warm_start=False)
randForest.fit(train_x, train_y)
# Using score method to get accuracy of model
score = randForest.score(test_x, test_y)
```

Checking the feature importance of individual attributes for feature selection into models in order to reduce computational load and improve performance of the models.

```
#feature importance against each variable
feature_imp = pd.Series(randForest.feature_importances_,index=train_x.columns).sort_values(ascending=False)
```

Boosting the Classifiers

```
# Load libraries
from sklearn.ensemble import AdaBoostClassifier
# Create adaboost classifer object
abc = AdaBoostClassifier(algorithm='SAMME.R',
                         base_estimator=DecisionTreeClassifier(class_weight=None,
                         criterion='gini', max_depth=6, max_features=None,
                         max_leaf_nodes=None, min_impurity_decrease=0.0,
                         min_impurity_split=None, min_samples_leaf=1,
                         min_samples_split=2, min_weight_fraction_leaf=0.0,
                         presort=False, random_state=10, splitter='best'),
                         learning_rate=1, n_estimators=120, random_state=None)
abc.fit(train_x, train_y)

# Using score method to get accuracy of model
score = abc.score(test_x, test_y)
```

```
# Load libraries
from sklearn.ensemble import GradientBoostingClassifier
# Create adaboost classifer object
gbc = GradientBoostingClassifier(criterion='friedman_mse', init=None,
                                 learning_rate=0.6, loss='deviance', max_depth=10,
                                 max_features=None, max_leaf_nodes=None,
                                 min_impurity_decrease=0.0, min_impurity_split=None,
                                 min_samples_leaf=1, min_samples_split=2,
                                 min_weight_fraction_leaf=0.0, n_estimators=100,
                                 n_iter_no_change=None, presort='auto',
                                 random_state=None, subsample=1.0, tol=0.0001,
                                 validation_fraction=0.1, verbose=0, warm_start=False)
gbc.fit(train_x, train_y)

# Using score method to get accuracy of model
score = gbc.score(test_x, test_y)
```

Creating Ensemble Models

Ensemble-1

```
# Voting Ensemble for Classification
from sklearn import model_selection
from sklearn.ensemble import VotingClassifier
kfold = model_selection.KFold(n_splits=15, random_state=10)
# creating the sub modelsestimators_1 = []
model1 = DecisionTreeClassifier(class_weight=None, criterion='gini',
                                max_depth=6, max_features=None,
                                max_leaf_nodes=None, min_impurity_decrease=0.0,
                                min_impurity_split=None, min_samples_leaf=1,
                                min_samples_split=2, min_weight_fraction_leaf=0.0,
                                presort=False, random_state=10, splitter='best')
estimators_1.append(('dec-tree', model1))
model2 = RandomForestClassifier(
                                    n_estimators=10, n_jobs=None,
                                    oob_score=False, random_state=10,
                                    verbose=0, warm_start=False)
estimators_1.append(('rand-forest', model2))
model3 = SVC(C=1, cache_size=200, class_weight=None,
                                coef0=0.0, decision_function_shape='ovr',
                                degree=3, gamma='auto', kernel='linear',
                                max_iter=-1, probability=False,
                                random_state=10, shrinking=True, tol=0.001,
                                verbose=False)
estimators_1.append(('svm', model3))
# create the ensemble model
ensemble_1 = VotingClassifier(estimators_1)
results = model_selection.cross_val_score(ensemble_1, train_x, train_y, cv=kfold)
ensemble_1.fit(train_x, train_y)
```

Ensemble-2

```
kfold = model_selection.KFold(n_splits=25, random_state=10)
# creating the sub modelsestimators_2 = []
model1 = DecisionTreeClassifier(class_weight=None, criterion='gini',
                                max_depth=6, max_features=None,
                                max_leaf_nodes=None, min_impurity_decrease=0.0,
                                min_impurity_split=None, min_samples_leaf=1,
                                min_samples_split=2, min_weight_fraction_leaf=0.0,
                                presort=False, random_state=10, splitter='best')

estimators_2.append(('dec-tree', model1))
```

```
model2 = RandomForestClassifier(
                                    n_estimators=10, n_jobs=None,
                                    oob_score=False, random_state=10,
                                    verbose=0, warm_start=False)

estimators_2.append(('rand-forest', model2))
model3 = LogisticRegression(C=1.0, class_weight=None,
                                    dual=False, fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='warn', n_jobs=None, penalty='l2',
                                    random_state=None, solver='warn', tol=0.0001,
                                    verbose=0, warm_start=False)
estimators_2.append(('log-regr', model3))
# create the ensemble model
ensemble_2 = VotingClassifier(estimators_2)
results = model_selection.cross_val_score(ensemble_2, train_x, train_y, cv=kfold)
ensemble_2.fit(train_x, train_y)
```

Ensemble-3:

```
kfold = model_selection.KFold(n_splits=25, random_state=10)
# creating the sub models
estimators_3 = []
model1 = DecisionTreeClassifier(class_weight=None, criterion='gini',
                        max_depth=6, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        presort=False, random_state=10, splitter='best')

estimators_3.append(('dec-tree', model1))
model2 = RandomForestClassifier(
                                    n_estimators=10, n_jobs=None,
                                    oob_score=False, random_state=10,
                                    verbose=0, warm_start=False)

estimators_3.append(('rand-forest', model2))
# create the ensemble model
ensemble_3 = VotingClassifier(estimators_3)
results = model_selection.cross_val_score(ensemble_3, train_x, train_y, cv=kfold)
ensemble_3.fit(train_x, train_y)
```

Ensemble -1 produced best results. Hence predict conditions using the first model using predict() function

```
input_1 = [[0.73, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0.0006, 0.015, 0.12, 0.082, 0.1
46]]
ensemble_model = ensemble_2.fit(x, y)
result_1 = ensemble_model.predict(input_1)
```

GUI for values input:

```
import ipywidgets as widgets
from ipywidgets import HBox, VBox, Layout
slider_1 = widgets.IntSlider(min=1, max=97, description='Age:')
row_1 = slider_1
radio_1 = widgets.RadioButtons(options=['male', 'female'], description='Gender:')
radio_2 = widgets.RadioButtons(options=['yes', 'no'], description='On thyroxine:')
radio_3 = widgets.RadioButtons(options=['yes', 'no'], description='Query on thyroxine
:')
radio_4 = widgets.RadioButtons(options=['yes', 'no'], description='On antithyroid med
ication:')
radio_5 = widgets.RadioButtons(options=['yes', 'no'], description='Sick:')
row_2 = HBox([radio_1, radio_2, radio_3, radio_4, radio_5], layout=Layout(margin='
20px 0 0 0'))
radio_6 = widgets.RadioButtons(options=['yes', 'no'], description='Pregnant:')
radio_7 = widgets.RadioButtons(options=['yes', 'no'], description='Thyroid_surgery:')
radio_8 = widgets.RadioButtons(options=['yes', 'no'], description='I131 treatment:')
radio_9 = widgets.RadioButtons(options=['yes', 'no'], description='Query hypothyroid
:')
radio_10 = widgets.RadioButtons(options=['yes', 'no'], description='Query hyperthyro
id:')
row_3 = HBox([radio_6, radio_7, radio_8, radio_9, radio_10], layout=Layout(margin
='20px 0 0 0'))
radio_11 = widgets.RadioButtons(options=['yes', 'no'], description='Lithium:')
radio_12 = widgets.RadioButtons(options=['yes', 'no'], description='Goitre:')
radio_13 = widgets.RadioButtons(options=['yes', 'no'], description='Tumor:')
radio_14 = widgets.RadioButtons(options=['yes', 'no'], description='Hypopituitary:')
radio_15 = widgets.RadioButtons(options=['yes', 'no'], description='Psych:')
row_4 = HBox([radio_11, radio_12, radio_13, radio_14, radio_15], layout=Layout(m
argin='20px 0 0 0'))
slider_2 = widgets.FloatSlider(min=0, max=53, description='TSH:')
slider_3 = widgets.FloatSlider(min=0.05, max=18, description='T3:')
slider_4 = widgets.FloatSlider(min=0.2, max=60, description='TT4:')
row_5 = HBox([slider_2, slider_3, slider_4], layout=Layout(margin='20px 0 0 0'))
slider_5 = widgets.FloatSlider(min=1.7, max=23.3, description='T4U:')
slider_6 = widgets.FloatSlider(min=0.2, max=64.2, description='FTI:')
row_6 = HBox([slider_5, slider_6], layout=Layout(margin='20px 0 0 0'))
gui = VBox([row_1, row_2, row_3, row_4, row_5, row_6], layout=Layout(margin='2
0px 0 20px 0'))
display(gui)button = widgets.Button(description="Check my condition")
output = widgets.Output()
display(button, output)
def on_button_clicked(b):
```

```
  with output:
    output.clear_output()

    Age = slider_1.value/100
    Sex = 1 if radio_1.value=='male' else 0
    On_thyroxine = 1 if radio_2.value=='yes' else 0
    Query_on_thyroxine = 1 if radio_3.value=='yes' else 0
    On_antithyroid_medication = 1 if radio_4.value=='yes' else 0
    Sick = 1 if radio_5.value=='yes' else 0
    Pregnant = 1 if radio_6.value=='yes' else 0
    Thyroid_surgery = 1 if radio_7.value=='yes' else 0
    I131_treatment = 1 if radio_8.value=='yes' else 0
    Query_hypothyroid = 1 if radio_9.value=='yes' else 0
    Query_hyperthyroid = 1 if radio_10.value=='yes' else 0
    Lithium = 1 if radio_11.value=='yes' else 0
    Goitre = 1 if radio_12.value=='yes' else 0
    Tumor = 1 if radio_13.value=='yes' else 0
    Hypopituitary = 1 if radio_1.value=='yes' else 0
    Psych = 1 if radio_15.value=='yes' else 0
    TSH = slider_2.value/100
    T3 = slider_3.value/100
    TT4 = slider_4.value/100
    T4U = slider_5.value/100
    FTI = slider_6.value/100

    input_1 = [[Age, Sex, On_thyroxine, Query_on_thyroxine, On_antithyroid_medi
cation, Sick, Pregnant, Thyroid_surgery, I131_treatment, Query_hypothyroid, Query_
hyperthyroid, Lithium, Goitre, Tumor, Hypopituitary, Psych, TSH, T3, TT4, T4U, FT
I]]
    ensemble_model = ensemble_1.fit(x, y)
    result = ensemble_model.predict(input_1)
    if result == [2]:
      print("The person is suffering from hyper-thyroidism")
    elif result == [3]:
      print("The person is suffering from hypo-thyroidism")
    else:
      print("The person is normal")
button.on_click(on_button_clicked)
```

6. TESTING AND VALIDATION

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of software as a system element and attendant costs associated with a software failure are motivating factors for we planned, through testing. Testing is the process of executing a program with the intent of finding an error. The design of tests for software and other engineered products can be as challenging as the initial design of the product itself.There of basically two types of testing approaches.

One is Black-Box testing – the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operated.The other is White-Box testing – knowing the internal workings of the product ,tests can be conducted to ensure that the internal operation of the product performs according to specifications and all internal components have been adequately exercised.

White box and Black box testing methods have been used to test this package. The entire loop constructs have been tested for their boundary and intermediate conditions. The test data was designed with a view to check for all the conditions and logical decisions. Error handling has been taken care of by the use of exception handlers.

6.1 Testing Strategies

Testing is a set of activities that can be planned in advanced and conducted systematically. A strategy for software testing must accommodation low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.Software testing is one element of verification and validation. Verification refers to the set of activities that ensure that software correctly implements as specific function. Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.

The main objective of software is testing to uncover errors. To fulfill this objective, a series of test steps unit, integration, validation and system tests are

planned and executed. Each test step is accomplished through a series of systematic test technique that assist in the design of test cases. With each testing step, the level of abstraction with which software is considered is broadened.Testing is the only way to assure the quality of software and it is an umbrella activity rather than a separate phase. This is an activity to be preformed in parallel with the software effort and one that consists of its own phases of analysis, design, implementation, execution and maintenance.

6.1.1 Unit Testing

This testing method considers a module as single unit and checks the unit at interfaces and communicates with other modules rather than getting into details at statement level. Here the module will be treated as a black box, which will take some input and generate output. Outputs for a given set of input combination are pre-calculated and are generated by the module.

6.1.2 Integrated Testing

Testing is a major quality control measure employed during software development. Its basic function is to detect errors. Sub functions when combined may not produce than it is desired. Global data structures can represent the problems. Integrated testing is a systematic technique for constructing the program structure while conducting the tests. To uncover errors that are associated with interfacing the objective is to make unit test modules and built a program structure that has been detected by design.

In a non - incremental integration all the modules are combined in advance and the program is tested as a whole. Here errors will appear in an end less loop function. In incremental testing the program is constructed and tested in small segments where the errors are isolated and corrected.

Different incremental integration strategies are top – down integration, bottom – up integration, regression testing.

6.1.3 Top-Down Integration Test

Modules are integrated by moving downwards through the control hierarchy beginning with main program. The subordinate modules are incorporated into

structure in either a breadth first manner or depth first manner. This process is done in five steps:

- Main control module is used as a test driver and steps are substituted or all modules directly to main program.
- Depending on the integration approach selected subordinate is replaced at a time with actual modules.
- On completion of each set of tests another stub is replaced with the real module
- Regression testing may be conducted to ensure trha5t new errors have not been introduced. This process continuous from step 2 until entire program structure is reached. In top down integration strategy decision making occurs at upper levels in the hierarchy and is encountered first. If major control problems do exists early recognitions is essential.If depth first integration is selected a complete function of the software may be implemented and demonstrated.Some problems occur when processing at low levels in hierarchy is required to adequately test upper level steps to replace low-level modules at the beginning of the top down testing. So, no data flows upward in the program structure.

6.1.4 Bottom-Up Integration Test

- Begins construction and testing with atomic modules. As modules are integrated from the bottom up, processing requirement for modules subordinate to a given level is always available and need for stubs is eliminated. The following steps implements this strategy.
- Low-level modules are combined in to clusters that perform a specific software sub function.
- A driver is written to coordinate test case input and output.
- Cluster is tested.

7. RESULT ANALYSIS

The dataset can be viewed as a dataframe by using pandas library. To view the dataframe, import the dataset as a dataframe variable and call "df.head()". First 5 rows of the dataset will be visible.

```
df.head()
```

lication	Sick	Pregnant	Thyroid_surgery	I131_treatment	Query_hypothyroid	...	Goitre	Tumor	Hypopituitary	Psych	TSH	T3	TT4	T4U	FTI
0	0	0	0	1	0	...	0	0	0	0	0.00060	0.015	0.120	0.082	0.146
0	0	0	0	0	0	...	0	0	0	0	0.00025	0.030	0.143	0.133	0.108
0	0	0	0	0	0	...	0	0	0	0	0.00190	0.024	0.102	0.131	0.078
0	0	0	0	0	0	...	0	0	0	0	0.00090	0.017	0.077	0.090	0.085
0	0	0	0	0	0	...	0	0	0	0	0.00025	0.026	0.139	0.090	0.153

Fig.15: Dataset visualization

On calling the "describe()" function on the dataframe, common attributes regarding the values in the dataset are displayed as follows:

```
df.describe(include="all")
```

	Age	Sex	On_thyroxine	Query_on_thyroxine	On_antithyroid_medication	Sick	Pregnant	Thyroid_surgery	I131_treatment
count	7200.000000	7200.000000	7200.000000	7200.000000	7200.000000	7200.000000	7200.000000	7200.000000	7200.000000
mean	0.520518	0.304306	0.130556	0.015417	0.012778	0.038333	0.010833	0.014028	0.016806
std	0.189269	0.460145	0.336937	0.123212	0.112322	0.192013	0.103525	0.117613	0.128551
min	0.010000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.370000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.550000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.670000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	0.970000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Fig.16: Described dataset attributes

Now, coming to the performance of the algorithms trained over the 70% training data, the accuracies followed by the confusion matrices are displayed below.
In the confusion matrices, the X-axis refers to the true data and the Y-axis refers to the predicted data.

The best performing models are consolidated into groups to form ensembles of classifiers.

Logistic Regression:

```
# Using score method to get accuracy of model
score = logisticRegr.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  91.99074074074073
```

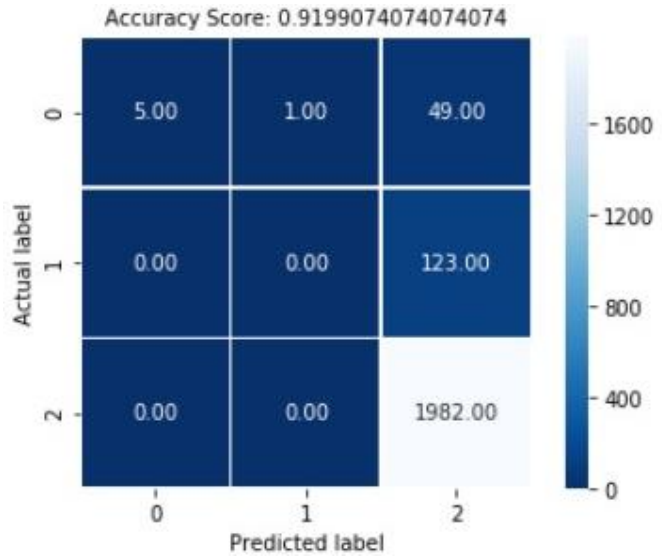


Fig.17: Logistic Regression Confusion Matrix

Support Vector Machines:

```
# Using score method to get accuracy of model
score = svm.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  92.12962962962963
```

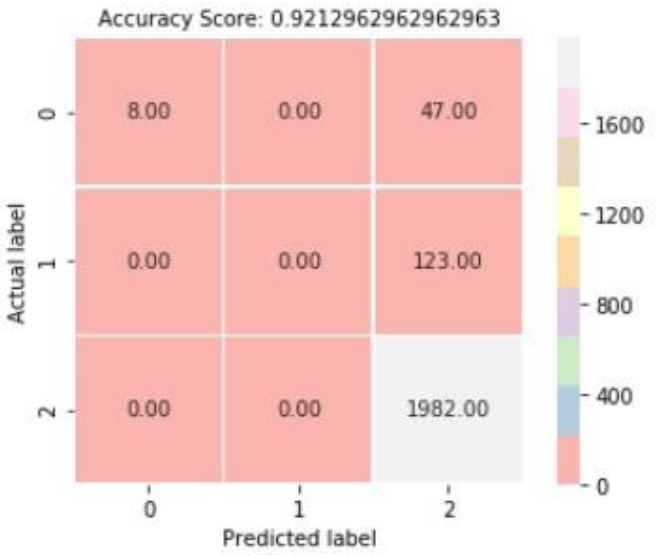


Fig.18: SVM Confusion Matrix

K-Nearest neighbours classifier:

```
# Using score method to get accuracy of model
score = knn.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  92.96296296296296
```

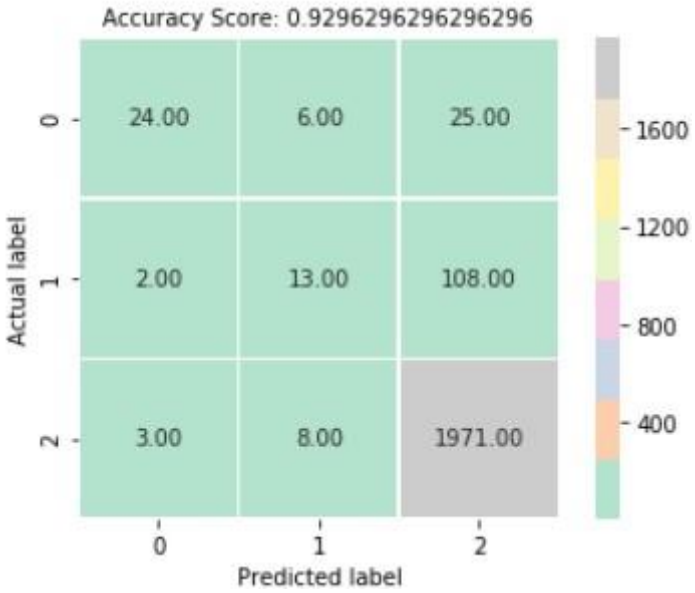


Fig.19: KNN Confusion Matrix

Multinomial Naïve Bayes:

```
# Using score method to get accuracy of model
score = mnb.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  91.75925925925927
```

Decision Tree Classifier:

To learn at what depth the decision trees lose efficacy in performance, we will plot a chart to depict the change in accuracies for each level increased in the depth.

	max_depth	train_acc	test_acc
0	1	0.952976	0.950926
1	2	0.975000	0.976389
2	3	0.991667	0.990741
3	4	0.995635	0.993056
4	5	0.997619	0.994444
5	6	0.998413	0.995370
6	7	0.999405	0.993981
7	8	1.000000	0.994907
8	9	1.000000	0.994907
9	10	1.000000	0.994907
10	11	1.000000	0.994907
11	12	1.000000	0.994907
12	13	1.000000	0.994907
13	14	1.000000	0.994907

```
plt.figure(figsize=(12,6))
plt.plot(frame['max_depth'], frame['train_acc'], marker='x')
plt.plot(frame['max_depth'], frame['test_acc'], marker='x')
plt.xlabel('Depth of tree')
plt.ylabel('performance')
plt.show()
```

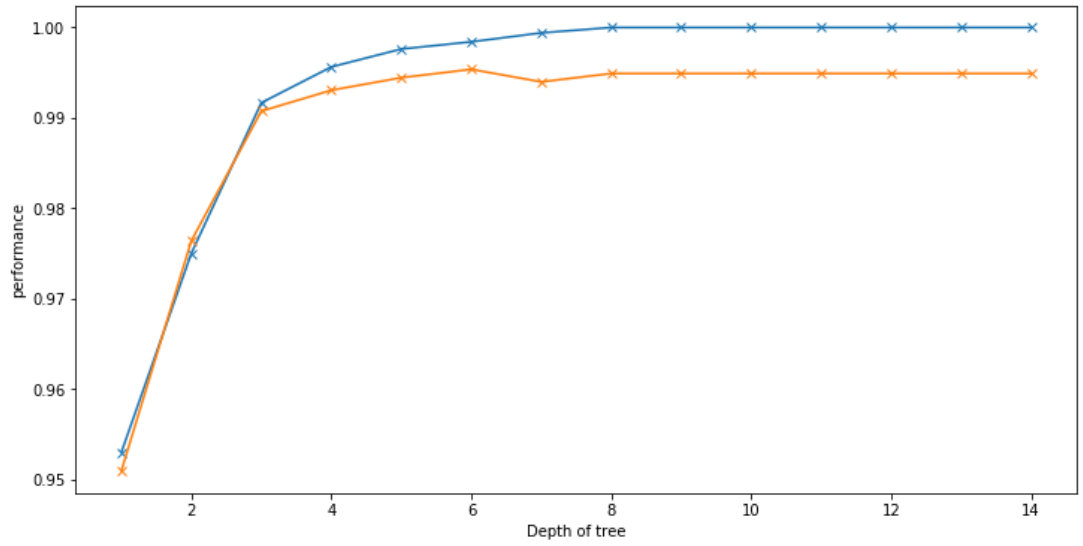


Fig.20: Depth vs. Accuracy for Dec.Trees

The decision trees performed best up until depth level = 6 and then stayed stagnant without any noticeable increase in the performance. Hence, the model is fit until depth=6. The accuracy of the classifier is:

```
# Using score method to get accuracy of model
score = decTree.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  99.53703703703704
```

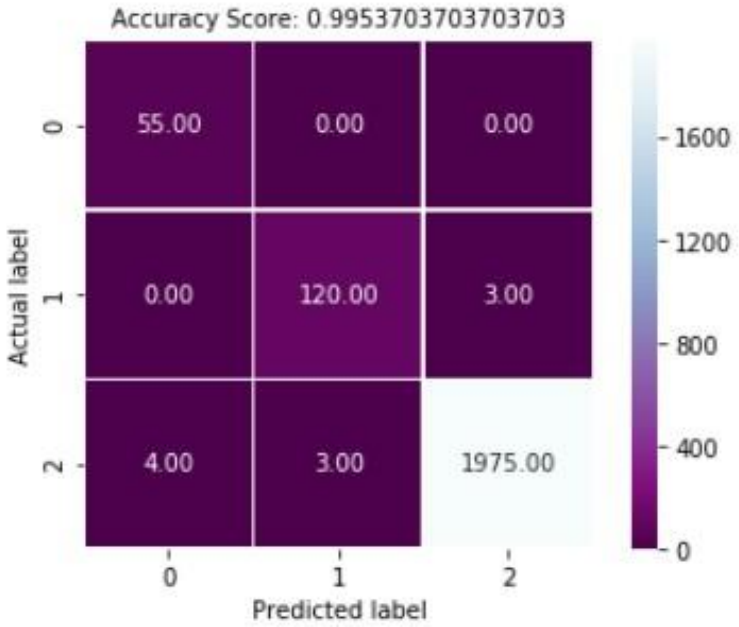


Fig.21: Decision Trees Confusion Matrix

Random Forest Classifier:

Random Forests are in themselves an ensemble of decision trees. To learn at what depth the decision trees lose efficacy in performance, we will plot a chart to depict the change in accuracies for each level increased in the depth.

	max_depth	train_acc	test_acc
0	1	0.929365	0.917593
1	2	0.929563	0.917593
2	3	0.948810	0.937500
3	4	0.987103	0.977778
4	5	0.987302	0.981944
5	6	0.996825	0.992593
6	7	0.997421	0.992130
7	8	0.999008	0.992130
8	9	0.999206	0.993056
9	10	0.999008	0.990741
10	11	0.999802	0.992593
11	12	1.000000	0.991667
12	13	1.000000	0.993056
13	14	1.000000	0.991667

```
plt.figure(figsize=(12,6))
plt.plot(frame['max_depth'], frame['train_acc'], marker='o')
plt.plot(frame['max_depth'], frame['test_acc'], marker='o')
plt.xlabel('Depth of tree')
plt.ylabel('performance')
plt.show()
```

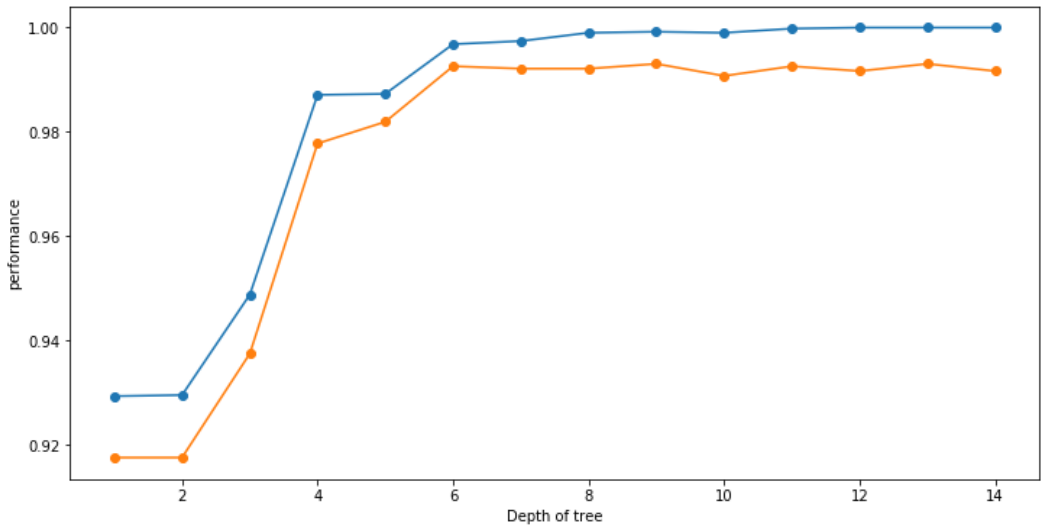


Fig.22: Depth vs. Accuracy for R.Forests

The random forests performed best up until depth level = 13 and then showed lower performance for the next changes without any noticeable increase. Hence, the model is fit until depth=13. The accuracy of the classifier is:

```
# Using score method to get accuracy of model
score = randForest.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  99.30555555555556
```

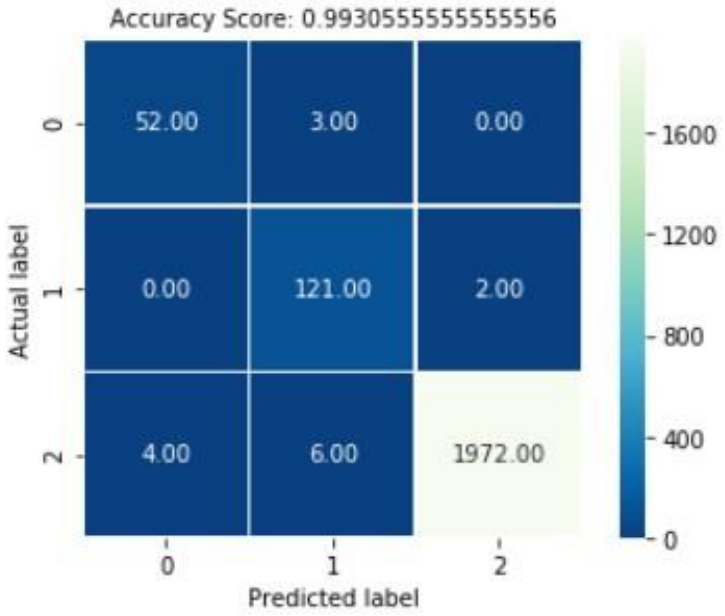


Fig.23: Random Forests Confusion Matrix

To understand what features play a major role in improving the performance of the models, the feature_importances are called to check scores of individual attributes. The scores are sorted in descending order to check the most important features first.

```
#feature importance against each variable

feature_imp = pd.Series(randForest.feature_importances_,index=train_x.columns).sort_values(ascending=False)
feature_imp
```

```
TSH                         0.541012
FTI                         0.140315
TT4                         0.103293
On_thyroxine                0.084909
T3                          0.047890
T4U                         0.036412
Age                         0.023102
Thyroid_surgery             0.005039
Sex                         0.004856
Query_hyperthyroid          0.002864
Query_hypothyroid           0.002272
On_antithyroid_medication   0.001521
Sick                        0.001475
I131_treatment              0.001402
Query_on_thyroxine          0.001201
Psych                       0.000961
Tumor                       0.000819
Lithium                     0.000281
Goitre                      0.000251
Pregnant                    0.000126
Hypopituitary               0.000000
dtype: float64
```

The same visualised in the form of a bargraph is as:

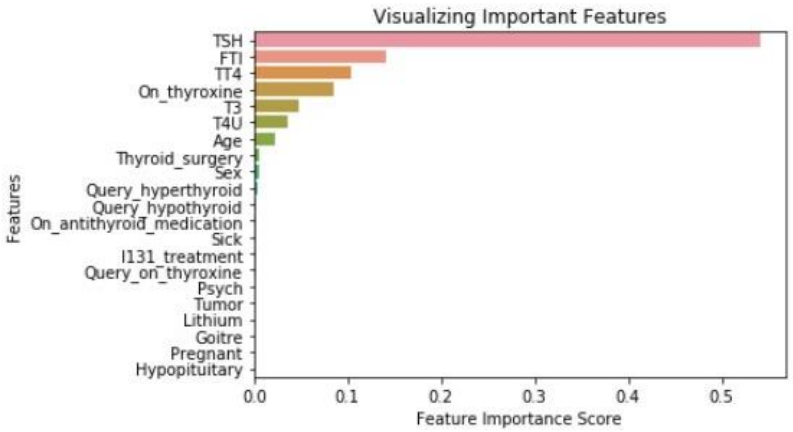


Fig.24: Feature Importances

And the accuracy of the random forest model when selecting only the top 5 most important features is:

```
rf_feat_imp_model = RandomForestClassifier(max_depth=10, random_state=10, n_estimators=10)
rf_feat_imp_model.fit(train_X, train_Y)

# Score
print("Accuracy: ",rf_feat_imp_model.score(test_X, test_Y)*100)
```

```
Accuracy:  99.62962962962963
```

Boosting Classifiers:

Boosting has the capability of turning weak learners into strong ones. In boosting, the resulting models built, depend on the performance of past built models.

During the process of boosting, the machine learning algorithm looks for to find misclassified instances, applies extra weights on each of the misclassified instances and then builds the fresh training data set for new model. During this process the new model built on fresh training data set becomes expert for misclassified instances.

Results of implementing the AdaBoost Classifier is:

```
# Using score method to get accuracy of model
score = abc.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  99.53703703703704
```

Result of implementing the Gradient Boosting Classifier is:

```
# Using score method to get accuracy of model
score = gbc.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  99.53703703703704
```

Ensembles:

Out of many combinations tested for ensembling, the following three ensembles produced the best results out of which ensemble-1 has the best performance.

Ensemble -1:

A consolidation of Decision Trees, Random Forests and Support Vector Machines is used to construct an ensemble classifier. Each individual model is appended to an estimator array which utilizes the Majority Voting Classifier technique.

```
# Using score method to get accuracy of model
score = ensemble_1.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  99.53703703703704
```

Ensemble -2:

A consolidation of Decision Trees, Random Forests and Logistic Regression is used to construct an ensemble classifier. Each individual model is appended to an estimator array which utilizes the Majority Voting Classifier technique.

```
# Using score method to get accuracy of model
score = ensemble_2.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  99.12037037037037
```

Ensemble -3:

A consolidation of Decision Trees and Random is used to construct an ensemble classifier. Each individual model is appended to an estimator array which utilizes the Majority Voting Classifier technique.

```
# Using score method to get accuracy of model
score = ensemble_3.score(test_x, test_y)
print("Accuracy: ", score*100)
```

```
Accuracy:  99.35185185185185
```

The best result obtained was from ensemble-1, a consolidation of decision trees, random forests and support vector machines. Now this model is utilized to make predictions on dynamic inputs related to the patient's details using the "predict()" function.

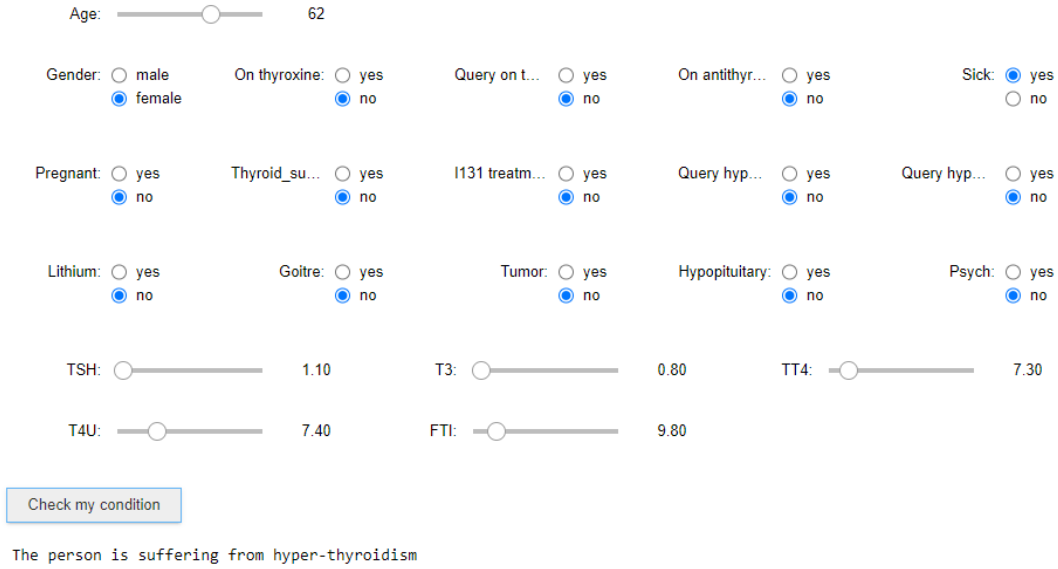
```
input_1 = [[0.73, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0.0006, 0.015, 0.12, 0.082, 0.146]]
ensemble_model = ensemble_2.fit(x, y)
result_1 = ensemble_model.predict(input_1)

print(result_1)
```

```
[3]
```

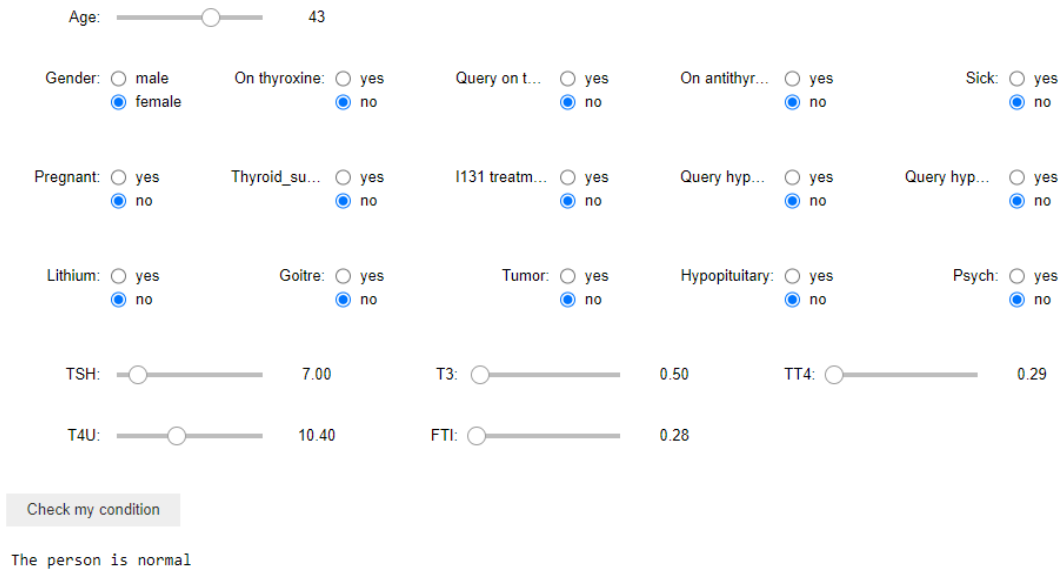
Here, the result "[3]" refers to the class label "Hypo-Thyroid".

The GUI constructed using widgets from the ipywidgets library gives a better mode of interaction for the user to input values and check their condition.
The following snippets show how the GUI reacts to each case scenario i.e. whether the person is Normal, Hyperthyroid or Hypothyroid.

Case-1:


Age: 62
Gender: male female
On thyroxine: yes no
Query on t...: yes no
On antithyr...: yes no
Sick: yes no
Pregnant: yes no
Thyroid_su...: yes no
I131 treatm...: yes no
Query hyp...: yes no
Query hyp...: yes no
Lithium: yes no
Goitre: yes no
Tumor: yes no
Hypopituitary: yes no
Psych: yes no
TSH: 1.10
T3: 0.80
TT4: 7.30
T4U: 7.40
FTI: 9.80
Check my condition
The person is suffering from hyper-thyroidism

Case-2:


Age: 43
Gender: male female
On thyroxine: yes no
Query on t...: yes no
On antithyr...: yes no
Sick: yes no
Pregnant: yes no
Thyroid_su...: yes no
I131 treatm...: yes no
Query hyp...: yes no
Query hyp...: yes no
Lithium: yes no
Goitre: yes no
Tumor: yes no
Hypopituitary: yes no
Psych: yes no
TSH: 7.00
T3: 0.50
TT4: 0.29
T4U: 10.40
FTI: 0.28
Check my condition
The person is normal

Case-3:

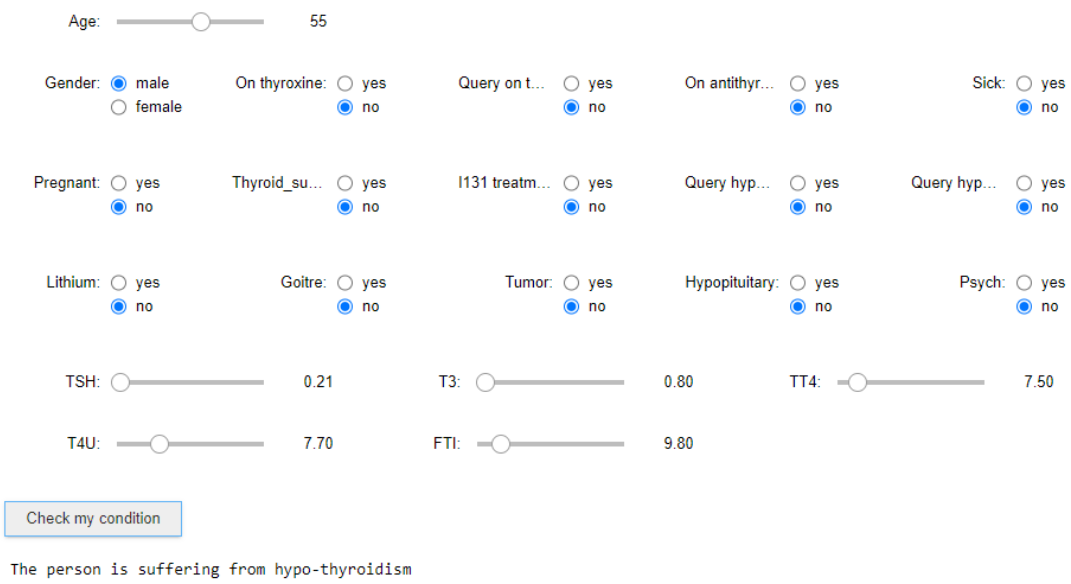


Fig.25: Predicting patient condition

8. CONCLUSION

In our current study, we have deployed supervised 2 tier ensemble method for prediction of thyroid disease. The proposed work is cost effective way of diagnosing thyroid disease among human population. The performance of the ensemble model relies on the features selected using wrapper method. The work combines the top classifiers into ensemble model consisting of three base classifiers with the majority scheme to increase the performance of the base classifiers. The ensemble models built are evaluated on the basis of accuracy, sensitivity, specificity, precision, recall and ROC. The final model obtained is 99.53% accurate in prediction whether a patient is normal or is suffering from hypothyroidism or hyperthyroidism. The intent of our work to be done further is to cater the research of idiosyncratic techniques of machine learning that can be mobilized in the diagnosis of thyroid diseases. There are numerous approachable analyses that are delineated and are being used in the latter years of adequate and competent thyroid disease diagnosis. The analysis shows that different technologies are used in all the papers showing different accuracies. In most research papers it is shown that neural network outperforms over other techniques. On the other hand, this is also given that random forests and decision tree has also performed well.

In the future, the predication techniques can be further improved by adding more instances to the dataset leading to more strong results and also for enhancing this project with different approaches such as implementing this using neural network algorithms like ANN. Furthermore, the model can be improved by applying various features selection algorithms to increase the performance of Thyroid Disease prediction. There is no doubt that researchers worldwide have attained a lot of success to diagnose thyroid diseases, but it is suggested to decrease the number of parameters used by the patients for diagnosis of thyroid diseases. More attributes mean a patient has to undergo a greater number of clinical tests which is both cost effective as well time consuming. Thus, there is a need to develop such type of algorithms and thyroid disease predictive models which require minimum number of parameters of a person to diagnose thyroid disease and saves both money and time of the patient.

REFERENCES

[1] Thyroid disorders in India: An epidemiological perspective. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3169866/#ref1.

[2] Farwell AP http://www.healthywomen.org/condition/thyroid disorders.

[3] Rao, N., Razia, S.: Machine learning techniques for thyroid disease diagnosis. Indian J. Sci. Technol. 9(28) (2016). https://doi.org/10.17485/ijst/2016/v9i28/93705.

[4] Prerana, P.S., Taneja, K.: Predictive data mining for diagnosis of thyroid disease using neural network. Int. J. Res. Manag. Sci. Technol. 3(2), 75–80 (2015). E-ISSN: 2321-3264.

[5] Umadevi, S., JeenMarseline, K.S.: Applying classification algorithms to predict thyroid disease. Int. J. Eng. Sci. Comput. 7(10), 15118–15120 (2017).

[6] Ammulu, K., Venugopal, T.: Thyroid data prediction using data classification algorithm. Int. J. Innov. Res. Sci. Technol. 4(2), 208–212 (2017).

[7] Ahmed, J., Abdul Rehman Soomrani, M.: TDTD: Thyroid Disease Type Diagnostics. In: 2016, International Conference on Intelligent Systems Engineering (ICISE) (2016). https://doi.org/10.1109/INTELSE.2016.7475160.

[8] Mahajan, P., Madhe, S.: Hypo and hyperthyroid disorder detection from thermal images using Bayesian classifier. In: 2014 International Conference on Advances in Communication and Computing Technologies (2014). https://doi.org/10.1109/EIC.2015.7230721.

[9] Saiti, F., Naini, A.A., Tehran, I., Shoorehdeli, M.A., Teshnehlab, M.: Thyroid disease diagnosis based on genetic algorithms using PNN and VM. In: 2009 3rd International Conference on Bioinformatics and Biomedical Engineering (2009). https://doi.org/10.1109/ ICBBE.2009.5163689.

[10] Verma AK, Pal S, Kumar S (2019). Classification of skin disease using ensemble data mining techniques. Asian Pac J Cancer Prev 20(6):1887–1894.

[11] Yadav DC, Pal S (2019) To generate an ensemble model for women thyroid prediction using data mining techniques. Asian Pac J Cancer Prev 20(4):1275–1281.

[12] Ahmad W, Huang L, Ahmad A, Shah F, Iqbal A, Saeed A (2017) Thyroid diseases forecasting using a hybrid decision support system based on ANFIS, k-NN and information gain method. J Appl Environ Biol Sci 7(10):78–85.

[13] Chaurasia V, Pal S, Tiwari BB (2018) Prediction of benign and malignant breast cancer using data mining techniques. J Algorithm Comput Technol 12(2):119–126.
[14] Shankar K, Lakshmanaprabu SK, Gupta D, Maseleno A, de Albuquerque VH (2018) Optimal featurebased multi-kernel SVM approach for thyroid disease classification. J Supercomput. https://doi.org/10.1007/s11227-018-2469-4.
[15] Sumathi A, Nithya G, Meganathan S (2018) Classification of thyroid disease using data mining techniques. Int J Pure Appl Math 119(12):13881–13890.
[16] Ali Z, Shahzad W (2017) Comparative analysis and survey of ant colony optimization based rule miners (IJACSA). Int J Adv Comput Sci Appl 8(1):49–60.
[17] Sudhan M, Kannan M, Sinha D (2017) Hybrid disease diagnosis using multiobjective optimization with evolutionary parameter optimization Hindawi. J Healthc Eng Article ID 5907264.