

**DIABETES PREDICTION AT EARLY STAGE USING  
MACHINE LEARNING**

**BY**

**MD. ZAMAN CHOWDHURY  
ID: 181-15-1788**

**MIRAJUL ISLAM  
ID: 181-15-1720**

**ISRAT JAHAN TULY  
ID: 181-15-1908**

This report is presented in partial compliance with the Qualifications  
Requirements for Computer Science and Engineering.

Supervised By

**Mushfiqur Rahman**  
Senior Lecturer  
Department of CSE  
Daffodil International University

Co-Supervised By

**Md. Reduanul Haque**  
Assistant Professor  
Department of CSE  
Daffodil International University



**DAFFODIL INTERNATIONAL UNIVERSITY**

**DHAKA, BANGLADESH**

**JANUARY 2022**

## **APPROVAL**

The project entitled “**DIABETES PREDICTION AT EARLY STAGE USING MACHINE LEARNING**”, submitted by **Md. Zaman Chowdhury, ID No: 181-15-1788, Mirajul Islam, ID No: 181-15-1720, and Israt Jahan Tuly, ID No: 181-15-1908**, to the Department of Computer Science and Engineering, Daffodil International University, has been accepted as satisfactory for the partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The Presentation has been held on 18.09.2021.

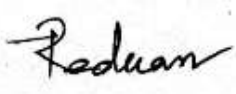
## **BOARD OF EXAMINERS**



---

**Professor Dr. Touhid Bhuiyan**  
**Professor & Head**  
Department of CSE  
Faculty of Science & Information Technology  
Daffodil International University

**Chairman**



---

**Md. Reduanul Haque**  
**Assistant Professor**  
Department of CSE  
Faculty of Science & Information Technology  
Daffodil International University

**Internal Examiner**



---

**Shah Md. Tanvir Siddiquee**  
**Assistant Professor**  
Department of CSE  
Faculty of Science & Information Technology  
Daffodil International University

**Internal Examiner**



---

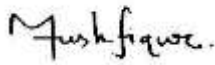
**Dr. Dewan Md. Farid**  
**Professor**  
Department of CSE  
United International University

**External Examiner**

## DECLARATION

We therefore declare that this work has been done by us under the watchful eye of **Mushfiqur Rahman**, a lecturer in the Department of CSE Daffodil International University. We also announce that neither this project nor any part of this project has been relocated to be awarded any degree or diploma.

### Supervised by:



---

**Mushfiqur Rahman**  
Senior Lecturer  
Department of CSE  
Daffodil International University

### Co-Supervised by:



---

**Md. Reduanul Haque**  
Senior Lecturer  
Department of CSE  
Daffodil International University

### Submitted by:



---

**Md. Zaman Chowdhury**  
ID: 181-15-1788  
Department of CSE  
Daffodil International University

Mirajul Islam

---

**Mirajul Islam**

ID: 181-15-1720

Department of CSE

Daffodil International University

Israt Jahan Tuly

---

**Israt Jahan Tuly**

ID: 181-15-1908

Department of CSE

Daffodil International University

## ACKNOWLEDGEMENT

First we express our heartiest thanks and gratefulness to almighty God for His divine blessing makes us possible to complete the final year project/internship successfully.

We really grateful and wish our profound our indebtedness to **Mushfiqur Rahman, Lecturer**, Department of CSE Daffodil International University, Dhaka. Deep Knowledge & keen interest of our supervisor in the field of “Machine Learning” to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior draft and correcting them at all stage have made it possible to complete this project.

We would like to express our heartiest gratitude to Mushfiqur Rahman, Reduanul Haque, and Head, Department of CSE, for his kind help to finish our project and also to other faculty member and the staff of CSE department of Daffodil International University.

We would like to thank our entire course mate in Daffodil International University, who took part in this discuss while completing the course work.

Finally, we must acknowledge with due respect the constant support and patients of our parents.

## **ABSTRACT**

The categorization of medical datasets using machine learning has piqued the interest of the academic community in recent years, despite the fact that it is a difficult undertaking. The use of a large number of machine learning algorithms to a collection of data aids in the completion of the processes. Many studies have been conducted in the past to predict disease using machine learning. However, there are several opportunities for improvement. The goal of this study is to show how pre-processing approaches, as well as traditional and ensemble classifiers, may be used to compare different machine learning-based models for diabetes prediction. The pre-processing procedures for processing the dataset include encoding categorical data, imputing missing values, handling imbalanced data, and scaling features are taken place in this exploration. Five classification techniques, including Support Vector Machine (SVM), Naive Bayes (NB), Logistic Regression (LR), Decision Tree (DT), and Extra Tree (ET), are used to classify the dataset using a 10-fold cross-validation technique, as well as hyperparameter tuning in each classifier to assign the best parameters. Ensemble methods are used to improve the performance of traditional algorithms and prevent them from being over fitted and biased. The experimental study indicates diabetes predictions with a higher degree of accuracy, as well as evaluated the findings of other current studies, with 98.44% accuracy being the best.

# TABLE OF CONTENTS

CONTENTS	PAGE
APPROVAL	i
DECLARATION	ii
ACKNOWLEDGEMENT	iii
ABSTRACT	iv
<b>CHAPTER</b>	
<b>CHAPTER 01: INTRODUCTION</b>	<b>1-3</b>
1.1 Overview	1-2
1.2 Motivation	2
1.3 Research Objectives	3
1.4 Research Questions	3
1.5 Report Layout	3
<b>CHAPTER 02: BACKGROUND</b>	<b>4-5</b>
2.1 Overview	4-5
<b>CHAPTER 03: Research Methodology</b>	<b>6-14</b>
3.1 Overview	6
3.2 Pre- processing	7
3.2.1 Categorical data encoding	7
3.2.2 Missing Value Imputation	7
3.2.3 Handling Imbalanced Data	7



3.2.4 Feature Scaling	7
3.3 Classification Algorithm	7
3.3.1 Logistic Regression (LR)	7
3.3.2 Support Vector Machine (SVM)	8
3.3.3 Naive Bayes (NB)	8-9
3.3.4 Decision Tree (DT)	10
3.3.5 Extra Tree (ET)	10
3.3.6 Bagging	11
3.3.7 Boosting	11
3.4 Research Subject and Instrumentation	12
3.5 Data Description	12
3.6 Performance Evaluation Measure	14
<b>CHAPTER 04: Experimental Results &amp; Discussion</b>	<b>15-21</b>
4.1 Traditional Classification	15
4.2 ROC Curve (Traditional)	22-24
4.3 Bagging Classifiers and ROC Curve	25-33
4.4 Boosting Classifiers and ROC Curve	33-40
<b>CHAPTER 05: Conclusion &amp; Future Work</b>	<b>41</b>
5.1 Conclusion	41
5.2 Future Work	41
5.3 Challenges	41
<b>REFERENCES</b>	<b>42-43</b>

## LIST OF FIGURES

<b>FIGURES NAME</b>	<b>PAGE NO</b>
Figure 1: Proposed Model	6
Figure 2: Naive Bayes	9
Figure 3: Decision Tree	10
Figure 4: Extra tree	10
Figure 5: Bagging and Boosting	11
Figure 6: Experimental Results of Traditional Classifiers	21
Figure 7: ROC Curve Analysis of Traditional Classifiers	24
Figure 8: Experimental Results of Bagging Classifiers	29
Figure 9: ROC Curve Analysis of Bagging Classifiers	32
Figure 10: Experimental Results of Boosting Classifiers	38
Figure 11: ROC Curve Analysis of Boosting Classifiers	40

## LIST OF TABLES

FIGURES NAME	PAGE NO
Table 1. Details of datasets	13

# CHAPTER 1

## Introduction

### 1.1 Overview

Since the disease is a part of life and there are many chronic diseases present, it is very important to identify the disease's initial status. Machine learning can play a significant aspect in predicting the status of disease from responsive healthcare datasets by analyzing several features and patient lab records as well as determine the severity of the disease. The machine learning techniques can be divided into two categories; supervised and unsupervised. <sup>[1]</sup> Supervised Learning infers a function from labeled training data and maps an input to an output based on example input-output pairs whereas Unsupervised Learning deals with the unlabeled data and allows the model to act with its patterns and information that was previously undetected.

In this exploration, we use diabetes dataset for prognosis. Insulin, a hormone produced by the pancreas, is known to aid in the conversion of glucose (also known as blood sugar) to energy. Diabetes develops when Insulin fails to function correctly, causing the blood glucose level to rise too high. The etiology of diabetes is unclear; however, genetics may play a role. Humans with diabetes have frequent urination and thirst as well as weight loss, blurry vision, tiredness, etc.

Many researchers have collaborated on an effort to use machine learning to forecast the state of diabetes. They haven't been able to achieve perfection in all of the key areas, though. Some of them haven't paid enough attention to the dataset's pre-processing, such as handling unbalanced data and feature scaling. Some researchers have reported decreased accuracy and failure to use ensemble methods to improve the result. Some researchers haven't tried to manually assign the optimum parameters via hyperparameter tuning, and they haven't utilized cross-validation to minimize bias or overfitting. In this article, we have taken a careful look at these processes and conducted our research using the following sentences.

- We have pre-processed the dataset by different steps such as, encoding categorical data, handling imbalanced data, feature scaling.
- We have comprised a comparative analysis among the algorithms with three different feature scaling methods.
- We have predicted the disease status using traditional machine learning algorithms (LR, SVM, NB, DT, ET) with better accuracy.

- We have used 10-fold cross validation to ensure the result as well as remove biased problems.
- We have added hyperparameter tuning to assign the best parameters so that we can achieve better accuracy.
- We have used the ensemble learning classifiers (Bagging and Boosting) to improve as well as ensure real performances.
- We have determined the results of many evaluation measures to compare with other's performances.

We have embellished the rest of the paper by adding six more sections. We have referred to the literature review in Section 2. In Section 3, the dataset description has been mentioned. We have briefed about our experimental setup in section 4. In Section 5, we have included the methodology. We have discussed the results of our experiment in Section 6. We finally conclude our research by Section 7.

## **1.2 Motivation**

Wherever in this world diabetes is the most common chronic diseases and day by day expanding in the number of people affected by diabetes. The primary cause is changing life style whichever lessen physical activity which can force to obesity. Key facts. Diabetes attack rose from 108 million in 1980 to 422 million in 2014. It's a vital cause of blindness kidney failure stroke and heart attack.in 2019 by who estimated 1.5 million deaths for only diabetes. Most of them are know when they are 1<sup>st</sup> affected by this disease. There are a Little numbers of research paper about early prediction. So we do about lots of study about diabetes prediction. Most of them are not achieve higher accuracy. So for achieve best result we are motivated and finally found the best accuracy most of other than in our using Method.

### **1.3 Research Objectives**

- To find out the diabetic result.
- To analyze the correlated problem
- Quality of life.
- To spread the awareness to the people about diabetes

### **1.4 Research Questions**

- What is the key features of this database?
- How the algorithm works on this research?
- How you can predict the early detection of diabetes?
- What will be the success rate if a person will have diabetes or not?

### **1.5 Report Layout**

- Background
- Research Methodology
- Experimental Result and Discussion
- Review, Conclusion and Future Analysis
- Reference

## **CHAPTER 2**

### **Background Study**

#### **2.1 Overview**

Huge studies on diabetes prediction have recently risen to the top of researchers' priority lists in the medical research sector. M. M. F. Islam et al.<sup>[2]</sup> used 10-fold cross-validation and percentage split assessment approaches to evaluate the dataset using several Machine learning algorithms such as NB, LR, and RF. They discovered that RF by cross-validation had the greatest accuracy of 97.40% and percentage split evaluation had the best accuracy of 99%. They did not do hyperparameter tuning, which might be considered a limitation.

T. M. Le et al. proposed a novel wrapper-based feature selection technique based on Grey Wolf Optimization (GWO) and Adaptive Particle Swarm Optimization (APSO), and compared the results of this method with several conventional machine learning algorithms, achieving 96% for GWO-MLP and 97% accuracy for APGWO-MLP.<sup>[3]</sup>

M. T. Ogedengbe and C. O. Egbunu<sup>[4]</sup> also presented the CSE-DT feature selection approach, which achieved an accuracy of 81.64%.

Abdul et al. performed a classification utilizing the DT with Linear Sampling and Information Gain optimization and achieved a 99.04%. Pro.<sup>[5]</sup>

Sakshi et al.<sup>[6]</sup> got from his examination he achieved best result in SVM & Ad Boost algorithm which is around 94.44%, he obtained this result in two categories without Bootstrapping & After Bootstrapping.

Hang et al<sup>[7]</sup> use logistic regression and gradient boosting machine for prediction and achieved the result in proposed GBM model is 84.7% with sensivity of 71.6%. In Logistic regression accuracy is 84.0% with sensivity73.4%.<sup>[7]</sup>

Tafa et al work for implementation with Naïve Bayes and SVM arithmetical modeling, from the given dataset which is collected from the medical examinations of 402 patients, in order to more reliability the computer-supported diagnosis reliability and got the classifier performance in SVM 95.52% and in Naïve Bayes 94.52% respectively. After this they use combined execution of two algorithms and achieve the average rate of the valid outcomes which is 94.77% with the projected lower bound of nearly 90.3%.<sup>[8]</sup>

A Negi et al in this paper they work for obtain the best accuracy different steps were took which are subsist of data collection, preprocessing, normalization, model validation, model generation, feature selection(It was done using Fselect script from LIBSVM package and using weka. This F-select method is use for feature selection for F-score and it is the best advance method in prospective of SVM Algorithm). For Feature selection they use WEKA and LIBSVM in WEKA wrapper and ranker methods. And finally obtain best accuracy 71% in wrapper method and obtain best accuracy 72% using Ranker method selected features. <sup>[9]</sup>

TM Alam et al try to achieve best accuracy using Clustering method, Random Forest(RF) and Artificial Neural Network(ANN). K-means clustering method has given the result 73.6% accuracy The random forest method afford an accuracy of 74.7%, and ANN gave the accuracy 75.7%. So here ANN gives the best accuracy 75.7%. It is a nonlinear model which is straight forward and generally use for relating statistical methods. This is a non-parametric model, while the majority of statistical techniques are parametric and require a higher foundation of statistics. The main advantage of handling ANN over other statistical techniques is its capacity to catch the non-linear relationship among the concerned variables. <sup>[10]</sup>

G Tripathi works with four machine learning algorithm these are Linear Discriminant Analysis (LDA), K-nearest neighbor (KNN), Support Vector Machine (SVM), and Random Forest (RF) for predictive analysis of early-stage diabetes. Here with accuracy found precision, recall, Specificity and f-score. Using this Four algorithm here The experimental outcomes show that Random Forest (RF) gives the maximum best accuracy of 87.66 %. Others are SVM achieved 80.85%, KNN 79.24% and LDA achieve 76.86%. <sup>[11]</sup>



## CHAPTER 3

### Research Methodology

#### 3.1 Overview

In this study, the relevant information is collected from the UCI machine learning repository. The data collected from a survey carried out Special studies addressing topical issues from Sylhet diabetes center, our research methodology. The present method consists of the different steps which are data collection, pre-processing. In this present study we use different 4 algorithms Linear Regression (LR), Naïve Byes (NB), Support Vector Machine (SVM) and Extra Tree (ET). Where the best accuracy is in SVM 97.81%. But after in this study we use the 10-fold cross validation, hyper parameter tuning. Then bagging and boosting method is used and we got the best High accuracy in Decision Tree (DE) 98.44%.

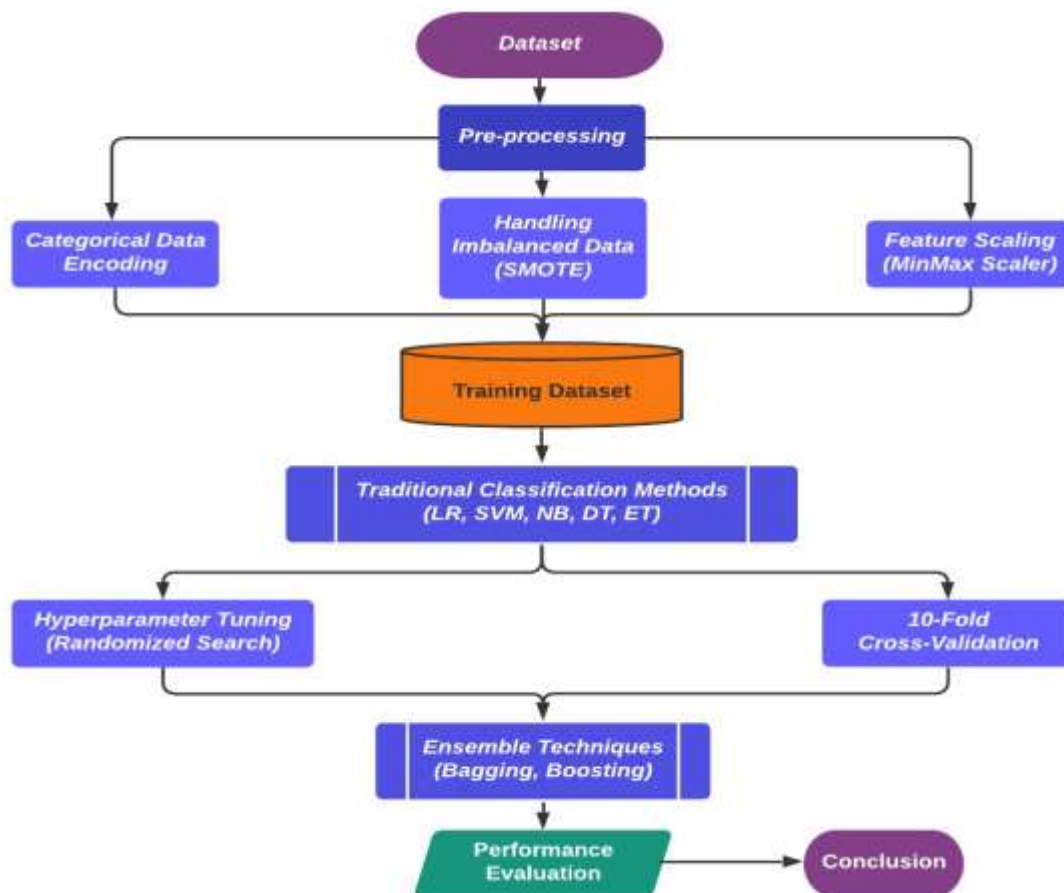


Fig: Proposed Model

## 3.2 Pre-processing

**3.2.1 Categorical data encoding:** Categorical data encoding is the process of transforming the categorical data to numerical value. We know that, in machine learning models, the input and output variables have to be numeric i.e., if the dataset contains categorical data, it is compulsory to encode it to numbers before fitting into a model. Our Diabetes dataset consists of both categorical and numerical data where all the columns contain nominal data except age. So, we have encoded the dataset.

**3.2.2 Missing Value Imputation:** Missing value imputation is the process of replacing missing data with substituted values imputed by a study concerning other data of the dataset. Quite fortunately our datasets have no missing value.

**3.2.3 Handling Imbalanced Data:** Handling imbalanced data refers to adjust the class distribution of a data set. We have used smote to handle imbalanced data which full form is Synthetic Minority Oversampling Technique. It handles data by increasing the number of cases in the dataset in a balanced way. It increases the data of minority while taking the full dataset as input.

**3.2.4 Feature Scaling:** FS (Feature scaling) is used to normalize the range of independent features of data. We have used MinMax scaler that scales all the data features in the range [0, 1] if there are no negative values or [-1, 1] for else.

## 3.3 Classification Algorithm

In this research, Machine Learning (ML) based classifiers like as, Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT), Naive Bayes (NB), and Extra Tree (ET) have been used to classify the datasets.

**3.3.1 Logistic Regression (LR):** It's a Machine Learning (ML) based algorithm where the class label has two categories, yes/no like binary (0/1). Logistic regression presume that the seer isn't enough to resolved the response variable, although resolve a possibility that's a logistic operation of a linear consolidation of them. LR is great in a small number of dimensions at the seer don't satisfy to deliver extra probabilistic evaluate of the feedback.

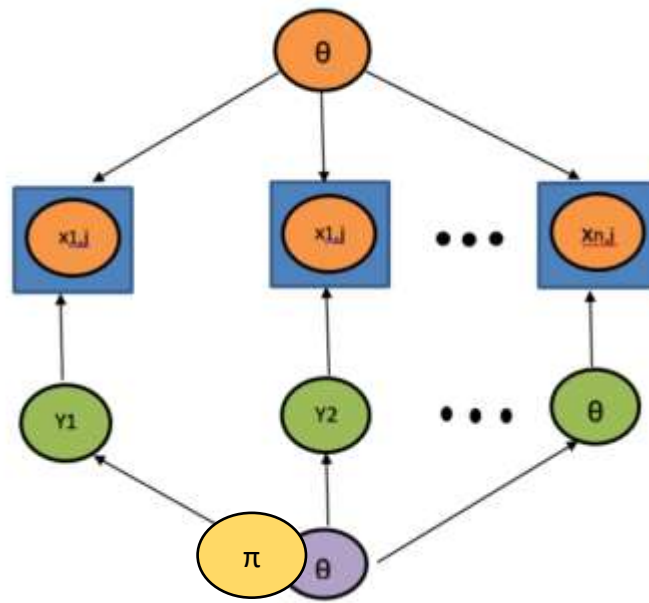
**3.3.2 Support Vector Machine (SVM):** SVM is a supervised machine learning model which is applicable as both linear and non-linear data. It constructs an N-dimensional hyperplane and transforms the original data to there as well as separates the data into two categories optimally which can be used for classification, regression, or other tasks. <sup>[12]</sup>

In many functional applications, SVM can ensure higher accuracy for a long-time period prediction as compared to different computational approaches.

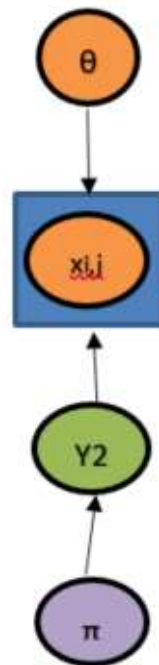
SVM take the phenomenal point that help in formulating the hyper plane. These flagrant cases are named uphold vectors, also hence the computation is termed as Support Vector Machine. Acknowledge the covered chart in where there are two particular classifications that are organized handling a preferred hyper plane.

**3.3.3 Naive Bayes (NB):** NB is a supervised Machine Learning (ML) classifier. It's created placed on the Bayes theorem. This classifier deals with the two classes perception with nominal features. It assigns class 0 to only one example and class 1 to the other examples, with probability 1. NB can handle both discrete and continuous variables. It can be applied to a few training data. <sup>[13]</sup>

$$P(A|B) = \text{probability of A given that B is true} = \frac{P(B|A) * P(A)}{P(B)} \quad (1)$$



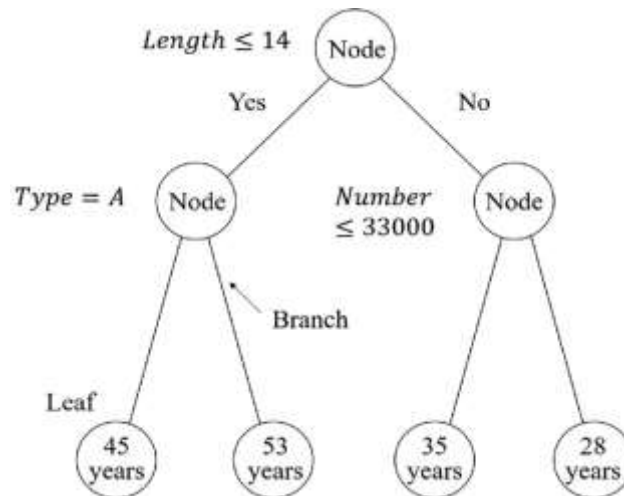
Naive Bayes (train)



Naive Bayes (train)

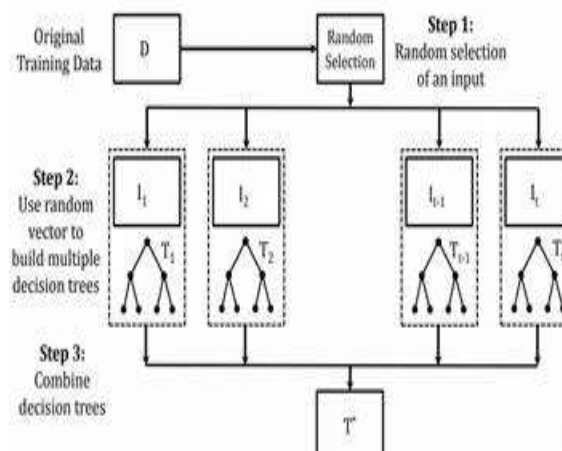
**Fig:** Naive Bayes

**3.3.4 Decision Tree (DT):** It is a supervised Machine Learning (ML) classifier. In this model, a structure is formed like a tree where all root and an internal node represents features and the leaf node represents class label in the form of True or False. It's completely a different approach to classify. It generally creates a series of if-then rules which assign an observation to one segment of the tree. It has more than two possible results and with ordinal outcome variables.



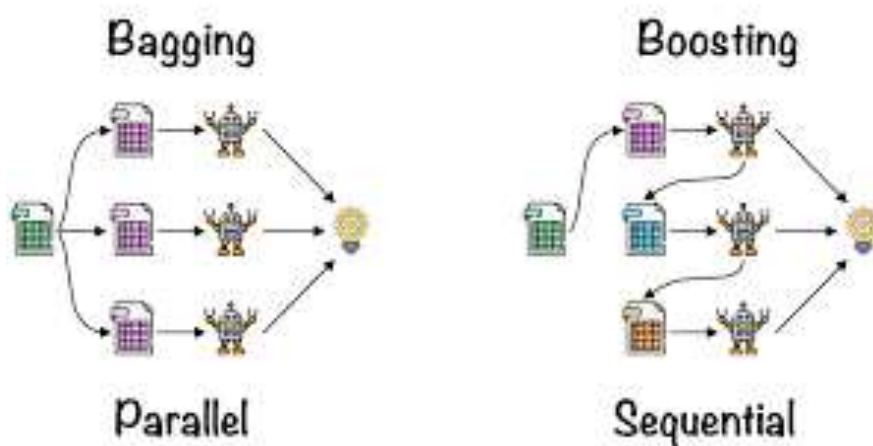
**Fig:** Decision Tree

**3.3.5 Extra Tree (ET):** Extra tree classifier combine many decors related decision trees results amassed in the forest to propagate the classifier result as well as handles a vast number of numerical features effectively.



**Fig:** Extra tree

**3.3.6 Bagging:** This bootstrap aggregating method generates some random bootstrap samples from the whole dataset. As a base estimator, a model is used to find out the evaluation report and simple voting is applied to obtain the overall evaluation report. Bagging is used generally when you desire to cut down the difference of opinion while having the bias. This arise when you equate predictions in other areas of the given feature space. Bagging is useful as long as you are developing the accuracy of a particular model by adopting numerous types of it skilled on numerous sets of data. Bagging is not suggested on models that have high bias. In alike cases Adaboost is used that drives a step forward and eradicates the achieve of a high bias produce in the criterion model.



**Fig:** Bagging and Boosting

**3.3.7 Boosting:** Boosting has different base learners and all the instances are misclassified by their previous base learners. We have chosen comparatively more applicable the Adaboost algorithm. The way of a boosting model generate predictions on current recent data is actual transparent. When we gain a new inspection with own its features, this moved over each one of the particular models, including all model create its own prediction.

### **3.4 Research Subject and Instrumentation**

All the experiments are implemented in Python language inside the platform Google Collaborator. It permits to write and perform arbitrary python code through the browser. Colab is a hosted Jupyter notebook without asking any setup. All the experiments were run into the system having the configuration of Intel(R) Core(TM) i5-8250U CPU @ 2.70GHz, the 64-bit operating system of Windows 10 pro with 8GB RAM.

### **3.5 Dataset Description**

For the operations take place in the analysis, one of the most acceptable and publicly available the Early stage diabetes risk prediction dataset is used retrieved from UCI Machine Learning Repository. [14] The dataset that was originated from Sylhet Diabetes Hospital, Sylhet, Bangladesh. This dataset holds 520 samples with 16 attributes. Patients from age 20 to 65 are present here that are separated by 2 conditions where the positive class indicates the diabetic patients whereas the negative class denotes non-diabetic patients. Specifically, 320 instances apprehend Diabetes whereas 200 are not. The dataset contains only nominal data. Fortunately, there exist no missing values in the dataset. The comprehensive explanation of the dataset is displayed in table 1.

Table No: 1. Details of datasets

No. of Attributes = 16	Total no. of Instances = 520	No. of Class Values = 2	No. of Missing Values = 0
<b>Description of Attributes</b>			
<b>Attributes</b>	<b>Description</b>	<b>Types of values</b>	
Age	Age of the person (in years)	Numerical [20,65]	
Gender	Gender of the person (male, female)	Nominal	
Polyuria	Yes, No	Nominal	
Polydipsia	Yes, No	Nominal	
Sudden weight loss	Yes, No	Nominal	
Weakness	Yes, No	Nominal	
Polyphagia	Yes, No	Nominal	
Genital thrush	Yes, No	Nominal	
Visual blurring	Yes, No	Nominal	
Itching	Yes, No	Nominal	
Irritability	Yes, No	Nominal	
Delayed healing	Yes, No	Nominal	
Partial paresis	Yes, No	Nominal	
Muscle stiffness	Yes, No	Nominal	
Alopecia	Yes, No	Nominal	
Obesity	Yes, No	Nominal	
Class	Negative = no disease,  Positive = disease	Nominal	



### 3.5 Performance Evaluation Measure

There are some performance evaluation measures method by these methods we can able to check the exact performance of our existing model. These methods estimate the overall performance which are perform on the unseen data.

**Accuracy:** It means the percentage of accurate prediction for the test data. Where accuracy performs the accessibility of the measurements with the real measurements. It based on a single factor. Accuracy deals with systematic errors.

**Precision:** It means the fraction of correctly predicted the positive observations. Precision actually identifies the real true fragment of the total instances where if they predicted true.

**Recall:** It's the fraction of correctly predicted the positive observations.

**F1 Score:** It's basically the harmonic mean of the precision and recall.

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

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

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

$$\text{F1-Score} = \frac{2 * P * R}{P+R}$$

**Fig. 4.** Performance evaluation measures formula

## CHAPTER 4

### Experimental Results & Discussion

#### 4.1 Traditional Classification

Here in this segment, we have to conducted on a comparative analysis on basis of the experimental results of the evaluation measures of the machine learning models of diabetes dataset. We have pre-processed our dataset to analyze the performances of traditional and ensemble classifiers which had gone through 10-fold cross validation and hyper parameter tuning. Consequently, we have observed different performances using traditional, bagging and boosting classifiers.

##### 4.1.1 Logistic Regression

```
# Importing Libraries
from sklearn.linear_model import LogisticRegression
LRclassifier = LogisticRegression()
```

```
#Randomized Search
from sklearn.model_selection import RandomizedSearchCV
params = [{
    'penalty':['l1', 'l2', 'elasticnet'],
    'C':np.linspace(0,10,50),
    'solver' : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    # 'max_iter':[100,1000,2500,5000,10000,20000]
    'max_iter':[i for i in range(1,100000,1)]
}]
grid = RandomizedSearchCV(LRclassifier,params,scoring='accuracy',cv=10, verbose=True,n_jobs=-1)
grid = grid.fit(X, y)
best_accuracy = grid.best_score_
best_params = grid.best_params_
print("Best-score on Grid : ",best_accuracy)
print("Best-parameters on Grid : ",best_params)
```

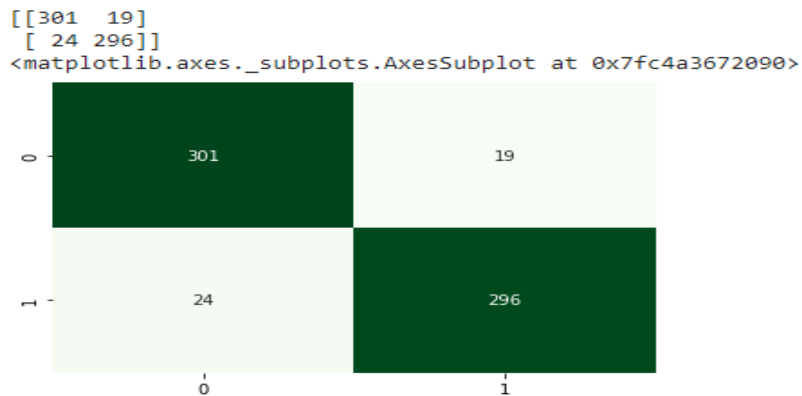
Fitting 10 folds for each of 10 candidates, totalling 100 fits  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.  
Best-score on Grid : 0.9328125  
Best-parameters on Grid : {'solver': 'liblinear', 'penalty': 'l2', 'max iter': 291627, 'C': 2.857142857142857}

```
[ ] #fitting LR to the training set
LRclassifier = LogisticRegression(solver=best_params['solver'], penalty=best_params['penalty'], max_iter=best_params['max_
LRclassifier.fit(X,y)
```

```
LogisticRegression(C=2.857142857142857, class_weight=None, dual=False,
                    fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                    max_iter=291627, multi_class='auto', n_jobs=None,
                    penalty='l2', random_state=None, solver='liblinear',
                    tol=0.0001, verbose=0, warm_start=False)
```

```
[ ] # Predicting the result
from sklearn.model_selection import cross_val_predict
LR_pred=cross_val_predict(LRclassifier,X,y,cv=10)
```

```
[ ] # Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, LR_pred)
print(cm)
sns.heatmap(cm, annot=True, fmt="d", cmap='Greens', cbar=False)
```



```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(LRclassifier, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))
```

```
CV accuracy: 0.9328125
CV precision: 0.9459080743734443
CV recall: 0.925
CV F1 score: 0.9327918961730436
```

First of all we import libraries of algorithm. Then we use hyper parameter tuning for get best parameter using randomized search algorithm. Then we train data using this parameter. Here for more accuracy predicting the result we use 10-fold cross validation method and make the confusion matrices and got the results.

### 4.1.2 SVM

- ▼ SVM

```
[ ] # Importing Libraries
    from sklearn.svm import SVC
    SVMclassifier = SVC()

[ ] #Randomized Search
    from sklearn.model_selection import RandomizedSearchCV
    params = [
        {'C':np.linspace(0,10,500),'kernel':['linear']},
        {'C':np.linspace(0,10,500),'kernel':['rbf'],'gamma':[0.5,0.100,0.101,0.102,0.103,0.104]}
    ]

    grid = RandomizedSearchCV(SVMclassifier,params,scoring='accuracy',cv=10, verbose=True,n_jobs=-1)
    grid = grid.fit(X, y)
    best_accuracy = grid.best_score_
    best_params = grid.best_params_
    print("Best-score on Grid : ",best_accuracy)
    print("Best-score on Grid : ",best_params)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=1)]: Using backend LokyBackend with 2 concurrent workers.
Best-score on Grid : 0.975
Best-score on Grid : {'kernel': 'rbf', 'gamma': 0.102, 'C': 3.947895791583166}
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed: 0.9s finished
```

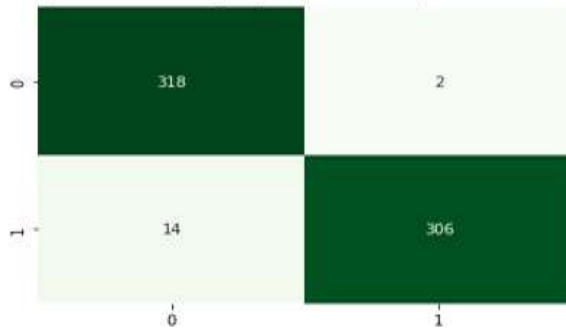
```
[ ] #fitting SVM to the training set
SVMclassifier = SVC(C=best_params['C'], kernel=best_params['kernel'], random_state=0, probability=True)
SVMclassifier.fit(X,y)
```

```
SVC(C=3.947895791583166, break_ties=False, cache_size=200, class_weight=None,
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='rbf', max_iter=-1, probability=True, random_state=0, shrinking=True,
    tol=0.001, verbose=False)
```

```
[ ] # Predicting the result
from sklearn.model_selection import cross_val_predict
SVM_pred=cross_val_predict(SVMclassifier,X,y,cv=10)
```

```
[ ] # Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, SVM_pred)
print(cm)
sns.heatmap(cm, annot=True, fmt="d", cmap='Greens', cbar=False)
```

```
[[318  2]
 [ 14 306]]
<matplotlib.axes._subplots.AxesSubplot at 0x7fc49f5e8450>
```



```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(SVMclassifier, X, y, cv=10, scoring=scoring)
```

```
print('CV accuracy:', np.mean(results['test_accuracy']))
print('CV precision:', np.mean(results['test_precision']))
print('CV recall:', np.mean(results['test_recall']))
print('CV F1 score:', np.mean(results['test_f1_score']))
```

```
CV accuracy: 0.975
CV precision: 0.9932183908045978
CV recall: 0.95625
CV F1 score: 0.974189351397178
```

Also here we do the same process what we done in previous Algorithm. Here the main difference is we get different parameter than others. Then set it in training set and finally got this results.

### 4.1.3 Naïve Bayes

In Naïve Bayes we done the same process. That's why we just presenting the results in all algorithms.

```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(NBclassifier, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))
```

```
CV accuracy: 0.865625
CV precision: 0.959925925925926
CV recall: 0.765625
CV F1 score: 0.8487444369536364
```

Here the highest accuracy is 86%

### 4.1.4 Decision Tree

```
▶ # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(DTclassifier, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))
```

```
CV accuracy: 0.9671875
CV precision: 0.9870346320346322
CV recall: 0.946875
CV F1 score: 0.9659760552039771
```

Here also same things happening and achieve the highest accuracy is 96%.

### 4.1.5 Extra Tree

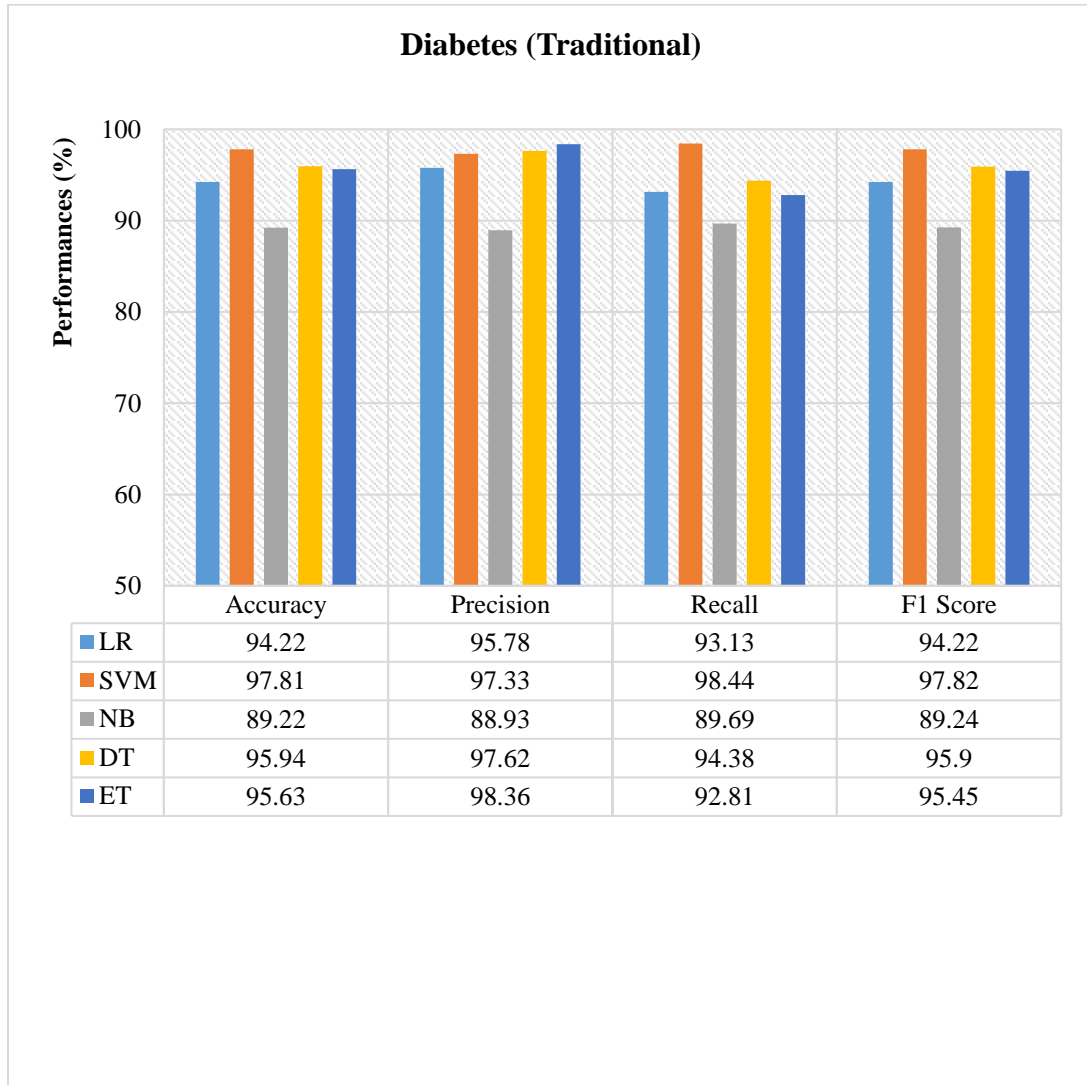
```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate, cross_val_predict
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(ETclassifier, X, y, cv=10, scoring=scoring)

print('CV accuracy:', np.mean(results['test_accuracy']))
print('CV precision:', np.mean(results['test_precision']))
print('CV recall:', np.mean(results['test_recall']))
print('CV F1 score:', np.mean(results['test_f1_score']))
```

CV accuracy: 0.959375  
CV precision: 0.993095238095238  
CV recall: 0.925  
CV F1 score: 0.9575807710709878

Here we obtain 95% highest accuracy.

#### 4.1.6 Traditional Classifiers analysis



**Fig. 3.** Experimental Results of Traditional Classifiers

First of all, considering the performances of traditional classifiers, the best accuracy has been obtained 97.81% using SVM classifier. Considering the values of precision, the best score of 98.36% has been obtained by applying ET. Looking at the recall section, again SVM has provided the best score of 98.44%. In the case of the f1-score, SVM classifier has performed the best of 97.82% as usual.



## 4.2 ROC Curve (Traditional)

In AUC ROC curve use for true positive rate and false positive rate where the range is 0 to 1. 1 is the highest performance & 0 is lowest. Here we use this formula.


### 4.2.1 Linear Regression ROC Result

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve


    LR_probs_y = LRclassifier.predict_proba(X)
    LR_auc_y = roc_auc_score(y, LR_probs_y[:,1])
    print(LR_auc_y)

0.984931640625
```


### 4.2.2 SVM ROC Result

```
 # AUC
    from sklearn.metrics import roc_auc_score, roc_curve


    SVM_probs_y = SVMclassifier.predict_proba(X)
    SVM_auc_y = roc_auc_score(y, SVM_probs_y[:,1])
    print(SVM_auc_y)
```

 1.0

### 4.2.3 Naïve Bayes ROC Result

```
 # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    NB_probs_y = NBclassifier.predict_proba(X)
    NB_auc_y = roc_auc_score(y, NB_probs_y[:,1])
    print(NB_auc_y)
```

 0.956064453125

### 4.2.4 Decision ROC Result

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    DT_probs_y = DTclassifier.predict_proba(X)
    DT_auc_y = roc_auc_score(y, DT_probs_y[:,1])
    print(DT_auc_y)

0.9989208984375
```

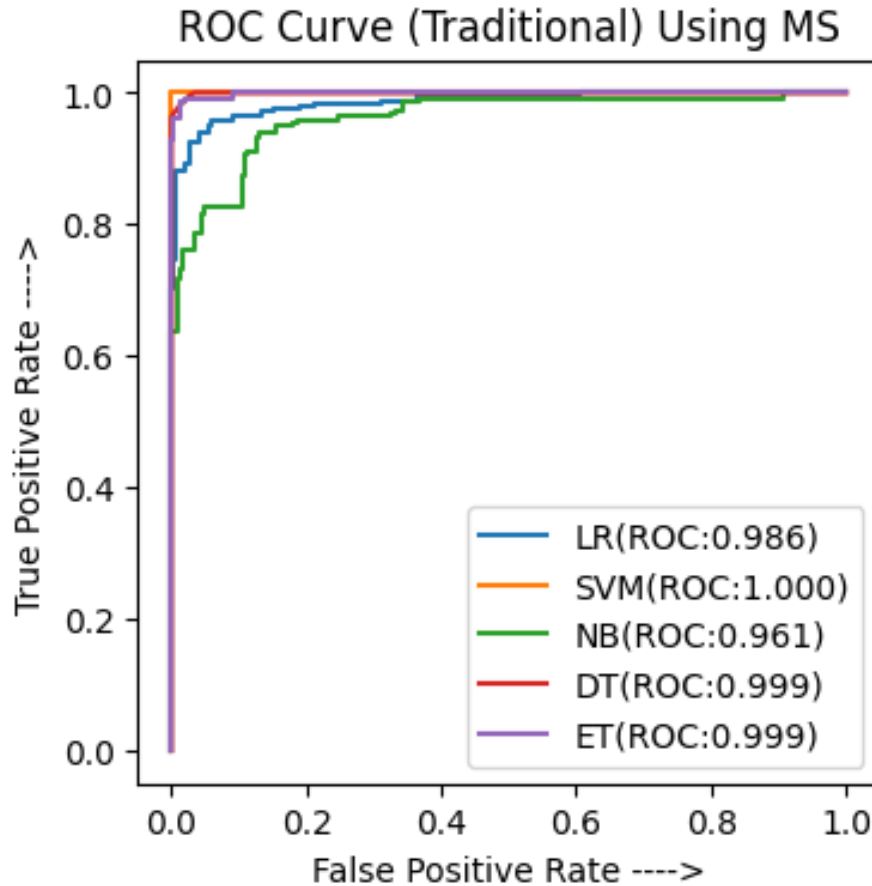
#### 4.2.5 Decision Tree ROC Result

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    ET_probs_y = ETclassifier.predict_proba(X)
    ET_auc_y = roc_auc_score(y, ET_probs_y[:,1])
    print(ET_auc_y)

0.9979589843750001
```

#### 4.2.6 ROC Curve Analysis of Traditional Classifiers



**Fig. 4.** ROC Curve Analysis of Traditional Classifiers

ROC Curve, depicted in Fig 4, has also drawn the same scenario where SVM has acquired the best score of 1. Hence, according to the above analysis as well as the detailed results with graphical representation depicted in Fig 3, the SVM classifier can be stamped as the best traditional classifier.

## 4.3 Bagging Classifiers and ROC Curve

Here in all Classifier we use traditional methods output as input in this Bagging Methods Classifiers.

### 4.3.1 Linear Regression (Bagging)

```

Bagging

Logistic Regression

[ ] # Importing Libraries
from sklearn.ensemble import BaggingClassifier
LR_BG = BaggingClassifier(LRClassifier)

[ ] # Randomized Search
from sklearn.model_selection import RandomizedSearchCV
params=[{
    'n_estimators':np.arange(1,101),
    'bootstrap':['True','False'],
    'bootstrap_features':['True','False']
}]
rand = RandomizedSearchCV(LR_BG,params,scoring='accuracy',cv=10,verbose=True,n_jobs=-1)
rand = rand.fit(X,y)
best_accuracy = rand.best_score_
best_params = rand.best_params_
print("Best score on Randomized Search: ", best_accuracy)
print("Best Parameters on Randomized Search: ", best_params)

Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 46 tasks | elapsed: 8.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 14.9s finished
Best score on Randomized Search: 0.9390625
Best Parameters on Randomized Search: {'n_estimators': 61, 'bootstrap_features': 'False', 'bootstrap': 'False'}

[ ] #fitting LR to the training set
LR_BG = BaggingClassifier(LRClassifier,n_estimators=best_params['n_estimators'],bootstrap=best_params['bootstrap'],bootstrap_features=best_params['bootstrap_features'])
LR_BG.fit(X,y)
```

```

BaggingClassifier(base_estimator=LogisticRegression(C=2.857142857142857,
                                                    class_weight=None,
                                                    dual=False,
                                                    fit_intercept=True,
                                                    intercept_scaling=1,
                                                    l1_ratio=None,
                                                    max_iter=291627,
                                                    multi_class='auto',
                                                    n_jobs=None, penalty='l2',
                                                    random_state=None,
                                                    solver='liblinear',
                                                    tol=0.0001, verbose=0,
                                                    warm_start=False),
                bootstrap='False', bootstrap_features='False',
                max_features=1.0, max_samples=1.0, n_estimators=61,
                n_jobs=None, oob_score=False, random_state=None, verbose=0,
                warm_start=False)

```

```

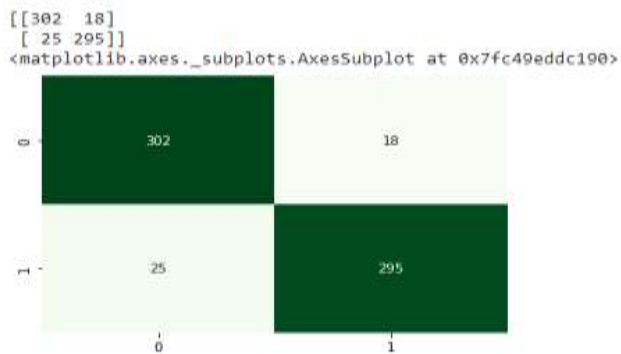
[ ] # Predicting the result
from sklearn.model_selection import cross_val_predict
LR_BG_pred = cross_val_predict(LR_BG,X,y,cv=10)

```

```

[ ] # Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, LR_BG_pred)
print(cm)
sns.heatmap(cm, annot=True, fmt="d", cmap='Greens', cbar=False)

```



```

[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(LR_BG, X, y, cv=10, scoring=scoring)

print('CV accuracy:', np.mean(results['test_accuracy']))
print('CV precision:', np.mean(results['test_precision']))
print('CV recall:', np.mean(results['test_recall']))
print('CV F1 score:', np.mean(results['test_f1_score']))

CV accuracy: 0.934375
CV precision: 0.9614101363093297
CV recall: 0.909375
CV F1 score: 0.93309379791878

```

As we said we use our traditional LR algorithm's output as input in bagging method and other Process are same what we have done in traditional algorithms.

### 4.3.2 SVM (Bagging)

We use here SVM's output as input and other process is same as like before. That's why we just presenting the results. Where we obtain the highest accuracy is 97%.

```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(SVM_BG, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))

CV accuracy: 0.9734375
CV precision: 0.986989247311828
CV recall: 0.959375
CV F1 score: 0.9729134664618535
```

### 4.3.3 Naïve Bayes (Bagging)

Same process also applied here and obtain the final result accuracy is 86%.

```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(NB_BG, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))

CV accuracy: 0.8625
CV precision: 0.9628218390804598
CV recall: 0.75625
CV F1 score: 0.8438587769867784
```

### 4.3.4 Decision Tree (Bagging)

Same things also happened in this process. Final achievement accuracy is 98.12%

```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(DT_BG, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))

CV accuracy: 0.98125
CV precision: 0.987460815047022
CV recall: 0.975
CV F1 score: 0.9808875277727737
```

### 4.3.5 Extra Tree (Bagging)

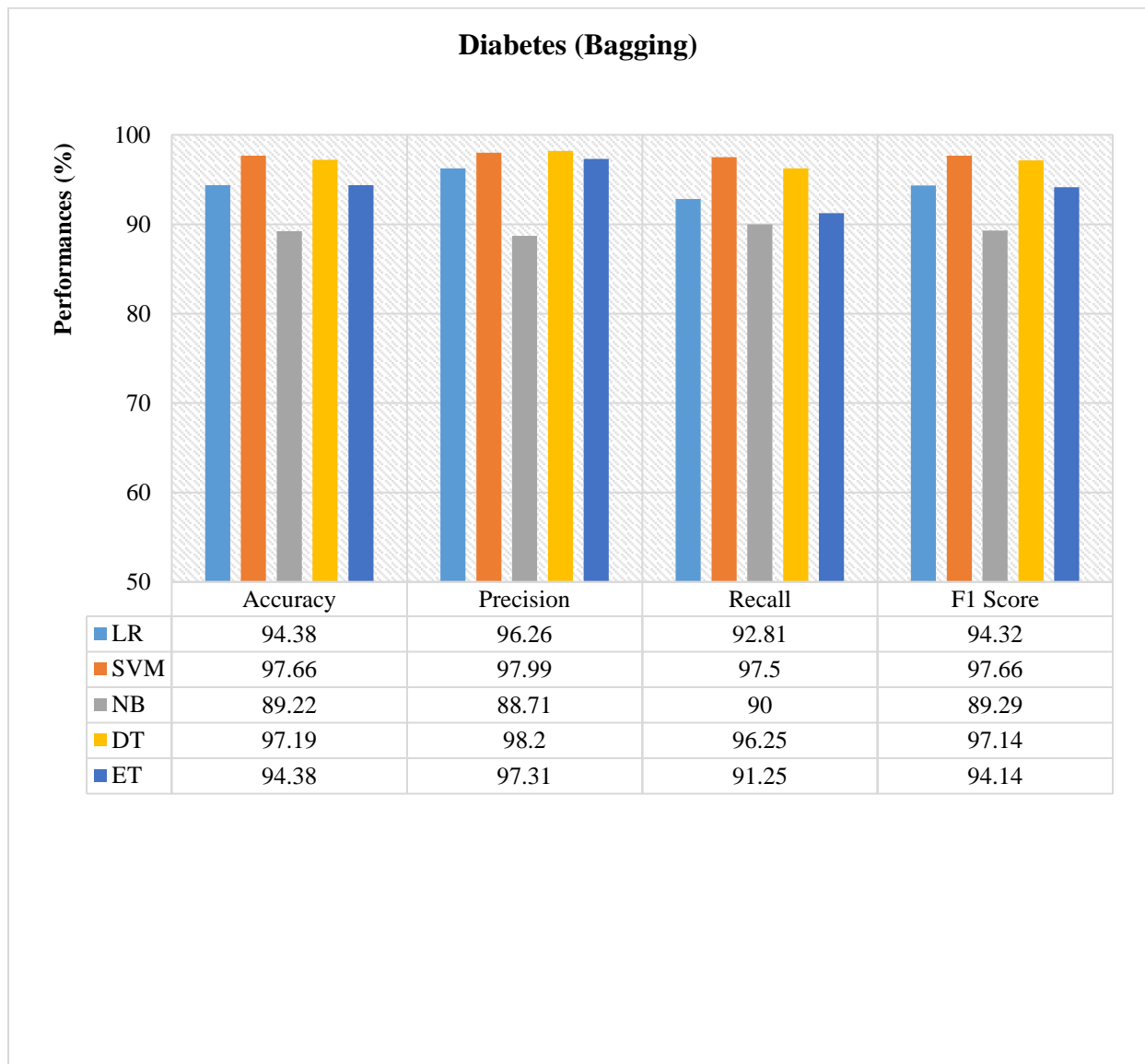
Here also we done the same process as we done other bagging methods and highest accuracy is 95.31%.

```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(ET_BG, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))

CV accuracy: 0.953125
CV precision: 0.9836350574712643
CV recall: 0.921875
CV F1 score: 0.9513342126027213
```

#### 4.3.6 Analysis Results of Bagging Classifiers (Bagging)



**Fig. 5.** Experimental Results of Bagging Classifiers

Secondly, the performances obtained by bagging classifiers should be talked over. Starting with accuracy, the elaboration has depicted that the best accuracy has been obtained 97.66% using SVM classifier. Then considering the performances of precision, the best score of 98.20% has been obtained by applying DT.



#### 4.3.6 LR ROC Curve Result (Bagging)

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    LR_BG_probs_y = LR_BG.predict_proba(X)
    LR_BG_auc_y = roc_auc_score(y, LR_BG_probs_y[:,1])
    print(LR_BG_auc_y)

0.981953125
```

#### 4.3.7 SVM ROC Curve Result (Bagging)

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    SVM_BG_probs_y = SVM_BG.predict_proba(X)
    SVM_BG_auc_y = roc_auc_score(y, SVM_BG_probs_y[:,1])
    print(SVM_BG_auc_y)

1.0
```

#### 4.3.8 Naïve Bayes ROC Curve Result (Bagging)

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    NB_BG_probs_y = NB_BG.predict_proba(X)
    NB_BG_auc_y = roc_auc_score(y, NB_BG_probs_y[:,1])
    print(NB_BG_auc_y)

0.9576660156250001
```

#### 4.3.9 Decision Tree ROC Curve Result (Bagging)

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    DT_BG_probs_y = DT_BG.predict_proba(X)
    DT_BG_auc_y = roc_auc_score(y, DT_BG_probs_y[:,1])
    print(DT_BG_auc_y)

0.99994140625
```

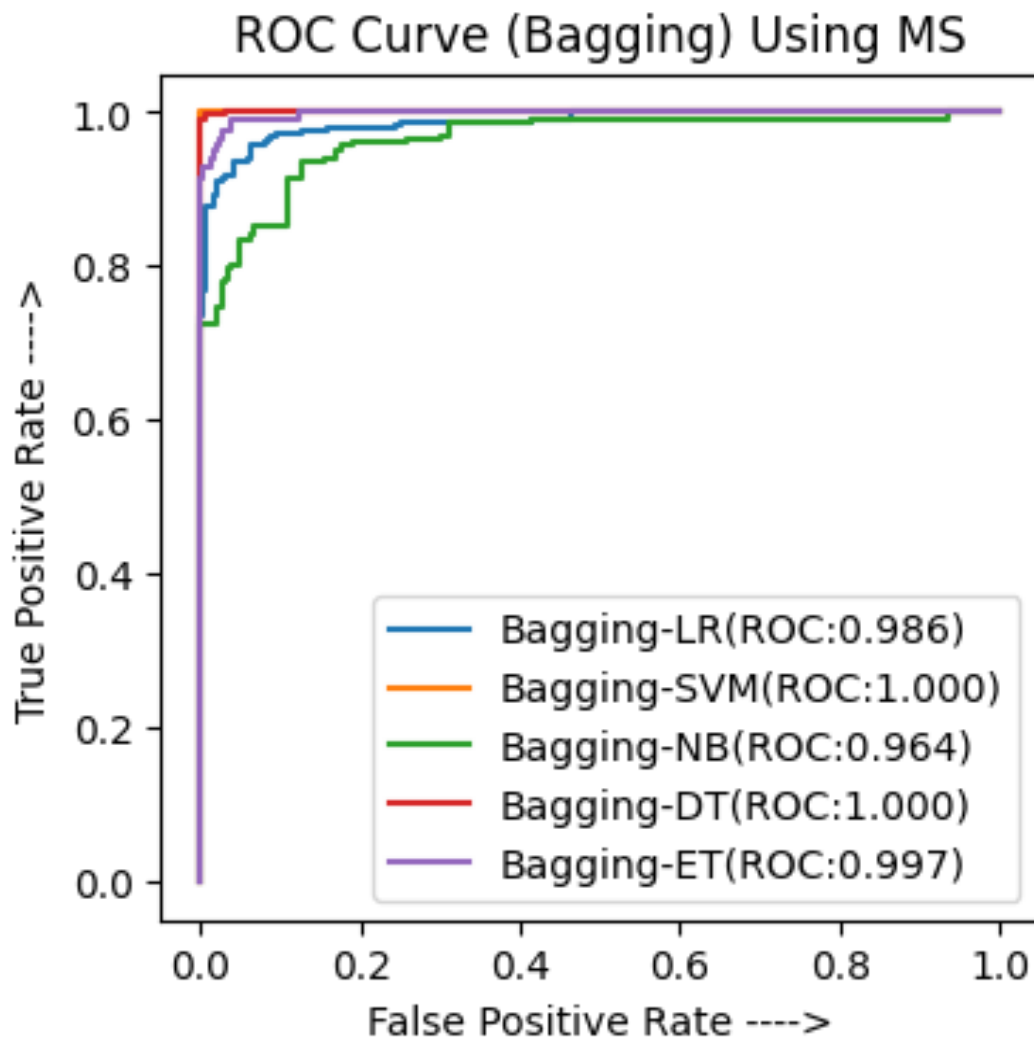
#### 4.3.10 Extra Tree ROC Curve Result (Bagging)

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    ET_BG_probs_y = ET_BG.predict_proba(X)
    ET_BG_auc_y = roc_auc_score(y, ET_BG_probs_y[:,1])
    print(ET_BG_auc_y)

0.9966992187499999
```

#### 4.3.11 Analysis of Bagging Classifiers ROC Curve



**Fig. 6.** ROC Curve Analysis of Bagging Classifiers

In the case of the recall section, SVM has obtained its position back showing the best score of 97.50%. In the case of the f1-score, once again the same scenario has been observed that SVM has performed the best of 97.66%.

ROC Curve, depicted in Fig 6, has drawn a different scenario that, both SVM and DT have provided the best score of 1 jointly. In this case, it is also noticeable that, after using bagging, the performances of the classifiers have been changed. The accuracy of LR and DT have been

improved whereas the accuracy of SVM and ET have been decreased and NB has remained the same.

The precision of LR, SVM, and DT have been increased though the results of NB and ET have been decreased. The values of recall have been also changed where LR, SVM and ET haven't abled to increase though NB and DT have. F1 score of LR, NB and DT have been improved a bit where the score of SVM and ET haven't. And finally, the ROC score of NB and DT have been improved where the score of ET have been decreased as well as LR and SVM have provided the same.

#### **4.4 Boosting Classifiers and ROC Curve**

Therefore, according to the above disputation as well as the detailed results with graphical representation picturized in Fig 5, the SVM classifier can be marked as the best bagging classifier. And the final consideration should be the performances obtained using boosting classifiers.

On setting with the accuracy section, the best accuracy has been obtained 98.44% using DT classifier. Then looking at the performances of precision, the best score of 98.78% has been obtained by DT. Considering the recall section, a little bit different result of best score of 98.75% using ET where DT has occupied the second position scoring 98.13%.

In the case of the f1-score, once again the same scenario has been observed that SVM classifier has performed the best of 98.42%. ROC Curve, depicted in Fig 8, has drawn a surprising scenario that SVM, DT and ET have served the score of 1. The changes of the performances of the classifiers due to using Adaboost have also been analyzed. Starting with accuracy, the scores of LR, SVM, and NB have been decreased whereas the scores of DT and ET have been improved. The SVM and DT have abled to improve the precision but LR, NB, and ET haven't. In case of recall, the values of LR and SVM have been decreased where the scores of DT and ET have been increased though NB have remained the same. Almost the same observation of having improved result by DT and ET as well as decreased result by LR, SVM, and NB has been noticed. The ROC scores of LR, SVM and DT have remained the same where the score of NB has been decreased and the ET has the opposite. Therefore, according to the above disputation as well as the detailed

results with graphical representation portrayed in Fig 7, the DT classifier can be considered as the best boosting classifier.

Final thing is in boosting we can't do any hyperparameter tuning. That's why we don't use any Randomized search algorithm here and other process is same.

#### 4.4.1 Logistic Regression (Boosting)

##### ▼ Logistic Regression

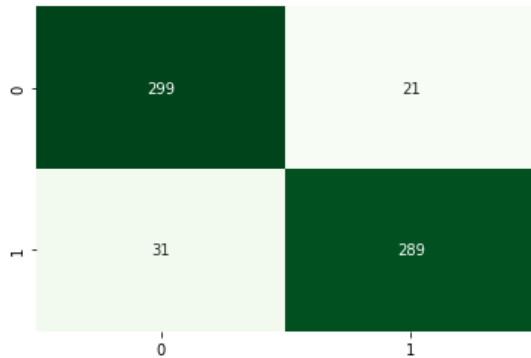
```
# Importing Libraries
from sklearn.ensemble import AdaBoostClassifier
LR_adb = AdaBoostClassifier(LRClassifier)
LR_adb.fit(X,y)

AdaBoostClassifier(algorithm='SAMME.R',
                    base_estimator=LogisticRegression(C=2.857142857142857,
                                                        class_weight=None,
                                                        dual=False,
                                                        fit_intercept=True,
                                                        intercept_scaling=1,
                                                        l1_ratio=None,
                                                        max_iter=291627,
                                                        multi_class='auto',
                                                        n_jobs=None, penalty='l2',
                                                        random_state=None,
                                                        solver='liblinear',
                                                        tol=0.0001, verbose=0,
                                                        warm_start=False),
                    learning_rate=1.0, n_estimators=50, random_state=None)

[ ] # Predicting the result
from sklearn.model_selection import cross_val_predict
LR_adb_pred = cross_val_predict(LR_adb,X,y,cv=10)

[ ] # Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, LR_adb_pred)
print(cm)
sns.heatmap(cm, annot=True, fmt="d", cmap='Greens', cbar=False)
```

```
[[299  21]
 [ 31 289]]
<matplotlib.axes._subplots.AxesSubplot at 0x7fc493d8da50>
```



```
] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(LR_adb, X, y, cv=10, scoring=scoring)

print('CV accuracy:', np.mean(results['test_accuracy']))
print('CV precision:', np.mean(results['test_precision']))
print('CV recall:', np.mean(results['test_recall']))
print('CV F1 score:', np.mean(results['test_f1_score']))

CV accuracy: 0.91875
CV precision: 0.9378105422571495
CV recall: 0.903125
CV F1 score: 0.9181236710877112
```

## 4.4.2 SVM (Boosting)

```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(SVM_adb, X, y, cv=10, scoring=scoring)

print('CV accuracy:', np.mean(results['test_accuracy']))
print('CV precision:', np.mean(results['test_precision']))
print('CV recall:', np.mean(results['test_recall']))
print('CV F1 score:', np.mean(results['test_f1_score']))

CV accuracy: 0.8484375
CV precision: 0.8648763739816371
CV recall: 0.846875
CV F1 score: 0.8478202193114539
```

Highest accuracy is 84.84%

### 4.4.3 Naïve Bayes (Boosting)

```
# Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(NB_adb, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))
```

CV accuracy: 0.7234375  
CV precision: 0.6730590473406863  
CV recall: 0.878125  
CV F1 score: 0.7595223524608226

Obtained accuracy is 72.34%

### 4.4.3 Decision Tree (Boosting)

```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(DT_adb, X, y, cv=10, scoring=scoring)

print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))
```

CV accuracy: 0.9890625  
CV precision: 0.9907196969696969  
CV recall: 0.9875  
CV F1 score: 0.9890369352869353

Last of all we get our most highest accuracy of our prediction in this method. Which is upper than 98.50

#### 4.4.4 Extra Tree (Boosting)

```
[ ] # Results
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
scoring={
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1_score': make_scorer(f1_score)
}
results=cross_validate(ET_adb, X, y, cv=10, scoring=scoring)

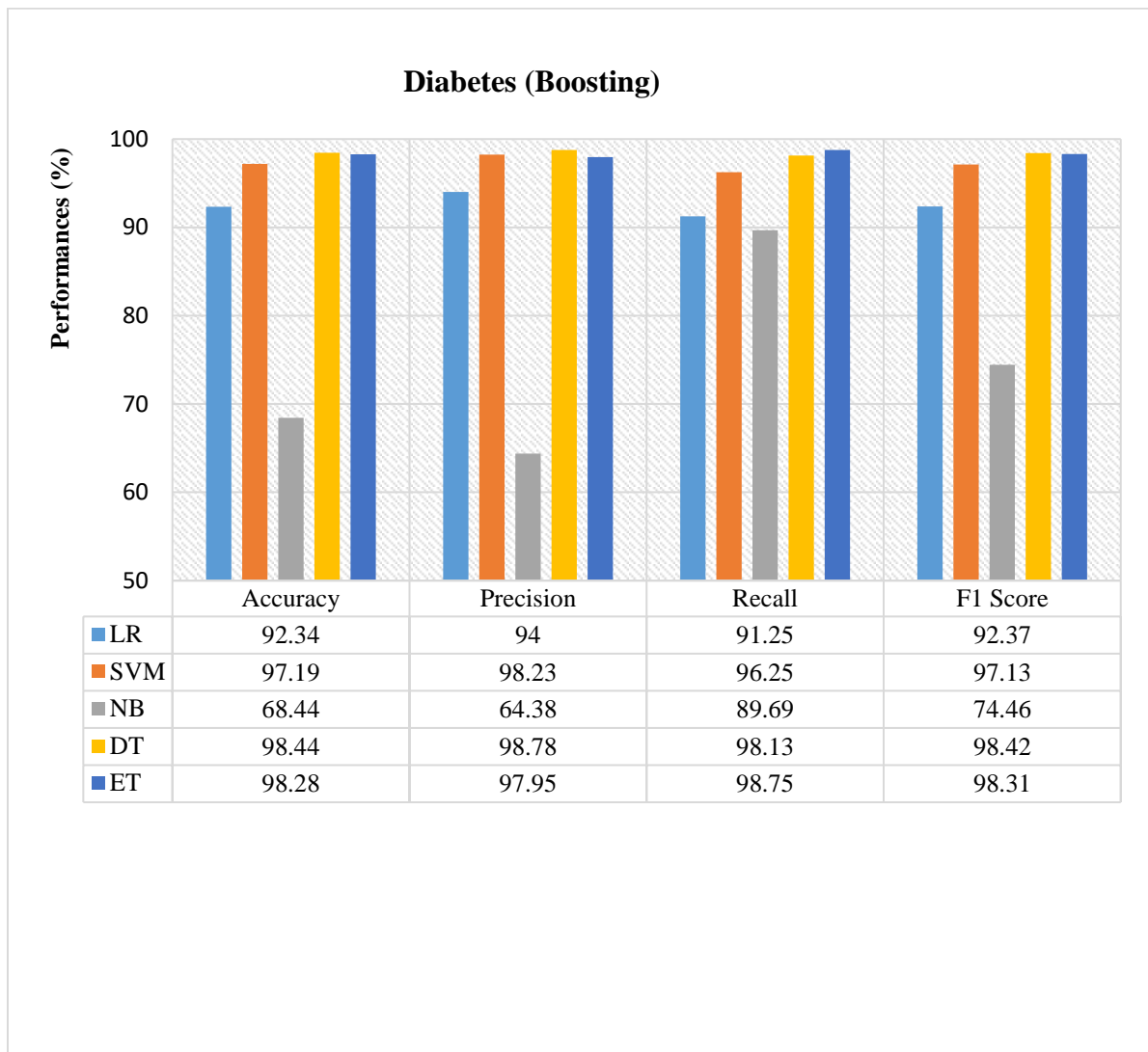
print('CV accuracy:',np.mean(results['test_accuracy']))
print('CV precision:',np.mean(results['test_precision']))
print('CV recall:',np.mean(results['test_recall']))
print('CV F1 score:',np.mean(results['test_f1_score']))

CV accuracy: 0.9828125
CV precision: 0.9934408602150537
CV recall: 0.971875
CV F1 score: 0.9823860727086533
```

Final accuracy is 98.28%.



#### 4.4.4 Analysis Results of Boosting Classifiers (Boosting)



**Fig. 7.** Experimental Results of Boosting Classifiers

#### 4.4.5 LR ROC Curve Result (Boosting)

```
[ ] # AUC
from sklearn.metrics import roc_auc_score, roc_curve

LR_adb_probs_y = LR_adb.predict_proba(X)
LR_adb_auc_y = roc_auc_score(y, LR_adb_probs_y[:,1])
print(LR_adb_auc_y)

0.9837011718750001
```

#### 4.4.6 SVM ROC Curve Result (Boosting)

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    SVM_adb_probs_y = SVM_adb.predict_proba(X)
    SVM_adb_auc_y = roc_auc_score(y, SVM_adb_probs_y[:,1])
    print(SVM_adb_auc_y)

0.9993945312499999
```

#### 4.4.7 Naïve Bayes ROC Curve Result (Boosting)

```
▶ # AUC
  from sklearn.metrics import roc_auc_score, roc_curve

  NB_adb_probs_y = NB_adb.predict_proba(X)
  NB_adb_auc_y = roc_auc_score(y, NB_adb_probs_y[:,1])
  print(NB_adb_auc_y)

☞ 0.805458984375
```

#### 4.4.8 Decision Tree ROC Curve Result (Boosting)

```
[ ] # AUC
    from sklearn.metrics import roc_auc_score, roc_curve

    DT_adb_probs_y = DT_adb.predict_proba(X)
    DT_adb_auc_y = roc_auc_score(y, DT_adb_probs_y[:,1])
    print(DT_adb_auc_y)

1.0
```

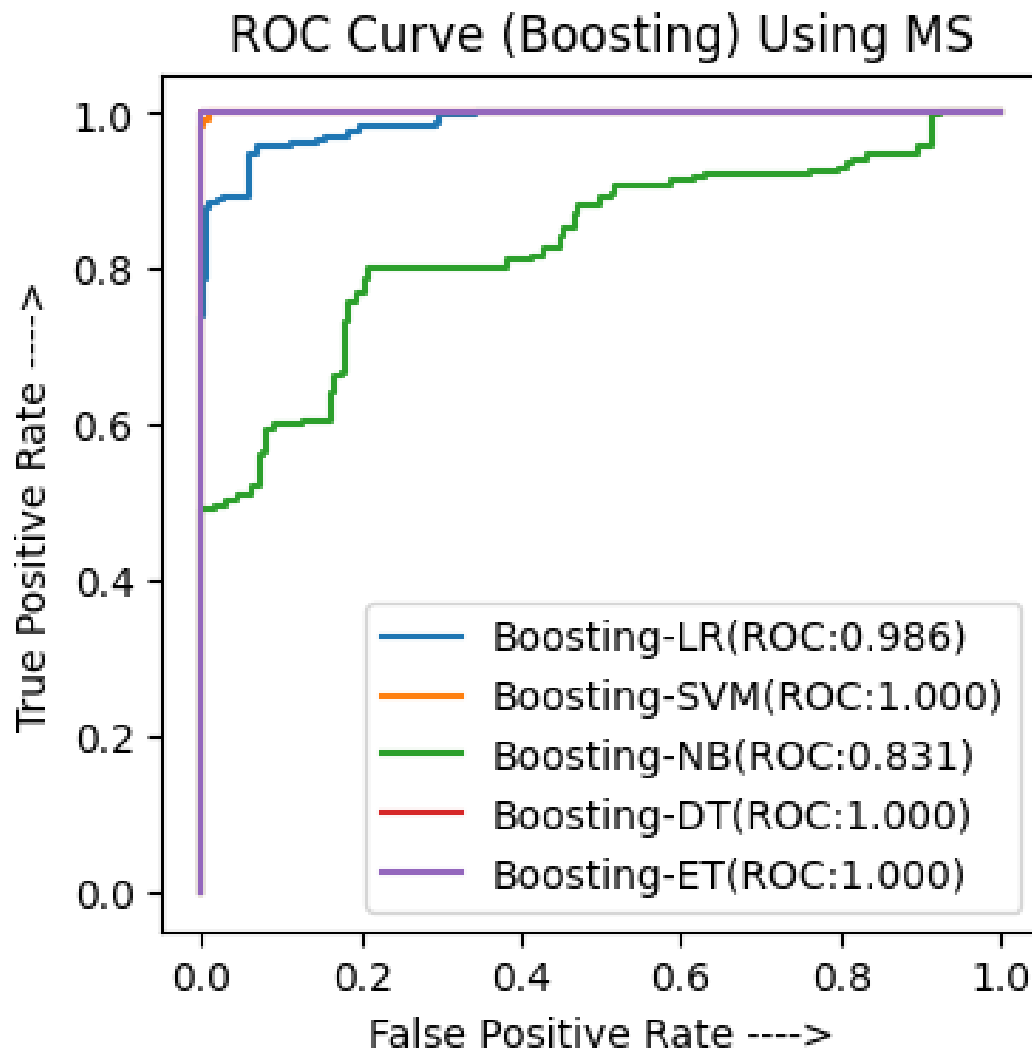
#### 4.4.9 Extra Tree ROC Curve Result (Boosting)

```
▶ # AUC
  from sklearn.metrics import roc_auc_score, roc_curve

  ET_adb_probs_y = ET_adb.predict_proba(X)
  ET_adb_auc_y = roc_auc_score(y, ET_adb_probs_y[:,1])
  print(ET_adb_auc_y)

☞ 1.0
```

#### 4.4.10 ROC Curve Analysis of Boosting Classifiers



**Fig. 8.** ROC Curve Analysis of Boosting Classifiers

However, according to the overall performances, our proposed model is Decision Tree (DT) that has provided the best accuracy of 98.44% after using boosting. DT is the lowest time-consuming classifier as well.

## **CHAPTER 5**

### **Conclusion & Future Work**

#### **5.1 Conclusion**

Medical data classification is supposed as amongst the most challenging tasks in medical informatics. Yet it is one of the most venerable works in the research field. Consequently, various methods have been proposed by others. Still there exists several scopes of improvement. Hence, we exhibited a correlative analysis for the diabetes dataset of the performance measures for various traditional, bagging, and boosting classification algorithms with 10-fold cross-validation where different pre-processing methods are used first. The performances of the models are obtained better than existing models because of assigning the best parameters by hyperparameter tuning. We have thoroughly examined the dataset. As a consequence, several performances have been observed using different algorithms. According to them, we have proposed the DT classifier for this dataset. Overall, the exploration gets better results as compared to the existing methods done by others because we have also compared according to the performance evaluation measures where it is also noticeable that, the other researchers haven't analyzed such a number of measures that we have.

#### **5.2 Future Work**

We can compile our experiment by adding new algorithms as well as using other datasets as our future work. Trying the different way of pre-processing techniques as well as using deep learning techniques should be our future target tasks as well.

We will make a website; on that website we will add our best algorithms to detect early stage of diabetes. It will be more user friendly so that anyone can use that website, so that they could predict diabetes.

#### **5.3 Challenges**

We looked very carefully so that there's no missing data. We used traditional algorithms but the result didn't come so good. We study a lot how to get the best result. Then we have figured it out then we apply hyperparameter tuning and use bagging & boosting classifiers. Our future challenges are to make the website using our source algorithms to find the best result.

## References:

- [1] Ramesh, D., & Katheria, Y. S. (2019). Ensemble method based predictive model for analyzing disease datasets: a predictive analysis approach. *Health and Technology*, 9(4), 533-545.
- [2] Islam, M. F., Ferdousi, R., Rahman, S., & Bushra, H. Y. (2020). Likelihood prediction of diabetes at early stage using data mining techniques. In *Computer Vision and Machine Intelligence in Medical Image Analysis* (pp. 113-125). Springer, Singapore.
- [3] Le, Tuan Minh, et al. "A Novel Wrapper-Based Feature Selection for Early Diabetes Prediction Enhanced With a Metaheuristic." *IEEE Access* 9 (2020): 7869-7884.
- [4] Ogedengbe, M. T., & Egbunu, C. O. (2020). CSE-DT Features selection technique for Diabetes classification. *Applications of Modelling and Simulation*, 4, 101-109.
- [5] Abdillah, A. A. (2021). Optimasi Linear Sampling dan Information Gain pada Algoritma Decision Tree untuk Diagnosis Penyakit Diabetes. *MULTINETICS*, 7(1), 21-29.
- [6] Aada, A., and Sakshi Tiwari. "Predicting diabetes in medical datasets using machine learning techniques." *Int. J. Sci. Eng. Res* 5.2 (2019).
- [7] Lai, H., Huang, H., Keshavjee, K., Guergachi, A., & Gao, X. (2019). Predictive models for diabetes mellitus using machine learning techniques. *BMC endocrine disorders*, 19(1), 1-9.
- [8] Tafa, Z., Pervetica, N., & Karahoda, B. (2015, June). An intelligent system for diabetes prediction. In *2015 4th Mediterranean Conference on Embedded Computing (MECO)* (pp. 378-382). IEEE.
- [9] Negi, A., & Jaiswal, V. (2016, December). A first attempt to develop a diabetes prediction method based on different global datasets. In *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)* (pp. 237-241). IEEE.
- [10] Alam, T. M., Iqbal, M. A., Ali, Y., Wahab, A., Ijaz, S., Baig, T. I., ... & Abbas, Z. (2019). A model for early prediction of diabetes. *Informatics in Medicine Unlocked*, 16, 100204.
- [11] Tripathi, G., & Kumar, R. (2020, June). Early prediction of diabetes mellitus using machine learning. In *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)* (pp. 1009-1014). IEEE.
- [12] Ramani, R. G., & Sivagami, G. (2011). Parkinson disease classification using data mining algorithms. *International journal of computer applications*, 32(9), 17-22.
- [13] Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, No. 22, pp. 41-46).
- [14] "Uci machine learning repository: Data set," <https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset>, (Accessed on 06/26/2021).