DETECTION OF CHRONICAL KIDNEY DISEASES

GALGOTIAS UNIVERSITY

Made by:

Ashish Kumar **19SCSE1010857**

Pawan Kumar **19SCSE1010869**

Guided by:

Dr Naresh Kumar(Professer)

Table of Contents:

Contents

Title			Page No.
Candidates Declaration			**I**
Acknowledgement			**II**
Abstract			**III**
Contents			**IV**
List of Table			**V**
List of Figures			**VI**
Acronyms			**VII**
Chapter 1	**Introduction**		**1**
	1.1	Introduction	**2**
	1.2	Formulation of Problem	**3**
		1.2.1 Tool and Technology Used	
Chapter 2	**Literature Survey/Project Design**		**5**
Chapter 3	**Functionality/Working of Project**		**9**
Chapter 4	**Results and Discussion**		**11**
Chapter 5	**Conclusion and Future Scope**		**41**
	5.1	Conclusion	**41**
	5.2	Future Scope	**42**
	Reference		**43**

Abstract - The objective of this paper is to build a CKD prediction model using machine learning techniques that can predict the risk of chronic kidney disease (CKD) in patients with Cardiovascular Disease (CVD) or at high risk of CVD. CVD is associated with worsening of renal functions. But patients with CVD remains often underdiagnosed and undertreated for CKD because mostly the clinical diagnosis and treatment are single organ centered in earlier stages. Machine learning algorithms have been widely used to predict and classify diseases in healthcare. Healthcare data is often imbalanced. In this analysis, the CKD prediction model is built using CVD data with imbalanced distribution of positive and negative cases. The analysis involves three stages: Stage I involves selecting the best model based on performance metrics that support imbalanced class distribution without applying any resampling techniques. Stage II involves oversampling the training data of the minority class using Synthetic Minority Oversampling Technique (SMOTE) and stage III involves randomly under-sampling the training data of the majority class to solve the class imbalance. The experimental results show that the MLP (Multi-Layer Perceptron)-SMOTE model performs better in predicting CKD with a better F-score, recall, precision, G-mean, balanced accuracy and RUC-AUC when compared to other models.

***Keywords*:** Chronic kidney disease; class imbalance; machine learning; sampling techniques.

Introduction

CKD is one of the leading causes of morbidity and mortality for individuals with CVD. A precise CKD risk prediction model developed from CVD patient data is critical for secondary prevention of CKD. Artificial Intelligence (AI) is enhancing the astuteness of medical professionals in diagnosis and prognosis in the field of nephrology. Early diagnosis of kidney disease by AI will help the health practitioners to screen potential kidney disease patients according to their risk levels. The availability of a huge volume of data and also with high quality is the greatest challenge in building an accurate and most efficient machine predictive model .

Various classifiers are deployed to create classification models for classifying the clinical CKD data. Supervised learning algorithms like logistic regression (LR), neural network (NN) and support vector machine (SVM) are used for creating classification models in. Classification and association rule mining techniques

are unified and utilized to paradigm a system for predicting and diagnosing CKD and its roots. Among, naive bayes (NB), decision tree (DT), SVM, k-nearest neighbor (K-NN) and JRip experimented on the medical data, K-NN achieved the highest accuracy. Apriori algorithm is applied to selected attributes of CKD data to extract strong rules based on the lift matrix.

Statistical method like multivariable cox's proportional hazards analysis is applied in over the high-risk CVD patients to determine the independent risk aspects like older age, history of coronary heart disease (CHD), diabetes mellitus, and smoking associated with developing CKD stages 3 to 5. Machine learning methods like multivariate regression model, classification and regression tree (CART), NB, bagged trees, ada boost and random forest are used in to build a prediction model for CVD disease. The CVD prediction model provided a three-year risk assessment of CVD. The prediction performance becomes unsatisfactory when prediction models are deployment into the local population. A knowledge-enhanced localized risk model is developed by to solve the localization issue. [Niehaus and Clifton , (2016)] proposed an extreme value theory (EVT) that can be applied to better quantify severity and risk in chronic disease.

The increasing research importance on novel machine learning approaches recommends that the modeling of chronic disease will continue to yield valuable discoveries for patients and doctors . Machine learning algorithms perform better when there is an equal number of records collected for all the target classes . But in reality, medical data for disease prediction cannot always be collected with equal distributions for diseased and non-diseased target classes. In the case of binary classification problem, one target class may have many instances than another, thus leading to an imbalance in the dataset . This problem is referred to as class imbalance.

The rest of the paper is organized as follows: section 2 elucidates the related work. Section 3 presents the materials and methods. Section 4 describes the results and discussions. Finally, section 5 presents the conclusion and outlines future work.

. PROPOSED MODEL

In this section, the classifiers were first established by different machine learning algorithms to diagnose the data samples. Among these models, those with better performance were selected as potential components. By analyzing their misjudgments, the component models were determined. An integrated model was then established to achieve higher performance.

A. ESTABLISHING AND EVALUATING INDIVIDUAL MODELS

The following machine learning models have been obtained by using the corresponding subset of features or predictors on the complete CKD data sets for diagnosing CKD.

1) Regression-based model: LOG
2) Tree-based model: RF

3) Decision plane-based model: SVM
4) Distance-based model: KNN
5) Probability-based model: NB
6) Neural network: FNN

Generally, in disease diagnosis, diagnostic samples are distributed in a multidimensional space. This space comprises predictors that are used for data classification (ckd or notckd). Samples of data in the space are clustered in different regions due to their different categories. Therefore, there is a boundary between the two categories, and the distances between samples in the same category are smaller. According to the effectiveness of classification, we choose the aforementioned methods for disease diagnosis. LOG is based on linear regression, and it obtains the weight of each predictor and a bias. If the sum of the effects of all predictors exceeds a threshold, the category of the sample will be classified as ckd or notckd. RF generates a large number of decision trees by randomly sampling training samples and predictors. Each decision tree is trained to find a boundary that maximises the difference between ckd and notckd. The final decision is determined by the predictions of all trees in the disease diagnosis. SVM divides different kinds of samples by establishing a decision surface in a multidimensional space that comprises the predictors of the samples. KNN finds the nearest training samples by calculating the distances between the test sample and the training samples and then determines the diagnostic category by voting. Naive Bayes classifier calculates the conditional probabilities of the sample under the interval by the number of ckd and notckd samples in each different measurement interval. FNN can analyse non-linear relationships in the data sets due to its complex structure, and the sigmoid activation function was used in the hidden layer and the output layer.

To evaluate model performance comprehensively, in the case of retaining the sample distribution in the original data, a complete data set was divided into four subsets evenly. For all of the above models, each subset was utilized once for testing, and other subsets were utilized for training, the overall result was taken as the final performance. With the exception of RF, the rest of the models were established using the selected variables by feature extraction. RF does not require prior feature extraction, because predictors are selected randomly when each decision tree is established. In addition, when using KNN and FNN, all the categorical variables were converted into numeric types: categories 0 and 1 were converted to values 0 and 1, respectively, and the complete data sets were then normalised with the mean that is equal to 0 and the standard deviation that is equal to 1. Details of all are as follows:

1) The output of LOG was the probability that the sample belongs to notckd, and the threshold was set to 0.5.

2) RF was established using all variables. Two strategies were used to determine the number of decision trees generated. One is to use the default 500 trees and the other is to use the number of trees corresponding to the minimum error in the training stage. The RF was established using both strategies and evaluated on the data sets obtained by KNN imputation. The same random number seed 1234 was used to

divide data and establish model, and the accuracy is shown in Table 4. It can be seen that the default number of trees is a better choice, therefore we selected the default 500 trees to establish RF.

TABLE 4. The accuracy of two types of RF after the KNN imputation was run.

Number of trees	K = 3	K = 5	K = 7	K = 9	K = 11
default	99.75%	99.75%	99.50%	99.75%	99.50%
error minimum	99.50%	98.75%	99.00%	99.75%	99.25%

3) The models of SVM were generated by using the RBF kernel function, and the function is described as follow:

$$K(x1,x2) = e^{-\gamma\|x1-x2\|^2}$$

where γ was set to [0.1, 0.5, 1, 2, 3, 4]. Parameter C represents the weight of misjudgment loss, and it was set to [0.5, 1, 2, 3]. In each calculation of the model training, the algorithm selects the best combination of parameters to establish the model by grid search. 4) For the NB, the value of Laplace was equal to 1. 5) For the KNN, due to the nearest Euclidean distance with the detected sample, when the number of samples that are selected in training data set is an even number, the algorithm randomly selects a category as the output result of the detected sample in the situation wherein the number of selected samples belonging to ckd and notckd are the same. To avoid this in the work, the nearest neighbor parameter was set to $[1,3,5,\ldots,19]$. In each calculation of model training, the algorithm selected the best parameter to establish the model by grid search.

6) For the FNN, the network had a hidden layer. Presently, there is no clear theory in determining the best number of hidden layer nodes in a neural network. A method proposed in the previous study that was used to evaluate the performance of neural networks by increasing the number of hidden layer nodes one by one [35] was used in this study. The number of hidden layer nodes was increased one by one from 1 to 30. Then, the best result was selected.

To ensure the repeatability and comparability of the results, in the division of data, the establishment of RF with FNN, and the selection of the best parameters of SVM with KNN, the same seed of 1234 was used. For the random imputation, the step of feature extraction was run on the complete data set obtained. Then, the models were established and evaluated by using the extracted features. Because of the randomness of the random imputation, the whole process was repeated five times to get the average result. For the KNN imputation and the mean and mode imputation, due to the certainty of data, the evaluation of models was executed once. After the feature extraction methods of optimal subset regression and RF were run, the accuracy of the basic models on the complete data sets are shown in Table 5 and Table 6, respectively.

TABLE 5. The accuracy (%) of the basic models after the optimal subset regression.

Imputation	LOG	RF	SVM	NB	FNN	KNN
KNN with K=3	98.75	99.75	99.50	94.25	98.75	97.75
KNN with K=5	98.75	99.75	99.50	93.75	98.50	98.00
KNN with K=7	98.00	99.50	99.50	94.00	98.25	98.25
KNN with K=9	98.75	99.75	99.50	93.75	98.75	98.00
KNN with K=11	99.00	99.50	99.50	93.75	98.50	97.75
Mean and mode	95.25	98.75	98.50	96.75	98.25	98.25
Random	96.25	97.85	97.00	95.00	97.40	95.60

For the FNN, we selected the model with the highest accuracy. For the RF, the model was established using all variables.

TABLE 6. The accuracy (%) of the basic models after the features extraction of RF was run.

Imputation	LOG	RF	SVM	NB	FNN	KNN
KNN with K=3	96.25	99.75	98.25	88.75	99.00	99.00
KNN with K=5	96.00	99.75	99.00	88.75	99.00	98.50
KNN with K=7	96.50	99.50	98.00	88.75	99.00	98.50
KNN with K=9	97.00	99.75	98.50	88.50	99.00	99.00
KNN with K=11	97.00	99.50	98.25	88.25	99.50	99.25
Mean and mode	96.50	98.75	99.25	95.75	98.75	99.25
Random	95.60	97.90	97.65	93.90	97.45	96.90

For the FNN, we selected the model with the highest accuracy. For the RF, the model was established using all variables.

It can been seen from Tables 5 and 6 that the optimal subset regression is more suitable for LOG and SVM when the KNN imputation is used, and the feature extraction method of RF is more suitable for FNN and KNN. When the KNN imputation is used, the accuracy of LOG and SVM is significantly improved (Table 5). In Table 6, the accuracy of LOG and SVM is relatively low, which might be due

to the fact that there are too many redundant variables compared to the optimal subset regression. The accuracy of FNN is slightly improved and RF shows better performance when the KNN imputation is used both in Tables 5 and 6. For the NB and the KNN, the performance of the models when using KNN imputation is not very ideal compared to using random imputation or mean and mode imputation in Tables 5 and 6. The above result also proves the validity of the KNN imputation, since KNN imputation does improve the accuracy of some models, such as LOG, RF and SVM (Table 5). From Tables 5 and 6, LOG and SVM with the use of optimal subset regression, KNN and FNN with the use of the feature extraction of random forest and RF have better performance. Therefore, they are selected as the potential component models.

B. MISJUDGMENT ANALYSIS AND SELECTING COMPONENT MODELS

After evaluating the above models, the potential component models were extracted for misjudgment analysis to determine which would be used as the components. The misjudgment analysis here refers to find out and compare the samples misjudged by different models, and then determine which model is suitable to establish the final integrated model. The misjudgment analysis was performed on the extracted models. The prerequisite for generating an integrated model is that the misjudged samples from each component model are different. If each component model misjudges the same samples, the generated integrated model would not make a correct judgement for the samples either. When the data were read, each sample was given a unique number ranging from 1 to 400. The numbers of misjudgments for the extracted models on each complete data are shown in Table 7, and the black part indicates that the samples were misjudged by other models except FNN.

In Table 7, for the FNN, it can be seen that most of the misjudgments are simultaneously misjudged by other models. In addition, the performance of FNN is affected by the number of nodes in the hidden layer. It is not easy to establish a unified model for different data. Therefore, the FNN was excluded firstly. For the best model (RF), when K equaling to 7, only one misjudgment is simultaneously misjudged by the LOG. In other cases, all the samples that are misjudged by RF can be correctly judged by the rest of the models. Hence, the combinations of the RF with the rest of the models could be used to establish an integrated model. Next, we investigate which specific model combination could generate the best integrated model for diagnosing CKD. From Tables 5 and 6, it can be seen that there is no significant difference between LOG, SVM and KNN. In the case where the performance of the models is similar, the models are evaluated by the complexity of the algorithm, the running time and the computational resources consumed. LOG, RF, SVM and KNN were run five times on each complete data,

TABLE 7. The numbers of misjudgments of the extracted models.

K value	Model	The numbers of misjudgments

K=3	LOG	189, 225, 42, 193, 212
	RF	166
	SVM	225, 118
	FNN (23)	90, 189, 225, 210
	KNN	193, 106, 103, 210
K=5	LOG	189, 225, 193, 212, 267
	RF	166
	SVM	225, 118
	FNN (27)	90, 225, 229, 210
	KNN	183, 193, 229, 132, 103, 210
K=7	LOG	90, 92, 189, 225, 193, 212, 267, 340
	RF	90, 166
	SVM	225, 118
	FNN (27)	90, 225, 229, 210
	KNN	183, 193, 229, 132, 103, 210
K=9	LOG	92, 189, 225, 193, 212
	RF	166
	SVM	225, 118
	FNN (23)	90, 225, 229, 210
	KNN	183, 132, 103, 210
K=11	LOG	189, 225, 193, 212
	RF	90, 166
	SVM	225, 118
	FNN (5)	225, 229
	KNN	183, 132, 103

For the FNN, the best model is selected, and the number in bracket represents the number of nodes in the hidden layer.

TABLE 8. The time spent by RF, LOG, SVM and KNN on the complete data.

KNN imputation with K	RF (s)	LOG (s)	SVM (s)	KNN (s)
3	0.382	0.138	16.114	2.796
5	0.376	0.144	15.836	2.788
7	0.386	0.140	16.222	2.864
9	0.396	0.128	16.276	2.822
11	0.394	0.132	16.104	2.766

and the average time taken are summarized in Table 8. It can be seen that the SVM and KNN take more time than the LOG and RF. In addition, SVM and KNN are also effected by their respective model parameters, so the parameters

need to be adjusted before the models are established, which means more manual intervention is needed. For the LOG, there was no additional parameter that need to be adjusted. For the RF, the default parameters of the model were used. Hence, a combination of the LOG and the RF was selected to generate the final integrated model.

C. ESTABLISHING THE INTEGRATED MODEL

LOG and RF were selected as underlying components to generate the integrated model to improve the performance of judging. The probabilities that each sample was judged as notckd in LOG and RF were used as the outputs of underlying components. These two probabilities of each sample were obtained and could be expressed in a two-dimensional plane. In the complete CKD data sets, the probability distributions of the samples in a two-dimensional plane are similar. Therefore, the probability distribution of samples when K equaling to 11 is shown in Fig. 3.

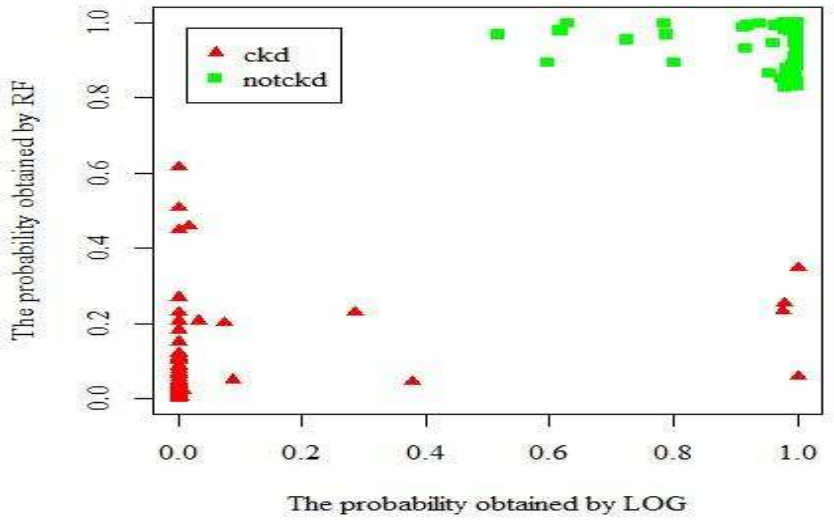


FIGURE 3. The probability distribution of the samples in the complete CKD data set (at K = 11), the horizontal axis and the vertical axis represent the probabilities that the samples were judged as notckd by the LOG and the RF, respectively.

It can be seen from Fig. 3 that the samples have different aggregation regions in the two-dimensional plane due to the different categories (ckd or notckd). In general, samples with ckd are concentrated in the lower left part, while the notckd samples are distributed in the top right part. Due to the fact that the results in the two models are different, some samples are located at the top left and lower right, and one of the two models makes the misjudgments. Perceptron can be used to separate samples of two categories by plotting a decision line in the two-dimensional plane of the probability distribution. Ciaburro and Venkateswaran defined perceptron as the basic building block of a neural network, and it can be understood as anything that requires multiple

inputs and produces an output [36]. The perceptron used in this study is shown in Fig. 4.

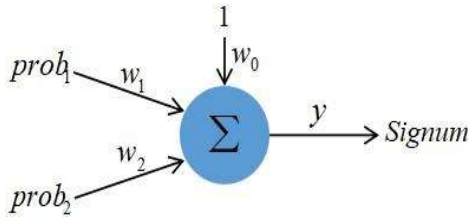


FIGURE 4. The structure of the perceptron used in this study.

In Fig. 4, $prob_1$ and $prob_2$ are the probabilities that a sample was judged as notckd by LOG and RF, respectively. w_0, w_1 and w_2 are the weights of input signals. w_0 corresponds to 1, w_1 corresponds to $prob_1$ and w_2 corresponds to $prob_2$, respectively. y is calculated according to (7):

$$y = w_0 + w_1 \times prob_1 + w_2 \times prob_2. \quad (7)$$

The input signal corresponding to the weight w_0 is 1, which is a bias. The function of Signum is used to calculate output by processing the value of y as follows: If $y > 0$, then the output = 1, whereas if $y < 0$, then the output = -1. For the output, 1 corresponds to notckd, whereas -1 corresponds to ckd. A single perceptron is a linear classifier that can be used to detect binary targets. The weights are the core of the perceptron and adjusted in the training stage. $y = 0$ is the decision line, and this line can be described as (8):

$$prob_2 = -\frac{w_1}{w_2} prob_1 - \frac{w_0}{w_2} \quad (8)$$

.

In the training stage, the models of LOG and RF were established by the training data at first. Then, a new training data set was generated though combining the probabilities of output of the two component models on the training data and the labels of the samples. This new training data set was used to establish the perceptron. For the binary classification, the samples have two types of labels, i.e. $Y = \pm 1$. The output of perceptron is calculated according to (7),

we use $g(X)$ to represent the matrix form of this calculation, where $W = [w_1, w_2]$, $X = [prob_1, prob_2]^T$, and $b = w_0$. When the $g(X) > 0$, the output = 1, whereas the $g(X) < 0$, then the output = -1. Therefore, for all samples correctly judged by the model, the following equation is valid:

$$Y \times g(X) = Y(WX + b) > 0. \quad (9)$$

For all misjudgments, the value of (9) is less than zero, and the large the absolute value, the more serious the model misjudges the samples. Hence, for a misjudged sample (X_i, Y_i), the loss of the perceptron can be expressed as (10):

$$L = -Y_i(WX_i + b). \qquad (10)$$

The perceptron is trained by the gradient descent method to adjust the weight and bias. The partial derivative of the weight and bias of the loss function are expressed as follows:

$$\frac{\partial L}{\partial W} = -Y_i X_i^T \qquad (11)$$

$$\frac{\partial L}{\partial b} = -Y_i \qquad (12)$$

Therefore, in the training stage, for each misjudgment, the weight and bias are updated by (13) and (14):

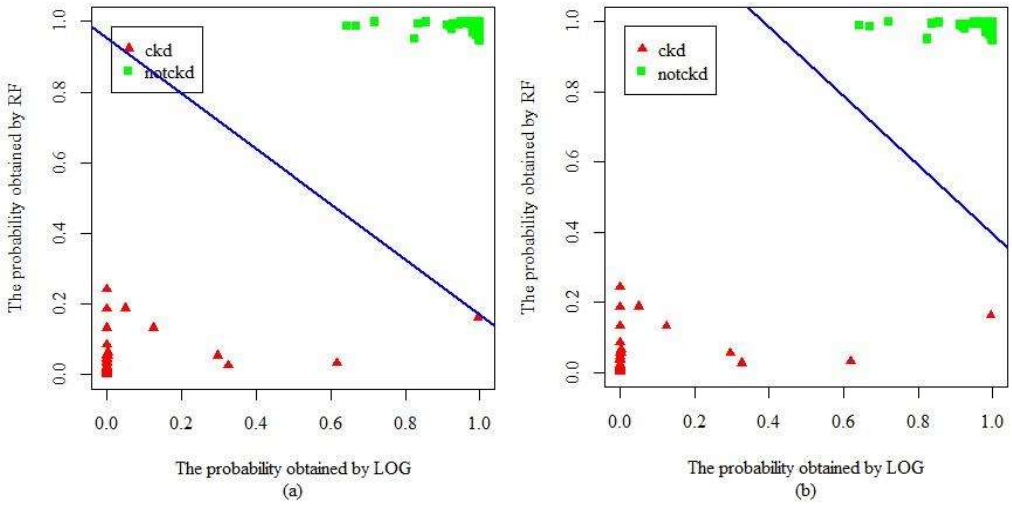
$$W = W + \eta Y_i X_i^T \qquad (13)$$

$$b = b + \eta Y_i \qquad (14)$$

where the η is the learning rate.

However, for the bias, when the updating method of (14) was used, the obtained decision line could classify the samples, but the line was located at the edge of the solution area, so it is not reliable. To solve this problem, a new bias adjustment strategy proposed in chapter 4 of the previous literature [36] was referred and used, which is expressed in (15):

$$b = b + \eta Y_i R^2 \qquad (15)$$



where the R is the maximum of the L2 norm of the eigenvectors in all training samples. When the (15) was used, the obtained decision line could correctly classify the samples, and the line was located in the middle of the solution area, so it is more reliable than (14). When the second subset was utilized for testing (at K = 11), the above phenomenon was obvious. Figs. 5(a) and (b) plot the decision line

constructed by the perceptron on the training data set when the updating strategies of (14) and (15) were used, respectively. It can be seen that the updating strategy of (15) in Fig. 5(b) is more reliable than (14) in Fig. 5(a). Therefore, (13) and (15) were used as the updating strategies of the perception. The pseudo code of the model is described as follows:

2. Related Work

1.Data collection

In machine learning, the quality and quantity of input data that is used for training the classifiers are very important. Most algorithms perform well when the prior probabilities of the target classes are similar. Data is said to be imbalanced if at least one of the target variable values has a significantly smaller number of instances when compared to the other values. Class imbalance is one of the vital issues in machine learning classification tasks. Machine learning algorithms trained on imbalanced data emphasize exploiting the total accuracy over the entire dataset leading to more attention being paid to the majority class samples . Due to this scenario, the minority class samples are poorly projected by the learning model.

proposed a two-phase classification model to solve the class imbalance problem for predicting type II diabetes. SMOTE was used to rebalance the data. The pre-processed data was then trained using a DT classifier. The DT showed increased performance by reducing class imbalance. Class imbalance causes difficulties for classifiers. studied data-driven methods and the algorithm-driven approaches for dealing with class imbalance problems for the autism diagnosis. The approaches on data fine-tune the class proportion in the input data to generate a balanced dataset. In algorithm-driven approaches the classification algorithm is fine- tuned to create a model that learns more from the minority class. Thus, the dataset will remain as imbalanced. In this approach, no changes are made to the input data distribution. proposed a concordant partial area under the receiver operating characteristic (ROC) for measuring the performance of the machine learning algorithms on imbalanced data with low prevalence.

experimented with SMOTE, borderline SMOTE (BLSMOTE), majority weighted minority oversampling technique (MWMOTE), and k-means SMOTE (KMSMOTE) to handle the class imbalance issue in the prediction of injury severity using machine learning techniques. tackled class imbalance with

SMOTE for predicting diabetes. Experiments were conducted with NB, SVM-radial basis function, C4.5 and repeated incremental pruning to produce error reduction (RIPPER). SMOTE sampled data applied to C4.5 DT produced better results than the other three classifiers.

developed an active balancing mechanism for the biomedical data under-sampling. Gaussian NB and entropy were used to evaluate sample information and retain valuable samples of the majority class to achieve under-sampling. proposed a reduced universum twin SVM for dealing with class imbalance learning. The model makes a balanced environment for the classification by incorporating prior information from the universum data points.

esigned an affinity and class probability based fuzzy SVM (ACFSVM) approach for imbalanced datasets classification tasks. The proposed technique gives more importance to majority class samples with higher affinities and class probabilities ultimately skewing the final classification boundary toward the majority class. But the minority class samples are dispensed with high memberships to promise their importance for the learning. resolved the problem of imbalance by under-sampling the training neuro imaging data during their work to analyze Alzheimer's disease. proposed a trainable under sampling method that applies evaluation metric optimization into the data sampling procedure. By using such an optimization, the method learns the instances to be discarded and the instances to be preserved.

Previous researches on class imbalance have focused on a few diseases but not on predicting CKD. In this work, a CKD predictive model is built by balancing the imbalanced CVD data by applying SMOTE over minority class and then training with MLP (Multi-layer Perceptron) classifier.

2. Data source

Table introduce Imbalanced class distribution of CVD dataset

	lass	
	No	**Yes**
Count	435	56
Percentage	88.6	11.4

The data for analysis is an electronic medical record (EMR) collected by during their research on CKD. The CVD dataset has 23 features of CVD or at- risk CVD patient and 491 instances. The target class "EVENTCKD35" has two values "Yes" and "No". CKD cases are represented as "Yes" and normal cases are represented as "No". Table 1 shows the number of positive cases for CKD in CVD patients' data (class 1) is lesser than the number of negative cases for CKD

(class 0). This makes it clear that the dataset that is taken for experimentation has class imbalance problem.

3. Proposed CKD prediction experimental framework

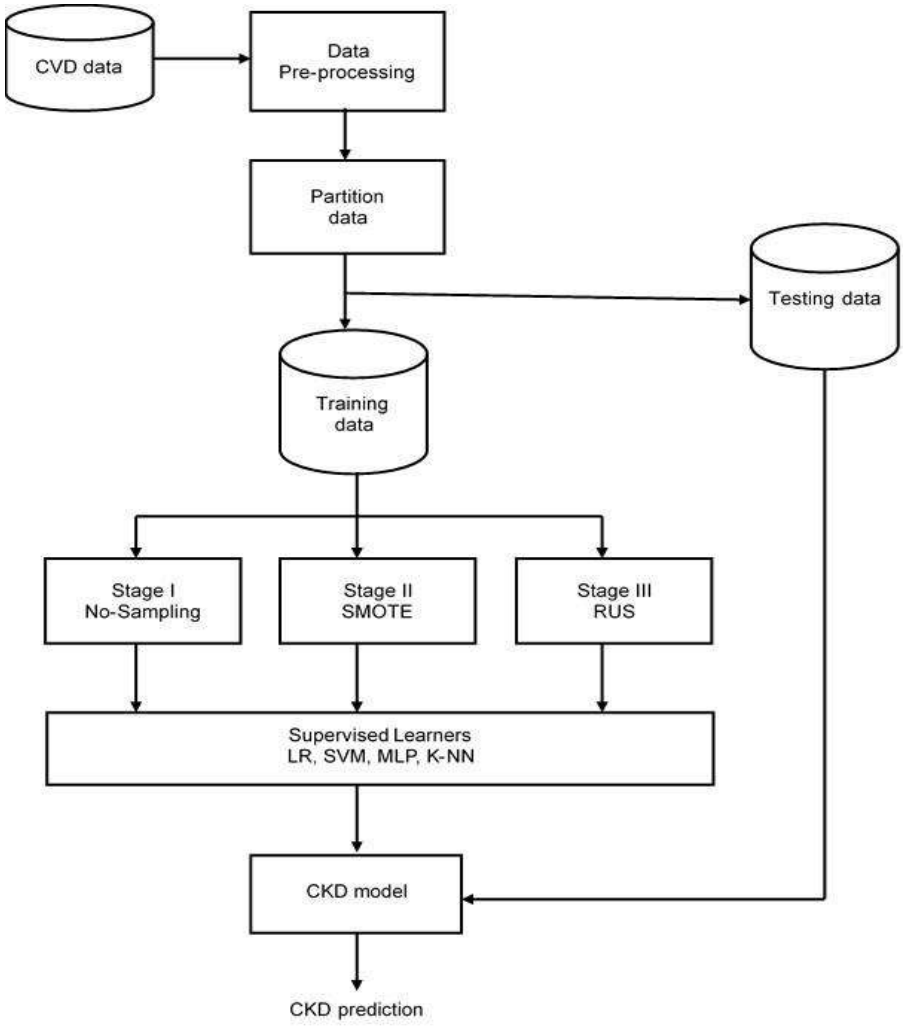


Fig. 1. Proposed CKD prediction experimental framework

The proposed CKD prediction experimental framework on class imbalanced CVD data is shown in Figure 1. The proposed framework involves steps like data pre-processing, solving class imbalance problem using sampling techniques, building the model using supervised learners and selecting the best models that outperforms others in terms of performance metrics. The framework also involves in applying the imbalanced data in building the model. Results obtained on predictions using no sampling and sampling techniques are compared.

4. Data pre-processing

The dataset is pre-processed before feeding it to a classification algorithm. The features in the CVD dataset consist of patient history details like gender, age, age-categories (split into three age groups) and the presence or absence of health

conditions like diabetes mellitus (DM), CHD, vascular disease, smoking status, hypertension (HTN), dyslipidemia (DLD), obesity. It also includes the history of intake of medicine for DLD, DM, HTN and angiotensin-converting enzyme inhibitors/angiotensin II receptor blockers (ACEI/ARB). Laboratory values includes cholesterol, triglycerides, HgbA1C (glycosylated Hemoglobin, type A1C), Creatinine, sBP (Systolic blood pressure), dBP (Diastolic blood pressure), eGFR (estimated Glomerular Filtration Rate), BMI (Body mass index). Further, it includes the patient observation time in months and the target class label 'EventCKD35' [Shamsi *et al*., (2018)]. The input features age-categories and age represent the same information about age. So, the age-categories feature is removed. Also, the feature 'eGFR' calculated using standard formulas may determine CKD directly [6]. This may create a hindrance to learn about the other features that may contribute to detect CKD. So, it was removed. The dataset contains a few missing values for the features 'triglycerides' and 'HgbA1C'. Since the count of missing values is very less, the instance with missing values is ignored. After removing missing values, 460 instances were representing the medical records of CVD patients.

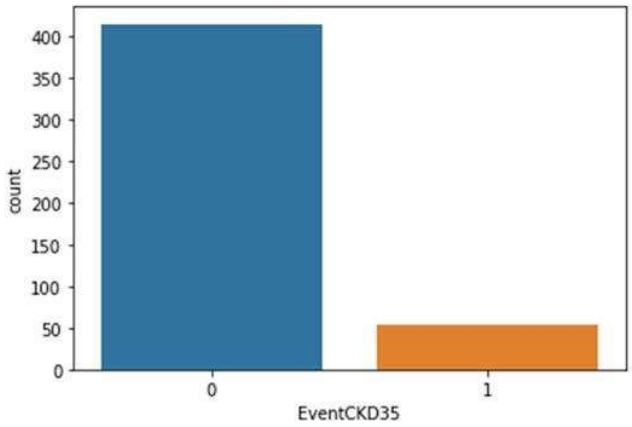


Fig. 2. Class distribution

The class distribution is shown in Figure 2. There are 415 class 0 records and 54 class 1 records. The dataset has 21 input features and one target feature named as 'EventCKD35'. Among the 21 input features, 9 features are numerical in nature and 12 features are categorical with values "yes" and "no". The values for categorical features are encoded to numerical 1 or 0 using label and one hot encoding technique. The range in the distribution of numerical features varies widely. To normalize the values of features, all numerical feature is scaled between values -1 and 1 using the min-max scaler as shown in Figure 3. After data pre-processing, the imbalanced data is divided as 70% training data and 30% testing data.

5. Class imbalance

Imbalanced classification problems represent a classification model created with data in which the distribution of class values across the classes is not equal . A classification problem gets skewed because of the nature of class imbalance . In this analysis 88.6% of the class values belong to the majority class "No" and

11.4% of the class values belong to the minority class “Yes”. But the prediction model is intended to create a model to predict the CKD patients from the CVD or high-risk CVD patient’s data. In this case, classification accuracy (A) can mislead to select the best performing model. Techniques to select the best model for data with class imbalance are: Choosing the performance metrics those that focus on the minority class, oversampling the minority class using SMOTE to rebalance the class, undersampling the majority class to rebalance the class and selecting classification algorithms such as those that penalize misclassification errors differently. The classification algorithms such as LR, SVM, MLP and K-NN are used for creating classification model.

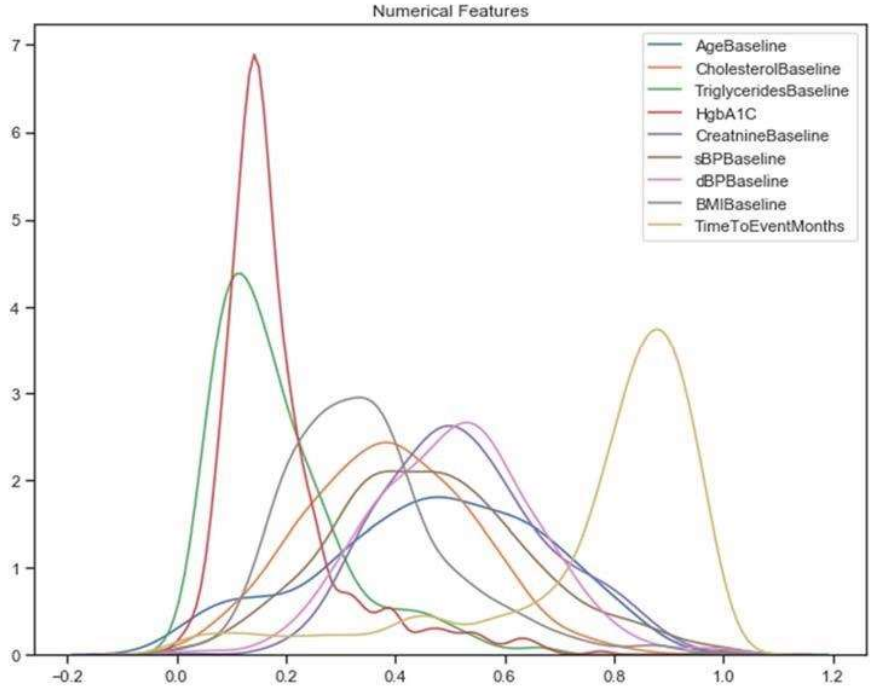


Fig. 3. Numerical features distribution

6. Oversampling using SMOTE

In this oversampling approach, the minority class is over-sampled by creating “synthetic” instances rather than by over-sampling with replacement [Chawla *et a*l., (2001)]. Oversampling encompasses accumulating samples to the minority class in an exertion to reduce the skew in the class distribution [Pan *et al*.,(2019)]. The minority class “Yes” is over-sampled, which means the number of samples is increased. SMOTE iterates through the existing minority instance. At each iteration, one of the ‘X’ closest minority class neighbors are chosen. A new minority instance is synthesized at some point between the minority instance and that neighbor. Synthetic examples are inserted along the line segments joining any of the X minority class nearest neighbor or all of the X minority class nearest neighbors. Depending upon the amount of over-sampling, N neighbors are chosen. Synthetic samples are created by taking the difference between the feature vector under consideration and its nearest neighbor. The difference is then multiplied by a random number between 0 and 1. The obtained result is then

added to the feature vector under consideration. This causes the selection of a random point along the line segment between two specific features .

8. Under sampling using random under sampling

Under-sampling techniques eliminate instances from the training dataset that belong to the majority class in order to reduce the skew in class distribution . The instances can be removed in the ration 1:1, 1:2, 1:100 or any ratio according to requirements. There are different techniques in under-sampling such as random majority under sampling, near miss, cluster centroid and Tomek link. In this work, random under sampling (RUS) is used. The samples of the majority class "No" of the training data are randomly removed such that a balanced 1:1 class distribution is created. Classifiers are trained on this balanced dataset.

9. Stages of Analysis

To build an efficient CKD prediction model, the experiment is done in three stage's as shown in Figure 1.

- Stage I: Building a classification model with the imbalanced data (No Sampling).
- Stage II: Building a classification model by oversampling the minority class using SMOTE to rebalance the class.
- Stage III: Building a classification model by under-sampling the majority class using RUS to rebalance the class.

In stage I the training data is fed to classifiers. The classification algorithms such as LR, K-NN, SVM, MLP were fit on the training data. The training data was validated with 10-fold cross-validation. The models obtained were evaluated using various performance metrics that support the minority class 1 to choose the best model. Stage II aims in building a classification model by oversampling the minority class using SMOTE to rebalance the class. The minority class "1" is oversampled using SMOTE to balance the class distribution. The training data is fed to SMOTE sampler to oversample the minority class 1. Class 1 is oversampled with instances equal to the number of majority class 0. After this pre-processing stage the data becomes balanced. The balanced training data is fed to classifiers and classification model is created. The test data is then fed to the models to predict the risk of CKD.

Stage III intends to build a classification model by under sampling the majority class using RUS to rebalance the class distributions. The majority class "0" in the training data is under-sampled randomly by deleting instances in the class"0". By deleting the instances of majority class, the information that classifiers can learn is lost [Koziarski, (2020)]. But now the class becomes balanced. With equal distributions in both the class, the classifiers build the balanced model. The imbalanced test data is fed to the models to predict CKD.

4. Results and Discussion

Total samples	Predicted No	Predicted Yes
Actual No	TN "Don't have CKD"	FP "Have CKD"
Actual Yes	FN "Don't, have CKD"	TP "Have CKD"

Fig. 4. Confusion matrix

Performance metrics that focus on the minority class are sensitivity or recall (R), precision (P), F-score, balanced accuracy (BA) and geometric mean (G-Mean). The confusion matrix shown in Figure.4 is a valuable tool in analyzing the predicted and actual values and supports to measure the performance of the predictive model [He and Garcia, (2009)]. True positive (TP) gives the number of observations correctly predicted as CKD, false positives (FP) gives the number of observations that are incorrectly predicted as CKD which are not CKD. True negative (TN) tells the number of observations predicted correctly as not having CKD and false negative (FN) gives the number of observations incorrectly predicted as not having CKD.

$$TP\ R1$$
$$TP\ \ FN$$
$$TP\ P2$$
$$TP\ \ FP$$

R is the True Positive Rate (TPR). It gives the information about how well the positive class "Yes" is predicted as shown in Eq. (1). P states the information about the fraction of observations that are really positive out of all the observations that are predicted as positive as in Eq. (2). F-score in Eq. (3) gives the balance between P and R. True negative rate (TNR) in Eq. (4) can be measured to know how well TN is predicted.

$$\mathrm{F-score} = \frac{2\mathrm{PR}}{(\mathrm{P}+\mathrm{R})} \quad 3$$

$$TNR = \frac{TN}{(TN+FP)} \quad 4$$

$$BA = \frac{(TPR+TNR)}{2} \quad 5$$

$$G\ mean\ \sqrt{TPR\ TNR} \quad 6$$

The BA metric in Eq. (5) is a more suitable metric to measure the performance of classifiers on imbalanced data. [Luquea *et al*.,(2019)]. The G-mean in Eq. (6). shows a balance in classification performance in terms of R and TNR [Wang *et al*., (2018)]. The performance of classifiers in phase I analysis is evaluated. Table 2 shows that the classifiers LR, SVM, MLP and K-NN perform with A>=90%. Among the four classification algorithms, MLP has the highest A - 93%. But R and F-score are very less. It is clear that the accuracy is biased by the majority class value 0. This is because the CVD data collected for predicting CKD suffers from severe class imbalance problem. If A alone be used as a metric to select the best model, it can mislead the classification task. So, the focus must be on the other performance metrics that can be used to evaluate the CVD data which has class imbalance issues. The evaluation metrics that can be used on the balanced data treat all classes with equal importance. But in imbalanced classification task classification errors with the minority class are more important than those with the majority class.

Table 2. Performance of classifiers on imbalanced training data

Model	**A**	**P**	**R**	**F-score**	**ROC-AUC**
LR	0.90	0.67	0.25	0.36	0.62
SVM	0.91	1.00	0.25	0.40	0.62
MLP	0.91	0.67	0.50	0.57	0.73
K-NN	0.87	1.00	0.12	0.22	0.56

Table 3. Predictions on imbalanced testing data (No sampling)

Model	**A**	**P**	**R**	**F-score**	**ROC-AUC**
LR	0.90	0.67	0.25	0.36	0.62
SVM	0.91	1.00	0.25	0.40	0.62
MLP	0.91	0.67	0.50	0.57	0.73
K-NN	0.87	1.00	0.12	0.22	0.56

Table 4. Comparison of predictions of imbalanced models

Model	**TN**	**FP**	**FN**	**TP**	**TPR**	**FPR**	**TNR**	**FNR**	**BA**	**G-mean**
LR	123	2	12	4	0.25	0.02	0.98	0.75	0.62	0.50
SVM	125	0	12	4	0.25	0.00	1.00	0.75	0.63	0.50
MLP	121	4	8	8	0.50	0.03	0.97	0.50	0.73	0.70
K-NN	125	0	14	2	0.12	0.00	1.00	0.88	0.56	0.35

Table 5. Performance of classifiers on SMOTE balanced training data

Model	A	P	R	F-score	ROC-AUC
LR-SMOTE	0.92	0.75	0.93	0.51	0.91
SVM-SMOTE	0.93	0.91	0.96	0.94	0.94
MLP - SMOTE	0.91	0.90	0.98	0.94	0.93
K-NN-SMOTE	0.90	0.63	0.97	0.38	0.77

Table 6. Performance of SMOTE models on testing data

Model	A	P	R	F-score	ROC-AUC
LR-SMOTE	0.87	0.43	0.56	0.49	0.73
SVM-SMOTE	0.89	0.53	0.62	0.57	0.78
MLP - SMOTE	0.93	0.64	0.56	0.60	0.76
K-NN-SMOTE	0.81	0.31	0.56	0.40	0.70

Table 7. Comparison of predictions of SMOTE models

Model	TN	FP	FN	TP	TPR	FPR	TNR	FNR	BA	G-mean
LR-SMOTE	113	12	7	9	0.56	0.10	0.90	0.44	0.73	0.71
SVM-SMOTE	116	9	6	10	0.62	0.07	0.93	0.38	0.76	0.76
MLP - SMOTE	120	5	7	9	0.56	0.04	0.96	0.44	0.76	0.73
K-NN-SMOTE	105	20	7	9	0.56	0.16	0.84	0.44	0.70	0.69

Table 8. Performance of classifiers on RUS balanced training data

Model	A	P	R	F-score	ROC-AUC
LR-RUS	0.75	0.78	0.77	0.74	0.88
SVM-RUS	0.77	0.76	0.82	0.78	0.83
MLP-RUS	0.79	0.82	0.81	0.78	0.90
K-NN-RUS	0.73	0.78	0.66	0.71	0.77

Table 9. Performance of RUS model on testing data

Model	A	P	R	F-score	ROC-AUC
LR-RUS	0.84	0.38	0.62	0.48	0.75
SVM-RUS	0.83	0.36	0.75	0.49	0.79
MLP-RUS	0.85	0.42	0.81	0.55	0.83
K-NN-RUS	0.83	0.34	0.75	0.47	0.78

Table 10. Comparison of predictions of RUS models

Model	TN	FP	FN	TP	TPR	FPR	TNR	FNR	BA	G-mean
LR-RUS	109	16	6	10	0.62	0.13	0.87	0.38	0.75	0.87
SVM-RUS	106	19	5	11	0.69	0.15	0.85	0.31	0.77	0.88
MLP-RUS	107	18	3	13	0.81	0.14	0.86	0.19	0.84	0.92
K-NN-RUS	107	18	6	10	0.62	0.14	0.86	0.38	0.74	0.86

Table 11. Comparison of imbalanced and balanced models

Classifier	Imbalanced model			SMOTE model			RUS model		
	F-score	G-mean	BA	F-score	G-mean	BA	F-score	G-mean	BA

LR	0.36	0.50	0.62	0.49	0.71	0.73	0.48	0.87	0.75
SVM	0.40	0.50	0.63	0.57	**0.76**	**0.76**	0.49	0.88	0.77
MLP	**0.57**	**0.70**	**0.73**	**0.60**	0.73	**0.76**	**0.55**	**0.92**	**0.84**
K-NN	0.22	0.35	0.56	0.40	0.69	0.70	0.47	0.86	0.74

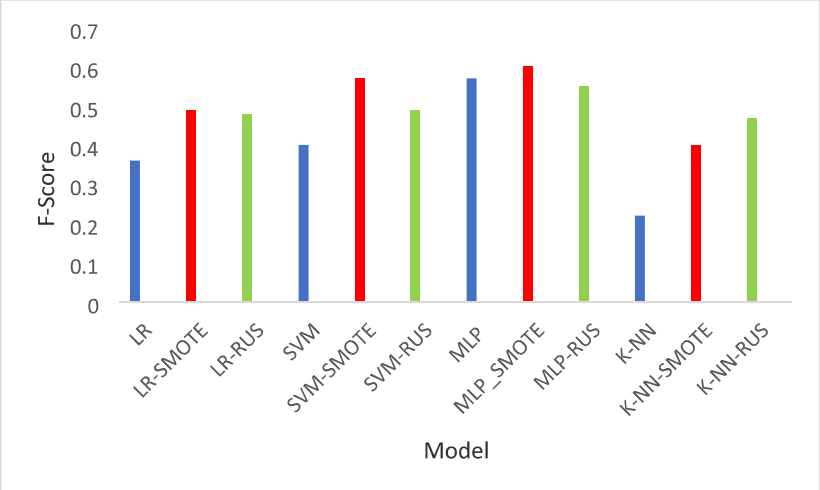


Fig.5. F-score of imbalanced and balanced models

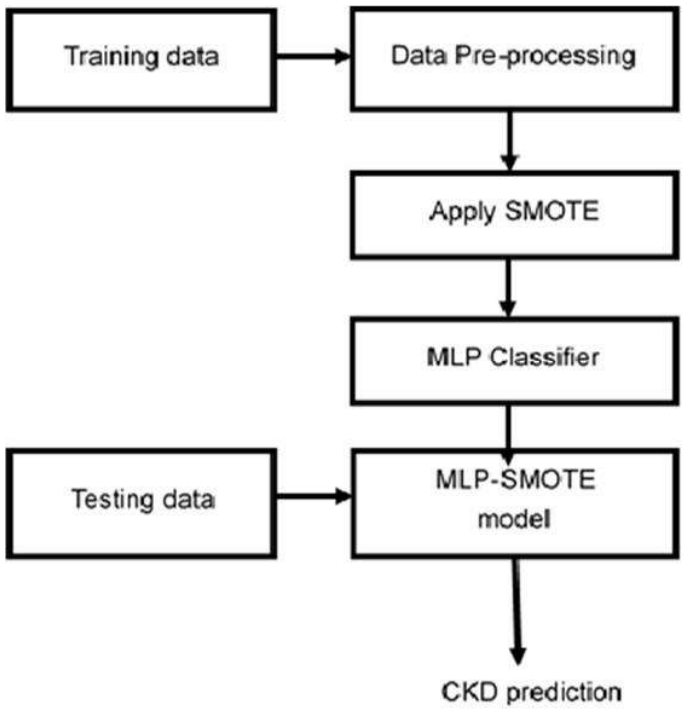


Fig. 6. Proposed CKD prediction model

The challenge is to choose the metric that focuses on the minority class where we dearth observations required to train an effectual classification model. P, R, F-score are the metrics that can measure the effectiveness of the model created when there is a skew in the class distribution. Based on this, MLP produces the highest R of 79%, P of 69% and F-score of 70%. The classification model build

is fed with the testing data to observe how it performs on the test data. Table 3 shows that the MLP classifier has a better R and F-score when compared with the other models. MLP and SVM performed better than LR and KNN classifiers by achieving 69% P, 79% R and 70% F-score. SVM and LR attained 25% R. MLP and LR reached 67% P. But SVM has the highest P of 100% with a low R of 25%. The reason for the very low R rate in all classifiers is due to the skew distribution of the class. MLP performed better than other classifiers in terms of area under the receiver operating characteristic curve (ROC-AUC) in phase I. Table 4 shows that the MLP also achieved the highest BA -73% and G-mean – 70 % when compared with the other three models.

In phase II, classifiers are applied to the SMOTE oversampled CVD data. The balanced data is 10-fold crossvalidated. In the training phase SVM -SMOTE model produced the highest A- 93%, P-91%, F score - 94% and ROC-AUC 94%. Table 5 shows the performance of the classifiers on the oversampled training data. In all the four classifiers R rate has increased when compared with model performance on imbalanced data in Table 2. This is because the data is now balanced. MLP-SMOTE achieved the highest R rate. These models were applied to the test data and evaluated as shown in Table 6. The test data is not balanced. On the test data, MLP attained better performance than the other classifiers in terms of P and F-score. Table 7 shows that SVM -SMOTE produced the highest G-mean of 76%. But MLP-SMOTE has better TNR and BA. In phase III, classifiers were trained with under- sampled data. Table 8 shows that the accuracy of all the RUS models on training data has decreased when compared with no-sampling and SMOTE models. But the MLP-RUS performs better than other RUS models in terms of A, P, R, F-score and ROC- AUC as shown in Table 9.

Table 10 shows that the G-mean of MLP-RUS has increased which shows that the performance is validated with equal importance to both TPR and TNR. The BA of MLP-RUS is also higher when compared with all other models. MLP classifier performs better when oversampled or under-sampled when compared with other classifiers as shown in Table 11. But when the class 0 data is under-sampled for sake of balancing with class 1, the classifiers miss the opportunity to learn from more data. The F-score of MLP- RUS model has decreased when compared with imbalanced models as shown in Figure. 5. But in MLP-SMOTE model the F-score has increased when compared with imbalanced models.

The proposed methodology offers MLP - SMOTE as the better classifier for predicting CKD from imbalanced CVD as illustrated in Figure. 6. The MLP model is trained with SMOTE balanced data to create an MLP-SMOTE model. The created model is fed with the testing data which is not balanced. By oversampling the imbalanced data, the MLP classifier performance increases. The

MLP-SMOTE predicts whether the patient has CKD or not. This prediction result may help the health care practitioners and the patients for the earlier identification of CKD from CVD data.

5. Conclusion

In this work, the CKD prediction model is built using imbalanced CVD data. Initially, the models LR, SVM, MLP and KNN build on imbalanced data, performed with good accuracy in predicting the CVD test data. But R and F-score were low. The low values are due to the fact that the number of CKD positive cases is too low. On the imbalanced data, the MLP model performed better than other models. The CVD training data is then balanced by applying resampling techniques like SMOTE and RUS. On the test data, MLP-SMOTE performed better with the highest and increased F-score when compared with other models. The proposed MLP-SMOTE model can predict CKD better by solving the imbalanced distribution of CVD data. This can help the medical practitioners and patients for the early prediction of CKD and save a life. In the future, the model can be further tuned by applying feature selection methods to increase the performance in prediction

SOURCE CODE:

```
#Import the libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.matrics import roc_curve,aue,confusion_matrix, classification_report,accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from Sklearn.neighbors import kNeighborsClassifier
# Reading the dataset
kidney = pd.read_csv(" ")
kidney.head()
#information about the dataset
kideny.info()
#Description of the dataset
kideny.describe()
# To see what are the columan name in our dataset
print(kideny.columan)
# Mapping the text to 1/0 and cleaning the dataset
```

```
    kidney[['htn','dm','cad','pe','ane']] = kidney[['htn','dm','cad','pe','ane']].
    replace(to_replace={'yes':1,'no':0})
    kidney[['rbc','pc']] = kidney[['rbc','pc']].replace(to_replace={'abnormal':1,'
    normal':0})
    kidney[['pcc','ba']] = kidney[['pcc','ba']].replace(to_replace={'present':1,'n
    otpresent':0})
    kidney[['appet']] = kidney[['appet']].replace(to_replace={'good':1,'poor':0,'n
    o':np.nan})
    kidney['classification'] = kidney['classification'].replace(to_replace={'ckd':
    1.0,'ckd\t':1.0,'notckd':0.0,'no':0.0})
    kidney.rename(columns={'classification':'class'},inplace=True)

    kidney['pe'] = kidney['pe'].replace(to_replace='good',value=0) # Not having pe
    dal edema is good
    kidney['appet'] = kidney['appet'].replace(to_replace='no',value=0)
    kidney['cad'] = kidney['cad'].replace(to_replace='\tno',value=0)
    kidney['dm'] = kidney['dm'].replace(to_replace={'\tno':0,'\tyes':1,' yes':1, '
    ':np.nan})
    kidney.drop('id',axis=1,inplace=True)
kidney.head()
# This helps us to count how many NaN are there in each column
len(kidney)-kidney.count()
# This shows number of rows with missing data
kidney.isnull().sum(axis = 1)
#This is a visualization of missing data in the dataset
sns.heatmap(kidney.isnull(),yticklabels=False,cbar=False,cmap='viridis')
# This shows number of complete cases and also removes all the rows with NaN
kidney2 = kidney.dropna()
print(kidney2.shape)
# Now our dataset is clean
sns.heatmap(kidney2.isnull(),yticklabels=False,cbar=False,cmap='viridis')
sns.heatmap(kidney2.corr())
# Counting number of normal vs. abnormal red blood cells of people having chronic
kidney disease
print(kidney2.groupby('rbc').rbc.count().plot(kind="bar"))
#This plot shows the patient's sugar level compared to their ages
kidney2.plot(kind='scatter', x='age',y='su');
plt.show()
# Shows the maximum blood pressure having chronic kidney disease
print(kidney2.groupby('class').bp.max())
print(kidney2['dm'].value_counts(dropna=False))
    X_train, X_test, y_train, y_test = train_test_split(kidney2.iloc[:,:-1], kidney2['
    class'], test_size=0.33, random_state=44, stratify= kidney2['class'])
```

In [19]:

```
    linkcode
    print(X_train.shape)
```

```
    X_train, X_test, y_train, y_test = train_test_split(kidney2.iloc[:,:-1], kidney2['
    class'], test_size=0.33, random_state=44, stratify= kidney2['class'])
```

In [19]:

```
    linkcode
    print(X_train.shape)
y_train.value_counts()
    rfc = RandomForestClassifier(random_state = 22)
    rfc_fit = rfc.fit(X_train,y_train)
    rfc_pred = rfc_fit.predict(X_test)
```

In [23]:

linkcode
print(confusion_matrix(y_test,rfc_pred))

```
print(classification_report(y_test,rfc_pred))
accuracy_score( y_test, rfc_pred)
knn = KNeighborsClassifier(n_neighbors=1)
```

In [27]:

linkcode

```
knn.fit(X_train,y_train)
pred = knn.predict(X_test)
```

In [29]:

linkcode
print(confusion_matrix(y_test,pred))

```
print(classification_report(y_test,pred))
accuracy_score( y_test,pred)
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
```

In [34]:

linkcode
print(classification_report(y_test,predictions))

```
print(confusion_matrix(y_test,predictions))
accuracy_score( y_test, predictions)
feature_importances = pd.DataFrame(rfc.fit(X_train,y_train).feature_importances_,
index = X_train.columns,
                                   columns=['importance']).sort_values('importance
', ascending=False)
print(feature_importances)
kidney3 = kidney.drop(columns=['rbc', 'pc', 'sod', 'pot', 'pcv', 'wc', 'rc'])
kidney3. shape
kidney3.head()
kidney3.isnull().sum()
kidney3.mode()
# Fill in the NaNs with the mode for each column.
kidney3_imp = kidney3.apply(lambda x:x.fillna(x.value_counts().index[0]))
kidney3_imp.isnull().sum()
X_train, X_test, y_train, y_test = train_test_split(kidney3_imp.iloc[:,:-1], kidne
y3_imp['class'],
                                                    test_size = 0.33, random_state
=44,
                                                   stratify = kidney3_imp['class']
)
```

In [45]:

linkcode
y_train.value_counts()

```
rfc = RandomForestClassifier(random_state = 22)
rfc_fit = rfc.fit(X_train,y_train)
rfc_pred = rfc_fit.predict(X_test)
print(confusion_matrix(y_test,rfc_pred))
print(classification_report(y_test,rfc_pred))
accuracy_score( y_test, rfc_pred)
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
```

In [53]:

linkcode

```
    print(classification_report(y_test,predictions))
```

```
accuracy_score( y_test, rfc_pred)
```

```
    dtree=DecisionTreeClassifier()
```

In [57]:

```
    dtree.fit(X_train,y_train)
```

```
    predictions=dtree.predict(X_test)
```

In [59]:

```
    linkcode
    print(classification_report(y_test,predictions))
```

```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydot
import os
os.environ["PATH"] += os.pathsep + 'C:\Program Files (x86)/Graphviz2.38/bin/'

features = list(kidney3.columns[1:])
features
dot_data = StringIO()
export_graphviz(dtree, out_file = dot_data,feature_names = features,filled = True,
rounded=True)

graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train,y_train)
```

```
    rfc_pred = rfc.predict(X_test)
```

In [64]:

```
    linkcode
    print(confusion_matrix(y_test,rfc_pred))
```

```
print(classification_report(y_test,rfc_pred))
accuracy_score( y_test, rfc_pred)
```

```
    # Choosing a K Value.
    # Let's go ahead and use the elbow method to pick a good k value.
    error_rate = []

    # Will take some time
    for i in range(1,40):

        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train,y_train)
        pred_i = knn.predict(X_test)
        error_rate.append(np.mean(pred_i != y_test))
```

In [68]:

```
    linkcode
    plt.figure(figsize=(10,6))
    plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o', mark
    erfacecolor='red',markersize=10)
    plt.title('Error Rate vs. K Value')
    plt.xlabel('K')
    plt.ylabel('Error Rate')
```

```
    knn = KNeighborsClassifier(n_neighbors=1)
```

In [70]:

```
    linkcode
    knn.fit(X_train,y_train)
```

```
    knn = KNeighborsClassifier(n_neighbors=1)
```

In [70]:

```
    linkcode
    knn.fit(X_train,y_train)
```

```
print(classification_report(y_test,pred))
accuracy_score( y_test,pred)
```

Out[3]:

```
(400, 26)
```

In [4]:

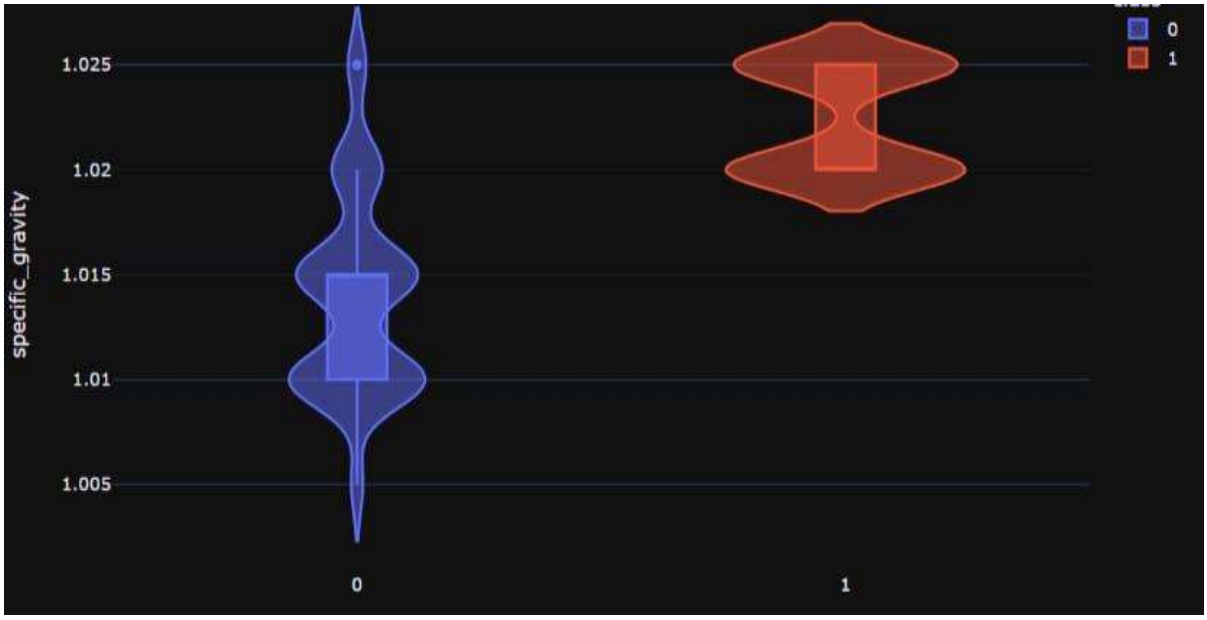
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   age                     391 non-null    float64
 1   blood_pressure          388 non-null    float64
 2   specific_gravity        353 non-null    float64
 3   albumin                 354 non-null    float64
 4   sugar                   351 non-null    float64
 5   red_blood_cells         248 non-null    object
 6   pus_cell                335 non-null    object
 7   pus_cell_clumps         396 non-null    object
 8   bacteria                396 non-null    object
 9   blood_glucose_random    356 non-null    float64
 10  blood_urea              381 non-null    float64
 11  serum_creatinine        383 non-null    float64
 12  sodium                  313 non-null    float64
 13  potassium               312 non-null    float64
 14  haemoglobin             348 non-null    float64
 15  packed_cell_volume      330 non-null    object
 16  white_blood_cell_count  295 non-null    object
 17  red_blood_cell_count    270 non-null    object
 18  hypertension            398 non-null    object
 19  diabetes_mellitus       398 non-null    object
 20  coronary_artery_disease 398 non-null    object
 21  appetite                399 non-null    object
 22  peda_edema              399 non-null    object
 23  aanemia                 399 non-null    object
 24  class                   400 non-null    object
dtypes: float64(11), object(14)
memory usage: 78.2+ KB
```

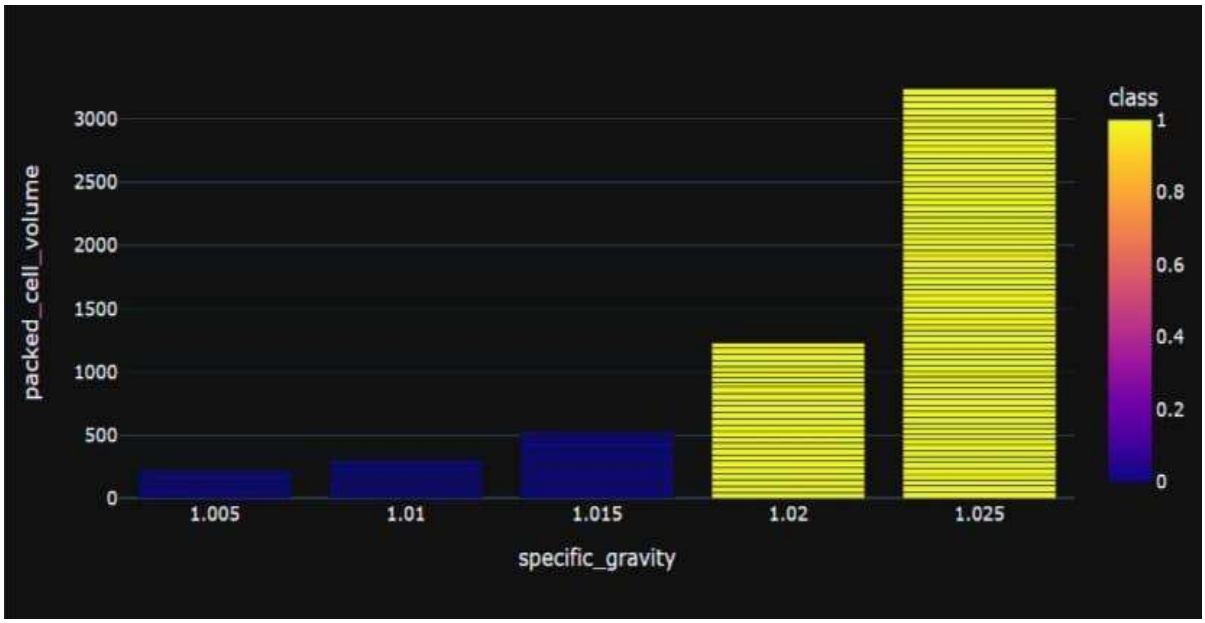
```
red_blood_cells has [nan 'normal' 'abnormal'] values

pus_cell has ['normal' 'abnormal' nan] values

pus_cell_clumps has ['notpresent' 'present' nan] values

bacteria has ['notpresent' 'present' nan] values

hypertension has ['yes' 'no' nan] values

diabetes_mellitus has ['yes' 'no' ' yes' '\tno' '\tyes' nan] values

coronary_artery_disease has ['no' 'yes' '\tno' nan] values

appetite has ['good' 'poor' nan] values

peda_edema has ['no' 'yes' nan] values

aanemia has ['no' 'yes' nan] values

class has ['ckd' 'ckd\t' 'notckd'] values
```

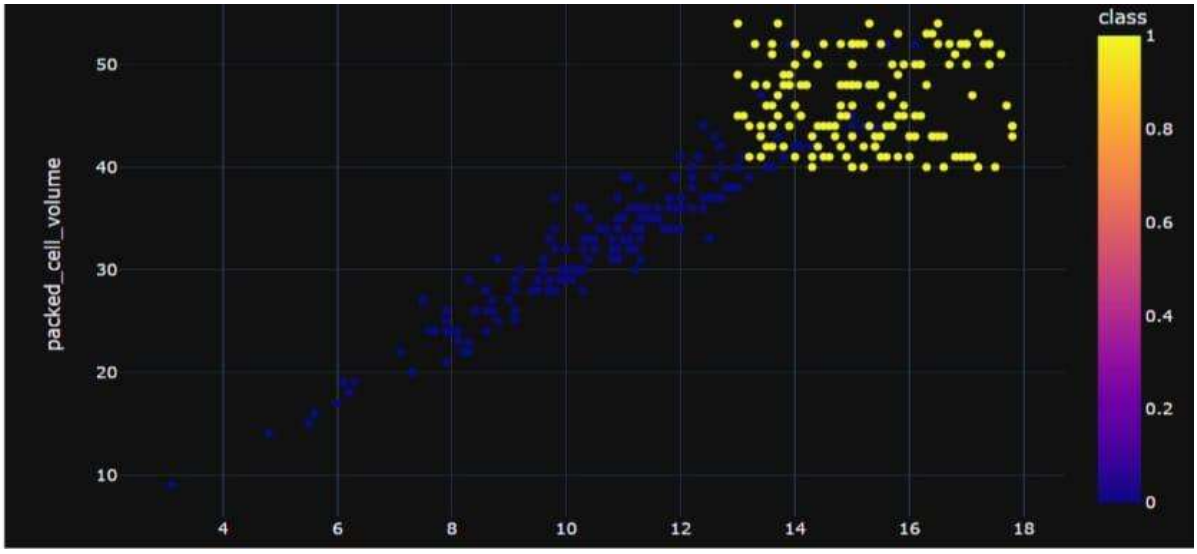
```
diabetes_mellitus has ['yes' 'no' nan] values

coronary_artery_disease has ['no' 'yes' nan] values

class has [0 1] values
```

```
Index(['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar',
       'red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria',
       'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium',
       'potassium', 'haemoglobin', 'packed_cell_volume',
       'white_blood_cell_count', 'red_blood_cell_count', 'hypertension',
       'diabetes_mellitus', 'coronary_artery_disease', 'appetite',
       'peda_edema', 'aanemia', 'class'],
      dtype='object')
```

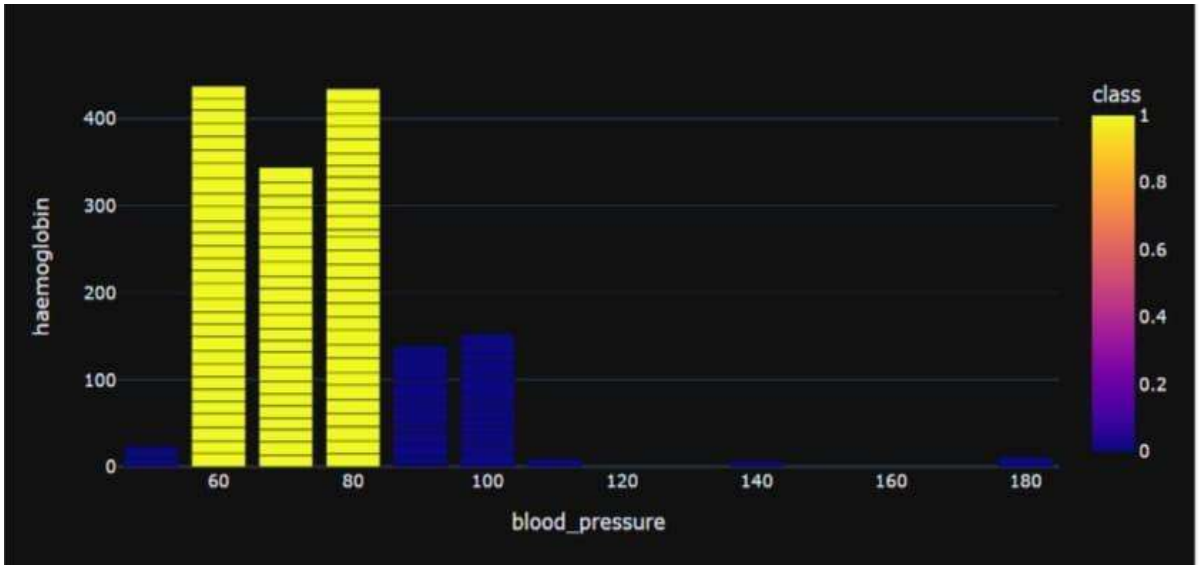
```
Index(['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar',
       'red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria',
       'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium',
       'potassium', 'haemoglobin', 'packed_cell_volume',
       'white_blood_cell_count', 'red_blood_cell_count', 'hypertension',
       'diabetes_mellitus', 'coronary_artery_disease', 'appetite',
       'peda_edema', 'aanemia', 'class'],
      dtype='object')
```

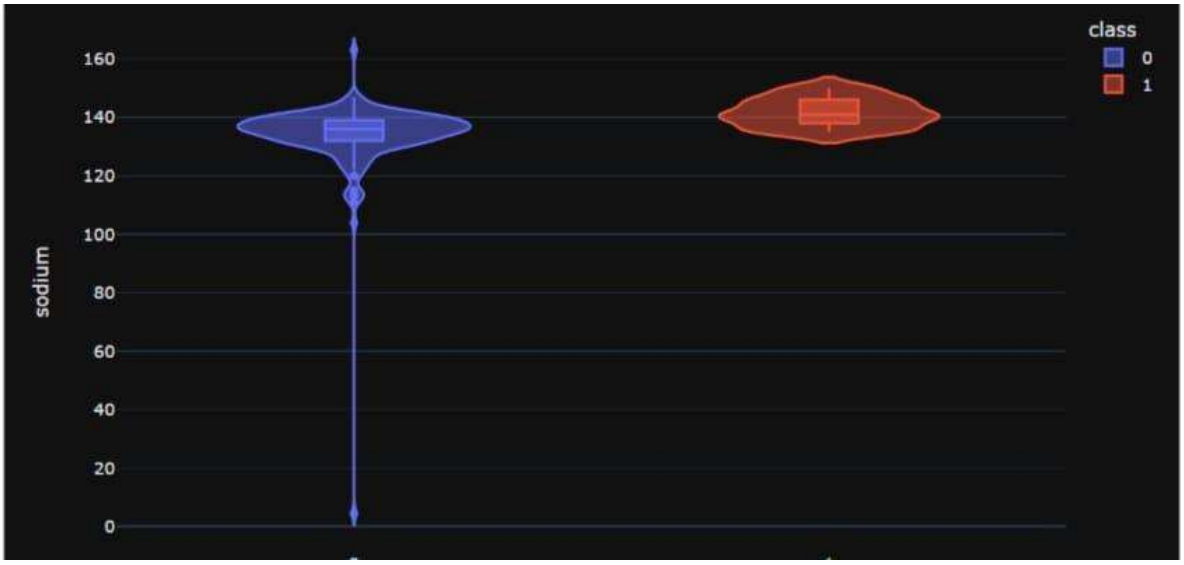


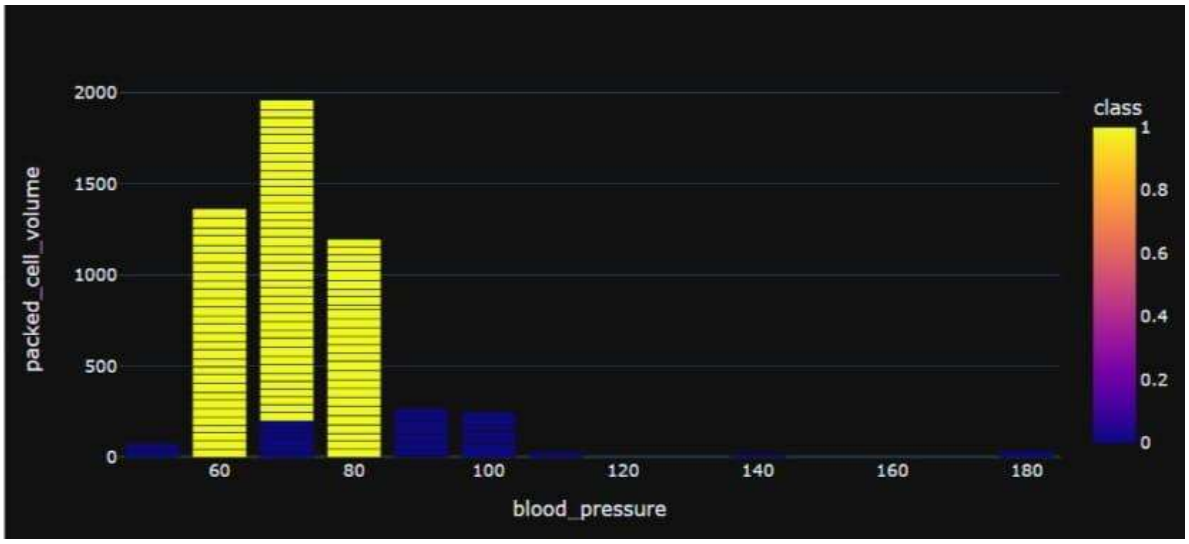



class
1
0.8
0.6
0.4
0.2
0
packed_cell_volume
50
40
30
20
10
4
6
8
10
12
14
16
18


class
0
1
albumin
6
5
4
3
2
1
0


haemoglobin
400
300
200
100
0
60
80
100
120
140
160
180
blood_pressure
class
1
0.8
0.6
0.4
0.2
0


sodium
160
140
120
100
80
60
40
20
0
class
0
1


packed_cell_volume
2000
1500
1000
500
0
60
80
100
120
140
160
180
blood_pressure
class
1
0.8
0.6
0.4
0.2
0

```
red_blood_cells           152
red_blood_cell_count      131
white_blood_cell_count    106
potassium                  88
sodium                     87
packed_cell_volume         71
pus_cell                   65
haemoglobin                52
sugar                      49
specific_gravity           47
albumin                    46
blood_glucose_random       44
blood_urea                 19
serum_creatinine           17
blood_pressure             12
age                         9
```

```
blood_pressure            12
age                        9
bacteria                   4
pus_cell_clumps            4
hypertension               2
diabetes_mellitus          2
coronary_artery_disease    2
appetite                   1
peda_edema                 1
aanemia                    1
class                      0
dtype: int64
```