## CHRONIC KIDNEY DISEASE PREDICTION USING MACHINE LEARNING

*Report submitted to the SASTRA Deemed to be University as the requirement for the course*

## CSE300 - MINI PROJECT

*Submitted by*

### Gandrothu Pavan Sai Kishore

### (Reg. No.: 122003073, CSE)

### Maddu Bhaskar Vamsi Krishna

### (Reg. No.: 122003138, CSE)

### Tamalampudi Venkata Rama Reddy

### (Reg. No.: 122014053, ICT)

# January 2022



**SCHOOL OF COMPUTING**

### THANJAVUR, TAMIL NADU, INDIA – 613 401



**SCHOOL OF COMPUTING THANJAVUR – 613 401**

**Bonafide Certificate**

This is to certify that the report titled “**Chronic Kidney Disease Prediction Using Machine Learning**” submitted as a requirement for the course, CSE300: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by

**Mr. Gandrothu Pavan Sai Kishore (Reg. No. 122003073, CSE)**

**Mr. Maddu Bhaskar Vamsi Krishna (Reg. No. 122003138, CSE)**

**Mr. Tamalampudi Venkata Rama Reddy (Reg. No. 122014053, ICT)**

during the academic year 2021-22, in the School of Computing, under my supervision.

**Signature of Project Supervisor :**

**Name with Affiliation :** Dr. Renuga Devi. T, Asst. Professor - II, School of Computing

**Date :** 09/01/2022

Mini Project *Viva voc*e held on

**Examiner 1 Examiner 2**

**Acknowledgements**

First of all, I would like to thank *God Almighty* for his endless blessings.

I would like to express my sincere gratitude to **Dr S. Vaidya Subramaniam,** *Vice-Chancellor* for his encouragement during the span of my academic life at SASTRA Deemed University.

I would forever remain grateful and I would like to thank **Dr A. Uma Maheshwari**, *Dean, School of Computing* and **R. Chandramouli**, *Registrar* for their overwhelming support provided during my course span in *SASTRA Deemed University*.

I am extremely grateful to **Dr. Shankar Sriram**, *Associate Dean, School of Computing* for his constant support, motivation and academic help extended for the past three years of my life in *School of Computing*.

I would specially thank and express my gratitude to **Dr. RenugaDevi T**, *Assistant Professor-II, School of Computing* for providing me an opportunity to do this project and for her guidance and support to successfully complete the project.

I also thank all the *Teaching and Non-teaching faculty*, and all other people who have directly or indirectly help me through their support, encouragement and all other assistance extended for completion of my project and for successful completion of all courses during my academic life at

*SASTRA Deemed University*.

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| 1.1 | Work Flow model for CKD Prediction before applying SMOTE | 5 |
| 1.2 | Work Flow model for CKD Prediction after applying SMOTE | 5 |
| 4.1 | Sample Image of Execution of Source code in Jupyter Notebook | 50 |
| 4.2 | Heat Maps of Correlation between Selected Features in case of Full feature selection | 51 |
| 4.3 | Heat Maps of Correlation between Selected Features in case of CFS | 51 |
| 4.4 | Heat Maps of Correlation between Selected Features in case of Wrapper FS | 51 |
| 4.5 | Heat Maps of Correlation between Selected Features in case of LASSO FS | 52 |
| 4.6 | Heat Maps of Correlation between Selected Features in case of Full features with SMOTE | 52 |
| 4.7 | Heat Maps of Correlation between Selected Features in case of LASSO FS with SMOTE | 52 |
| 4.8 | Accuracy, Precision, Recall for ML classifiers in case of Full feature selection | 53 |
| 4.9 | Accuracy, Precision, Recall for ML classifiers in case of CFS | 53 |
| 4.10 | Accuracy, Precision, Recall for ML classifiers in case of Wrapper FS | 54 |
| 4.11 | Accuracy, Precision, Recall for ML classifiers in case of LASSO FS | 54 |
| 4.12 | Accuracy, Precision, Recall for ML classifiers in case of Full features with SMOTE | 55 |
| 4.13 | Accuracy, Precision, Recall for ML classifiers in case of LASSO FS with SMOTE | 55 |
| 4.14 | Classification Error for ML classifiers in case of Full feature selection | 56 |
| 4.15 | Classification Error for ML classifiers in case of CFS | 56 |
| 4.16 | Classification Error for ML classifiers in case of Wrapper FS | 56 |
| 4.17 | Classification Error for ML classifiers in case of LASSO FS | 57 |
| 4.18 | Classification Error for ML classifiers in case of Full features with SMOTE | 57 |
| 4.19 | Classification Error for ML classifiers in case of LASSO FS with SMOTE | 57 |
| 4.20 | F- Measure for ML classifiers in case of Full feature selection | 58 |
| 4.21 | F- Measure for ML classifiers in case of CFS | 58 |
| **Figure No.** | **Title** | **Page No.** |
| 4.22 | F- Measure for ML classifiers in case of Wrapper FS | 58 |
| 4.23 | F- Measure for ML classifiers in case of LASSO FS | 59 |
| 4.24 | F- Measure for ML classifiers in case of Full features with SMOTE | 59 |
| 4.25 | F- Measure for ML classifiers in case of LASSO FS with SMOTE | 59 |
| 4.26 | AUC for ML classifiers in case of Full feature selection | 60 |
| 4.27 | AUC for ML classifiers in case of CFS | 60 |
| 4.28 | AUC for ML classifiers in case of Wrapper FS | 60 |
| 4.29 | AUC for ML classifiers in case of LASSO FS | 61 |
| 4.30 | AUC for ML classifiers in case of Full features with SMOTE | 61 |
| 4.31 | AUC for ML classifiers in case of LASSO FS with SMOTE | 61 |
| 4.32 | GINI Index for ML classifiers in case of Full feature selection | 62 |
| 4.33 | GINI Index for ML classifiers in case of CFS | 62 |
| 4.34 | GINI Index for ML classifiers in case of Wrapper FS | 62 |
| 4.35 | GINI Index for ML classifiers in case of LASSO FS | 63 |
| 4.36 | GINI Index for ML classifiers in case of Full features with SMOTE | 63 |
| 4.37 | GINI Index for ML classifiers in case of LASSO FS with SMOTE | 63 |
| 6.1 | Screenshot of Indexing | 66 |

## List of Tables

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Table name** | **Page No.** |
| 1.1 | Confusion Matrix for prediction of CKD | 3 |
| 1.2 | Description of Attributes in the Dataset | 6 |
| 2.1 | Accuracy Table of Classifiers in case of different FS | 9 |

**Abbreviations**

AI Artificial Intelligence

ANN Artificial Neural Network

AUC Area Under Curve

CFS Correlation based Feature Selection

CHAID Chi-square Automatic Interaction Detection

CKD Chronic Kidney disease

FN False Negative

FP False Positive

FS Feature Selection

KNN K-Nearest Neighbor

LASSO Least Absolute Shrinkage and Selection Operator

LASSO FS Least Absolute Shrinkage and Selection Operator Feature Selection

LR Logistic Regression

LSVM Linear Support Vector Machine

LSVM L1 Linear Support Vector Machine with penalty L1

LSVM L2 Linear Support Vector Machine with penalty L2

ML Machine Learning

RF Random Forest

ROC Receiver Operating a Characteristic

SMOTE Synthetic Minority class Oversampling Technique

TN True Negative

TP True Positive

UCI UC Irvine repository

## Abstract

Since curing a disease is possible only after detection, disease detection has become an essential factor in the medical sector. Chronic kidney disease (CKD), one among the major diseases that needs to be detected at early stages. Nowadays, machine learning (ML) is playing a crucial role in the detection of diseases. So, in this paper, we mainly focus on finding the best ML model for prediction of CKD. For this perspective, a dataset was collected from UCI repository. We applied 7 ML classifier algorithms, namely, K-Nearest Neighbor (KNN), Logistic Regression (LR), Linear Support Vector Machine (LSVM) with penalties L1 and L2, Random Forest (RF), CHAID, C 4.5, and Artificial Neural Network (ANN) for prediction. We apply Feature Selection (FS) methods such as full feature selection, correlation-based Feature Selection (CFS), wrapper FS, LASSO FS for selecting the features. A dataset for the SMOTE, a data balancing technique is applied to the dataset for balancing the target class. Classifier performance is measured using accuracy, precision, recall, error, F-measure, AUC, and GINI index.

In this paper, the results are computed for all classifier algorithms applied to each feature selection method. From the results, the best-performing algorithms with the best feature selection methods are selected, and their performances are computed with the application of SMOTE in each case. From the results, we observed that LR, LSVM L1, LSVM L2, CHAID, C 4.5, and RF perform better in measure of accuracy and other evaluation metrics than Full Features and LASSO Feature Selection in measure of accuracy and other evaluation metrics. After applying SMOTE with LASSO Feature Selection, Random Forest performs best with an accuracy of 98.49%, and LSVM with penalty L2 has an accuracy of 97.73%.

**KEY WORDS:** CKD, prediction, ML Classifier algorithms, FS methods, SMOTE.

|  |  |  |
| --- | --- | --- |
|  | **Table of Contents** |  |
| **Title** |  | **Page No.** |
| Bonafide Certificate |  | ii |
| Acknowledgements |  | iii |
| List of Figures |  | iv |
| List of Tables |  | vi |
| Abbreviations |  | vii |
| Abstract |  | viii |

1. Summary of the base paper 1

1. Merits and Demerits of the base paper 7
2. Source Code 10
3. Snapshots 50
4. Conclusion and Future Plans 64
5. References . 65
6. Appendix -Base Paper 66

## CHAPTER 1

### SUMMARY OF THE BASE PAPER

* 1. **Problem Statement**

CKD cannot be detected easily until the kidney is damaged badly. In this high-risk population, we are not able to check our status of the disease regularly.

By existing methods, we cannot predict the disease in advance. Using ML classifier algorithms, we can predict the disease on time.

* 1. **Data set collection & Pre-processing**
     1. **Data Set**

The CKD dataset for this paper was collected from the UCI website, which UC Irvine Machine Learning Repository was providing. This dataset consists of 25 attributes and 400 instances. Of those 25 attributes, 24 are feature attributes and 1 is a target attribute. The target attribute has 2 values: CKD or Not CKD. The 24 feature attributes and their descriptions are provided in Table 1.2.

* + 1. **KNN Imputation**

Identification of missing values and replacing them is the first and foremost thing we have to do to the dataset. This is known as data cleaning. One of the best approaches for data cleaning is to predict the missing values using an ML model. In this paper, we use the KNN algorithm for predicting missing values.

* + 1. **Label Encoder**

Label Encoding can be defined as the conversion of all the labels into numeric form, which will be further converted into a machine-readable form. It is one of the best approaches for data preprocessing. In this paper, we convert all the labels into either int or float types.

* + 1. **Train- Test split**

Train-Test Split is the most frequently used approach for preprocessing. In this process, the data is divided into two parts, one for training the model and the other for testing. It is mainly used for estimating the performance of ML classifiers. In this paper, we consider the train-test split ratio to be 50:50.

* 1. **Proposed Methodology**
     1. **Feature Selection Methods**
        1. **Full Feature Selection**

In this case, we will consider all features as input for ML classifiers for prediction.

* + - 1. **Correlation based FS (CFS)**

CFS is one of the filter methods to select the features based on the correlation between them. In this process, it will eliminate the highly correlated feature with the other feature. It selects only the remaining features after eliminating the highly correlated ones.

* + - 1. **Wrapper FS**

In Wrapper FS, the feature selection takes place based on a precise ML classifier. It uses a greedy approach for making subsets. Subsets can be formed either by the forward selection method or the backward elimination method. In this paper, we use "forward selection." In forward selection, the subset is initially null and, iteratively, the features are added.

* + - 1. **LASSO FS**

LASSO (least absolute shrinkage and selection operator) FS is an embedded method of FS. In the embedded method, features are selected based on decision trees. In this method, two processes take place: regularization and feature selection. In regularization, the coefficients of unimportant features are shrunk to zero, and in feature selection, the features with zero coefficients are eliminated.

* + 1. **Machine Learning Classifiers**
       1. **LR**

Logistic regression is one of supervised ML classifiers. It's a mathematical model. It predicts the probability of achieving the target value. It categorizes the target attribute as either successful or unsuccessful. It returns 1 when successful, and 0 when unsuccessful.

* + - 1. **KNN**

KNN is a simplest supervised algorithm. For both classification and regression problems KNN is applicable. It is, however, primarily used for solving classification problems. The distance between two already labelled data sets is calculated. The distance aids in locating the new data nearest neighbor. Distance is calculated using the Euclidian method.

* + - 1. **LSVM**

LSVM is a latest, uniquely fast machine learning algorithm that uses a simple iterative approach to solve multiclass classification problems for large datasets. The SVM model is built in the dataset's linear CPU time. There are two penalty cases, L1 and L2.

* + - 1. **CHAID**

This decision tree technique is chi-square automatic interaction detection (CHAID). It's used to figure out how variables are related to one another. CHAID can find the outcome using nominal, ordinal, and continuous data.

* + - 1. **C 4.5**

C4.5 is a decision tree in the sense that from input it generates the decision tree. The tree will have the specified count of branches. The structure of tree is used to find out the relationship between potential outcomes and features. This process is iterative and continues until no further splits are found.

* + - 1. **ANN**

ANN is a type of AI. It falls under the category of supervised ML. It has the same structure as the human brain. It can solve problems that human or statistical standards have failed to solve.

* + - 1. **RF**

Random forests, also called as random decision forests, is an ensemble learning classifier for regression and classification that works by constructing a big number of decision trees during training. For classification tasks, the random forest output is the class chosen by the majority of trees. The mean prediction value of the different trees is returned for regression tasks. RF outperform decision trees in general.

* + 1. **SMOTE**

Data balancing is the major problem one is facing while using dataset for prediction. There are 2 methods for data balancing, oversampling and under sampling. In this case we use oversampling. For oversampling the minority class, the SMOTE is used. It is also referred to as a data balancer. It accepts the entire dataset as input but works only on the minority class. It raises the proportion of minorities in the population. This methodology expands the features available for target class.

* + 1. **Performance Evaluation Metrics**
       1. **Confusion Matrix for prediction of CKD**

|  |  |  |
| --- | --- | --- |
|  | CKD (1) | Not CKD (0) |
| CKD (1) | **TP** | **FN** |
| Not CKD (0) | **FP** | **TN** |

Table. 1.1. Confusion Matrix for prediction of CKD

TP: It indicates that output was CKD and the prediction was classified correctly.

TN: It indicates that output was not CKD and the prediction was classified correctly.

FP: It indicates that output was CKD and the prediction was classified incorrectly.

FN: It indicates that output was not CKD and the prediction was classified incorrectly.

* + - 1. **Accuracy:**

It shows the right rate of prediction findings.

Accuracy = ((TP+TN)/(TP+TN+FP+FN)) \*100

* + - 1. **Precision:**

It is the proportion of related instances in the total number of retrieved instances. It is a predicted positive value.

Precision= (TP/(TP+FP)) \*100

* + - 1. **Recall:**

It is the proportion of related instances among all retrieved instances.

Recall= (TP/(TP+FN)) \*100

* + - 1. **F- Measure:**

It is referred to as the F Score. It is used to determine the test accuracy.

F- Measure= 2\* ((precision\*recall)/ (precision + recall))

* + - 1. **AUC:**

ROC curve, a graph that compares the true positive rate to the false positive rate at various classification thresholds. The whole area under the ROC curve is referred to as the AUC.

* + - 1. **GINI Index:**

It is used to assess the disparity in attribute value distributions. It calculates the impurity of a specific attribute as probability.

GINI= 2\* AUC-1

* + - 1. **Error:**

The rate of incorrect predicted results is shown by the classification error.

Error = 1- accuracy

Fig. 1.2. Work Flow model for CKD Prediction after applying SMOTE

**CKD**

**Not CKD**

**Best Performed Algorithm is used for Prediction**

**Performance Evaluation Measures**

**ML Classifiers**

**Feature Selection**

**Best Performed algorithms in best feature selection techniques are selected.**

**Performance Evaluation Measures**

**LR**

**ANN**

**C4.5**

**LSVM L1**

**LSVM L2**

**KNN**

**RF**

**CHAID**

**LR**

**Selected Features**

**LASSO FS**

**Wrapper FS**

**Full Features**

**Correlation based FS**

**Target Class instances are converted to Binary**

Fig. 1.1. Work Flow model for CKD Prediction before applying SMOTE

**ML Classifiers**

**LASSO FS**

**Full Features**

**Data Pre-Processing**

**CKD Data Set**

**SMOTE**

**Selected Features**

**LR**

**CHAID**

**RF**

**LSVM L2**

**C4.5**

**LSVM L1**

**LR**

**Feature Selection**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Attribute Name** | **Description** |
| 1 | Age | Age of patient |
| 2 | Bp | Blood pressure of patient |
| 3 | Sg | Urine specific gravity of patient |
| 4 | Al | Albumin level of patient |
| 5 | Su | Sugar level of patient |
| 6 | Rbc | Red blood cells normality in patient |
| 7 | Pc | Pus cell normality in patient |
| 8 | Pcc | Pus cell clumps existence in patient |
| 9 | Ba | Bacteria existence in patient |
| 10 | Bgr | Blood glucose random value of patient |
| 11 | Bu | Blood urea value of patient |
| 12 | Sc | Serum creatinine value of patient |
| 13 | Sod | Sodium level of patient |
| 14 | Pot | Potassium level of patient |
| 15 | Hemo | Hemoglobin level of patient |
| 16 | Pcv | Packed cell volume % of RBCs in circulating blood of patient |
| 17 | Wc | White blood cells count in patient |
| 18 | Rc | Red blood cells count in patient |
| 19 | Htn | Hypertension Yes or No for patient |
| 20 | Dm | Diabetes mellitus Yes or No for patient |
| 21 | Cad | Coronary artery disease Yes or No for patient |
| 22 | Appet | Appetite Yes or No for patient |
| 23 | Pe | Pedal edema Yes or No for patient |
| 24 | Ane | Anemia Yes or No for patient |
| 25 | Class | Target Variable (CKD or Not CKD) |

Table. 1.2. Description of Attributes in the Dataset.

## CHAPTER 2

**MERITS AND DEMERITS OF THE BASE PAPER**

**2.1. Literature Review**

G. Parthiban and K. R. A. Padmanaban et al. [9] identified Kidney disorder in its early stages by using Nave Bayes and Decision tree techniques on the selected features. However, the neural network prediction system's accuracy could be improved.

K. D. M. Perera, W. Gunarathne and K. A. D. C. P Kahandawaarachchi et al. [8] conducted a performance evaluation on Machine Learning Classification Techniques for classification of disease and Forecasting via Data Analytics for CKD and discovered that the Multiclass Decision Forest algorithm has the best accuracy of 98.1 percent.

The effects of using clinical features to classify patients with chronic kidney disease using the SVM algorithm were investigated by S. Shamiluulu, Y. Amirgaliyev and A. Serek et al. [7]. The implemented classifier is capable of achieving an overall performance of 94.602 percent.

M. Almasoud and T. E. Ward et al. [6] used 10-fold cross-validation to train and test logistic regression, support vector machines, random forest, and gradient boosting algorithms in this study. The gradient boosting algorithm outperformed other algorithms in terms of F1-measure (99.1 percent), sensitivity (98.8 percent), and specificity (98.8 percent) (99.3 percent).

L. Lessa, A. Peixoto, L. Kilvia De Almeida, R. Gomes, and J. Celestino, among others, [3] They devised a method for detecting kidney failure early on, increasing the chances of successful treatment for these patients. Several Machine Learning techniques (DT, SVM, and RF) were employed for this purpose. However, the accuracy of the prediction was low and could be improved further.

S. Elavarthy, T. Kiran, S. Shankar, S. Verma, P. Ghuli, and colleagues [4] Logistic Regression, Random Forest Tree, K-Nearest Neighbor, and Neural Network are four ML techniques that they used. To determine the correct classifier for CKD diagnosis, they conducted a systematic study of the outcomes of all three classifiers. However, certain noisy values are missing from the data collection.

**2.2. Merits**

ML classifiers are commonly used for prediction. We used several FS methods to improve accuracy. Not every algorithm performs well with a well-balanced data set. As a result, we used SMOTE to balance the dataset.

**2.3. Demerits**

In general, ANN and KNN are better performing classifiers, but their performance on our data set was poor.

**2.4. Results**

In case of Full feature selection, for LR the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97, 100, 94.96, 0.96, 97.47, 0.94 and 3.0 respectively. for KNN the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 66.5, 66.25, 89.07, 0.44, 61.20, 0.22 and 33.5 respectively. for LSVM penalty L1 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97.0, 99.1, 95.7, 0.96, 97.2, 0.94 and 3.0 respectively. for LSVM penalty L2 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97.5,100, 95.8, 0.97, 97.9, 0.95 and 2.5 respectively. for RF the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 99.5, 99.1, 100, 0.99, 99.3, 0.98 and 0.5 respectively. for CHAID the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 96.2, 97.5, 96.3, 0.96, 96.1, 0.92 and 3.78 respectively. for C 4.5 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 96.9, 97.5, 97.5, 0.97, 96.7, 0.93 and 3.03 respectively. for ANN the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 83, 94.3, 61.7, 0.74, 79.6, 0.59 and 17.0 respectively.

In case of CFS, for LR the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 94.5, 97.3,93.27, 0.93, 94.7, 0.89 and 5.5 respectively. for KNN the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 64.5, 68.4, 74.7, 0.52, 62.08, 0.24 and 35.5 respectively. for LSVM penalty L1 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 95.5, 97.4, 94.9, 0.94, 95.6, 0.91 and 4.5 respectively. for LSVM penalty L2 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 91.5, 97.2, 88.2, 0.90, 92.2, 0.84 and 8.5 respectively. for RF the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97, 95.1, 100, 0.96, 96.29, 0.92 and 3.0 respectively. for CHAID the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 95.45, 95.23, 97.56, 0.96, 94.7, 0.89 and 4.54 respectively. for C 4.5 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 96.9, 98.75, 96.34, 0.97, 97.1, 0.94 and 3.0 respectively. for ANN the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 67.5, 60.0, 59.25, 0.59, 66.1, 032 and 32.5 respectively.

In case of Wrapper FS, for LR the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 90.5, 92.3, 91.5, 0.88, 90.24, 0.80 and 9.5 respectively. for KNN the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 64.0, 68.2, 73.9, 0.52, 61.6, 0.23 and 36 respectively. for LSVM penalty L1 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 90.0, 93.04, 89.9, 0.87, 90.0, 0.8 and 10 respectively. for LSVM penalty L2 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 90.5, 92.3, 91.5, 0.8, 90.2, 0.8 and 9.5 respectively. for RF the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 91.5, 91.8, 94.1, 0.89, 90.8, 0.81 and 8.5 respectively. for CHAID the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 94.6, 94.1, 97.5, 0.95, 93.7, 0.87 and 5.3 respectively. for C 4.5 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 96.2, 96.3, 97.5, 0.96, 95.7, 0.91 and 3.7 respectively. for ANN the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 79.5, 66.94, 97.5, 0.88, 90.88, 0.81 and 20.5 respectively.

In case of LASSO FS, for LR the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97.5, 100, 95.7, 0.97, 97.8, 0.95 and 2.5 respectively. for KNN the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 67.0, 70.8, 75.6, 0.57, 64.97, 0.29 and 33 respectively. for LSVM penalty L1 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97.5, 99.1, 96.6, 0.96, 97.7, 0.95 and 2.5 respectively. for LSVM penalty L2 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97.5, 99.13, 96.63, 0.96, 97.7, 0.95 and 2.5 respectively. for RF the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 98.0, 98.3, 98.3, 0.97, 97.9, 0.95 and 2 respectively. for CHAID the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 98.4, 98.7, 98.7, 0.98, 98.3, 0.96 and 1.5 respectively. for C 4.5 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 93.1, 95.0, 93.9, 0.94, 92.9, 0.85 and 6.8 respectively. for ANN the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 87.5, 87.8, 80.2, 0.83, 86.3, 0.72 and 12.5 respectively.

In case of Full features with SMOTE, for LR the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 96.5, 100, 94.1, 0.95, 97.0, 0.94 and 3.5 respectively. for LSVM penalty L1 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 95.5, 98.3, 98.3, 0.97, 85.8, 0.91 and 4.5 respectively. for LSVM penalty L2 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 96.5, 99.1, 94.9, 0.95, 96.8, 0.93 and 3.5 respectively. for RF the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97.5, 98.3, 97.4, 0.96, 97.5, 9.95 and 2.5 respectively. for CHAID the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 95.4, 98.7, 93.9, 0.96, 95.9, 0.9 and 4.5 respectively. for C 4.5 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 62.1, 62.1, 100, 0.76, 50, 0 and 37.8 respectively.

In case of LASSO FS with SMOTE, for LR the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 96.9, 98.75, 96.3, 0.97, 97.1, 0.91 and 3.03 respectively. for LSVM penalty L1 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 96.9, 98.7, 96.3, 0.97, 97.1, 0.91 and 3.03 respectively. for LSVM penalty L2 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 97.7, 98.7, 97.5, 0.98, 97.7, 0.93 and 2.27 respectively. for RF the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 98.4, 97.6, 100, 0.98, 98, 0.95 and 1.51respectively. for CHAID the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 95.45, 96.34, 96.34, 0.96, 95.17, 0.91 and 4.5 respectively. for C 4.5 the accuracy, precision, recall, F-measure, AUC, GINI Index and Error are 62.12, 62.12, 100, 0.76, 50, 0 and 37.8 respectively.

In the Table. 2.1. the accuracies of all classifiers in all cases are compared and for each case the best accurate algorithm is marked.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Classifiers / FS** | **Full Features** | **CFS** | **Wrapper** | **LASSO** | **Full Features with SMOTE** | **LASSO with SMOTE** |
| **LR** | 97.0% | 94.5% | 90.5% | 97.5% | 96.5% | 96.97% |
| **LSVM L1** | 97.0% | 95.5% | 90.0% | 97.5% | 95.5% | 96.97% |
| **LSVM L2** | 97.5% | 91.5% | 90.5% | 97.5% | 96.5% | 97.73% |
| **CHAID** | 96.22% | 95.45% | 94.67% | **98.49%** | 95.46% | 95.46% |
| **C 4.5** | 96.97% | 96.97% | **96.22%** | 93.19% | 62.13% | 62.13% |
| **RF** | **99.5%** | **97.0%** | 91.5% | 98.0% | **97.5%** | **98.49%** |
| **KNN** | 66.5% | 64.5% | 64.0% | 67.0% | ---- | ---- |
| **ANN** | 83.0% | 67.5% | 79.5% | 87.5% | ---- | ---- |
| **Best Accuracy** | **RF** | **RF** | **C 4.5** | **CHAID** | **RF** | **RF** |

Table. 2.1. Accuracy Table of Classifiers in case of different FS

## CHAPTER 3

### SOURCE CODE

**# Data Pre-Processing**

import pandas as pd

data = pd.read\_csv("DataSet.csv")

data.head()

data.info()

dec = data['iSCKD']

list1 = []

for i in data:

if data[i].dtype == 'object':

list1.append(i)

list1

**# Label Encoder**

from sklearn.preprocessing import LabelEncoder

labelencoder = LabelEncoder()

for i in list1:

data[i] = labelencoder.fit\_transform(data[i])

data

type(data)

list2 = [x for x in data]

data.info()

**# KNN Imputer**

from sklearn.impute import KNNImputer

imputer = KNNImputer(n\_neighbors=20,metric = 'nan\_euclidean') #l

DataNoNull= pd.DataFrame(imputer.fit\_transform(data))

Data = pd.DataFrame()

for i in range(len(list2)):

Data[list2[i]] = DataNoNull[i]

Data[list2[i]] = Data[list2[i]].astype(data[list2[i]].dtype)

Data.head()

Data.info()

x\_all\_features = Data.iloc[:,0:24]

y\_all\_features = Data['iSCKD']

**# Train-Test Split**

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split( x\_all\_features, y\_all\_features, test\_size =0.50)

display(x\_train.shape,y\_train.shape,x\_test.shape,y\_test.shape)

**# Heat Map for Full Features**

import seaborn as sns

sns.set(rc={'figure.figsize':(24,14)})

sns.heatmap(data.corr(),annot = True)

**# LR in case of Full Features**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import precision\_score, accuracy\_score, recall\_score,confusion\_matrix, f1\_score, roc\_auc\_score

import sys

lr = LogisticRegression(solver = 'lbfgs', max\_iter = sys.maxsize ) #liblong

lr.fit(x\_train, y\_train)

y\_predict = lr.predict(x\_test)

accuracy\_lr = accuracy\_score(y\_test,y\_predict)

precison\_lr = precision\_score(y\_test, y\_predict,pos\_label= 0 )

recall\_lr = recall\_score(y\_test, y\_predict,pos\_label=0)

fmes\_lr = f1\_score(y\_test,y\_predict,zero\_division =1)

accuracy\_lr\*=100

precison\_lr\*=100

recall\_lr\*=100

error\_lr = 100 - accuracy\_lr

print("Accuracy :",accuracy\_lr)

print("Precision",precison\_lr)

print("Recall :",recall\_lr)

print("Error : ",error\_lr)

print("F1 Score : ",fmes\_lr)

auc\_lr = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_lr)

gini\_lr = 2\*(auc\_lr/100) - 1

print("GINI : ",gini\_lr)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

**# KNN in case of Full features**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=20)

knn.fit(x\_train, y\_train)

y\_predict = knn.predict(x\_test)

accuracy\_knn = accuracy\_score(y\_test,y\_predict)

precison\_knn = precision\_score(y\_test, y\_predict,pos\_label= 0 )

recall\_knn = recall\_score(y\_test, y\_predict,pos\_label= 0)

fmes\_knn = f1\_score(y\_test,y\_predict,zero\_division =1)

accuracy\_knn\*= 100

precison\_knn\*= 100

recall\_knn \*= 100

error\_knn = 100 - accuracy\_knn

print("Accuracy :",accuracy\_knn)

print("Precision :",(precison\_knn))

print("Recall :",recall\_knn )

print("Error : ",error\_knn)

print("F1 Score : ",fmes\_knn)

auc\_knn = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_knn)

gini\_knn = 2\*(auc\_knn/100) - 1

print("GINI : ",gini\_knn)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

**# LSVM L1 in case of Full features**

from sklearn.svm import LinearSVC

lsvmL1 = LinearSVC(dual = False, penalty = 'l1',max\_iter = 10000)

lsvmL1.fit(x\_train, y\_train)

y\_predict = lsvmL1.predict(x\_test)

accuracy\_lsvmL1 = accuracy\_score(y\_test,y\_predict)

precison\_lsvmL1 = precision\_score(y\_test, y\_predict,pos\_label=0 )

recall\_lsvmL1 = recall\_score(y\_test, y\_predict,pos\_label=0)

fmes\_lsvmL1 = f1\_score(y\_test,y\_predict,zero\_division =1)

accuracy\_lsvmL1 \*= 100

precison\_lsvmL1 \*= 100

recall\_lsvmL1\*= 100

error\_lsvmL1 = 100 - accuracy\_lsvmL1

print("Accuracy :",accuracy\_lsvmL1)

print("Precision",precison\_lsvmL1)

print("Recall :",recall\_lsvmL1 ) #max\_iter = sys.maxsize

print("F1 Score : ",fmes\_lsvmL1)

print("Error : ",error\_lsvmL1)

auc\_lsvmL1 = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_lsvmL1)

gini\_lsvmL1 = 2\*(auc\_lsvmL1/100) - 1

print("GINI : ",gini\_lsvmL1)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

**# LSVM L2 in case of Full Features**

from sklearn.svm import LinearSVC

import sys

lsvmL2 = LinearSVC( dual=False,penalty='l2',max\_iter = 10000)

lsvmL2.fit(x\_train, y\_train)

y\_predict = lsvmL2.predict(x\_test)

accuracy\_lsvmL2 = accuracy\_score(y\_test,y\_predict)

precison\_lsvmL2 = precision\_score(y\_test, y\_predict,pos\_label=0 )

recall\_lsvmL2 = recall\_score(y\_test, y\_predict,pos\_label=0)

fmes\_lsvmL2 = f1\_score(y\_test,y\_predict,zero\_division =1)

accuracy\_lsvmL2 \*= 100

precison\_lsvmL2 \*= 100

recall\_lsvmL2 \*= 100

error\_lsvmL2 = 100 - accuracy\_lsvmL2

print("Accuracy :",accuracy\_lsvmL2)

print("Precision",precison\_lsvmL2)

print("Recall :",recall\_lsvmL2 )

print("Error : ",error\_lsvmL2)

auc\_lsvmL2 = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_lsvmL2)

gini\_lsvmL2 = 2\*(auc\_lsvmL2/100) - 1

print("GINI : ",gini\_lsvmL2)

print("F1 Score : ",fmes\_lsvmL2)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

**# RF in case of Full features**

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators= 20, criterion="entropy")

classifier.fit(x\_train, y\_train)

y\_predict= classifier.predict(x\_test)

accuracy\_rf = accuracy\_score(y\_test,y\_predict)

precision\_rf = precision\_score(y\_test, y\_predict,pos\_label=0)

recall\_rf = recall\_score(y\_test, y\_predict,pos\_label=0)

fmes\_rf = f1\_score(y\_test,y\_predict,zero\_division =1)

accuracy\_rf \*= 100

recall\_rf \*= 100

precision\_rf \*=100

error\_lsvmL2 = 100 - accuracy\_rf

print("Accuracy :",accuracy\_rf)

print("Precision",precision\_rf)

print("Recall :",recall\_rf )

print("Error : ",error\_lsvmL2)

print("F Score : ",fmes\_rf)

auc\_rf = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_rf)

gini\_rf = 2\*(auc\_rf/100) - 1

print("GINI : ",gini\_rf)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

Data.columns = [\*Data.columns[:-1], 'Decision']

Data.head()

DataTree = pd.DataFrame()

DataTree = Data

DataTree['Decision'] = dec

DataTree

X\_all\_features = DataTree.iloc[:,0:24]

Y\_all\_features = DataTree['Decision']

x1\_train, x1\_test, y1\_train, y1\_test = train\_test\_split( X\_all\_features, Y\_all\_features, test\_size = 0.33)

display(x1\_train.shape,y1\_train.shape,x1\_test.shape,y1\_test.shape)

x1\_train

y1\_train

df = pd.DataFrame()

df = x1\_train

print(type(df))

df.join(y1\_train)

df.info()

**# CHAID in case of Full Features**

from chefboost import Chefboost as cf

config = {"algorithm" : "CHAID"}

ChaidTree = cf.fit(df.join(y1\_train), config)

print(type(ChaidTree))

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test.index:

pred = cf.predict(ChaidTree,x1\_test.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

tn += 1

y\_predict.append(0)

y\_testing.append(0)

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

accuracy\_chaid = (tp+tn)/(tp+tn+fp+fn)

precision\_chaid = tp/(tp+fp)

recall\_chaid = tp/(tp+fn)

accuracy\_chaid \*= 100

precision\_chaid \*= 100

recall\_chaid \*= 100

print(accuracy\_chaid)

print(precision\_chaid)

print(recall\_chaid)

error\_chaid = 100 - accuracy\_chaid

print("Error : ",error\_chaid)

fmes\_chaid = 2\*((precision\_chaid\*recall\_chaid)/(precision\_chaid+recall\_chaid))/100

print(fmes\_chaid)

auc\_chaid = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_chaid)

gini\_chaid = 2\*(auc\_chaid/100) - 1

print("GINI : ",gini\_chaid)

**# C4.5 in case of Full Features**

config = {"algorithm" : "C4.5"}

C45Tree = cf.fit(df.join(y1\_train), config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test.index:

pred = cf.predict(C45Tree,x1\_test.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

fp+=1

y\_predict.append(1)

y\_testing.append(0)

else:

fn+=1

y\_predict.append(0)

y\_testing.append(1)

accuracy\_c45 = (tp+tn)/(tp+tn+fp+fn)

accuracy\_c45 \*= 100

print(accuracy\_c45)

precision\_c45 = tp/(tp+fp)

precision\_c45 \*= 100

recall\_c45 = tp/(tp+fn)

recall\_c45 \*= 100

print(precision\_c45)

print(recall\_c45)

error\_c45 = 100 - accuracy\_c45

print("Error : ",error\_c45)

fmes\_c45 = 2\*((precision\_c45\*recall\_c45)/(precision\_c45+recall\_c45))/100

print(fmes\_c45)

auc\_c45 = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_c45)

gini\_c45 = 2\*(auc\_c45/100) - 1

print("GINI : ",gini\_c45)

**# ANN in case of Full Features**

import keras

from keras.models import Sequential

from keras.layers import Dense

from sklearn.metrics import confusion\_matrix, accuracy\_score,precision\_score,recall\_score

classifier = Sequential()

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu', input\_dim =24))

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu'))

classifier.add(Dense(units = 1, kernel\_initializer = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

Y\_train = y\_train

Y\_test = y\_test

y\_train = y\_train.astype('category')

y\_train = y\_train.cat.codes

y\_test = y\_test.astype('category')

y\_test = y\_test.cat.codes

classifier.fit(x\_train, y\_train, batch\_size = 10, epochs = 100)

y\_pred = classifier.predict(x\_test)

y\_pred = (y\_pred > 0.5)

accuracy\_ann = accuracy\_score(y\_test,y\_pred)

accuracy\_ann \*= 100

precision\_ann = precision\_score(y\_test,y\_pred)

precision\_ann \*= 100

recall\_ann = recall\_score(y\_test,y\_pred)

recall\_ann \*= 100

print("Accuracy :",accuracy\_ann)

print("Precision",precision\_ann)

print("Recall :",recall\_ann)

error\_ann = 100 - accuracy\_ann

print("Error : ",error\_ann)

fmes\_ann = f1\_score(y\_test,y\_pred,zero\_division =1)

print("F1 Score : ",fmes\_ann)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_pred))

auc\_ann = roc\_auc\_score(y\_test, y\_pred)\*100

print("AUC : ",auc\_ann)

gini\_ann = 2\*(auc\_ann/100) - 1

print("GINI : ",gini\_ann)

**# Evaluation Measures Plotting in case of Full Features**

import matplotlib.pyplot as plt

import numpy as np

barWidth = 0.25

fig = plt.subplots(figsize =(12, 8))

accuracy = [accuracy\_lr,accuracy\_knn,accuracy\_lsvmL1 ,accuracy\_lsvmL2,accuracy\_chaid,accuracy\_c45, accuracy\_ann, accuracy\_rf ]

precision = [precison\_lr,precison\_knn,precison\_lsvmL1,precison\_lsvmL2, precision\_chaid,precision\_c45,precision\_ann, precision\_rf ]

recall = [recall\_lr, recall\_knn, recall\_lsvmL1,recall\_lsvmL2,recall\_chaid, recall\_c45, recall\_ann, recall\_rf ]

br1 = np.arange(len(accuracy))

br2 = [x + barWidth for x in br1]

br3 = [x + barWidth for x in br2]

plt.bar(br1, accuracy, color ='r', width = barWidth,

edgecolor ='grey', label ='Accuracy')

plt.bar(br2, precision,color ='g', width = barWidth,

edgecolor ='grey', label ='Precision')

plt.bar(br3, recall, color ='b', width = barWidth,

edgecolor ='grey', label ='Recall')

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Performance', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(accuracy))],

['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN', 'RF' ] )

plt.title("Accuracy Precision Recall for all features")

plt.legend()

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

f\_mesures = [fmes\_lr, fmes\_knn, fmes\_lsvmL1,fmes\_lsvmL2,fmes\_chaid, fmes\_c45, fmes\_ann, fmes\_rf]

x.bar(algo,f\_mesures)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('F Measure', fontweight ='bold', fontsize = 15)

plt.title("Variation of F Measure for Various Classifiers using Full Features")

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

auc = [auc\_lr, auc\_knn, auc\_lsvmL1,auc\_lsvmL2,auc\_chaid, auc\_c45, auc\_ann, auc\_rf]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('AUC', fontweight ='bold', fontsize = 15)

plt.title("Variation of AUC for Various Classifiers using Full Features")

plt.show()

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

gini = [gini\_lr, gini\_knn, gini\_lsvmL1,gini\_lsvmL2,gini\_chaid, gini\_c45, gini\_ann, gini\_rf]

plt.rcParams["figure.figsize"] = (30,10)

plt.plot(algo, gini)

plt.xlabel("Algorithm", fontweight ='bold', fontsize = 25)

plt.ylabel("GINI",fontweight ='bold', fontsize = 25)

plt.title("Variation of GINI for Various Classifiers using Full Features", fontweight ='bold', fontsize = 25)

plt.show()

err = plt.figure()

x = err.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

errors = [100-accuracy\_lr,100-accuracy\_knn,100-accuracy\_lsvmL1 ,100 accuracy\_lsvmL2,100-accuracy\_chaid,100-accuracy\_c45, 100-accuracy\_ann, 100-accuracy\_rf ]

x.bar(algo,errors)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Error', fontweight ='bold', fontsize = 15)

plt.title("Variation of Error for Various Classifiers using Full Features")

plt.show()

**# CFS**

def correlation(dataSeT, threshold):

col\_corr = set()

corr\_matrix = dataSeT.corr()

for i in range(len(corr\_matrix.columns)):

for j in range(i):

if abs(corr\_matrix.iloc[i,j])>threshold :

col\_corr.add(corr\_matrix.columns[i])

return col\_corr

corr\_features = correlation(x\_train,0.5)

corr\_features

x\_train\_corr = x\_train.drop(corr\_features, axis = 1)

x1\_train\_corr = x1\_train.drop(corr\_features, axis = 1)

x\_test\_corr = x\_test.drop(corr\_features, axis = 1)

x1\_test\_corr = x1\_test.drop(corr\_features, axis = 1)

x1\_train\_corr

y1\_train

x\_train\_corr

y\_train

sns.heatmap(x\_train\_corr.corr(),annot = True)

x\_train\_corr

y1\_train

**# LR**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import precision\_score, accuracy\_score, recall\_score

import sys

lr = LogisticRegression(solver = 'lbfgs', max\_iter = sys.maxsize )

lr.fit(x\_train\_corr, y\_train)

y\_predict = lr.predict(x\_test\_corr)

accuracy\_lr\_corr = accuracy\_score(y\_test,y\_predict)

precison\_lr\_corr = precision\_score(y\_test, y\_predict,pos\_label= 0 )

recall\_lr\_corr = recall\_score(y\_test, y\_predict,pos\_label=0)

fmes\_lr\_corr = f1\_score(y\_test,y\_predict,zero\_division =1)

accuracy\_lr\_corr \*= 100

precison\_lr\_corr \*= 100

recall\_lr\_corr \*= 100

error\_lr\_corr = 100 - accuracy\_lr\_corr

print("Error : ",error\_lr\_corr)

print("Accuracy :",accuracy\_lr\_corr)

print("Precision",precison\_lr\_corr)

print("Recall :",recall\_lr\_corr )

print("F1 Score : ",fmes\_lr\_corr)

auc\_lr\_corr = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_lr\_corr)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

auc\_lr\_corr = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_lr\_corr)

gini\_lr\_corr = 2\*(auc\_lr\_corr/100) - 1

print("GINI : ",gini\_lr\_corr)

**# KNN**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(x\_train\_corr, y\_train)

y\_predict\_corr = knn.predict(x\_test\_corr)

accuracy\_knn\_corr = accuracy\_score(y\_test,y\_predict\_corr)

precison\_knn\_corr = precision\_score(y\_test, y\_predict\_corr,pos\_label= 0 )

recall\_knn\_corr = recall\_score(y\_test, y\_predict\_corr,pos\_label= 0)

fmes\_knn\_corr = f1\_score(y\_test,y\_predict\_corr,zero\_division =1)

accuracy\_knn\_corr \*= 100

precison\_knn\_corr \*= 100

recall\_knn\_corr \*= 100

print("Accuracy :",accuracy\_knn\_corr)

print("Precision",precison\_knn\_corr)

print("Recall :",recall\_knn\_corr )

error\_knn\_corr = 100 - accuracy\_knn\_corr

print("Error : ",error\_knn\_corr)

print("F1 Score : ",fmes\_knn\_corr)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_corr))

print("Correct Predictions :",sum(y\_predict\_corr == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_corr != y\_test))

auc\_knn\_corr = roc\_auc\_score(y\_test, y\_predict\_corr)\*100

print("AUC : ",auc\_knn\_corr)

gini\_knn\_corr = 2\*(auc\_knn\_corr/100) - 1

print("GINI : ",gini\_knn\_corr)

**# LSVM L1**

from sklearn.svm import LinearSVC

lsvmL1 = LinearSVC(dual = False, penalty = 'l1',max\_iter = 10000)

lsvmL1.fit(x\_train\_corr, y\_train)

y\_predict\_corr = lsvmL1.predict(x\_test\_corr)

accuracy\_lsvmL1\_corr = accuracy\_score(y\_test,y\_predict\_corr)

precison\_lsvmL1\_corr = precision\_score(y\_test, y\_predict\_corr,pos\_label=0 )

recall\_lsvmL1\_corr = recall\_score(y\_test, y\_predict\_corr,pos\_label=0)

fmes\_lsvmL1\_corr = f1\_score(y\_test,y\_predict\_corr,zero\_division =1)

accuracy\_lsvmL1\_corr \*= 100

precison\_lsvmL1\_corr \*= 100

recall\_lsvmL1\_corr \*= 100

error\_lsvmL1\_corr = 100 - accuracy\_lsvmL1\_corr

print("Error : ",error\_lsvmL1\_corr)

print("Accuracy :",accuracy\_lsvmL1\_corr)

print("Precision",precison\_lsvmL1\_corr)

print("Recall :",recall\_lsvmL1\_corr ) #max\_iter = sys.maxsize

print("F1 Score : ",fmes\_lsvmL1\_corr)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_corr))

print("Correct Predictions :",sum(y\_predict\_corr == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_corr != y\_test))

auc\_lsvmL1\_corr = roc\_auc\_score(y\_test, y\_predict\_corr)\*100

print("AUC : ",auc\_lsvmL1\_corr)

gini\_lsvmL1\_corr = 2\*(auc\_lsvmL1\_corr/100) - 1

print("GINI : ",gini\_lsvmL1\_corr)

**# LSVM L2**

from sklearn.svm import LinearSVC

import sys

lsvmL2 = LinearSVC( dual=False,penalty='l2',max\_iter = 10000)

lsvmL2.fit(x\_train\_corr, y\_train)

y\_predict\_corr = lsvmL2.predict(x\_test\_corr)

accuracy\_lsvmL2\_corr = accuracy\_score(y\_test,y\_predict\_corr)

precison\_lsvmL2\_corr = precision\_score(y\_test, y\_predict\_corr,pos\_label=0 )

recall\_lsvmL2\_corr = recall\_score(y\_test, y\_predict\_corr,pos\_label=0)

fmes\_lsvmL2\_corr = f1\_score(y\_test,y\_predict\_corr,zero\_division =1)

accuracy\_lsvmL2\_corr \*= 100

precison\_lsvmL2\_corr \*= 100

recall\_lsvmL2\_corr \*= 100

print("Accuracy :",accuracy\_lsvmL2\_corr)

print("Precision",precison\_lsvmL2\_corr)

print("Recall :",recall\_lsvmL2\_corr )

error\_lsvmL2\_corr = 100 - accuracy\_lsvmL2\_corr

print("Error : ",error\_lsvmL2\_corr)

print("F1 Score : ",fmes\_lsvmL2\_corr)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_corr))

print("Correct Predictions :",sum(y\_predict\_corr == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_corr != y\_test))

auc\_lsvmL2\_corr = roc\_auc\_score(y\_test, y\_predict\_corr)\*100

print("AUC : ",auc\_lsvmL2\_corr)

gini\_lsvmL2\_corr = 2\*(auc\_lsvmL2\_corr/100) - 1

print("GINI : ",gini\_lsvmL2\_corr)

**# RF**

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy")

classifier.fit(x\_train\_corr, y\_train)

y\_predict\_corr= classifier.predict(x\_test\_corr)

accuracy\_rf\_corr = accuracy\_score(y\_test,y\_predict\_corr)

precision\_rf\_corr = precision\_score(y\_test, y\_predict\_corr,pos\_label=0)

recall\_rf\_corr = recall\_score(y\_test, y\_predict\_corr,pos\_label=0)

fmes\_rf\_corr = f1\_score(y\_test,y\_predict\_corr,zero\_division =1)

accuracy\_rf\_corr \*= 100

precision\_rf\_corr \*= 100

recall\_rf\_corr \*= 100

print("Accuracy :",accuracy\_rf\_corr)

print("Precision",precision\_rf\_corr)

print("Recall :",recall\_rf\_corr )

error\_rf\_corr = 100 - accuracy\_rf\_corr

print("Error : ",error\_rf\_corr)

print("F1 Score : ",fmes\_rf\_corr)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_corr))

print("Correct Predictions :",sum(y\_predict\_corr == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_corr != y\_test))

auc\_rf\_corr = roc\_auc\_score(y\_test, y\_predict\_corr)\*100

print("AUC : ",auc\_rf\_corr)

gini\_rf\_corr = 2\*(auc\_rf\_corr/100) - 1

print("GINI : ",gini\_rf\_corr)

**# ANN**

import keras

from keras.models import Sequential

from keras.layers import Dense

from sklearn.metrics import confusion\_matrix, accuracy\_score,precision\_score,recall\_score

classifier = Sequential()

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu', input\_dim = 15))

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu'))

classifier.add(Dense(units = 1, kernel\_initializer = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

Y\_train = y\_train

Y\_test = y\_test

y\_train = y\_train.astype('category')

y\_train = y\_train.cat.codes

y\_test = y\_test.astype('category')

y\_test = y\_test.cat.codes

classifier.fit(x\_train\_corr, y\_train, batch\_size = 10, epochs = 100)

y\_pred = classifier.predict(x\_test\_corr)

y\_pred = (y\_pred > 0.5)

accuracy\_ann\_corr = accuracy\_score(y\_test,y\_pred)\*100

precision\_ann\_corr = precision\_score(y\_test,y\_pred)\*100

recall\_ann\_corr = recall\_score(y\_test,y\_pred)\*100

print("Accuracy :",accuracy\_ann\_corr)

print("Precision",precision\_ann\_corr)

print("Recall :",recall\_ann\_corr)

error\_ann\_corr = 100 - accuracy\_ann\_corr

print("Error : ",error\_ann\_corr)

fmes\_ann\_corr = f1\_score(y\_test,y\_pred,zero\_division =1)

print("F1 Score : ",fmes\_ann\_corr)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_pred))

auc\_ann\_corr = roc\_auc\_score(y\_test, y\_pred)\*100

print("AUC : ",auc\_ann\_corr)

gini\_ann\_corr = 2\*(auc\_ann\_corr/100) - 1

print("GINI : ",gini\_ann\_corr)

y1\_train

x1\_train\_corr

y1\_train

**# CHAID**

from chefboost import Chefboost as cf

from pandas.api.types import CategoricalDtype

config = {"algorithm" : "CHAID"}

df = pd.DataFrame()

df = x1\_train\_corr

print(y1\_train)

print(df)

ChaidTree = cf.fit(df.join(y1\_train),config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test.index:

pred = cf.predict(ChaidTree,x1\_test\_corr.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

accuracy\_chaid\_corr = (tp+tn)/(tp+tn+fp+fn)

accuracy\_chaid\_corr \*= 100

precision\_chaid\_corr = tp/(tp+fp)

precision\_chaid\_corr \*= 100

recall\_chaid\_corr = tp/(tp+fn)

recall\_chaid\_corr \*= 100

print(accuracy\_chaid\_corr)

print(precision\_chaid\_corr)

print(recall\_chaid\_corr)

error\_chaid\_corr = 100 - accuracy\_chaid\_corr

print("Error : ",error\_chaid\_corr)

fmes\_chaid\_corr =2\*((precision\_chaid\_corr\*recall\_chaid\_corr)/(precision\_chaid\_corr+recall\_chaid\_corr))/100

print(fmes\_chaid\_corr)

auc\_chaid\_corr = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_chaid\_corr)

gini\_chaid\_corr = 2\*(auc\_chaid\_corr/100) - 1

print("GINI : ",gini\_chaid\_corr)

**# C4.5**

config = {"algorithm" : "C4.5"}

C45Tree = cf.fit(df.join(y1\_train), config)

#print(type(ChaidTree))

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test\_corr.index:

pred = cf.predict(C45Tree,x1\_test\_corr.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

accuracy\_c45\_corr = (tp+tn)/(tp+tn+fp+fn)

accuracy\_c45\_corr \*=100

print(accuracy\_c45\_corr)

precision\_c45\_corr = tp/(tp+fp)

precision\_c45\_corr\*=100

recall\_c45\_corr = tp/(tp+fn)

recall\_c45\_corr\*=100

print(precision\_c45\_corr)

print(recall\_c45\_corr)

error\_c45\_corr = 100 - accuracy\_c45\_corr

print("Error : ",error\_c45\_corr)

fmes\_c45\_corr = 2\*((precision\_c45\_corr\*recall\_c45\_corr)/(precision\_c45\_corr+recall\_c45\_corr))/100

print(fmes\_c45\_corr)

auc\_c45\_corr = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_c45\_corr)

gini\_c45\_corr = 2\*(auc\_c45\_corr/100) - 1

**# Evaluation Measures Plotting in case of CFS**

import matplotlib.pyplot as plt

import numpy as np

barWidth = 0.25

fig2 = plt.subplots(figsize =(12, 8))

accuracy = [accuracy\_lr\_corr,accuracy\_knn\_corr,accuracy\_lsvmL1\_corr ,accuracy\_lsvmL2\_corr,accuracy\_ann\_corr,accuracy\_chaid\_corr,accuracy\_c45\_corr,accuracy\_rf\_corr ]

precision = [precison\_lr\_corr,precison\_knn\_corr,precison\_lsvmL1\_corr,precison\_lsvmL2\_corr, precision\_ann\_corr,precision\_chaid\_corr,precision\_c45\_corr,precision\_rf\_corr ]

recall = [recall\_lr\_corr, recall\_knn\_corr, recall\_lsvmL1\_corr,recall\_lsvmL2\_corr, recall\_ann\_corr,recall\_chaid\_corr, recall\_c45\_corr,recall\_rf\_corr ]

br1 = np.arange(len(accuracy))

br2 = [x + barWidth for x in br1]

br3 = [x + barWidth for x in br2]

plt.bar(br1, accuracy, color ='r', width = barWidth,

edgecolor ='grey', label ='Accuracy')

plt.bar(br2, precision,color ='g', width = barWidth,

edgecolor ='grey', label ='Precision')

plt.bar(br3, recall, color ='b', width = barWidth,

edgecolor ='grey', label ='Recall')

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Performance', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(accuracy))],

['LR', 'KNN', 'LSVML1', 'LSVML2', 'ANN', 'CHAID', 'C4.5', 'RF' ] )

plt.legend()

plt.title("Accuracy Precision Recall for correlation based feature selection")

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

f\_mesures = [fmes\_lr\_corr, fmes\_knn\_corr, fmes\_lsvmL1\_corr,fmes\_lsvmL2\_corr,fmes\_chaid\_corr, fmes\_c45\_corr, fmes\_ann\_corr, fmes\_rf\_corr]

x.bar(algo,f\_mesures)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('F Measure', fontweight ='bold', fontsize = 15)

plt.title("Variation of F Measure for Various Classifiers using Correlation Based Feature selection")

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

auc = [auc\_lr\_corr, auc\_knn\_corr, auc\_lsvmL1\_corr,auc\_lsvmL2\_corr,auc\_chaid\_corr, auc\_c45\_corr, auc\_ann\_corr, auc\_rf\_corr]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('AUC', fontweight ='bold', fontsize = 15)

plt.title("Variation of AUC for Various Classifiers using Correlation Based Feature selection")

plt.show()

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

gini = [gini\_lr\_corr, gini\_knn\_corr, gini\_lsvmL1\_corr,gini\_lsvmL2\_corr,gini\_chaid\_corr, gini\_c45\_corr, gini\_ann\_corr,gini\_rf\_corr]

plt.rcParams["figure.figsize"] = (30,10)

plt.plot(algo, gini)

plt.xlabel("Algorithm", fontweight ='bold', fontsize = 25)

plt.ylabel("GINI", fontweight ='bold', fontsize = 25)

plt.title("Variation of GINI for Various Classifiers using Correlation Based Feature selection", fontweight ='bold', fontsize = 25)

plt.show()

error = plt.figure()

x = error.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

auc = [100-accuracy\_lr\_corr,100-accuracy\_knn\_corr,100-accuracy\_lsvmL1\_corr ,100-accuracy\_lsvmL2\_corr,100-accuracy\_chaid\_corr,100-accuracy\_c45\_corr,100-accuracy\_ann\_corr,100-accuracy\_rf\_corr ]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Error', fontweight ='bold', fontsize = 15)

plt.title("Variation of Error for Various Classifiers using Correlation Based Feature selection")

plt.show()

Data.info()

**# Wrapper FS**

x\_lasso = Data.iloc[:,0:24]

y\_lasso = Data['Decision']

y\_lasso = y\_lasso.astype('category')

y\_lasso = y\_lasso.cat.codes

import statsmodels.api as sm

def forward\_selection(data, target, significance\_level=0.05):

initial\_features = data.columns.tolist()

best\_features = []

while (len(initial\_features)>0):

remaining\_features = list(set(initial\_features)-set(best\_features))

new\_pval = pd.Series(index=remaining\_features,dtype = "float64")

pd.to\_numeric(new\_pval, errors = 'coerce')

for new\_column in remaining\_features:

model = sm.OLS(target.astype(float), sm.add\_constant(data[best\_features+[new\_column]]).astype(float)).fit()

new\_pval[new\_column] = model.pvalues[new\_column]

min\_p\_value = new\_pval.min()

if(float(min\_p\_value)<significance\_level):

best\_features.append(new\_pval.idxmin())

else:

break

return best\_features

X = forward\_selection(x\_lasso,y\_lasso)

print(X)

x\_train\_wrp = x\_train.drop(X, axis = 1)

x1\_train\_wrp = x1\_train.drop(X, axis = 1)

x\_test\_wrp = x\_test.drop(X, axis = 1)

x1\_test\_wrp = x1\_test.drop(X, axis = 1)

x\_train\_wrp

y\_train

sns.heatmap(x\_train\_wrp.corr(),annot = True)

**# LR**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import precision\_score, accuracy\_score, recall\_score

import sys

lr = LogisticRegression(solver = 'lbfgs', max\_iter = sys.maxsize )

lr.fit(x\_train\_wrp, y\_train)

y\_predict = lr.predict(x\_test\_wrp)

accuracy\_lr\_wrp = accuracy\_score(y\_test,y\_predict)\*100

precison\_lr\_wrp = precision\_score(y\_test, y\_predict,pos\_label= 0 )\*100

recall\_lr\_wrp = recall\_score(y\_test, y\_predict,pos\_label=0)\*100

fmes\_lr\_wrp = f1\_score(y\_test,y\_predict,zero\_division =1)

print("Accuracy :",accuracy\_lr\_wrp)

print("Precision",precison\_lr\_wrp)

print("Recall :",recall\_lr\_wrp )

error\_lr\_wrp = 100 - accuracy\_lr\_wrp

print("Error : ",error\_lr\_wrp)

print("F1 Score : ",fmes\_lr\_wrp)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

auc\_lr\_wrp = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_lr\_wrp)

gini\_lr\_wrp = 2\*(auc\_lr\_wrp/100) - 1

print("GINI : ",gini\_lr\_wrp)

**# KNN**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(x\_train\_wrp, y\_train)

y\_predict\_wrp = knn.predict(x\_test\_wrp)

accuracy\_knn\_wrp = accuracy\_score(y\_test,y\_predict\_wrp)\*100

precison\_knn\_wrp = precision\_score(y\_test, y\_predict\_wrp,pos\_label= 0 )\*100

recall\_knn\_wrp = recall\_score(y\_test, y\_predict\_wrp,pos\_label= 0)\*100

fmes\_knn\_wrp = f1\_score(y\_test,y\_predict\_wrp,zero\_division =1)

print("Accuracy :",accuracy\_knn\_wrp)

print("Precision",precison\_knn\_wrp)

print("Recall :",recall\_knn\_wrp )

error\_knn\_wrp = 100 - accuracy\_knn\_wrp

print("Error : ",error\_knn\_wrp)

print("F1 Score : ",fmes\_knn\_wrp)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_wrp))

print("Correct Predictions :",sum(y\_predict\_wrp == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_wrp != y\_test))

auc\_knn\_wrp = roc\_auc\_score(y\_test, y\_predict\_wrp)\*100

print("AUC : ",auc\_knn\_wrp)

gini\_knn\_wrp = 2\*(auc\_knn\_wrp/100) - 1

print("GINI : ",gini\_knn\_wrp)

**# LSVM L1**

from sklearn.svm import LinearSVC

lsvmL1 = LinearSVC(dual = False, penalty = 'l1',max\_iter = 10000)

lsvmL1.fit(x\_train\_wrp, y\_train)

y\_predict\_wrp = lsvmL1.predict(x\_test\_wrp)

accuracy\_lsvmL1\_wrp = accuracy\_score(y\_test,y\_predict\_wrp)\*100

precison\_lsvmL1\_wrp = precision\_score(y\_test, y\_predict\_wrp,pos\_label=0 )\*100

recall\_lsvmL1\_wrp = recall\_score(y\_test, y\_predict\_wrp,pos\_label=0)\*100

fmes\_lsvmL1\_wrp = f1\_score(y\_test,y\_predict\_wrp,zero\_division =1)

print("Accuracy :",accuracy\_lsvmL1\_wrp)

print("Precision",precison\_lsvmL1\_wrp)

print("Recall :",recall\_lsvmL1\_wrp ) #max\_iter = sys.maxsize

error\_lsvmL1\_wrp = 100 - accuracy\_lsvmL1\_wrp

print("Error : ",error\_lsvmL1\_wrp)

print("F1 Score : ",fmes\_lsvmL1\_wrp)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_wrp))

print("Correct Predictions :",sum(y\_predict\_wrp == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_wrp != y\_test))

auc\_lsvmL1\_wrp = roc\_auc\_score(y\_test, y\_predict\_wrp)\*100

print("AUC : ",auc\_lsvmL1\_wrp)

gini\_lsvmL1\_wrp = 2\*(auc\_lsvmL1\_wrp/100) - 1

print("GINI : ",gini\_lsvmL1\_wrp)

**# LSVM L2**

from sklearn.svm import LinearSVC

import sys

lsvmL2 = LinearSVC( dual=False,penalty='l2',max\_iter = 120000)

lsvmL2.fit(x\_train\_wrp, y\_train)

y\_predict\_wrp = lsvmL2.predict(x\_test\_wrp)

accuracy\_lsvmL2\_wrp = accuracy\_score(y\_test,y\_predict\_wrp)\*100

precison\_lsvmL2\_wrp = precision\_score(y\_test, y\_predict\_wrp,pos\_label=0 )\*100

recall\_lsvmL2\_wrp = recall\_score(y\_test, y\_predict\_wrp,pos\_label=0)\*100

fmes\_lsvmL2\_wrp = f1\_score(y\_test,y\_predict\_wrp,zero\_division =1)

print("Accuracy :",accuracy\_lsvmL2\_wrp)

print("Precision",precison\_lsvmL2\_wrp)

print("Recall :",recall\_lsvmL2\_wrp )

error\_lsvmL2\_wrp = 100 - accuracy\_lsvmL2\_wrp

print("Error : ",error\_lsvmL2\_wrp)

print("F1 Score : ",fmes\_lsvmL2\_wrp)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_wrp))

auc\_lsvmL2\_wrp = roc\_auc\_score(y\_test, y\_predict\_wrp)\*100

print("AUC : ",auc\_lsvmL2\_wrp)

print("Correct Predictions :",sum(y\_predict\_wrp == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_wrp != y\_test))

gini\_lsvmL2\_wrp = 2\*(auc\_lsvmL2\_wrp/100) - 1

print("GINI : ",gini\_lsvmL2\_wrp)

**# RF**

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy")

classifier.fit(x\_train\_wrp, y\_train)

y\_predict\_wrp= classifier.predict(x\_test\_wrp)

accuracy\_rf\_wrp = accuracy\_score(y\_test,y\_predict\_wrp)\*100

precision\_rf\_wrp = precision\_score(y\_test, y\_predict\_wrp,pos\_label=0)\*100

recall\_rf\_wrp = recall\_score(y\_test, y\_predict\_wrp,pos\_label=0)\*100

fmes\_rf\_wrp = f1\_score(y\_test,y\_predict\_wrp,zero\_division =1)

print("Accuracy :",accuracy\_rf\_wrp)

print("Precision",precision\_rf\_wrp)

print("Recall :",recall\_rf\_wrp )

error\_rf\_wrp = 100 - accuracy\_rf\_wrp

print("Error : ",error\_rf\_wrp)

print("F1 Score : ",fmes\_rf\_wrp)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_wrp))

print("Correct Predictions :",sum(y\_predict\_wrp == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_wrp != y\_test))

auc\_rf\_wrp = roc\_auc\_score(y\_test, y\_predict\_wrp)\*100

print("AUC : ",auc\_rf\_wrp)

gini\_rf\_wrp = 2\*(auc\_rf\_wrp/100) - 1

print("GINI : ",gini\_rf\_wrp)

**# ANN**

import keras

from keras.models import Sequential

from keras.layers import Dense

from sklearn.metrics import confusion\_matrix, accuracy\_score,precision\_score,recall\_score

classifier = Sequential()

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu', input\_dim = 14))

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu'))

classifier.add(Dense(units = 1, kernel\_initializer = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

Y\_train = y\_train

Y\_test = y\_test

y\_train = y\_train.astype('category')

y\_train = y\_train.cat.codes

y\_test = y\_test.astype('category')

y\_test = y\_test.cat.codes

classifier.fit(x\_train\_wrp, y\_train, batch\_size = 10, epochs = 100)

y\_pred = classifier.predict(x\_test\_wrp)

y\_pred = (y\_pred > 0.5)

accuracy\_ann\_wrp = accuracy\_score(y\_test,y\_pred)\*100

precision\_ann\_wrp = precision\_score(y\_test,y\_pred)\*100

recall\_ann\_wrp = recall\_score(y\_test,y\_pred)\*100

print("Accuracy :",accuracy\_ann\_wrp)

print("Precision",precision\_ann\_wrp)

print("Recall :",recall\_ann\_wrp)

error\_ann\_wrp = 100 - accuracy\_ann\_wrp

print("Error : ",error\_ann\_wrp)

fmes\_ann\_wrp = f1\_score(y\_test,y\_predict,zero\_division =1)

print("F1 Score : ",fmes\_ann\_wrp)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_pred))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

auc\_ann\_wrp = roc\_auc\_score(y\_test, y\_predict\_wrp)\*100

print("AUC : ",auc\_ann\_wrp)

gini\_ann\_wrp = 2\*(auc\_ann\_wrp/100) - 1

print("GINI : ",gini\_ann\_wrp)

**# CHAID**

from chefboost import Chefboost as cf

from pandas.api.types import CategoricalDtype

config = {"algorithm" : "CHAID"}

df = pd.DataFrame()

df = x1\_train\_wrp

print(df)

ChaidTree = cf.fit(df.join(y1\_train),config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test.index:

pred = cf.predict(ChaidTree,x1\_test\_wrp.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

accuracy\_chaid\_wrp = ((tp+tn)/(tp+tn+fp+fn))\*100

precision\_chaid\_wrp = (tp/(tp+fp))\*100

recall\_chaid\_wrp = (tp/(tp+fn))\*100

print(accuracy\_chaid\_wrp)

print(precision\_chaid\_wrp)

print(recall\_chaid\_wrp)

error\_chaid\_wrp = 100 - accuracy\_chaid\_wrp

print("Error : ",error\_chaid\_wrp)

fmes\_chaid\_wrp = 2\*(precision\_chaid\_wrp\*recall\_chaid\_wrp)/((precision\_chaid\_wrp+recall\_chaid\_wrp)\*100)

print(fmes\_chaid\_wrp)

auc\_chaid\_wrp = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_chaid\_wrp)

gini\_chaid\_wrp = 2\*(auc\_chaid\_wrp/100) - 1

print("GINI : ",gini\_chaid\_wrp)

**# C4.5**

config = {"algorithm" : "C4.5"}

C45Tree = cf.fit(df.join(y1\_train), config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test\_corr.index:

pred = cf.predict(C45Tree,x1\_test\_wrp.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

accuracy\_c45\_wrp = (tp+tn)\*100/(tp+tn+fp+fn)

print(accuracy\_c45\_wrp)

precision\_c45\_wrp = tp\*100/(tp+fp)

recall\_c45\_wrp = tp\*100/(tp+fn)

print(precision\_c45\_wrp)

print(recall\_c45\_wrp )

error\_c45\_wrp = 100 - accuracy\_c45\_wrp

print("Error : ",error\_c45\_wrp)

fmes\_c45\_wrp = 2\*(precision\_c45\_wrp\*recall\_c45\_wrp)/((precision\_c45\_wrp+recall\_c45\_wrp)\*100)

print(fmes\_c45\_wrp)

auc\_c45\_wrp = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_c45\_wrp)

gini\_c45\_wrp = 2\*(auc\_c45\_wrp/100) - 1

print("GINI : ",gini\_c45\_wrp)

**# Evaluation Measures Plotting in case of Wrapper FS**

fig2 = plt.subplots(figsize =(12, 8))

accuracy = [accuracy\_lr\_wrp,accuracy\_knn\_wrp,accuracy\_lsvmL1\_wrp ,accuracy\_lsvmL2\_wrp,accuracy\_ann\_wrp,accuracy\_chaid\_wrp,accuracy\_c45\_wrp,accuracy\_rf\_wrp ]

precision = [precison\_lr\_wrp,precison\_knn\_wrp,precison\_lsvmL1\_wrp,precison\_lsvmL2\_wrp, precision\_ann\_wrp,precision\_chaid\_wrp,precision\_c45\_wrp,precision\_rf\_wrp]

recall = [recall\_lr\_wrp, recall\_knn\_wrp, recall\_lsvmL1\_wrp,recall\_lsvmL2\_wrp, recall\_ann\_wrp,recall\_chaid\_wrp, recall\_c45\_wrp,recall\_rf\_wrp ]

br1 = np.arange(len(accuracy))

br2 = [x + barWidth for x in br1]

br3 = [x + barWidth for x in br2]

plt.bar(br1, accuracy, color ='r', width = barWidth,

edgecolor ='grey', label ='Accuracy')

plt.bar(br2, precision,color ='g', width = barWidth,

edgecolor ='grey', label ='Precision')

plt.bar(br3, recall, color ='b', width = barWidth,

edgecolor ='grey', label ='Recall')

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Performance', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(accuracy))],

['LR', 'KNN', 'LSVML1', 'LSVML2', 'ANN', 'CHAID', 'C4.5', 'RF' ] )

plt.title("Accuracy Precision Recall for Wrapper feature selection")

plt.legend()

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

f\_mesures = [fmes\_lr\_wrp, fmes\_knn\_wrp, fmes\_lsvmL1\_wrp,fmes\_lsvmL2\_wrp,fmes\_chaid\_wrp, fmes\_c45\_wrp, fmes\_ann\_wrp,fmes\_rf\_wrp]

x.bar(algo,f\_mesures)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('F Measure', fontweight ='bold', fontsize = 15)

plt.title("Variation of F Measure for Various Classifiers using Wrapper Feature Selection")

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

auc = [auc\_lr\_wrp, auc\_knn\_wrp, auc\_lsvmL1\_wrp,auc\_lsvmL2\_wrp,auc\_chaid\_wrp, auc\_c45\_wrp, auc\_ann\_wrp,auc\_rf\_wrp]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('AUC', fontweight ='bold', fontsize = 15)

plt.title("Variation of AUC for Various Classifiers using Wrapper Feature Selection")

plt.show()

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

gini = [gini\_lr\_wrp, gini\_knn\_wrp, gini\_lsvmL1\_wrp,gini\_lsvmL2\_wrp,gini\_chaid\_wrp, gini\_c45\_wrp, gini\_ann\_wrp,gini\_rf\_wrp]

plt.rcParams["figure.figsize"] = (30,10)

plt.plot(algo, gini)

plt.xlabel("Algorithm", fontweight ='bold', fontsize = 25)

plt.ylabel("GINI", fontweight ='bold', fontsize = 25)

plt.title("Variation of GINI for Various Classifiers using Wrapper Feature Selection", fontweight ='bold', fontsize = 25)

plt.show()

error = plt.figure()

x = error.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

auc = [100-accuracy\_lr\_wrp,100-accuracy\_knn\_wrp,100-accuracy\_lsvmL1\_wrp ,100-accuracy\_lsvmL2\_wrp,100-accuracy\_chaid\_wrp,100-accuracy\_c45\_wrp,100-accuracy\_ann\_wrp,100-accuracy\_rf\_wrp ]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Error', fontweight ='bold', fontsize = 15)

plt.title("Variation of Error for Various Classifiers using Wrapper Feature Selection")

plt.show()

**# LASSO FS**

from sklearn.linear\_model import Lasso

features = [i for i in data]

features.remove('iSCKD')

print(features)

lasso = Lasso(alpha = 0.01)

lasso.fit(x\_train,y\_train)

coeff=lasso.coef\_

for i in range(len(coeff)):

coeff[i] = abs(coeff[i])

df\_lasso = pd.DataFrame({"Features":features,"Coefficient":coeff})

df\_lasso.sort\_values("Coefficient")

list\_lasso = []

for i in range(len(df\_lasso)):

if df\_lasso['Coefficient'][i] == 0:

list\_lasso.append(df\_lasso['Features'][i])

list\_lasso

x\_train\_lsso = x\_train.drop(list\_lasso, axis = 1)

x1\_train\_lsso = x1\_train.drop(list\_lasso, axis = 1)

x\_test\_lsso = x\_test.drop(list\_lasso, axis = 1)

x1\_test\_lsso = x1\_test.drop(list\_lasso, axis = 1)

sns.heatmap(x\_train\_lsso.corr(),annot = True)

**# LR**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import precision\_score, accuracy\_score, recall\_score

import sys

lr = LogisticRegression(solver = 'lbfgs', max\_iter = sys.maxsize )

lr.fit(x\_train\_lsso, y\_train)

y\_predict = lr.predict(x\_test\_lsso)

accuracy\_lr\_lsso = accuracy\_score(y\_test,y\_predict)\*100

precison\_lr\_lsso = precision\_score(y\_test, y\_predict,pos\_label= 0 )\*100

recall\_lr\_lsso = recall\_score(y\_test, y\_predict,pos\_label=0)\*100

fmes\_lr\_lsso = f1\_score(y\_test,y\_predict,zero\_division =1)

print("Accuracy :",accuracy\_lr\_lsso)

print("Precision",precison\_lr\_lsso)

print("Recall :",recall\_lr\_lsso )

error\_lr\_lsso = 100 - accuracy\_lr\_lsso

print("Error : ",error\_lr\_lsso)

print("F1 Score : ",fmes\_lr\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

auc\_lr\_lsso = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_lr\_lsso)

gini\_lr\_lsso = 2\*(auc\_lr\_lsso/100) - 1

print("GINI : ",gini\_lr\_lsso)

**# KNN**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(x\_train\_lsso, y\_train)

y\_predict\_lsso = knn.predict(x\_test\_lsso)

accuracy\_knn\_lsso = accuracy\_score(y\_test,y\_predict\_lsso)\*100

precison\_knn\_lsso = precision\_score(y\_test, y\_predict\_lsso,pos\_label= 0 )\*100

recall\_knn\_lsso = recall\_score(y\_test, y\_predict\_lsso,pos\_label= 0)\*100

fmes\_knn\_lsso = f1\_score(y\_test,y\_predict\_lsso,zero\_division =1)

print("Accuracy :",accuracy\_knn\_lsso)

print("Precision",precison\_knn\_lsso)

print("Recall :",recall\_knn\_lsso )

error\_knn\_lsso = 100 - accuracy\_knn\_lsso

print("Error : ",error\_knn\_lsso)

print("F1 Score : ",fmes\_knn\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_lsso))

print("Correct Predictions :",sum(y\_predict\_lsso == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_lsso != y\_test))

auc\_knn\_lsso = roc\_auc\_score(y\_test, y\_predict\_lsso)\*100

print("AUC : ",auc\_knn\_lsso)

gini\_knn\_lsso = 2\*(auc\_knn\_lsso/100) - 1

print("GINI : ",gini\_knn\_lsso)

**# LSVM L1**

from sklearn.svm import LinearSVC

lsvmL1 = LinearSVC(dual = False, penalty = 'l1',max\_iter = 10000)

lsvmL1.fit(x\_train\_lsso, y\_train)

y\_predict\_lsso = lsvmL1.predict(x\_test\_lsso)

accuracy\_lsvmL1\_lsso = accuracy\_score(y\_test,y\_predict\_lsso)\*100

precison\_lsvmL1\_lsso= precision\_score(y\_test, y\_predict\_lsso,pos\_label=0 )\*100

recall\_lsvmL1\_lsso = recall\_score(y\_test, y\_predict\_lsso,pos\_label=0)\*100

fmes\_lsvmL1\_lsso = f1\_score(y\_test,y\_predict\_lsso,zero\_division =1)

print("Accuracy :",accuracy\_lsvmL1\_lsso)

print("Precision",precison\_lsvmL1\_lsso)

print("Recall :",recall\_lsvmL1\_lsso) #max\_iter = sys.maxsize

error\_lsvmL1\_lsso = 100 - accuracy\_lsvmL1\_lsso

print("Error : ",error\_lsvmL1\_lsso)

print("F1 Score : ",fmes\_lsvmL1\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_lsso))

print("Correct Predictions :",sum(y\_predict\_lsso == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_lsso != y\_test))

auc\_lsvmL1\_lsso = roc\_auc\_score(y\_test, y\_predict\_lsso)\*100

print("AUC : ",auc\_lsvmL1\_lsso)

gini\_lsvmL1\_lsso = 2\*(auc\_lsvmL1\_lsso/100) - 1

print("GINI : ",gini\_lsvmL1\_lsso)

**# LSVM L2**

from sklearn.svm import LinearSVC

import sys

lsvmL2 = LinearSVC( dual=False,penalty='l2',max\_iter = 120000)

lsvmL2.fit(x\_train\_lsso, y\_train)

y\_predict\_lsso = lsvmL2.predict(x\_test\_lsso)

accuracy\_lsvmL2\_lsso = accuracy\_score(y\_test,y\_predict\_lsso)\*100

precison\_lsvmL2\_lsso = precision\_score(y\_test, y\_predict\_lsso,pos\_label=0 )\*100

recall\_lsvmL2\_lsso = recall\_score(y\_test, y\_predict\_lsso,pos\_label=0)\*100

fmes\_lsvmL2\_lsso = f1\_score(y\_test,y\_predict\_lsso,zero\_division =1)

print("Accuracy :",accuracy\_lsvmL2\_lsso)

print("Precision",precison\_lsvmL2\_lsso)

print("Recall :",recall\_lsvmL2\_lsso)

error\_lsvmL2\_lsso = 100 - accuracy\_lsvmL2\_lsso

print("Error : ",error\_lsvmL2\_lsso)

print("F1 Score : ",fmes\_lsvmL2\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_lsso))

print("Correct Predictions :",sum(y\_predict\_lsso == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_lsso != y\_test))

auc\_lsvmL2\_lsso = roc\_auc\_score(y\_test, y\_predict\_lsso)\*100

print("AUC : ",auc\_lsvmL2\_lsso)

gini\_lsvmL2\_lsso = 2\*(auc\_lsvmL2\_lsso/100) - 1

print("GINI : ",gini\_lsvmL2\_lsso)

**# RF**

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy")

classifier.fit(x\_train\_lsso, y\_train)

y\_predict\_lsso= classifier.predict(x\_test\_lsso)

accuracy\_rf\_lsso = accuracy\_score(y\_test,y\_predict\_lsso)\*100

precision\_rf\_lsso= precision\_score(y\_test, y\_predict\_lsso,pos\_label=0)\*100

recall\_rf\_lsso = recall\_score(y\_test, y\_predict\_lsso,pos\_label=0)\*100

fmes\_rf\_lsso = f1\_score(y\_test,y\_predict\_lsso,zero\_division =1)

print("Accuracy :",accuracy\_rf\_lsso)

print("Precision",precision\_rf\_lsso)

print("Recall :",recall\_rf\_lsso)

error\_rf\_lsso = 100 - accuracy\_rf\_lsso

print("Error : ",error\_rf\_lsso)

print("F1 Score : ",fmes\_rf\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict\_lsso))

print("Correct Predictions :",sum(y\_predict\_lsso == y\_test))

print("Incorrect Predictions :",sum(y\_predict\_lsso != y\_test))

auc\_rf\_lsso = roc\_auc\_score(y\_test, y\_predict\_lsso)\*100

print("AUC : ",auc\_rf\_lsso)

gini\_rf\_lsso = 2\*(auc\_rf\_lsso/100) - 1

print("GINI : ",gini\_rf\_lsso)

**# ANN**

import keras

from keras.models import Sequential

from keras.layers import Dense

from sklearn.metrics import confusion\_matrix, accuracy\_score,precision\_score,recall\_score

classifier = Sequential()

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu', input\_dim = 16))

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu'))

classifier.add(Dense(units = 1, kernel\_initializer = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

Y\_train = y\_train

Y\_test = y\_test

y\_train = y\_train.astype('category')

y\_train = y\_train.cat.codes

y\_test = y\_test.astype('category')

y\_test = y\_test.cat.codes

classifier.fit(x\_train\_lsso, y\_train, batch\_size = 10, epochs = 100)

y\_pred = classifier.predict(x\_test\_lsso)

y\_pred = (y\_pred > 0.5)

accuracy\_ann\_lsso = accuracy\_score(y\_test,y\_pred)\*100

precision\_ann\_lsso = precision\_score(y\_test,y\_pred)\*100

recall\_ann\_lsso = recall\_score(y\_test,y\_pred)\*100

print("Accuracy :",accuracy\_ann\_lsso)

print("Precision",precision\_ann\_lsso)

print("Recall :",recall\_ann\_lsso)

error\_ann\_lsso = 100 - accuracy\_ann\_lsso

print("Error : ",error\_ann\_lsso)

fmes\_ann\_lsso = f1\_score(y\_test,y\_pred,zero\_division =1)

print("F1 Score : ",fmes\_ann\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_pred))

auc\_ann\_lsso = roc\_auc\_score(y\_test, y\_pred)\*100

print("AUC : ",auc\_ann\_lsso)

gini\_ann\_lsso = 2\*(auc\_ann\_lsso/100) - 1

print("GINI : ",gini\_ann\_lsso)

**# CHAID**

from chefboost import Chefboost as cf

from pandas.api.types import CategoricalDtype

config = {"algorithm" : "CHAID"}

df = pd.DataFrame()

df = x1\_train\_lsso

ChaidTree = cf.fit(df.join(y1\_train),config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test.index:

pred = cf.predict(ChaidTree,x1\_test\_lsso.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

accuracy\_chaid\_lsso = (tp+tn)\*100/(tp+tn+fp+fn)

precision\_chaid\_lsso = tp\*100/(tp+fp)

recall\_chaid\_lsso = tp\*100/(tp+fn)

print(accuracy\_chaid\_lsso)

print(precision\_chaid\_lsso)

print(recall\_chaid\_lsso)

error\_chaid\_lsso = 100 - accuracy\_chaid\_lsso

print("Error : ",error\_chaid\_lsso)

fmes\_chaid\_lsso = 2\*(precision\_chaid\_lsso\*recall\_chaid\_lsso)/((precision\_chaid\_lsso+recall\_chaid\_lsso)\*100)

print(fmes\_chaid\_lsso)

auc\_chaid\_lsso = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_chaid\_lsso)

gini\_chaid\_lsso = 2\*(auc\_chaid\_lsso/100) - 1

print("GINI : ",gini\_chaid\_lsso)

**# C4.5**

config = {"algorithm" : "C4.5"}

C45Tree = cf.fit(df.join(y1\_train), config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test\_corr.index:

pred = cf.predict(C45Tree,x1\_test\_lsso.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

accuracy\_c45\_lsso = (tp+tn)\*100/(tp+tn+fp+fn)

print(accuracy\_c45\_lsso)

precision\_c45\_lsso = tp\*100/(tp+fp)

recall\_c45\_lsso = tp\*100/(tp+fn)

print(precision\_c45\_lsso)

print(recall\_c45\_lsso )

error\_c45\_lsso = 100 - accuracy\_c45\_lsso

print("Error : ",error\_c45\_lsso)

fmes\_c45\_lsso = 2\*(precision\_c45\_lsso\*recall\_c45\_lsso)/((precision\_c45\_lsso+recall\_c45\_lsso)\*100)

print(fmes\_c45\_lsso)

auc\_c45\_lsso = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_c45\_lsso)

gini\_c45\_lsso = 2\*(auc\_c45\_lsso/100) - 1

print("GINI : ",gini\_c45\_lsso)

**# Evaluation Measures Plotting in case of LASSO FS**

fig2 = plt.subplots(figsize =(12, 8))

accuracy = [accuracy\_lr\_lsso,accuracy\_knn\_lsso,accuracy\_lsvmL1\_lsso ,accuracy\_lsvmL2\_lsso,accuracy\_ann\_lsso,accuracy\_chaid\_lsso,accuracy\_c45\_lsso,accuracy\_rf\_lsso]

precision = [precison\_lr\_lsso,precison\_knn\_lsso,precison\_lsvmL1\_lsso,precison\_lsvmL2\_lsso, precision\_ann\_lsso,precision\_chaid\_lsso,precision\_c45\_lsso,precision\_rf\_lsso ]

recall = [recall\_lr\_lsso, recall\_knn\_lsso, recall\_lsvmL1\_lsso,recall\_lsvmL2\_lsso, recall\_ann\_lsso,recall\_chaid\_lsso, recall\_c45\_lsso,recall\_rf\_lsso ]

br1 = np.arange(len(accuracy))

br2 = [x + barWidth for x in br1]

br3 = [x + barWidth for x in br2]

plt.bar(br1, accuracy, color ='r', width = barWidth,

edgecolor ='grey', label ='Accuracy')

plt.bar(br2, precision,color ='g', width = barWidth,

edgecolor ='grey', label ='Precision')

plt.bar(br3, recall, color ='b', width = barWidth,

edgecolor ='grey', label ='Recall')

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Performance', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(accuracy))],

['LR', 'KNN', 'LSVML1', 'LSVML2', 'ANN', 'CHAID', 'C4.5','RF' ] )

plt.title("Accuracy Precision Recall for Lasso feature selection")

plt.legend()

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

f\_mesures = [fmes\_lr\_lsso, fmes\_knn\_lsso, fmes\_lsvmL1\_lsso,fmes\_lsvmL2\_lsso,fmes\_chaid\_lsso, fmes\_c45\_lsso, fmes\_ann\_lsso,fmes\_rf\_lsso]

x.bar(algo,f\_mesures)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('F Measure', fontweight ='bold', fontsize = 15)

plt.title("Variation of F Measure for Various Classifiers using Lasso Feature selection")

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

auc = [auc\_lr\_lsso, auc\_knn\_lsso, auc\_lsvmL1\_lsso,auc\_lsvmL2\_lsso,auc\_chaid\_lsso, auc\_c45\_lsso, auc\_ann\_lsso,auc\_rf\_lsso]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('AUC', fontweight ='bold', fontsize = 15)

plt.title("Variation of AUC for Various Classifiers using Lasso Feature selection")

plt.show()

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

gini = [gini\_lr\_lsso, gini\_knn\_lsso, gini\_lsvmL1\_lsso,gini\_lsvmL2\_lsso,gini\_chaid\_lsso, gini\_c45\_lsso, gini\_ann\_lsso,gini\_rf\_lsso]

plt.rcParams["figure.figsize"] = (30,10)

plt.plot(algo, gini)

plt.xlabel("Algorithm", fontweight ='bold', fontsize = 25)

plt.ylabel("GINI", fontweight ='bold', fontsize = 25)

plt.title("Variation of GINI for Various Classifiers using Lasso Feature selection", fontweight ='bold', fontsize = 25)

plt.show()

error = plt.figure()

x = error.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'KNN', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','ANN','RF']

auc = [100-accuracy\_lr\_lsso,100-accuracy\_knn\_lsso,100-accuracy\_lsvmL1\_lsso ,100-accuracy\_lsvmL2\_lsso,100-accuracy\_chaid\_lsso,100-accuracy\_c45\_lsso,100-accuracy\_ann\_lsso,100-accuracy\_rf\_lsso]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Error', fontweight ='bold', fontsize = 15)

plt.title("Variation of Error for Various Classifiers using Lasso Feature selection")

plt.show()

**# SMOTE**

Data.head()

Data

y\_train

np.bincount(y\_train)

np.bincount(y\_test)

from imblearn.over\_sampling import SMOTE

sm = SMOTE(random\_state=42)

x1\_smote\_train, y1\_smote\_train = sm.fit\_resample(x1\_train,y1\_train)

x\_smote\_train, y\_smote\_train = sm.fit\_resample(x\_train, y\_train)

np.bincount(y\_smote\_train)

x\_smote\_train

**# Full features with SMOTE**

sns.heatmap(x\_smote\_train.corr(),annot = True)

**# LR**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import precision\_score, accuracy\_score, recall\_score

import sys

lr = LogisticRegression(solver = 'lbfgs', max\_iter = sys.maxsize )

lr.fit(x\_smote\_train, y\_smote\_train)

y\_predict = lr.predict(x\_test)

accuracy\_smote\_lr = accuracy\_score(y\_test,y\_predict)\*100

precison\_smote\_lr = precision\_score(y\_test, y\_predict,pos\_label= 0 )\*100

recall\_smote\_lr = recall\_score(y\_test, y\_predict,pos\_label=0)\*100

fmes\_smote\_lr = f1\_score(y\_test,y\_predict,zero\_division =1)

print("Accuracy :",accuracy\_smote\_lr)

print("Precision",precison\_smote\_lr)

print("Recall :",recall\_smote\_lr )

error\_smote\_lr = 100 - accuracy\_smote\_lr

print("Error : ",error\_smote\_lr)

print("F1 Score : ",fmes\_smote\_lr)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

auc\_smote\_lr = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_smote\_lr)

gini\_smote\_lr = 2\*(auc\_smote\_lr/100) - 1

print("GINI : ",gini\_smote\_lr)

**# LSVM L1**

from sklearn.svm import LinearSVC

lsvmL1 = LinearSVC(dual = False, penalty = 'l1',max\_iter = 10000)

lsvmL1.fit(x\_smote\_train, y\_smote\_train)

y\_predict = lsvmL1.predict(x\_test)

accuracy\_smote\_lsvmL1 = accuracy\_score(y\_test,y\_predict)\*100

precison\_smote\_lsvmL1 = precision\_score(y\_test, y\_predict\_lsso,pos\_label=0 )\*100

recall\_smote\_lsvmL1 = recall\_score(y\_test, y\_predict\_lsso,pos\_label=0)\*100

fmes\_smote\_lsvmL1 = f1\_score(y\_test,y\_predict\_lsso,zero\_division =1)

print("Accuracy :",accuracy\_smote\_lsvmL1)

print("Precision",precison\_smote\_lsvmL1)

print("Recall :",recall\_smote\_lsvmL1) #max\_iter = sys.maxsize

error\_smote\_lsvmL1 = 100 - accuracy\_smote\_lsvmL1

print("Error : ",error\_smote\_lsvmL1)

print("F1 Score : ",fmes\_smote\_lsvmL1)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

auc\_smote\_lsvmL1 = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_smote\_lsvmL1)

gini\_smote\_lsvmL1 = 2\*(auc\_smote\_lsvmL1/100) - 1

print("GINI : ",gini\_smote\_lsvmL1)

**# LSVM L2**

from sklearn.svm import LinearSVC

import sys

lsvmL2 = LinearSVC( dual=False,penalty='l2',max\_iter = 120000)

lsvmL2.fit(x\_smote\_train, y\_smote\_train)

y\_predict = lsvmL2.predict(x\_test)

accuracy\_smote\_lsvmL2 = accuracy\_score(y\_test,y\_predict)\*100

precison\_smote\_lsvmL2 = precision\_score(y\_test, y\_predict,pos\_label=0 )\*100

recall\_smote\_lsvmL2 = recall\_score(y\_test, y\_predict,pos\_label=0)\*100

fmes\_smote\_lsvmL2 = f1\_score(y\_test,y\_predict,zero\_division =1)

print("Accuracy :",accuracy\_smote\_lsvmL2)

print("Precision",precison\_smote\_lsvmL2)

print("Recall :",recall\_smote\_lsvmL2)

error\_smote\_lsvmL2 = 100 - accuracy\_smote\_lsvmL2

print("Error : ",error\_smote\_lsvmL2)

print("F1 Score : ",fmes\_smote\_lsvmL2)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

auc\_smote\_lsvmL2 = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_smote\_lsvmL2)

gini\_smote\_lsvmL2 = 2\*(auc\_smote\_lsvmL2/100) - 1

print("GINI : ",gini\_smote\_lsvmL2)

**# RF**

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy")

classifier.fit(x\_smote\_train, y\_smote\_train)

y\_predict = classifier.predict(x\_test)

accuracy\_smote\_rf = accuracy\_score(y\_test,y\_predict)\*100

precision\_smote\_rf= precision\_score(y\_test, y\_predict,pos\_label=0)\*100

recall\_smote\_rf = recall\_score(y\_test, y\_predict,pos\_label=0)\*100

fmes\_smote\_rf = f1\_score(y\_test,y\_predict,zero\_division =1)

print("Accuracy :",accuracy\_smote\_rf)

print("Precision",precision\_smote\_rf)

print("Recall :",recall\_smote\_rf)

error\_smote\_rf = 100 - accuracy\_smote\_rf

print("Error : ",error\_smote\_rf)

print("F1 Score : ",fmes\_smote\_rf)

print("Confusion matrix:")

print(confusion\_matrix(y\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y\_test))

print("Incorrect Predictions :",sum(y\_predict != y\_test))

auc\_smote\_rf = roc\_auc\_score(y\_test, y\_predict)\*100

print("AUC : ",auc\_smote\_rf)

gini\_smote\_rf = 2\*(auc\_smote\_rf/100) - 1

print("GINI : ",gini\_smote\_rf)

x1\_smote\_train

df1 = pd.DataFrame()

df1 = x1\_smote\_train

y1\_smote\_train.rename = "Decision"

df1 = df1.join(y1\_smote\_train)

df1

df1.info()

df1["Hemo"] = pd.to\_numeric(df1.Hemo, errors = 'coerce')

print(df1.info())

y1\_test

x1\_test

**# CHAID**

from chefboost import Chefboost as cf

from pandas.api.types import CategoricalDtype

config = {"algorithm" : "CHAID"}

ChaidTree = cf.fit(df1,config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test.index:

pred = cf.predict(ChaidTree,x1\_test.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

tn += 1

y\_predict.append(0)

y\_testing.append(0)

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

print("TP : "+str(tp))

print("TN : "+str(tn))

print("FP : "+str(fp))

print("FN : "+str(fn))

accuracy\_smote\_chaid = (tp+tn)\*100/(tp+tn+fp+fn)

precision\_smote\_chaid = tp\*100/(tp+fp)

recall\_smote\_chaid = tp\*100/(tp+fn)

print(accuracy\_smote\_chaid)

print(precision\_smote\_chaid)

print(recall\_smote\_chaid)

error\_smote\_chaid = 100 - accuracy\_smote\_chaid

print("Error : ",error\_smote\_chaid)

fmes\_smote\_chaid = 2\*(recall\_smote\_chaid\*precision\_smote\_chaid)/((recall\_smote\_chaid+precision\_smote\_chaid)\*100)

print("F MEasure : ",fmes\_smote\_chaid)

auc\_smote\_chaid = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_smote\_chaid)

gini\_smote\_chaid = 2\*(auc\_smote\_chaid/100) - 1

print("GINI : ",gini\_smote\_chaid)

**# C4.5**

from chefboost import Chefboost as cf

from pandas.api.types import CategoricalDtype

config = {"algorithm" : "C4.5"}

C45Tree = cf.fit(df1, config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test.index:

pred = cf.predict(C45Tree,x1\_test.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

print("TP : "+str(tp))

print("TN : "+str(tn))

print("FP : "+str(fp))

print("FN : "+str(fn))

accuracy\_smote\_c45 = (tp+tn)\*100/(tp+tn+fp+fn)

print(accuracy\_smote\_c45)

precision\_smote\_c45 = tp\*100/(tp+fp)

recall\_smote\_c45 = tp\*100/(tp+fn)

print(precision\_smote\_c45)

print(recall\_smote\_c45 )

error\_smote\_c45 = 100 - accuracy\_smote\_c45

print("Error : ",error\_smote\_c45)

fmes\_smote\_c45 = 2\*(recall\_smote\_c45\*precision\_smote\_c45)/((recall\_smote\_c45+precision\_smote\_c45)\*100)

print("F MEasure : ",fmes\_smote\_c45)

auc\_smote\_c45 = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_smote\_c45)

gini\_smote\_c45 = 2\*(auc\_smote\_c45/100) - 1

print("GINI : ",gini\_smote\_c45)

**# Evaluation Measures Plotting in case of Full Features with SMOTE**

fig2 = plt.subplots(figsize =(12, 8))

accuracy = [accuracy\_smote\_lr,accuracy\_smote\_lsvmL1 ,accuracy\_smote\_lsvmL2,accuracy\_smote\_rf,accuracy\_smote\_chaid,accuracy\_smote\_c45]

precision = [precison\_smote\_lr,precison\_smote\_lsvmL1,precison\_smote\_lsvmL2, precision\_smote\_rf,precision\_smote\_chaid,precision\_smote\_c45 ]

recall = [recall\_smote\_lr, recall\_smote\_lsvmL1,recall\_smote\_lsvmL2, recall\_smote\_rf,recall\_smote\_chaid, recall\_smote\_c45]

br1 = np.arange(len(accuracy))

br2 = [x + barWidth for x in br1]

br3 = [x + barWidth for x in br2]

plt.bar(br1, accuracy, color ='r', width = barWidth,

edgecolor ='grey', label ='Accuracy')

plt.bar(br2, precision,color ='g', width = barWidth,

edgecolor ='grey', label ='Precision')

plt.bar(br3, recall, color ='b', width = barWidth,

edgecolor ='grey', label ='Recall')

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Performance', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(accuracy))],

['LR', 'LSVML1', 'LSVML2', 'RF', 'CHAID', 'C4.5' ] )

plt.title("Accuracy Precision Recall using Smote for Full features")

plt.legend()

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','RF']

f\_mesures = [fmes\_smote\_lr, fmes\_smote\_lsvmL1,fmes\_smote\_lsvmL2,fmes\_smote\_chaid, fmes\_smote\_c45, fmes\_smote\_rf]

x.bar(algo,f\_mesures)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('F Measure', fontweight ='bold', fontsize = 15)

plt.title("Variation of F Measure for Various Classifiers using Full Features with Smote")

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','RF']

auc = [auc\_smote\_lr, auc\_smote\_lsvmL1,auc\_smote\_lsvmL2,auc\_smote\_chaid, auc\_smote\_c45, auc\_smote\_rf]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('AUC', fontweight ='bold', fontsize = 15)

plt.title("Variation of AUC for Various Classifiers using Full Features with Smote")

plt.show()

algo = ['LR', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','RF']

gini = [gini\_smote\_lr, gini\_smote\_lsvmL1,gini\_smote\_lsvmL2,gini\_smote\_chaid, gini\_smote\_c45, gini\_smote\_rf]

plt.rcParams["figure.figsize"] = (30,10)

plt.plot(algo, gini)

plt.xlabel("Algorithm", fontweight ='bold', fontsize = 25)

plt.ylabel("GINI", fontweight ='bold', fontsize = 25)

plt.title("Variation of GINI for Various Classifiers using Full Features with Smote", fontweight ='bold', fontsize = 25)

plt.show()

error = plt.figure()

x = error.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','RF']

auc =[100-accuracy\_smote\_lr,100-accuracy\_smote\_lsvmL1 ,100-accuracy\_smote\_lsvmL2,100-accuracy\_smote\_chaid,100-accuracy\_smote\_c45,100-accuracy\_smote\_rf]

x.bar(algo,auc)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Error', fontweight ='bold', fontsize = 15)

plt.title("Variation of Error for Various Classifiers using Full Features with Smote")

plt.show()

**# LASSO with SMOTE**

x1\_smote\_train\_lsso = x1\_smote\_train.drop(list\_lasso, axis = 1)

x1\_train\_lsso = x1\_train.drop(list\_lasso, axis = 1)

x\_test\_lsso = x\_test.drop(list\_lasso, axis = 1)

x1\_test\_lsso = x1\_test.drop(list\_lasso, axis = 1)

x1\_smote\_train\_lsso

sns.heatmap(x1\_smote\_train\_lsso.corr(),annot = True)

**# LR**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import precision\_score, accuracy\_score, recall\_score

import sys

lr = LogisticRegression(solver = 'lbfgs', max\_iter = sys.maxsize )

lr.fit(x1\_smote\_train\_lsso, y1\_smote\_train)

y\_predict = lr.predict(x1\_test\_lsso)

accuracy\_smote\_lr\_lsso = accuracy\_score(y1\_test,y\_predict)\*100

precison\_smote\_lr\_lsso = precision\_score(y1\_test, y\_predict,pos\_label= 'ckd' )\*100

recall\_smote\_lr\_lsso = recall\_score(y1\_test, y\_predict,pos\_label='ckd')\*100

fmes\_smote\_lr\_lsso = f1\_score(y1\_test,y\_predict, pos\_label='ckd')

print("Accuracy :",accuracy\_smote\_lr\_lsso)

print("Precision",precison\_smote\_lr\_lsso)

print("Recall :",recall\_smote\_lr\_lsso )

error\_smote\_lr\_lsso = 100 - accuracy\_smote\_lr\_lsso

print("Error : ",error\_smote\_lr\_lsso)

print("F1 Score : ",fmes\_smote\_lr\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y1\_test,y\_predict))

print("Correct Predictions :",sum(y\_predict == y1\_test))

print("Incorrect Predictions :",sum(y\_predict != y1\_test))

print(y\_predict.dtype)

y2\_predict = pd.Series(y\_predict,dtype = "category")

y2\_predict = y2\_predict.cat.codes

auc\_smote\_lr\_lsso = roc\_auc\_score(y1\_test, y2\_predict)\*100

print("AUC : ",auc\_smote\_lr\_lsso)

gini\_smote\_lr\_lsso = 2\*(auc\_smote\_lr\_lsso/100) - 1

print("GINI : ",gini\_smote\_lr\_lsso)

**# LSVM L1**

from sklearn.svm import LinearSVC

lsvmL1 = LinearSVC(dual = False, penalty = 'l1',max\_iter = 10000)

lsvmL1.fit(x1\_smote\_train\_lsso, y1\_smote\_train)

y\_predict\_lsso = lsvmL1.predict(x1\_test\_lsso)

accuracy\_smote\_lsvmL1\_lsso = accuracy\_score(y1\_test,y\_predict\_lsso)\*100

precison\_smote\_lsvmL1\_lsso= precision\_score(y1\_test, y\_predict\_lsso,pos\_label='ckd' )\*100

recall\_smote\_lsvmL1\_lsso = recall\_score(y1\_test, y\_predict\_lsso,pos\_label='ckd')\*100

fmes\_smote\_lsvmL1\_lsso = f1\_score(y1\_test,y\_predict\_lsso,zero\_division =1,pos\_label='ckd')

print("Accuracy :",accuracy\_smote\_lsvmL1\_lsso)

print("Precision",precison\_smote\_lsvmL1\_lsso)

print("Recall :",recall\_smote\_lsvmL1\_lsso) #max\_iter = sys.maxsize

error\_smote\_lsvmL1\_lsso = 100 - accuracy\_smote\_lsvmL1\_lsso

print("Error : ",error\_smote\_lsvmL1\_lsso)

print("F1 Score : ",fmes\_smote\_lsvmL1\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y1\_test,y\_predict\_lsso))

print("Correct Predictions :",sum(y\_predict\_lsso == y1\_test))

print("Incorrect Predictions :",sum(y\_predict\_lsso != y1\_test))

y2\_predict = pd.Series(y\_predict\_lsso,dtype = "category")

y2\_predict = y2\_predict.cat.codes

auc\_smote\_lsvmL1\_lsso = roc\_auc\_score(y1\_test, y2\_predict)\*100

print("AUC : ",auc\_smote\_lsvmL1\_lsso)

gini\_smote\_lsvmL1 = 2\*(auc\_smote\_lsvmL1/100) - 1

print("GINI : ",gini\_smote\_lsvmL1)

**# LSVM L2**

from sklearn.svm import LinearSVC

import sys

lsvmL2 = LinearSVC( dual=False,penalty='l2',max\_iter = 120000)

lsvmL2.fit(x1\_smote\_train\_lsso, y1\_smote\_train)

y\_predict\_lsso = lsvmL2.predict(x1\_test\_lsso)

accuracy\_smote\_lsvmL2\_lsso = accuracy\_score(y1\_test,y\_predict\_lsso)\*100

precison\_smote\_lsvmL2\_lsso = precision\_score(y1\_test, y\_predict\_lsso,pos\_label='ckd' )\*100

recall\_smote\_lsvmL2\_lsso = recall\_score(y1\_test, y\_predict\_lsso,pos\_label='ckd')\*100

fmes\_smote\_lsvmL2\_lsso = f1\_score(y1\_test,y\_predict\_lsso,zero\_division =1,pos\_label='ckd')

print("Accuracy :",accuracy\_smote\_lsvmL2\_lsso)

print("Precision",precison\_smote\_lsvmL2\_lsso)

print("Recall :",recall\_smote\_lsvmL2\_lsso)

error\_smote\_lsvmL2\_lsso = 100 - accuracy\_smote\_lsvmL2\_lsso

print("Error : ",error\_smote\_lsvmL2\_lsso)

print("F1 Score : ",fmes\_smote\_lsvmL2\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y1\_test,y\_predict\_lsso))

print("Correct Predictions :",sum(y\_predict\_lsso == y1\_test))

print("Incorrect Predictions :",sum(y\_predict\_lsso != y1\_test))

y2\_predict = pd.Series(y\_predict\_lsso,dtype = "category")

y2\_predict = y2\_predict.cat.codes

auc\_smote\_lsvmL2\_lsso = roc\_auc\_score(y1\_test, y2\_predict)\*100

print("AUC : ",auc\_smote\_lsvmL2\_lsso)

gini\_smote\_lsvmL2 = 2\*(auc\_smote\_lsvmL2/100) - 1

print("GINI : ",gini\_smote\_lsvmL2)

**# RF**

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy")

classifier.fit(x1\_smote\_train\_lsso, y1\_smote\_train)

y\_predict\_lsso= classifier.predict(x1\_test\_lsso)

accuracy\_smote\_rf\_lsso = accuracy\_score(y1\_test,y\_predict\_lsso)\*100

precision\_smote\_rf\_lsso= precision\_score(y1\_test, y\_predict\_lsso,pos\_label='ckd')\*100

recall\_smote\_rf\_lsso = recall\_score(y1\_test, y\_predict\_lsso,pos\_label='ckd')\*100

fmes\_smote\_rf\_lsso = f1\_score(y1\_test,y\_predict\_lsso,zero\_division =1,pos\_label='ckd')

print("Accuracy :",accuracy\_smote\_rf\_lsso)

print("Precision",precision\_smote\_rf\_lsso)

print("Recall :",recall\_smote\_rf\_lsso)

error\_smote\_rf\_lsso = 100 - accuracy\_smote\_rf\_lsso

print("Error : ",error\_smote\_rf\_lsso)

print("F1 Score : ",fmes\_smote\_rf\_lsso)

print("Confusion matrix:")

print(confusion\_matrix(y1\_test,y\_predict\_lsso))

print("Correct Predictions :",sum(y\_predict\_lsso == y1\_test))

print("Incorrect Predictions :",sum(y\_predict\_lsso != y1\_test))

y2\_predict = pd.Series(y\_predict\_lsso,dtype = "category")

y2\_predict = y2\_predict.cat.codes

auc\_smote\_rf\_lsso = roc\_auc\_score(y1\_test, y2\_predict)\*100

print("AUC : ",auc\_smote\_rf\_lsso)

gini\_smote\_rf = 2\*(auc\_smote\_rf/100) - 1

print("GINI : ",gini\_smote\_rf)

**# CHAID**

from chefboost import Chefboost as cf

from pandas.api.types import CategoricalDtype

config = {"algorithm" : "CHAID"}

df = pd.DataFrame()

df = x1\_smote\_train\_lsso

df = df.join(y1\_smote\_train)

df

ChaidTree = cf.fit(df,config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test.index:

pred = cf.predict(ChaidTree,x1\_test\_lsso.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

print("TP : "+str(tp))

print("TN : "+str(tn))

print("FP : "+str(fp))

print("FN : "+str(fn))

accuracy\_smote\_chaid\_lsso = (tp+tn)\*100/(tp+tn+fp+fn)

precision\_smote\_chaid\_lsso = tp\*100/(tp+fp)

recall\_smote\_chaid\_lsso = tp\*100/(tp+fn)

print(accuracy\_smote\_chaid\_lsso)

print(precision\_smote\_chaid\_lsso)

print(recall\_smote\_chaid\_lsso)

error\_smote\_chaid\_lsso = 100 - accuracy\_smote\_chaid\_lsso

print("Error : ",error\_smote\_chaid\_lsso)

fmes\_smote\_chaid\_lsso = 2\*(precision\_smote\_chaid\_lsso\*recall\_smote\_chaid\_lsso)/(precision\_smote\_chaid\_lsso+recall\_smote\_chaid\_lsso)

fmes\_smote\_chaid\_lsso/=100.0

print("F Measure : ",fmes\_smote\_chaid\_lsso)

auc\_smote\_chaid\_lsso = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_smote\_chaid\_lsso)

gini\_smote\_chaid = 2\*(auc\_smote\_chaid/100) - 1

print("GINI : ",gini\_smote\_chaid)

**# C4.5**

config = {"algorithm" : "C4.5"}

C45Tree = cf.fit(df, config)

tp = 0

tn = 0

fp = 0

fn = 0

y\_predict = []

y\_testing = []

for i in x1\_test\_corr.index:

pred = cf.predict(C45Tree,x1\_test\_lsso.loc[i,:])

if pred == y1\_test[i]:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(1)

tp += 1

else:

y\_predict.append(0)

y\_testing.append(0)

tn += 1

else:

if pred == 'ckd':

y\_predict.append(1)

y\_testing.append(0)

fp+=1

else:

y\_predict.append(0)

y\_testing.append(1)

fn+=1

print("TP : "+str(tp))

print("TN : "+str(tn))

print("FP : "+str(fp))

print("FN : "+str(fn))

accuracy\_smote\_c45\_lsso = (tp+tn)\*100/(tp+tn+fp+fn)

print(accuracy\_smote\_c45\_lsso)

precision\_smote\_c45\_lsso = tp\*100/(tp+fp)

recall\_smote\_c45\_lsso = tp\*100/(tp+fn)

print(precision\_smote\_c45\_lsso)

print(recall\_smote\_c45\_lsso )

error\_smote\_c45\_lsso = 100 - accuracy\_smote\_c45\_lsso

print("Error : ",error\_smote\_c45\_lsso)

fmes\_smote\_c45\_lsso = 2\*(precision\_smote\_c45\_lsso\*recall\_smote\_c45\_lsso)/((precision\_smote\_c45\_lsso+recall\_smote\_c45\_lsso)\*100)

print("F Measure : ",fmes\_smote\_c45\_lsso)

auc\_smote\_c45\_lsso = roc\_auc\_score(y\_testing, y\_predict)\*100

print("AUC : ",auc\_smote\_c45\_lsso)

gini\_smote\_c45 = 2\*(auc\_smote\_c45/100) - 1

print("GINI : ",gini\_smote\_c45)

**# Evaluation Measures Plotting in case of LASSO with SMOTE**

fig2 = plt.subplots(figsize =(12, 8))

accuracy = [accuracy\_smote\_lr\_lsso,accuracy\_smote\_lsvmL1\_lsso ,accuracy\_smote\_lsvmL2\_lsso,accuracy\_smote\_rf\_lsso,accuracy\_smote\_chaid\_lsso,accuracy\_smote\_c45\_lsso ]

precision = [precison\_smote\_lr\_lsso,precison\_smote\_lsvmL1\_lsso,precison\_smote\_lsvmL2\_lsso, precision\_smote\_rf\_lsso,precision\_smote\_chaid\_lsso,precision\_smote\_c45\_lsso ]

recall = [recall\_smote\_lr\_lsso, recall\_smote\_lsvmL1\_lsso,recall\_smote\_lsvmL2\_lsso, recall\_smote\_rf\_lsso,recall\_smote\_chaid\_lsso, recall\_smote\_c45\_lsso ]

br1 = np.arange(len(accuracy))

br2 = [x + barWidth for x in br1]

br3 = [x + barWidth for x in br2]

plt.bar(br1, accuracy, color ='r', width = barWidth,

edgecolor ='grey', label ='Accuracy')

plt.bar(br2, precision,color ='g', width = barWidth,

edgecolor ='grey', label ='Precision')

plt.bar(br3, recall, color ='b', width = barWidth,

edgecolor ='grey', label ='Recall')

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Performance', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(accuracy))],

['LR', 'LSVML1', 'LSVML2', 'RF', 'CHAID', 'C4.5' ] )

plt.title("Accuracy Precision Recall using Smote for Lasso feature selection")

plt.legend()

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','RF']

f\_mesures = [fmes\_smote\_lr\_lsso, fmes\_smote\_lsvmL1\_lsso,fmes\_smote\_lsvmL2\_lsso,fmes\_smote\_chaid\_lsso, fmes\_smote\_c45\_lsso, fmes\_smote\_rf\_lsso]

x.bar(algo,f\_mesures)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('F Measure', fontweight ='bold', fontsize = 15)

plt.title("Variation of F Measure for Various Classifiers using Lasso Feature selection with Smote")

plt.show()

f\_mes = plt.figure()

x = f\_mes.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','RF']

f\_mesures = [auc\_smote\_lr\_lsso, auc\_smote\_lsvmL1\_lsso,auc\_smote\_lsvmL2\_lsso,auc\_smote\_chaid\_lsso, auc\_smote\_c45\_lsso, auc\_smote\_rf\_lsso]

x.bar(algo,f\_mesures)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('AUC', fontweight ='bold', fontsize = 15)

plt.title("Variation of AUC for Various Classifiers using Lasso Feature selection with Smote")

plt.show()

algo = ['LR', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','RF']

gini = [gini\_smote\_lr\_lsso, gini\_smote\_lsvmL1,gini\_smote\_lsvmL2,gini\_smote\_chaid, gini\_smote\_c45, gini\_smote\_rf]

plt.rcParams["figure.figsize"] = (30,10)

plt.plot(algo, gini)

plt.xlabel("Algorithm", fontweight ='bold', fontsize = 25)

plt.ylabel("GINI", fontweight ='bold', fontsize = 25)

plt.title("Variation of GINI for Various Classifiers using Lasso Feature selection with Smote", fontweight ='bold', fontsize = 25)

plt.show()

error = plt.figure()

x = error.add\_axes([0,0,0.3,0.3])

algo = ['LR', 'LSVML1', 'LSVML2', 'CHAID', 'C4.5','RF']

errorures = [100-accuracy\_smote\_lr\_lsso,100-accuracy\_smote\_lsvmL1\_lsso ,100-accuracy\_smote\_lsvmL2\_lsso,100-accuracy\_smote\_chaid\_lsso,100-accuracy\_smote\_c45\_lsso,100-accuracy\_smote\_rf\_lsso ]

x.bar(algo,errorures)

plt.xlabel('Algorithm', fontweight ='bold', fontsize = 15)

plt.ylabel('Error', fontweight ='bold', fontsize = 15)

plt.title("Variation of Error for Various Classifiers using Lasso Feature selection with Smote")

plt.show()

## CHAPTER 4

### SNAP SHOTS



Fig. 4.1. Sample Image of Execution of Source code in Jupyter Notebook

**Heat Maps:**

Here are the heat maps of correlation between selected features in case of Full feature selection, CFS, Wrapper FS, LASSO FS, Full features with SMOTE, LASSO with SMOTE respectively.

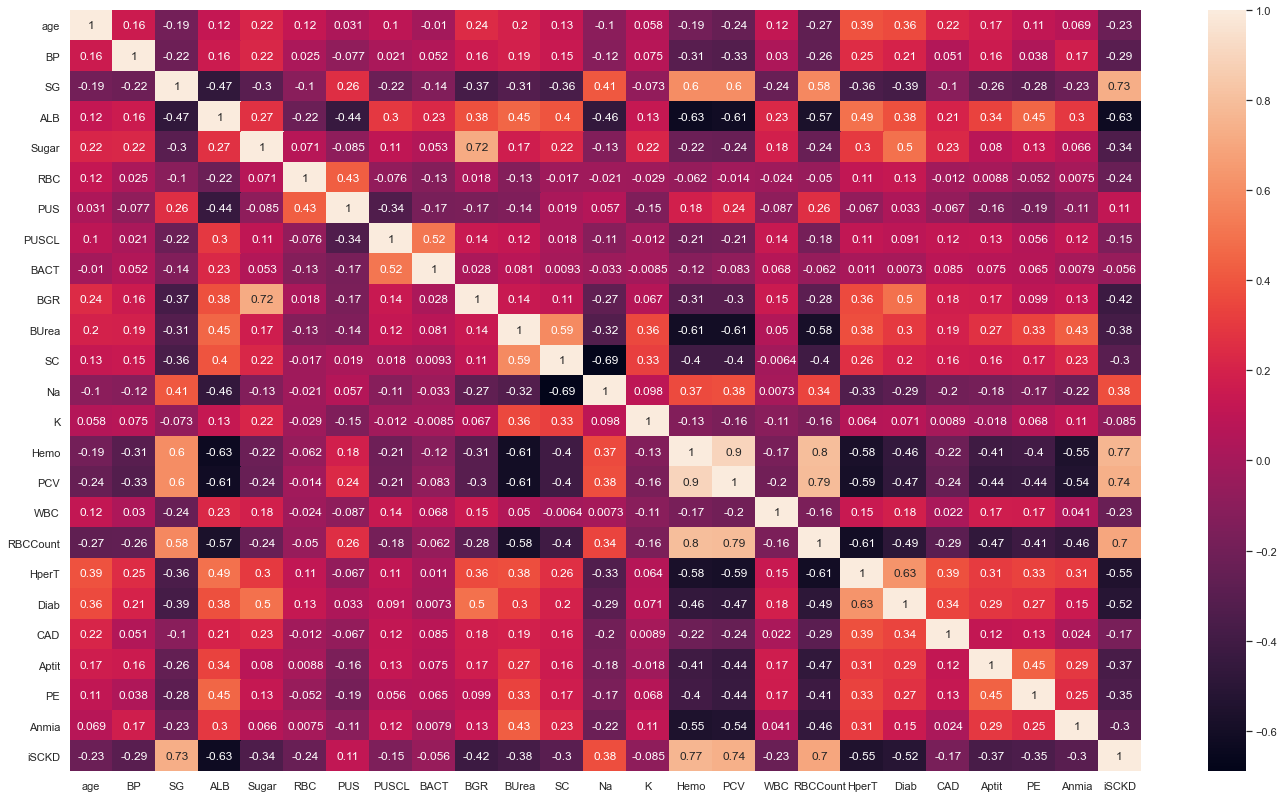


Fig. 4.2. Heat Maps of Correlation between Selected Features in case of Full feature selection

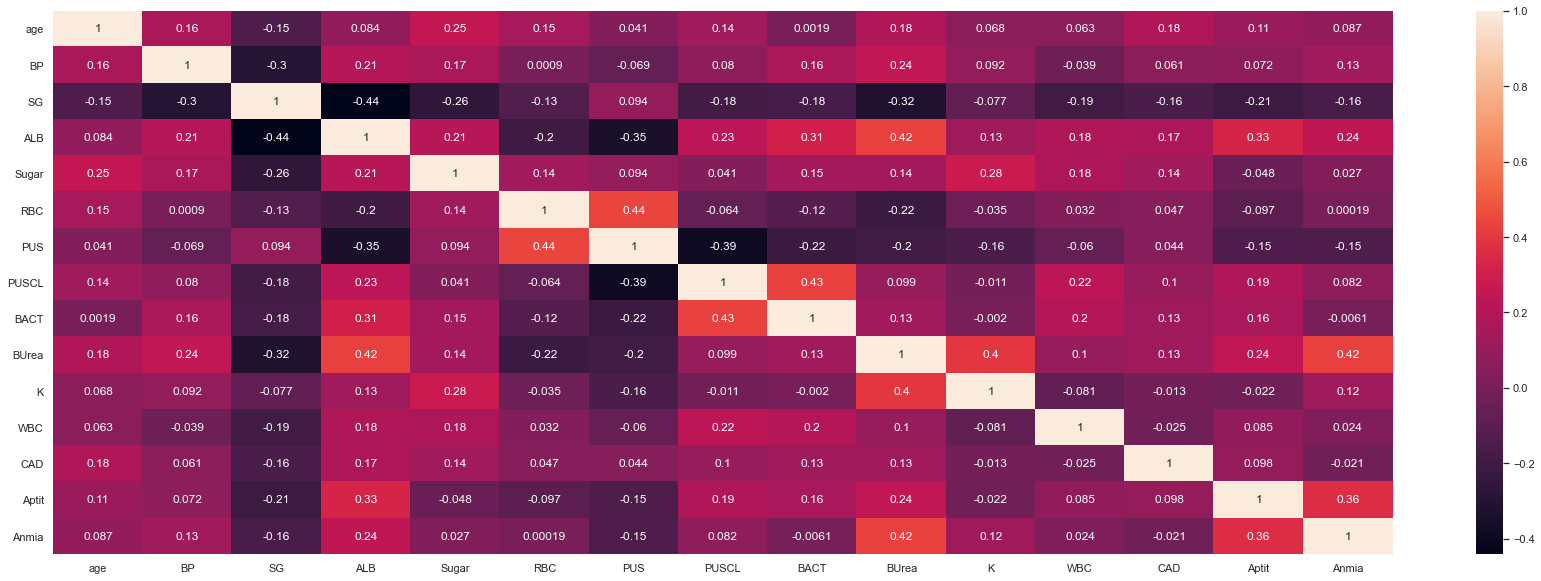


Fig. 4.3. Heat Maps of Correlation between Selected Features in case of CFS

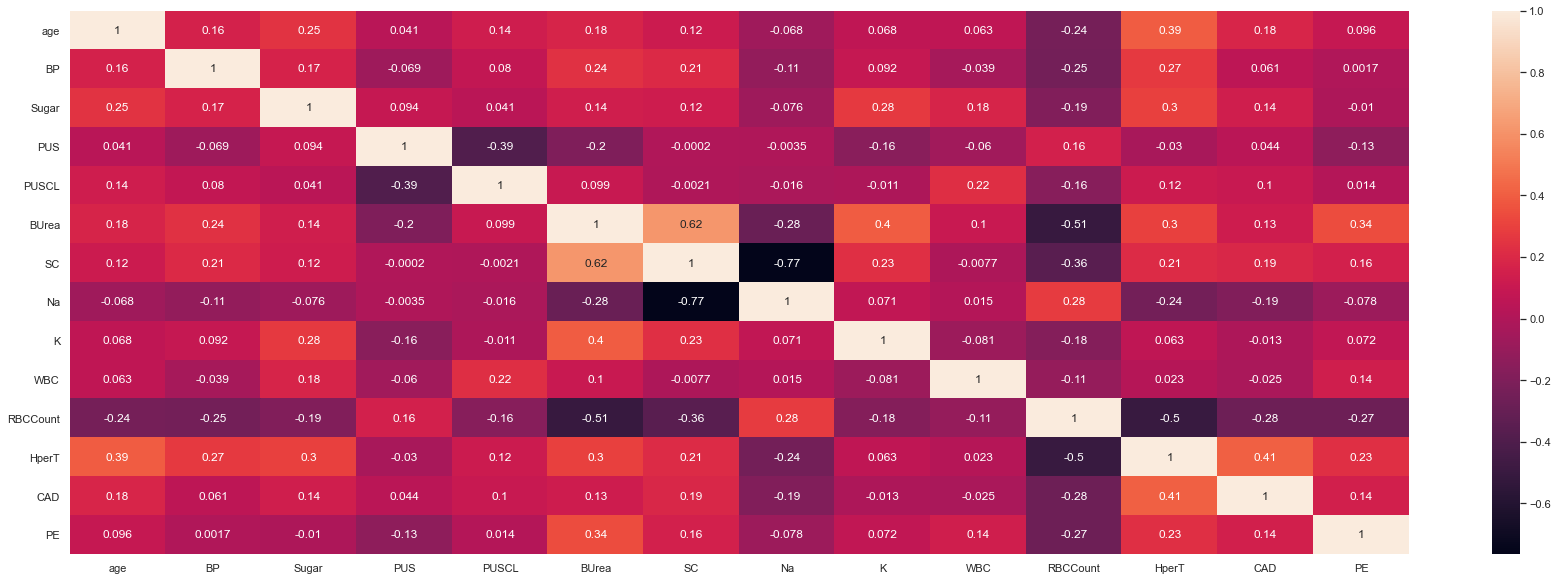


Fig. 4.4. Heat Maps of Correlation between Selected Features in case of Wrapper FS

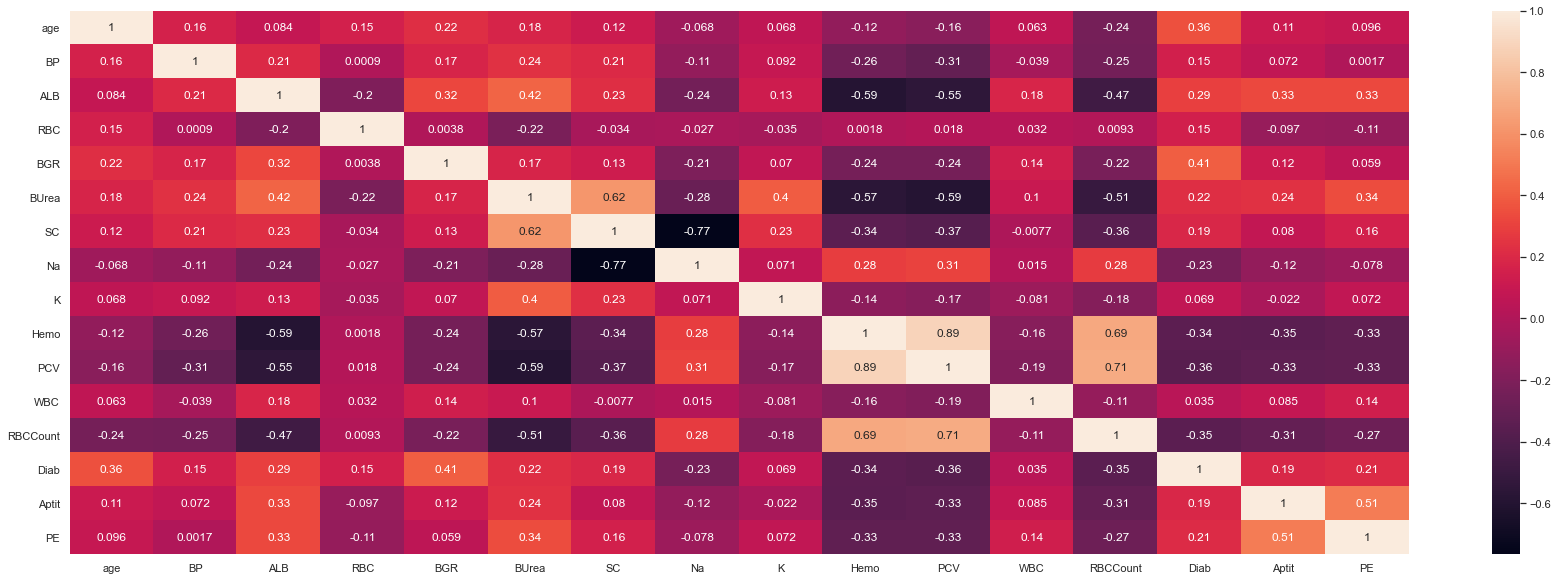


Fig. 4.5. Heat Maps of Correlation between Selected Features in case of LASSO FS

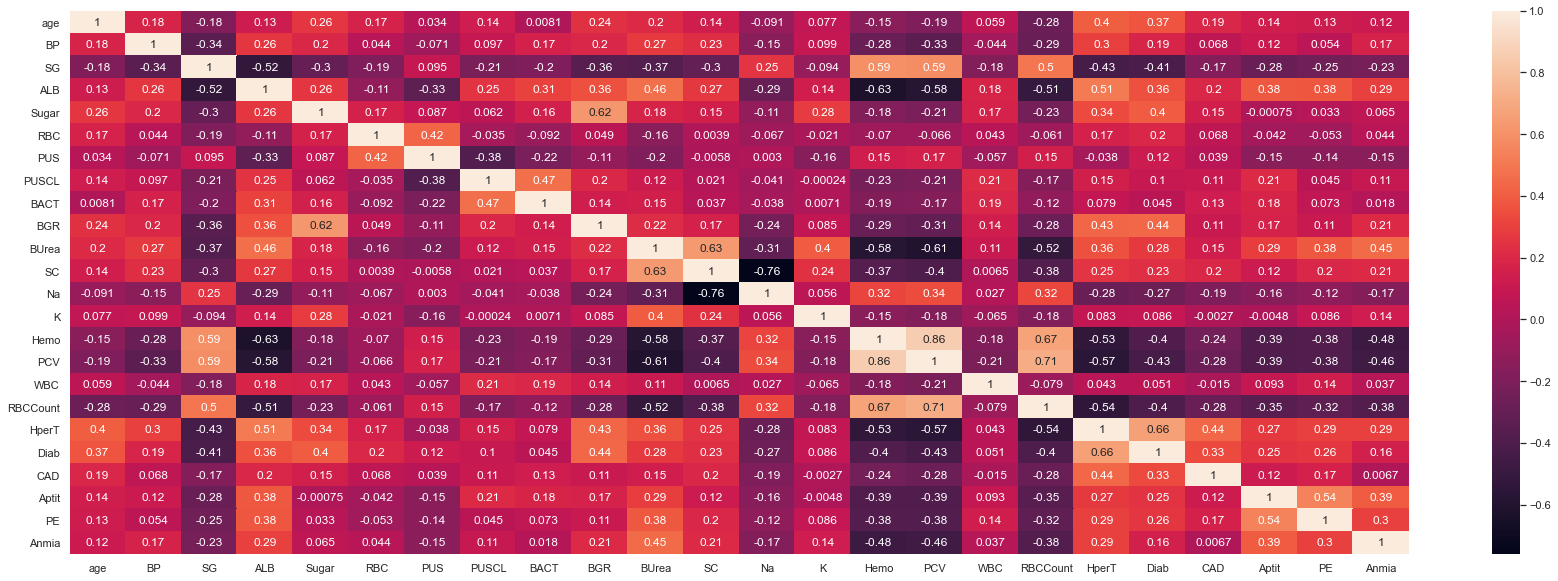


Fig. 4.6. Heat Maps of Correlation between Selected Features in case of Full features with SMOTE

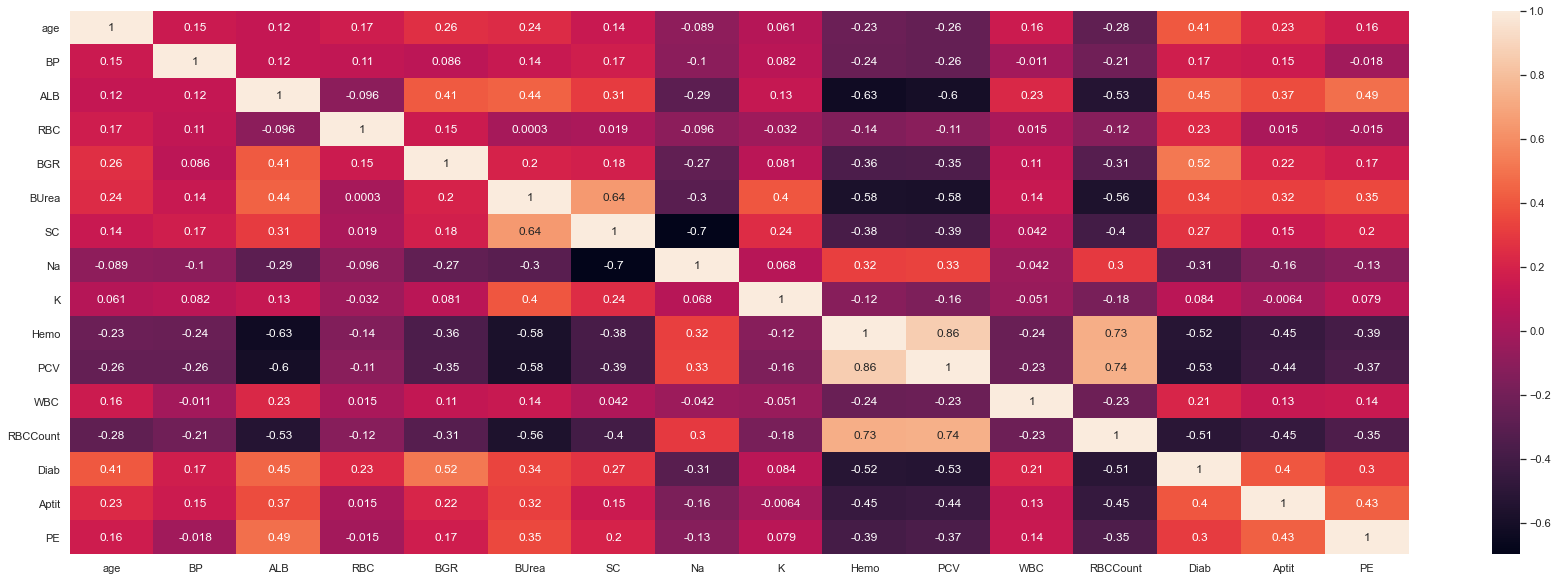


Fig. 4.7. Heat Maps of Correlation between Selected Features in case of LASSO FS with SMOTE

**Graphs of Accuracy, Precision, Recall:**

Here are the plotted graphs of Accuracy, Precision, Recall respectively for the Machine Learning classifiers in case of Full feature selection, CFS, Wrapper FS, LASSO FS, Full features with SMOTE, LASSO with SMOTE respectively.

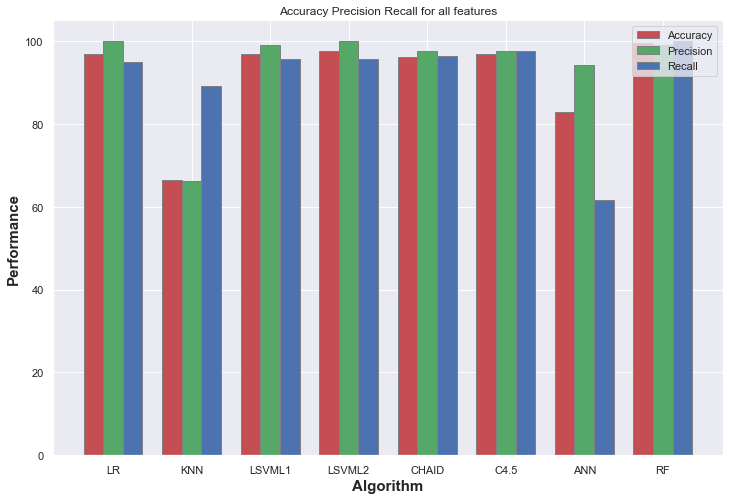


Fig. 4.8. Accuracy, Precision, Recall for ML classifiers in case of Full feature selection

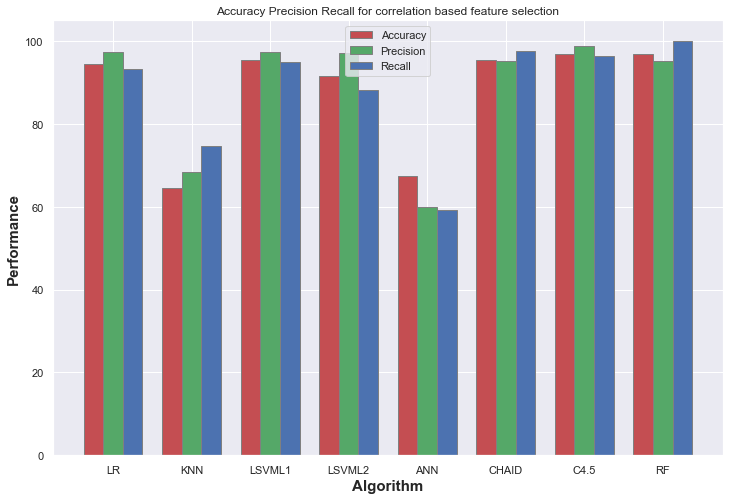


Fig. 4.9. Accuracy, Precision, Recall for ML classifiers in case of CFS

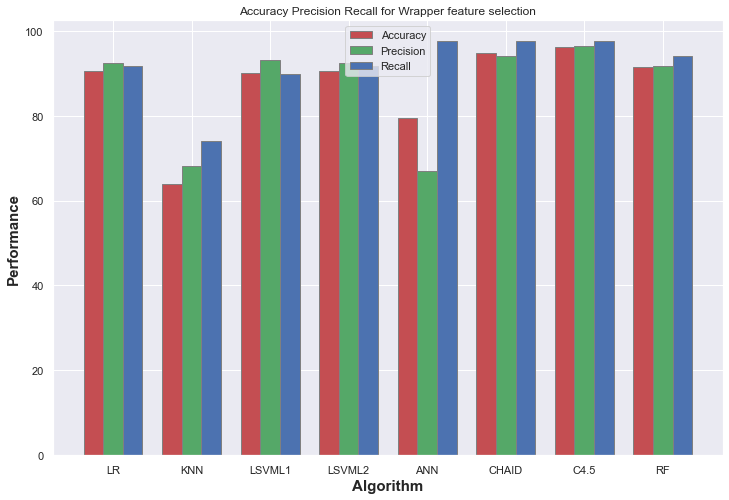


Fig. 4.10. Accuracy, Precision, Recall for ML classifiers in case of Wrapper FS

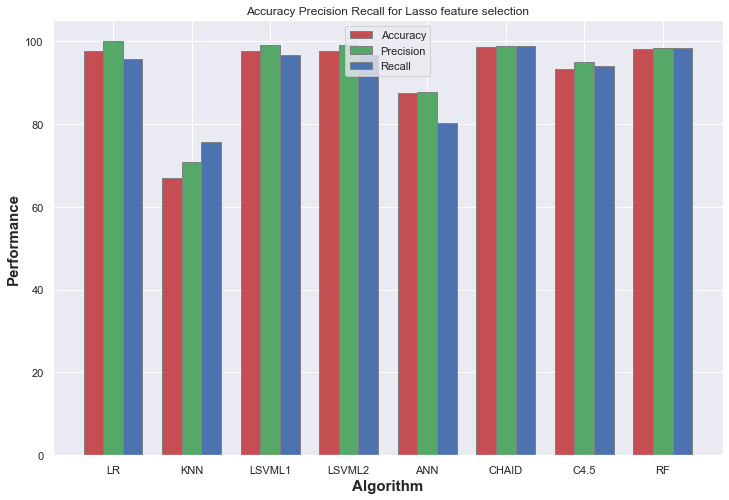


Fig. 4.11. Accuracy, Precision, Recall for ML classifiers in case of LASSO FS

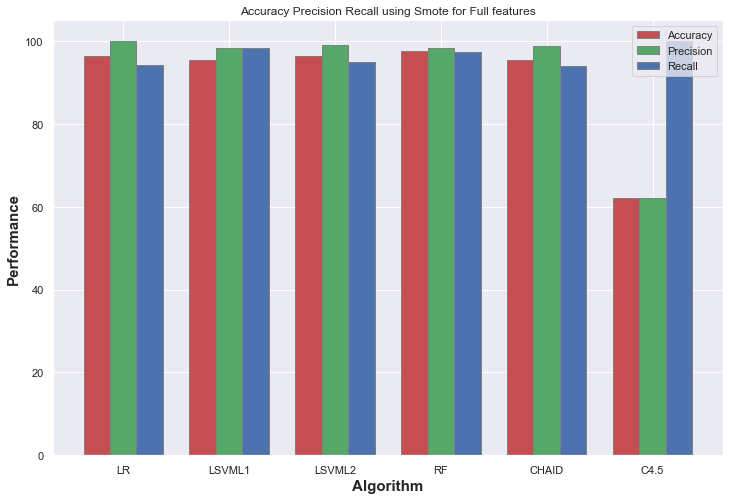


Fig. 4.12. Accuracy, Precision, Recall for ML classifiers in case of Full features with SMOTE

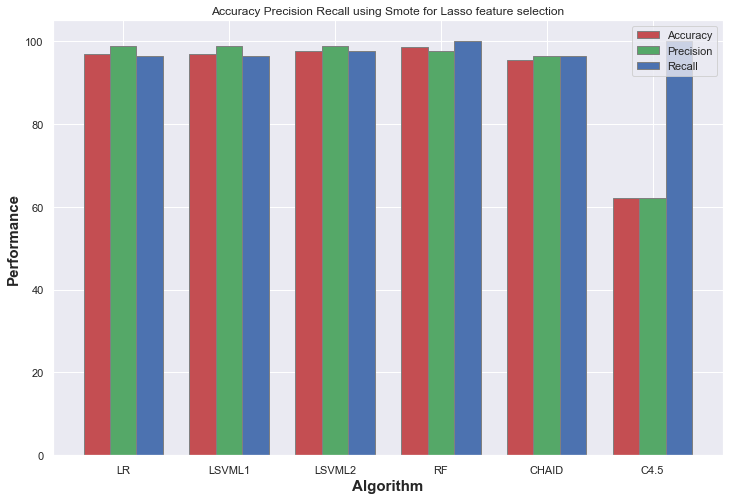


Fig. 4.13. Accuracy, Precision, Recall for ML classifiers in case of LASSO FS with SMOTE

**Classification Error:**

Here are the plotted graphs of Classification Error for the Machine Learning classifiers in case of Full feature selection, CFS, Wrapper FS, LASSO FS, Full features with SMOTE, LASSO with SMOTE respectively.

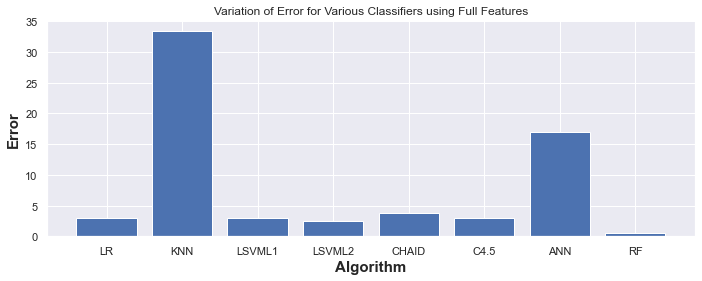


Fig. 4.14. Classification Error for ML classifiers in case of Full feature selection

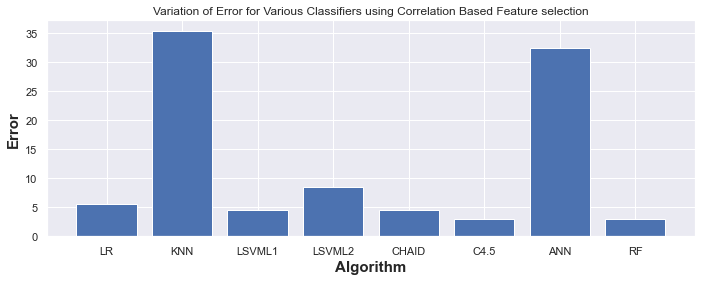


Fig. 4.15. Classification Error for ML classifiers in case of CFS

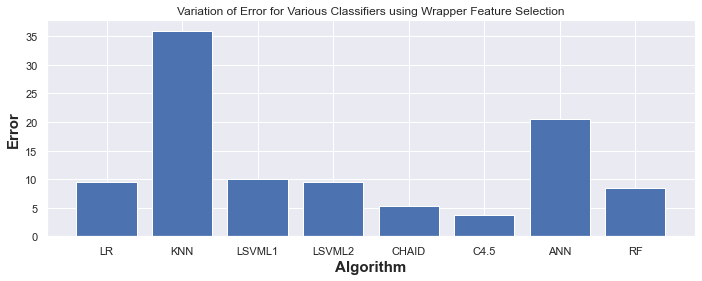


Fig. 4.16. Classification Error for ML classifiers in case of Wrapper FS

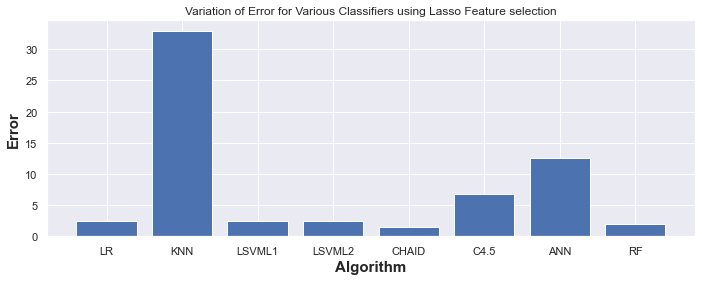


Fig. 4.17. Classification Error for ML classifiers in case of LASSO FS

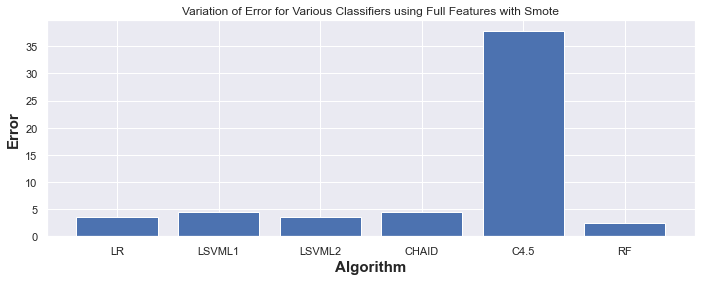


Fig. 4.18. Classification Error for ML classifiers in case of Full features with SMOTE

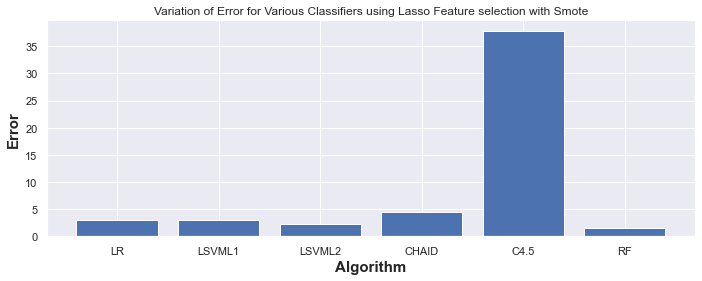


Fig. 4.19. Classification Error for ML classifiers in case of LASSO FS with SMOTE

**F- Measure:**

Here are the plotted graphs of F- Measure for the Machine Learning classifiers in case of Full feature selection, CFS, Wrapper FS, LASSO FS, Full features with SMOTE, LASSO with SMOTE respectively.

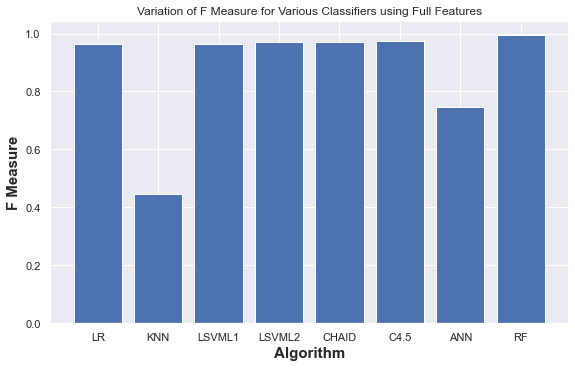


Fig. 4.20. F- Measure for ML classifiers in case of Full feature selection

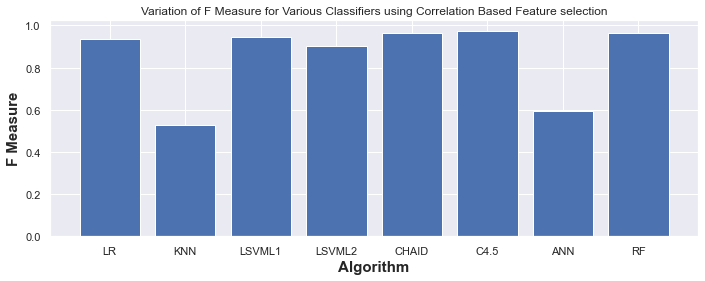


Fig. 4.21. F- Measure for ML classifiers in case of CFS

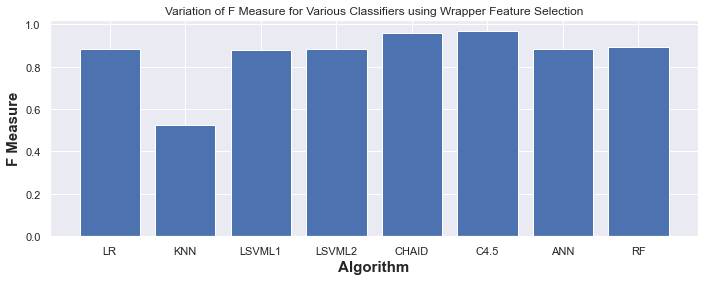


Fig. 4.22. F- Measure for ML classifiers in case of Wrapper FS

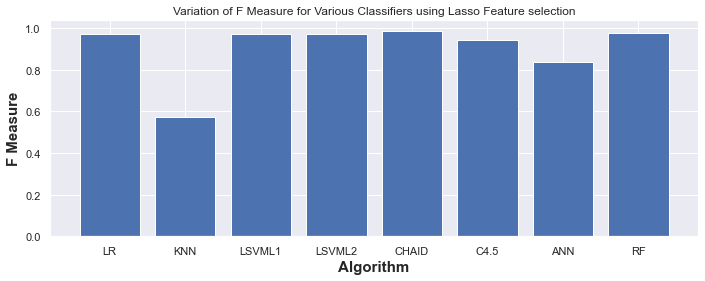


Fig. 4.23. F- Measure for ML classifiers in case of LASSO FS

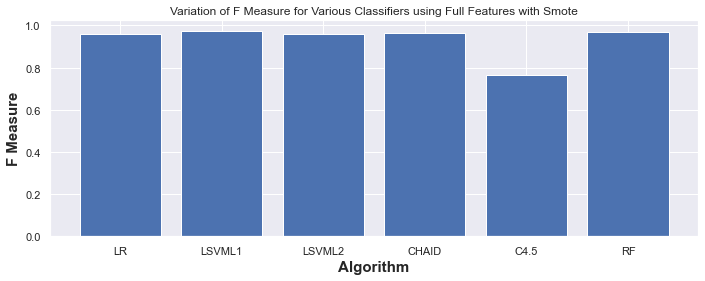


Fig. 4.24. F- Measure for ML classifiers in case of Full features with SMOTE

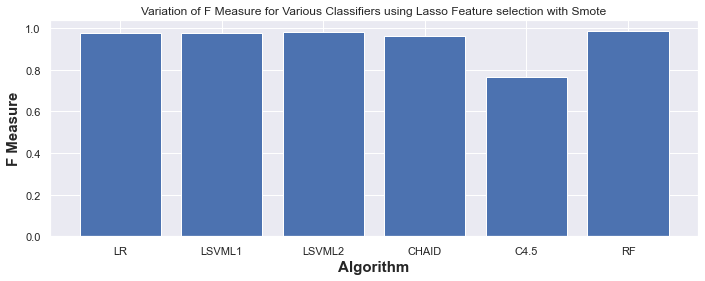


Fig. 4.25. F- Measure for ML classifiers in case of LASSO FS with SMOTE

**AUC:**

Here are the plotted graphs of AUC for the Machine Learning classifiers in case of Full feature selection, CFS, Wrapper FS, LASSO FS, Full features with SMOTE, LASSO with SMOTE respectively.

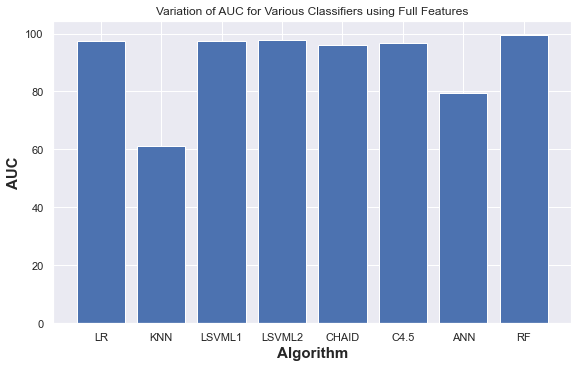


Fig. 4.26. AUC for ML classifiers in case of Full feature selection

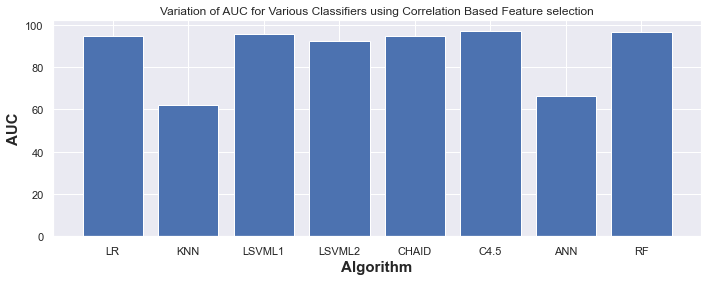


Fig. 4.27. AUC for ML classifiers in case of CFS

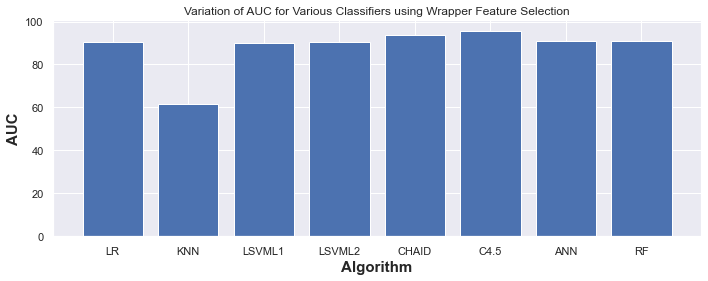


Fig. 4.28. AUC for ML classifiers in case of Wrapper FS

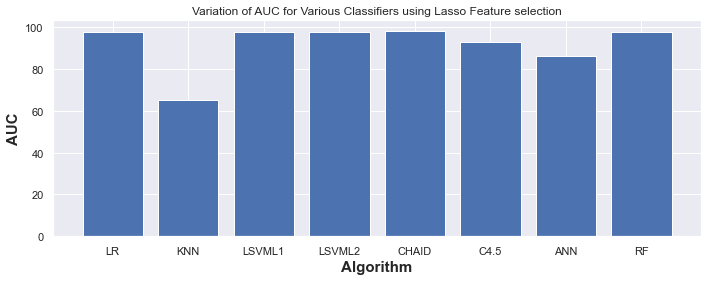


Fig. 4.29. AUC for ML classifiers in case of LASSO FS

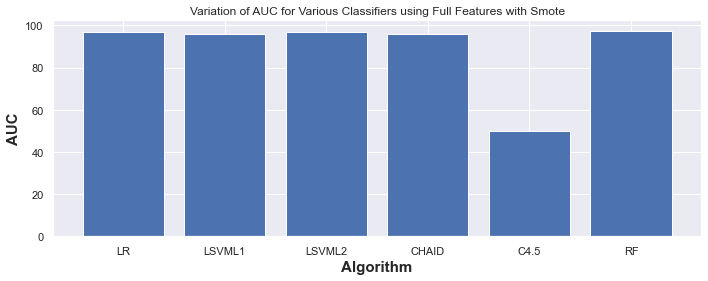


Fig. 4.30. AUC for ML classifiers in case of Full features with SMOTE

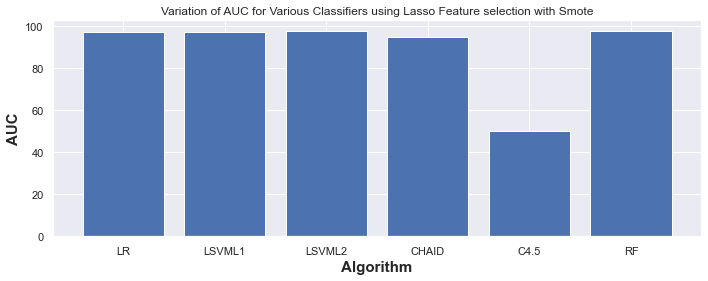


Fig. 4.31. AUC for ML classifiers in case of LASSO FS with SMOTE

**GINI Index:**

Here are the plotted graphs of GINI Index for the Machine Learning classifiers in case of Full feature selection, CFS, Wrapper FS, LASSO FS, Full features with SMOTE, LASSO with SMOTE respectively.

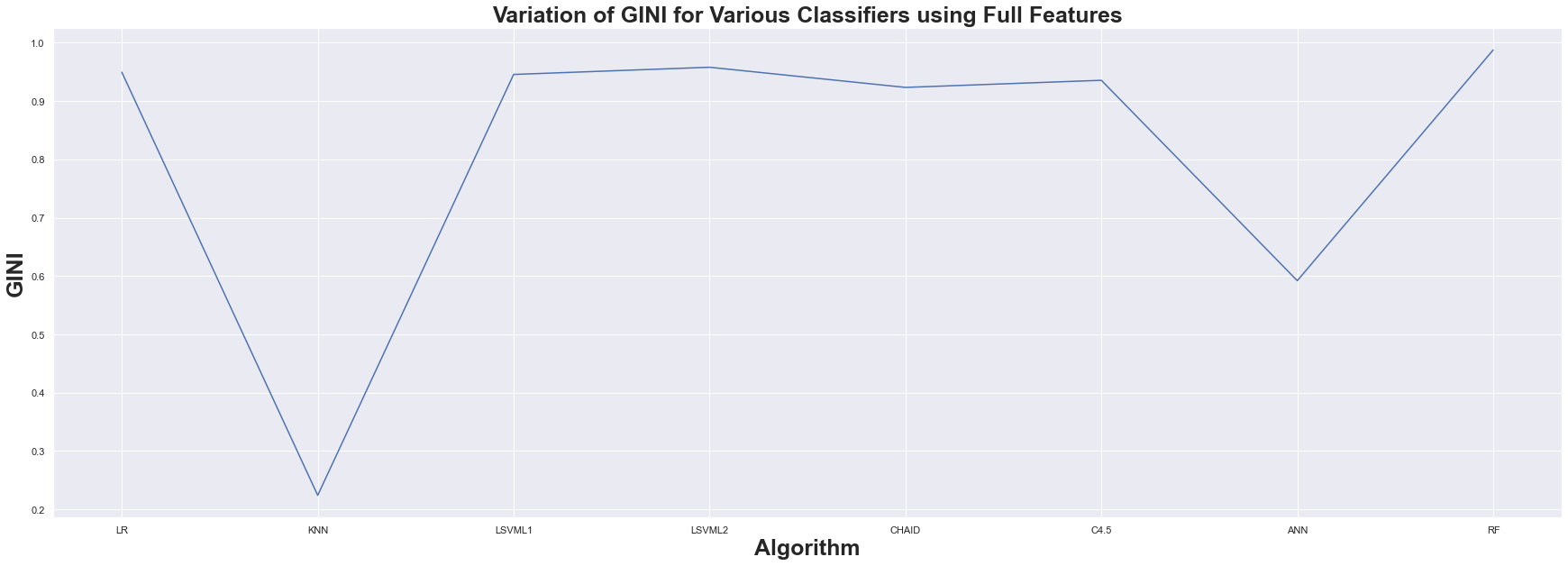


Fig. 4.32. GINI Index for ML classifiers in case of Full feature selection

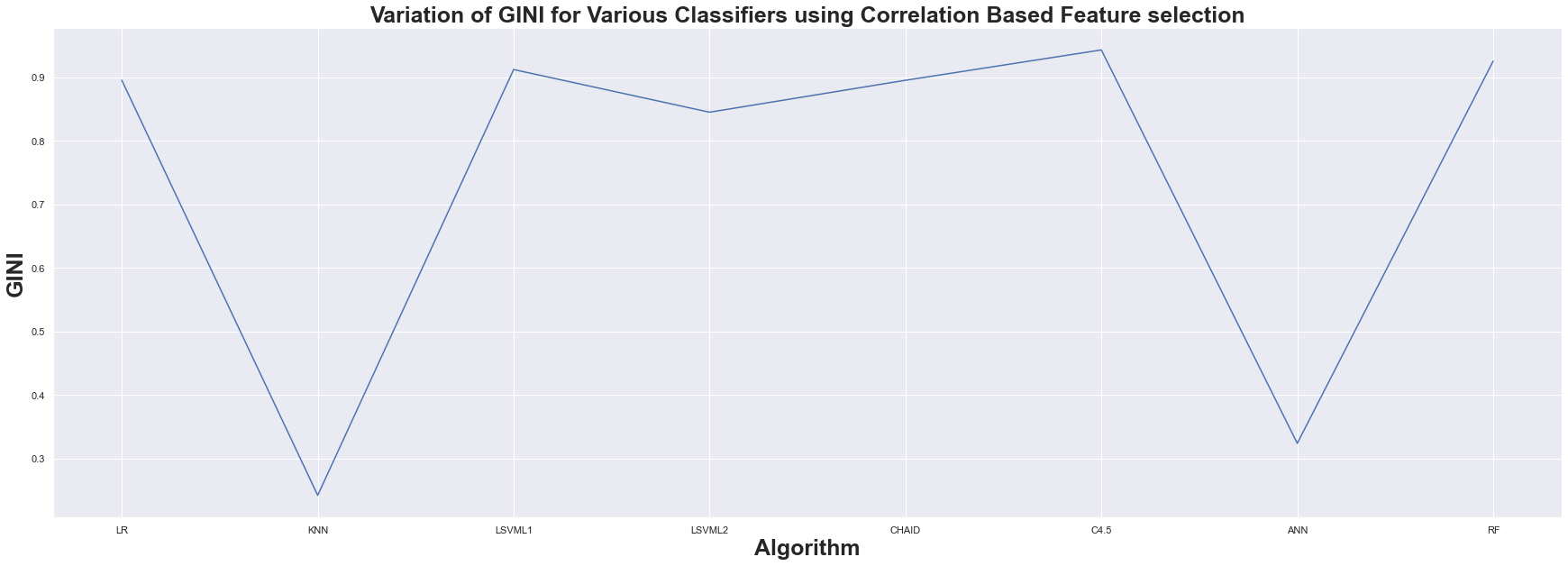


Fig. 4.33. GINI Index for ML classifiers in case of CFS

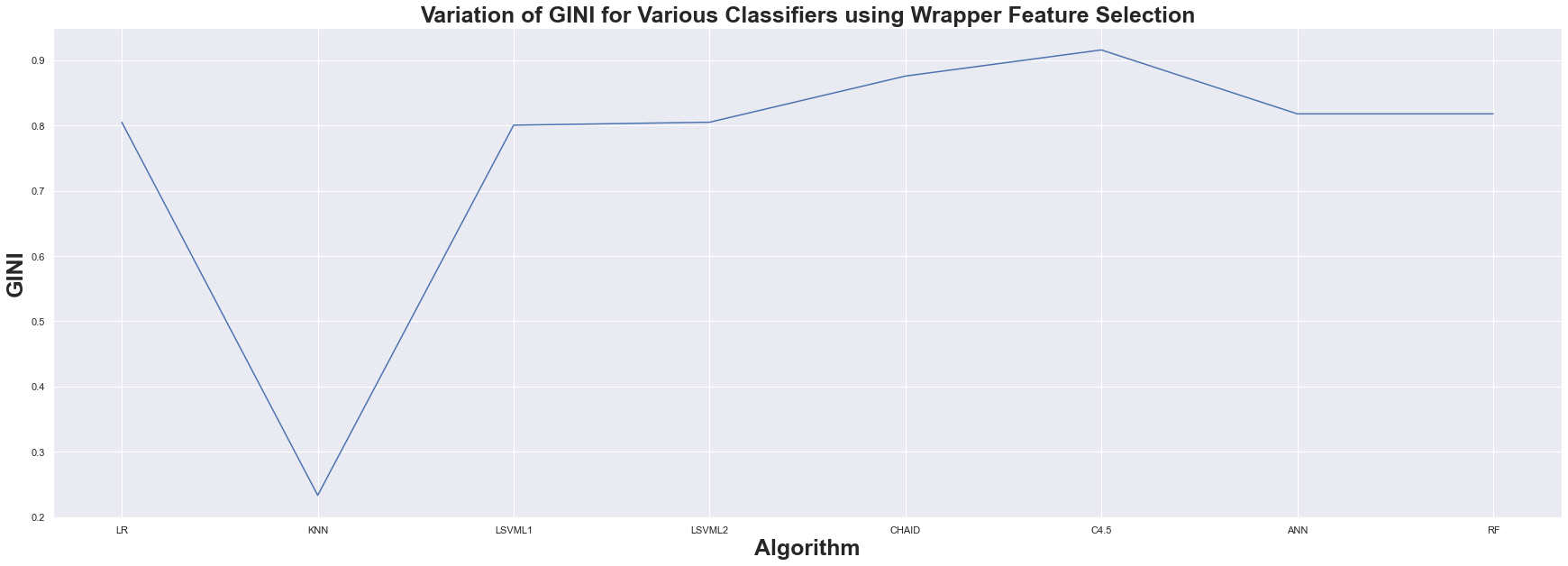


Fig. 4.34. GINI Index for ML classifiers in case of Wrapper FS

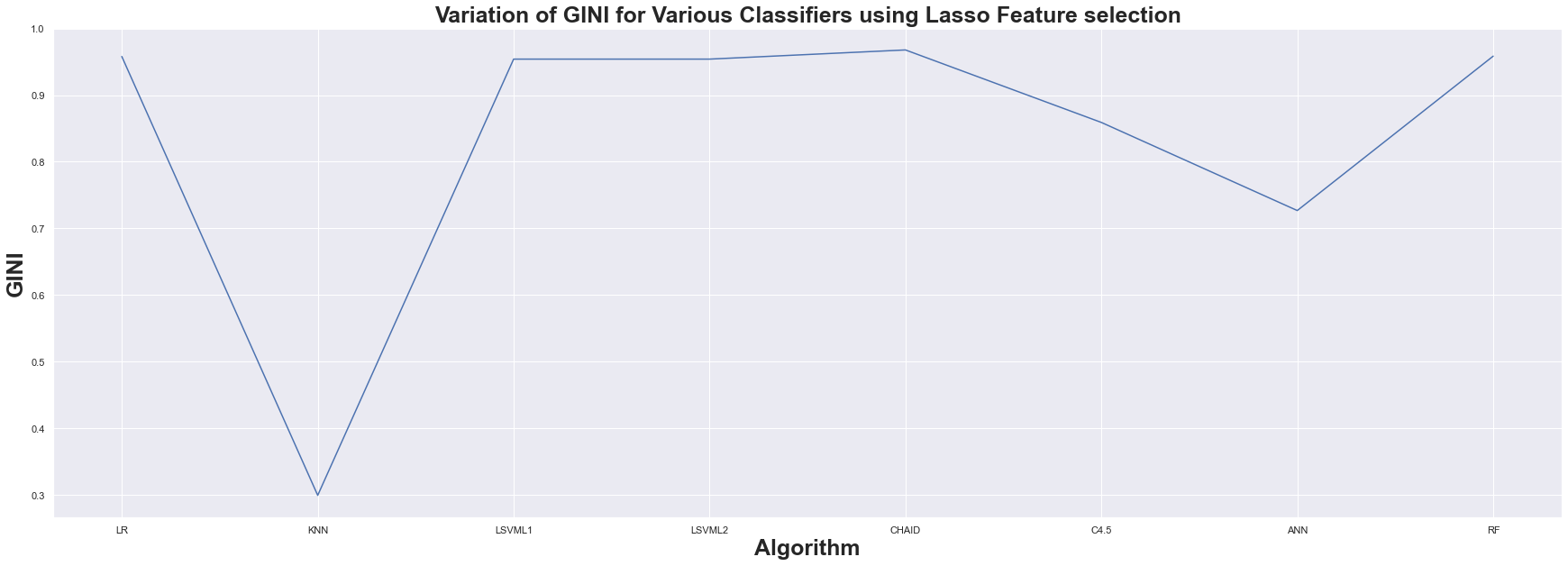


Fig. 4.35. GINI Index for ML classifiers in case of LASSO FS

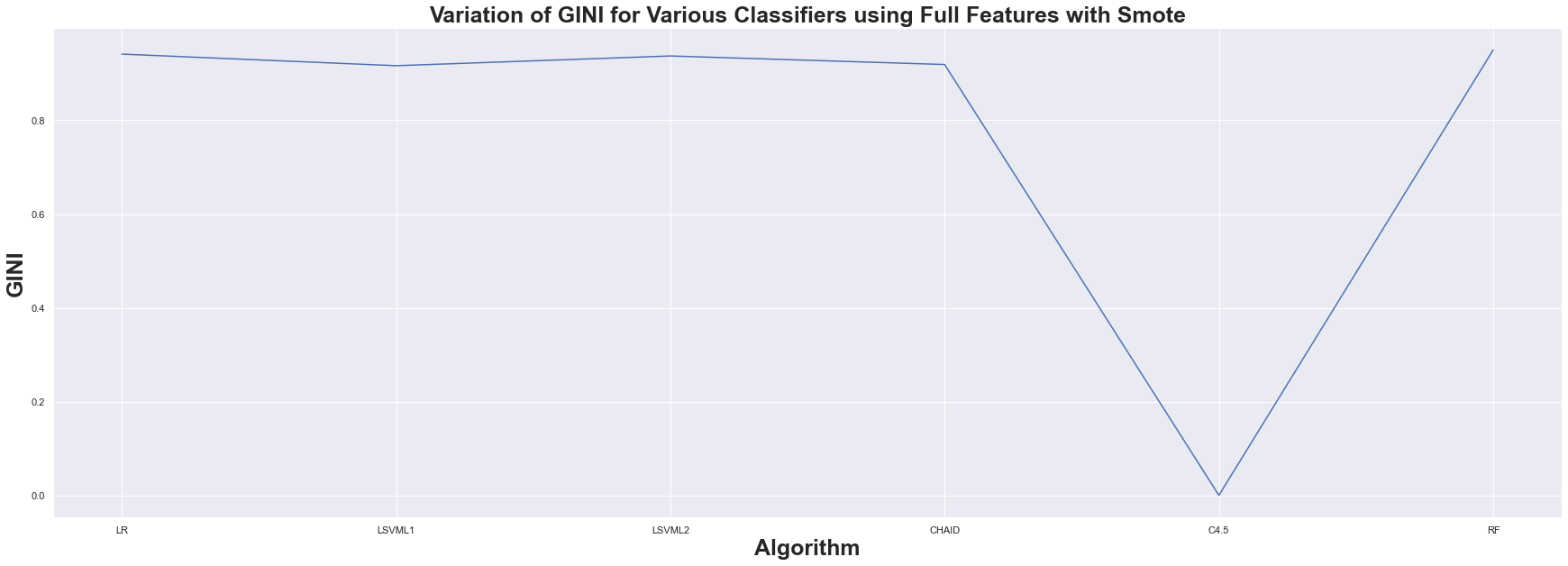


Fig. 4.36. GINI Index for ML classifiers in case of Full features with SMOTE

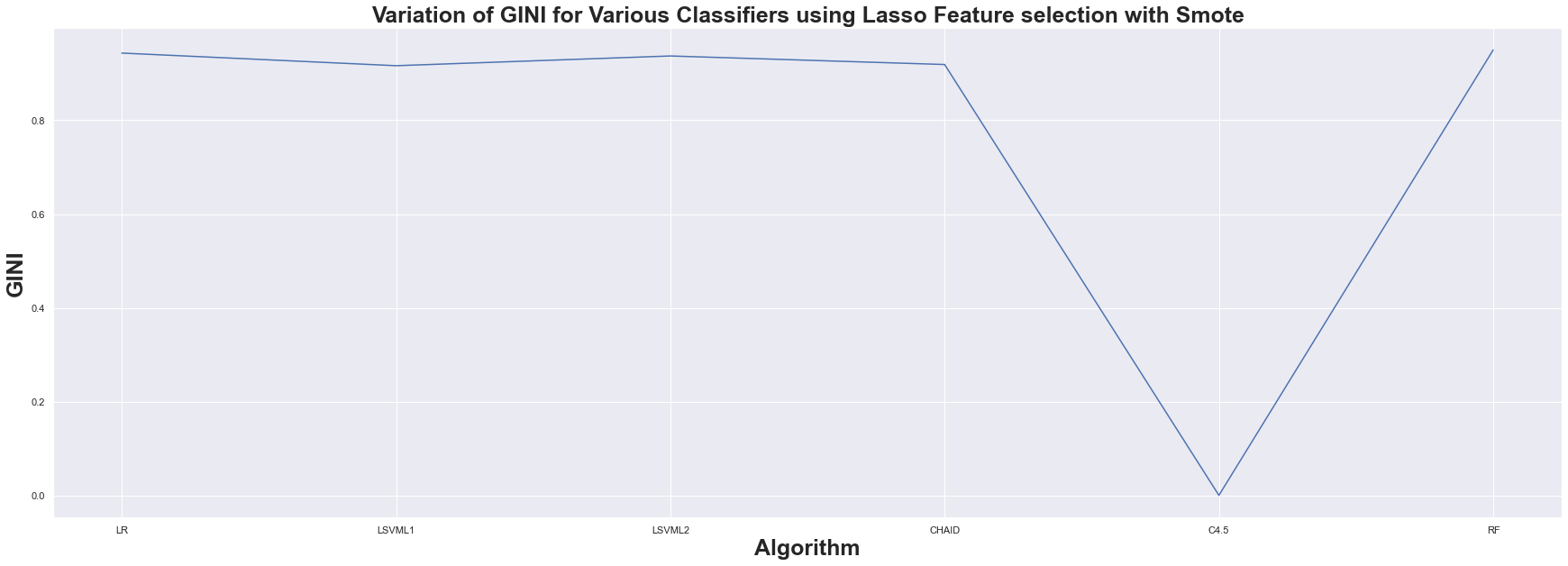


Fig. 4.37. GINI Index for ML classifiers in case of LASSO FS with SMOTE

## CHAPTER 5

### CONCLUSION AND FUTURE PLANS

**5.1. Conclusion**

In this paper, we primarily focus on CKD prediction using ML classifier algorithms. We used ML classifier algorithms such as LR, KNN, LSVM with penalty L1 & penalty L2, RF, CHAID, C 4.5, and ANN. Results for all FS methods, full feature, Correlation based, Wrapper, and LASSO FS are computed for each classifier.

In terms of accuracy and other metrics, LR, LSVM L1, LSVM L2, CHAID, C 4.5, and RF perform better than Full Features and LASSO Feature Selection.

The results of the above-mentioned classifiers and FS methods are computed using SMOTE. Because the KNN and ANN algorithms did not produce satisfactory results, we did not use them for SMOTE.

With LASSO Feature Selection and SMOTE, Random Forest performs best with an accuracy of 98.49 percent, followed by LSVM penalty L2 with an accuracy of 97.73 percent.

As a result of the findings, it was determined that Random Forest with LASSO Feature Selection with SMOTE provides more accurate results when predicting CKD.

**5.2. Future Plans**

Because our data set has an average number of instances, we oversampled it using SMOTE. In the future, considering a data set with a large number of instances, under sampling is possible and can be done using the same procedure.

## CHAPTER 6

### REFERENCES

[ 1] Pankaj Chittora, Sandeep Chaurasia, Prasun Chakrabarti, Gaurav Kumawat1,

Tulika Chakrabarti, Zbigniew Leonowicz, Michał jasiński, Lukasz Jasiński,

Radomir Gono, Elżbieta Jasińska, and Vadim Bolshev , (2021) “prediction of

chronic kidney disease - a machine learning perspective”, *ieee access.2021.3053763.*

[ 2] J. Qin, L. Chen, Y. Liu, C. Liu, C. Feng, and B. Chen, ‘‘A machine learning methodology for diagnosing chronic kidney disease,’’ *IEEE Access, vol. 8, pp. 20991–21002, 2020.*

[ 3] L. Kilvia De Almeida, L. Lessa, A. Peixoto, R. Gomes, and J. Celestino, ‘‘Kidney failure detection using machine learning techniques,’’ *in Proc. 8th Int. Workshop ADVANCEs ICT Infrastructures Services, 2020, pp. 1–8.*

[ 4] S. Shankar, S. Verma, S. Elavarthy, T. Kiran, and P. Ghuli, ‘‘Analysis and prediction of chronic kidney disease,’’ *Int. Res. J. Eng. Technol., vol. 7, no. 5, May 2020, pp. 4536–4541.*

[ 5] G. R. Vasquez-Morales, S. M. Martinez-Monterrubio, P. Moreno-Ger, and J. A. Recio-Garcia, ‘‘Explainable prediction of chronic renal disease in the colombian population using neural networks and case-based reasoning,’’ *IEEE Access, vol. 7, pp. 152900–152910, 2019.*

[ 6] M. Almasoud and T. E. Ward, ‘‘Detection of chronic kidney disease using machine learning algorithms with least number of predictors,’’ *Int. J. Adv. Comput. Sci. Appl., vol. 10, no. 8, pp. 89–96, 2019.*

[ 7] Y. Amirgaliyev, S. Shamiluulu, and A. Serek, ‘‘Analysis of chronic kidney disease dataset by applying machine learning methods,’’ in Proc. *IEEE 12th Int. Conf. Appl. Inf. Commun. Technol. (AICT), Oct. 2018, pp. 1–4.*

[ 8] W. Gunarathne, K. D. M Perera, and K. A. D. C. P Kahandawaarachchi,‘‘Performance evaluation on machine learning classification techniques for disease classification and forecasting through data analytics for chronic kidney disease (CKD),’’ *in Proc. IEEE 17th Int. Conf. Bioinf. Bioeng. (BIBE), Oct.2017, pp.291–296.*

[ 9] K. R. A. Padmanaban and G. Parthiban, ‘‘Applying machine learning techniques for predicting the risk of chronic kidney disease,’’ *Indian J. Sci. Technol., vol. 9, no. 29, Aug. 2016.*

## CHAPTER 6

### APPENDIX -BASE PAPER

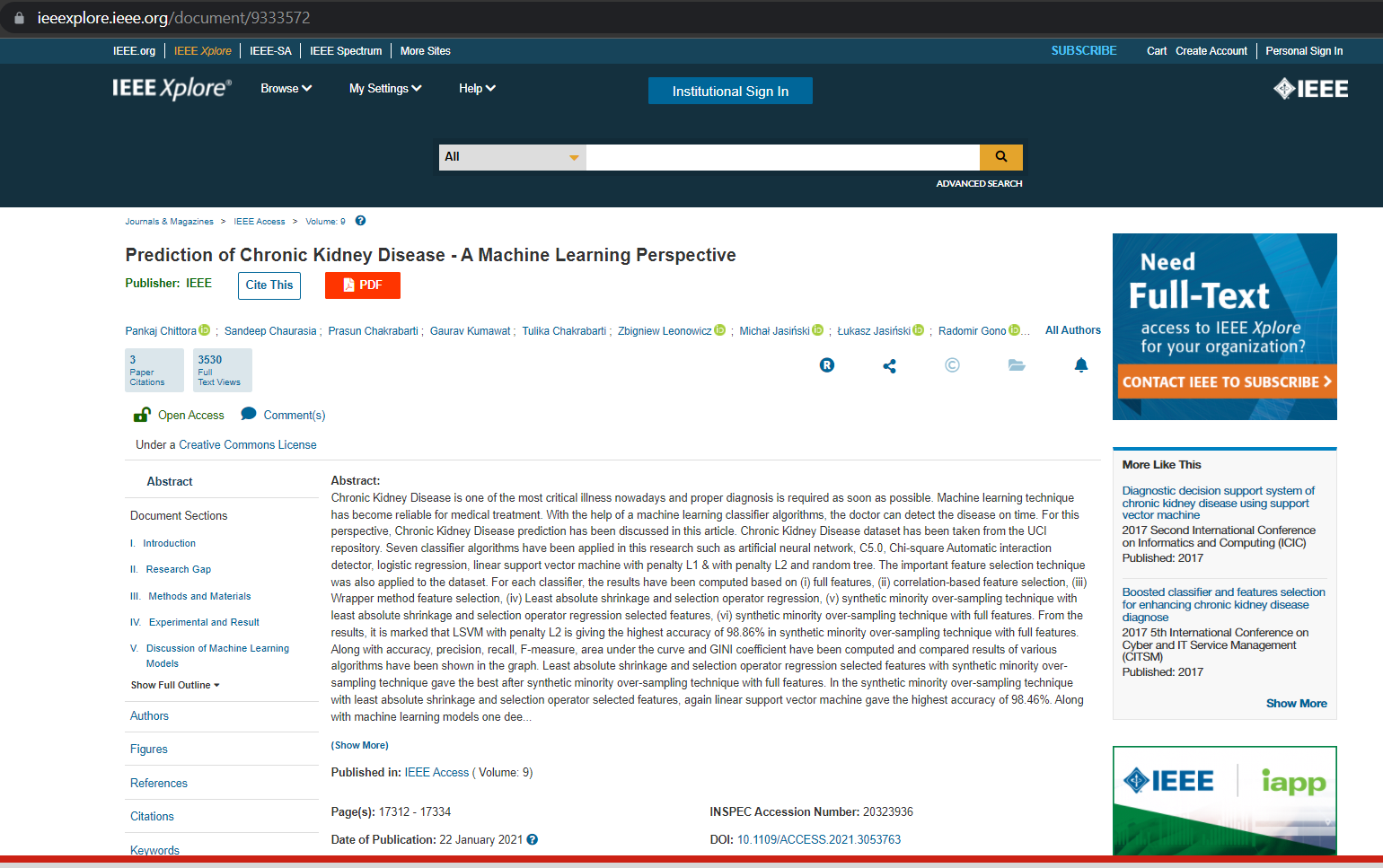


Fig. 6.1. Screenshot of Indexing