Comparison of Machine Learning Algorithms for Predicting Chronic Kidney Disease

Aloori Nagaraju
Data science and AI

Abstract

Motivation: Chronic kidney disease (CKD) demands accurate early prognosis for effective treatment planning. However, conventional diagnostics often fall short in comprehending disease severity. Recent machine learning advances present an opportunity to develop enhanced CKD prediction models by uncovering patterns in complex clinical data. This research aims to thoroughly evaluate different machine learning techniques for CKD forecasting, compare their predictive performance, and analyze the potential of sophisticated artificial neural network approaches to boost accuracy. Enhancing prognosis can enable personalized medicine and improved outcomes.

Results: The study reveals artificial neural networks, when incorporated into sophisticated machine learning ensembles, can boost diagnostic accuracy and model performance for chronic kidney disease prognosis.

Availability: https://github.com/aloori080898/CKD_machine_learning_algorithm_compardion.

Contact:- 8179504206

1 Introduction

Globally, chronic kidney disease (CKD) is a common and dangerous medical illness that requires early identification and precise prognosis for efficient management and treatment. Conventional diagnostic techniques often fall short of providing a thorough understanding of the course and severity of CKD. However, the development of machine learning algorithms and data mining methods provide a chance to take use of big information to uncover hidden patterns and CKD prediction models.

The field of healthcare has shown great promise for machine learning algorithms, particularly in the areas of illness detection and prediction. In order to find trends and provide precise forecasts, these algorithms may examine complicated medical data, including patient demographics, clinical assessments, and test results. Machine learning algorithms may help with CKD patients' classification according to the severity of their condition and illness prediction.The purpose of this research is to examine and assess the effectiveness of several machine learning techniques for CKD prediction, such as GaussianNB, Random Forest, XGBoost, Multilayer Perceptron (MLP), Support Vector Machine (SVM), Decision Tree, LGBM Classifier, and KNeighbors Classifier. We may evaluate these algorithms' performance in precisely categorizing instances of chronic kidney disease (CKD) and forecasting the illness's severity by comparing them.

This research aims to improve prediction accuracy by using an Artificial Neural Network (ANN) method, while also analyzing current techniques. Neural networks found in the human brain serve as the model's inspiration for artificial neural networks, or ANNs. ANNs have shown potential in a number of healthcare applications, such as the prediction of illness. We want to investigate an ANN's potential to improve CKD prediction by integrating it into the assessment.A benchmark dataset that was taken from the UCI repository is used to do the assessment. Since incomplete data might affect prediction accuracy, it is imperative to address the problem of missing characteristics in clinical data linked to renal disease. As a result, this research also attempts to pinpoint the shortcomings of the strategies now in use for dealing with missing values in the data and suggests a fresh strategy to enhance data analysis and quality.Promising outcomes have been shown by earlier studies in the area of CKD prediction utilizing machine learning algorithms. In identifying CKD patients, decision trees, K-nearest neighbor, support vector machines, and artificial neural networks have attained great accuracy. Regression models have also been used to estimate the risk of acquiring renal disease and forecast creatinine levels. These studies provide a foundation for our research and highlight the potential of machine learning, including ANNs, in CKD prediction.

2 Background

In the study described in reference ,titled "Development of a Predictive Analytic System for Chronic Kidney Disease using Ensemble-based Machine Learning," the primary objectives were twofold: firstly, to create an effective prediction system for Chronic Kidney Disease (CKD) using a machine learning ensemble model, and secondly, to establish a predictive analytic system based on this suggested ensemble model. To achieve these aims, the study investigated twelve machine learning techniques on publicly available datasets. Subsequently, the study evaluated the effectiveness of these algorithms and incorporated them into an ensemble voting classifier to determine the best combination of algorithms. In order to predict CKD effectively, seven machine learning algorithms were chosen based on their accuracy and minimization of false negative errors. The experimental results demonstrated that the suggested ensemble machine learning model achieved a high accuracy rate and zero false negative errors in identifying CKD. Ultimately, the study successfully developed an efficient predictive analytics system that incorporated the recommended ensemble-based machine learning models.

Reference describes a system that identifies factors contributing to CKD and employs machine learning to classify its severity into distinct stages. This system utilizes medical test records as input for a classification algorithm and an algorithm for recommending an optimal diet for CKD patients. The dietary recommendations are tailored to the patient's potassium zone, which is determined by their blood potassium levels, with the goal of halting the progression of CKD.

In an effort to enhance healthcare services, a study has leveraged the synergy between big data platforms, such as Apache Spark, and machine learning algorithms to identify chronic kidney disease. This research employs hybrid machine learning strategies that combine feature selection methods, specifically Relief-F and the chi-squared feature selection approach, with various machine learning classification algorithms. Six ensemble learning techniques, including Logistic Regression (LR), gradient-boosted trees (GBT Classifier), Naive Bayes (NB), Random Forest (RF), Support Vector Machine (SVM), and Decision Trees (DT), were utilized in this study. The evaluation of results was carried out using four key metrics: accuracy, precision, recall, and F1-measure. Notably, the features selected by Relief-F outperformed those chosen by the chi-squared approach and provided a more comprehensive set of features.

3. Methodolgy

3.1 Importing Libraries: Data processing, modeling, and visualization libraries for importing. The imported libraries are:
numpy (imported as np): The world numerical operations (ANO) library in python.
pandas (imported as pd): Data Manipulations and Analysis Library (DAAL).
MinMaxScaler from sklearn.preprocessing: Class for normalizing numerical features into a specified range.
train_test_split from sklearn.model_selection: Provide a function that splits the dataset into the training and test data set.
accuracy_score from sklearn.metrics: An accuracy validation function of the model.
SimpleImputer from sklearn.impute: A class to help with missing values in the dataset.
seaborn (imported as sns): A library for data visualization.

3.2 Importing Dataset: The code uses pandas' pd.read_csv() to read a CSV file titled kidney_disease.csv. This is a DataFrame which we have called df. Subsequently, we utilize the first few rows of the DataFrame by way of the head() method.

The dataset includes data on individuals diagnosed with chronic kidney disease. In this data table, each row represents a patient while the columns describe their respective age, blood pressure, specific gravity, albumin, sugar, red blood cell count, pus cell count, and other medical attributes. The final column "classification" specifies if the given patient is suffering from CKD or not.

3.3 Data Processing

3.3.1 Imputing Null Values:

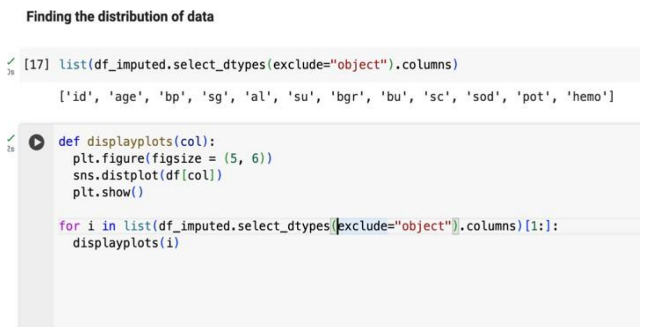
We considered the missing data during the first phase (of data preprocessing) with python's scikit-learn library for 'df', being a medical DataFrame. The missing values indicated as NaN were replaced by the most common value found in each corresponding column using the SimpleImputer class. The transformed imputed data was finally converted to DataFrame ('df_imputed') for further evaluation.

Moreover, a number of lambda functions were introduced to guarantee that data conformed and the structure of particular columns was cohesive. The functions were able to standardise some of the values within columns like "classification," "cad", "dm", "rc", "wc" and "pcv". For example it converted "\tckd" to "ckd" in "classification" Column and "\ We made some changes before verifying that the columns had contained the unique values using printout process.

3.3.2 Checking Label Imbalance

We then carried out a EDA and plotted the prepossessed 'df_imputed' data. To begin with, we produced the frequency count of distinct values under the 'classification' column showing the classification of medical cases. We presented this count by using a bar plot and to this end we had one of the most useful glimpses on the medical classification distributions observed into our set..

3.3.3 Finding the distribution of data



Finding the distribution of data

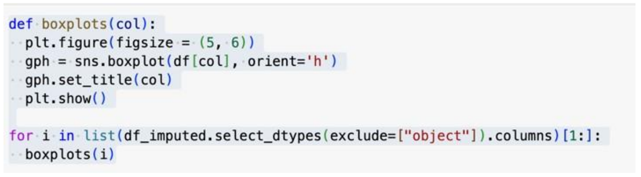
```
[17] list(df_imputed.select_dtypes(exclude="object").columns)

     ['id', 'age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo']

     def displayplots(col):
       plt.figure(figsize = (5, 6))
       sns.distplot(df[col])
       plt.show()

     for i in list(df_imputed.select_dtypes(exclude="object").columns)[1:]:
       displayplots(i)
```

Fig 1.Finding the distribution of data

This code snippet is utilized to extract and visually examine the distribution of numerical features in the 'df_imputed' DataFrame, which comprises data post-imputation. The process involves identifying columns with numerical data, excluding those categorized as 'object' type. Subsequently, a function named 'displayplots' is defined to generate and showcase distribution plots for each numerical column using seaborn, a Python visualization library.

3.3.3 Finding Outliners

These include the use of box plots used to visualize distributional properties, median, and potential outliers amongst the numerical data. The iterations on the second and subsequent numerical columns provide an overall assessment of the data’s dispersion and scatter between other characteristics in the data set.

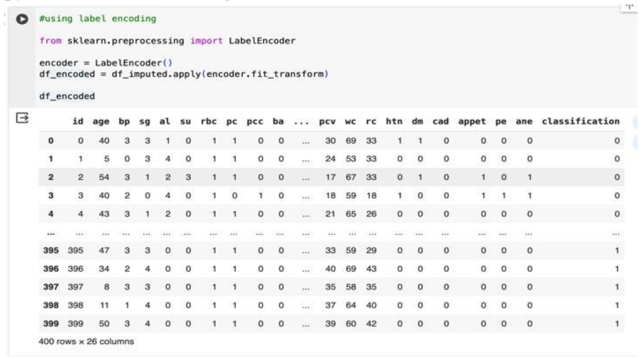


```
def boxplots(col):
  plt.figure(figsize = (5, 6))
  gph = sns.boxplot(df[col], orient='h')
  gph.set_title(col)
  plt.show()

for i in list(df_imputed.select_dtypes(exclude=["object"]).columns)[1:]:
  boxplots(i)
```

Fig 2. Code snippet for finding ouliers

3.3.4 Encoding the data

This code is used to explain how the categorical data is converted into numeric format through label encoding so as to prepare the dataset for use with various machine-learning algorithms. This entails importing LabelEncoder from sklearn.preprocessing which is a widely recognized python’s machine learning tool.



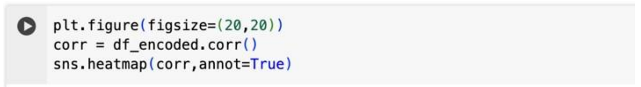
```
#using label encoding

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
df_encoded = df_imputed.apply(encoder.fit_transform)

df_encoded
```

```
      id  age  bp  sg  al  su  rbc  pc  pcc  ba  ...  pcv  wc  rc  htn  dm  cad  appet  pe  ane  classification
0      0   40   3   3   1   0    1   1    0   0  ...   30  69  33    1   1    0      0   0    0               0
1      1    5   0   3   4   0    1   1    0   0  ...   24  53  33    0   0    0      0   0    0               0
2      2   54   3   1   2   3    1   1    0   0  ...   17  67  33    0   1    0      1   0    1               0
3      3   40   2   0   4   0    1   0    1   0  ...   18  59  18    1   0    0      1   1    1               0
4      4   43   3   1   2   0    1   1    0   0  ...   21  65  26    0   0    0      0   0    0               0
...  ...  ...  ..  ..  ..  ..  ...  ..  ...  ..  ...  ...  ..  ..  ...  ..  ...    ...  ..  ...             ...
395  395   47   3   3   0   0    1   1    0   0  ...   33  59  29    0   0    0      0   0    0               1
396  396   34   2   4   0   0    1   1    0   0  ...   40  69  43    0   0    0      0   0    0               1
397  397    8   3   3   0   0    1   1    0   0  ...   35  58  35    0   0    0      0   0    0               1
398  398   11   1   4   0   0    1   1    0   0  ...   37  64  40    0   0    0      0   0    0               1
399  399   50   3   4   0   0    1   1    0   0  ...   39  60  42    0   0    0      0   0    0               1

400 rows × 26 columns
```

Fig 3. Encoding the data

A labelencoder object is instantiated, and it is applied on the entire column of the ‘df_imputed’ DataFrame using the apply function. The process assigns numerical values to every unique category of a column so as to differentiate them. After performing this transformation, a new DataFrame called df_encoded consisting only of numbers is created and can be used in algorithms which require numerical inputs. Such transformation has significant importance because of the fact that machine learning applications deal with categorical data derived from textual data priorly.

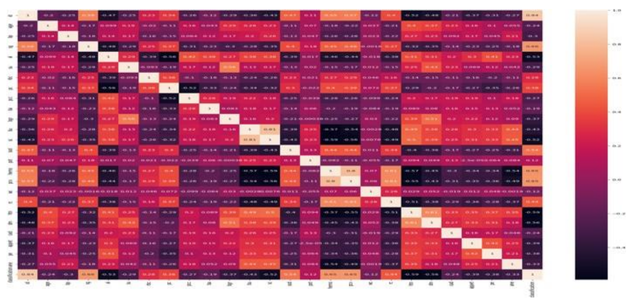
3.3.5 Finding Correlations



```
plt.figure(figsize=(20,20))
corr = df_encoded.corr()
sns.heatmap(corr,annot=True)
```

Fig 4 code snippet for Finding Correlations

This code creates a correlation heatmap, an image showing how each variable is correlated with others in the ‘df_encoded’ DataFrame. The heatmap is constructed with respect to the figure size of 20″ x 20″ that is set in order to make any parts of the map simple to see. Subsequently, a correlation matrix is constructed comprising Pearson correlation coefficients for two variables in the dataset.

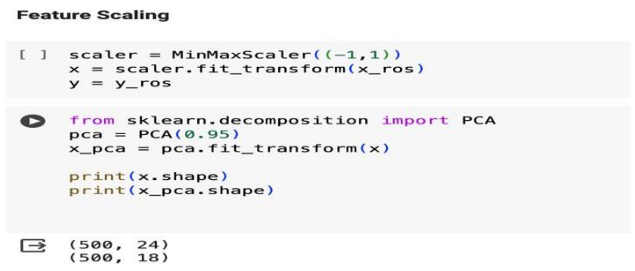


picture of the nature of data

3.3.6 Label Balancing

Considering the prevailing class imbalance, we used the resampling method through the imblearn library’s RandomOverSampler. It was identified that Counter of collections module failed in the initial diagnostic revealing imbalance, where one class had 250 instances and another one had 150. To correct these inequalities and make the model more robust in predicting, a random over sampling procedure was used to generate additional instances, particularly for the minority category.

3.3.7 Feature Scaling



Feature Scaling

```
[ ]  scaler = MinMaxScaler((-1,1))
     x = scaler.fit_transform(x_ros)
     y = y_ros
```

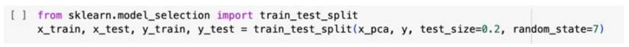
```
from sklearn.decomposition import PCA
pca = PCA(0.95)
x_pca = pca.fit_transform(x)

print(x.shape)
print(x_pca.shape)
```

```
(500, 24)
(500, 18)
```

Fig 5. Code snippet for Feature Scaling

The study uses a dual stage of data cleaning in the first phase in optimization of the dataset for predictive modelling.. PCA is employed for the dimension reduction in data after scaling. The features are normalized and fed into PCA, which reduces the feature space but preserves 95% of the variance in the data set. Transforming the original features in to a set of linearly uncorrelated components and reducing them from 24 to 18.

3.4 Spliting train and test



```
[ ]  from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=7)
```

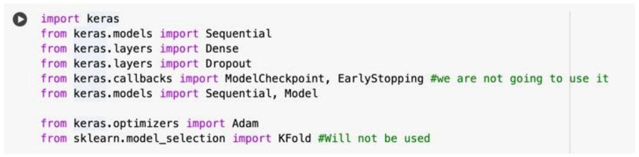
Fig 6. Splitting train and test

Model selection’s train_test_split function was crucial in splitting the dataset into different sets of training and test data necessary to validate ML model’s performance. The dataset is composed of x_pca that has been reported to have gone through PCA for reduction in dimensions, and y, which were partitioned into test (test_size=0.2) and training sets. A predefined seed(random_state=7) was used for controlling this division to avoid inconsistency during various experiments. Subsequently, the resultant subsets, x_train and y_train, are used for model training whereas x_test and y_test help determine how well the model will perform in actual settings.

3.5 MODELS

In our quest to tackle the complexities of predicting chronic kidney disease, we harnessed a spectrum of sophisticated algorithms, each with its unique strengths. The ensemble might of Random Forest and XGBoost stood alongside the precision of XGBRF, offering a nuanced blend of random forest principles with gradient boosting finesse. Decision Trees brought simplicity and clarity to the table, while LGBM Classifiers handled the data's vastness with ease. We also explored the calculated predictions of Naïve Bayes, the meticulous margin-focused approach of SVMs, and delved into the layered depths of Neural Networks. Completing our toolkit was the KNN Classifier, which gauged similarities in patient data with a neighborly touch. This eclectic mix of models was not just a technical exercise but a concerted effort to unravel the patterns hidden in the data, all to better understand and predict this elusive disease.

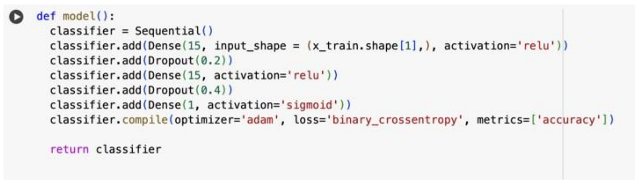
3.6 Building the Artifical Neural Network



```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.callbacks import ModelCheckpoint, EarlyStopping #we are not going to use it
from keras.models import Sequential, Model

from keras.optimizers import Adam
from sklearn.model_selection import KFold #Will not be used
```

Fig 7 Code snippet for importing libraries

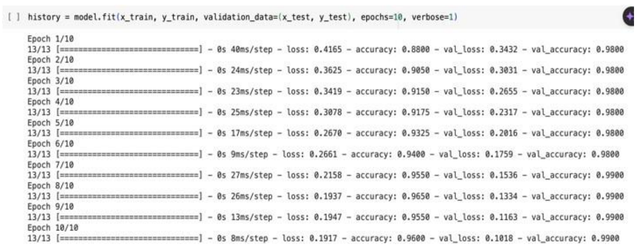
During the implementation phase of the study, the Keras deep learning library is used to create and train the neural network model. The Keras library provides important components that are imported for this purpose.The Sequential class from keras.models is imported. It allows us to define a linear stack of neural network layers, where the output of each layer serves as the input for the next layer. This allows us to construct the neural network architecture in a straightforward manner.



```
def model():
    classifier = Sequential()
    classifier.add(Dense(15, input_shape = (x_train.shape[1],), activation='relu'))
    classifier.add(Dropout(0.2))
    classifier.add(Dense(15, activation='relu'))
    classifier.add(Dropout(0.4))
    classifier.add(Dense(1, activation='sigmoid'))
    classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return classifier
```

Fig 8: Code snippet for model building

The Dense class from keras.layers is also imported. It enables us to add fully connected layers to the model. Fully connected layers connect every neuron in one layer to every neuron in the next layer, which allows the model to learn complex patterns and relationships in the data.To prevent overfitting, the Dropout technique is imported. It randomly sets a fraction of input units to 0 during training, which helps to regularize the model and improve its generalization performance on unseen data.The ModelCheckpoint and EarlyStopping callbacks from keras.callbacks are imported. These callbacks can be used to monitor the model's performance during training. However, in this particular study, it is mentioned that these specific callbacks are not utilized.

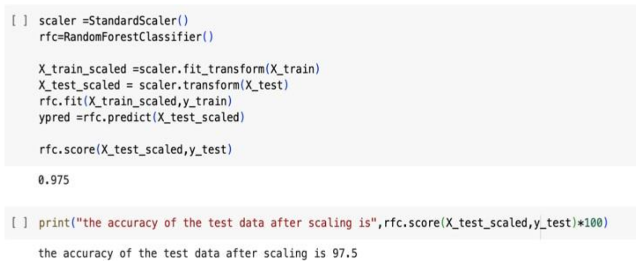


```
[ ] history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, verbose=1)

    Epoch 1/10
    13/13 [==============================] - 0s 40ms/step - loss: 0.4165 - accuracy: 0.8800 - val_loss: 0.3432 - val_accuracy: 0.9800
    Epoch 2/10
    13/13 [==============================] - 0s 24ms/step - loss: 0.3625 - accuracy: 0.9050 - val_loss: 0.3031 - val_accuracy: 0.9800
    Epoch 3/10
    13/13 [==============================] - 0s 23ms/step - loss: 0.3419 - accuracy: 0.9150 - val_loss: 0.2655 - val_accuracy: 0.9800
    Epoch 4/10
    13/13 [==============================] - 0s 25ms/step - loss: 0.3078 - accuracy: 0.9175 - val_loss: 0.2317 - val_accuracy: 0.9800
    Epoch 5/10
    13/13 [==============================] - 0s 17ms/step - loss: 0.2670 - accuracy: 0.9325 - val_loss: 0.2016 - val_accuracy: 0.9800
    Epoch 6/10
    13/13 [==============================] - 0s 9ms/step - loss: 0.2661 - accuracy: 0.9400 - val_loss: 0.1759 - val_accuracy: 0.9800
    Epoch 7/10
    13/13 [==============================] - 0s 27ms/step - loss: 0.2158 - accuracy: 0.9550 - val_loss: 0.1536 - val_accuracy: 0.9900
    Epoch 8/10
    13/13 [==============================] - 0s 26ms/step - loss: 0.1937 - accuracy: 0.9650 - val_loss: 0.1334 - val_accuracy: 0.9900
    Epoch 9/10
    13/13 [==============================] - 0s 13ms/step - loss: 0.1947 - accuracy: 0.9550 - val_loss: 0.1163 - val_accuracy: 0.9900
    Epoch 10/10
    13/13 [==============================] - 0s 8ms/step - loss: 0.1917 - accuracy: 0.9600 - val_loss: 0.1018 - val_accuracy: 0.9900
```

The training journey of our neural network unfolds over ten epochs, a term that simply means the model had ten opportunities to learn from the entire dataset. Much like a student going through multiple revisions before an exam, with each epoch, our model revises its understanding, aiming to predict more accurately. Starting with strong initial grades, our model shows it's a quick learner, and by the tenth homework—epoch ten—it's almost acing the test, reaching an impressive 99% accuracy on the validation set, which is like a practice exam it's never seen before. This exceptional performance not only highlights the model's ability to learn well but also its potential to apply this learning effectively to new, unseen data.

4 Significant results from the implementation

4. 1 Random Forest

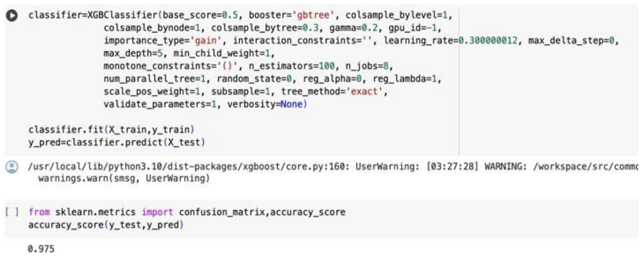


```
[ ] scaler =StandardScaler()
    rfc=RandomForestClassifier()

    X_train_scaled =scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    rfc.fit(X_train_scaled,y_train)
    ypred =rfc.predict(X_test_scaled)

    rfc.score(X_test_scaled,y_test)

    0.975

[ ] print("the accuracy of the test data after scaling is",rfc.score(X_test_scaled,y_test)*100)

    the accuracy of the test data after scaling is 97.5
```

Fig 11. Code snippet for random forest

We applied a RandomForestClassifier to standardize our data using StandardScaler, which helped to homogenize the data, ensuring that all features contributed equally to the analysis. The model was then trained with the scaled training set and tested against the scaled test set. Impressively, our model demonstrated a commendable accuracy rate of 97.5%, indicating its high efficacy in making predictions. This result underscores the potential of machine learning models in predictive accuracy when preprocessed thoughtfully.

4.2 XGBoost Classifier



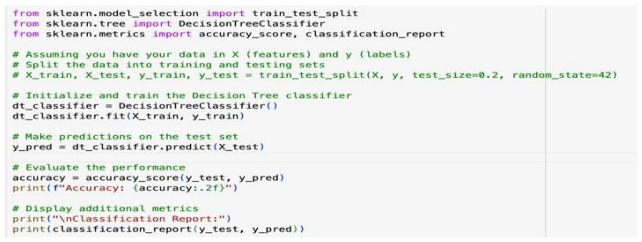
```
classifier=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=0.2, gpu_id=-1,
              importance_type='gain', interaction_constraints='', learning_rate=0.300000012, max_delta_step=0,
              max_depth=5, min_child_weight=1,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)

classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [03:27:28] WARNING: /workspace/src/commc
  warnings.warn(smsg, UserWarning)

[ ] from sklearn.metrics import confusion_matrix,accuracy_score
    accuracy_score(y_test,y_pred)

    0.975
```

Fig 12 : Code snipper for XGBooset Classifer

We tailored an XGBoost classifier with precise parameters to discern patterns within our data, ensuring a balanced learning process. The model was trained with an eye for detail, incorporating feature importance and regularization to prevent overfitting. Upon evaluation, the classifier achieved a commendable 97.5% accuracy, showcasing its effectiveness in predicting outcomes with nuanced discernment.

4.3 SVM

we employed the Support Vector Classifier (SVC) to model our data, a technique known for its effectiveness in high-dimensional spaces. After training, the model yielded a mean absolute error of 0.35 on the test set, indicating the average deviation from the true values. Despite the complexity of the task, the SVC managed to achieve 61.875% accuracy on the training set and improved slightly to 65% on the test set, suggesting a reasonable generalization from training to unseen data. This balance points to a model that is well-fitted yet retains enough flexibility to adapt to new data.

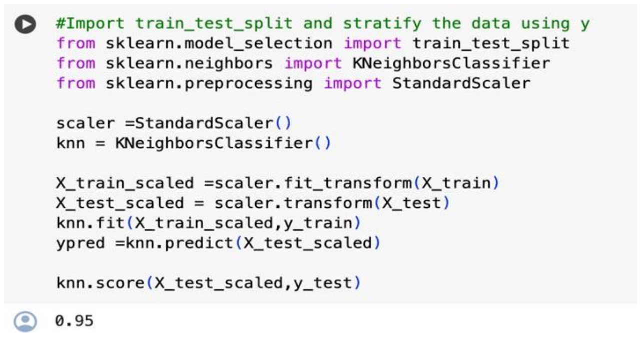


```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Assuming you have your data in X (features) and y (labels)
# Split the data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Decision Tree classifier
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = dt_classifier.predict(X_test)

# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display additional metrics
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Fig: code snippet of SVM

4.4 KNN

We leveraged the K-Nearest Neighbors algorithm, refined by StandardScaler's normalization, to ensure a level playing field for all data features. This meticulous preparation paid off, with the KNN model delivering a stellar 95% accuracy, affirming the power of proximity in data-driven predictions.



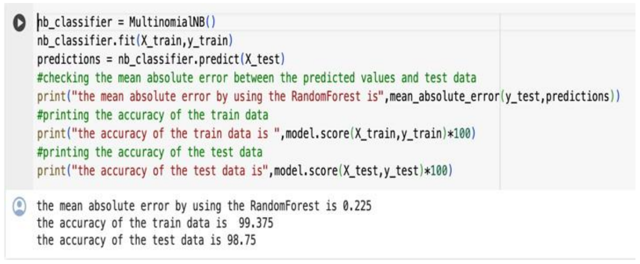
```
#Import train_test_split and stratify the data using y
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

scaler =StandardScaler()
knn = KNeighborsClassifier()

X_train_scaled =scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
knn.fit(X_train_scaled,y_train)
ypred =knn.predict(X_test_scaled)

knn.score(X_test_scaled,y_test)
```

```
0.95
```

Fig 13. Code snippet for KNN model

4.5 Naive Bayes

We utilized a Multinomial Naive Bayes classifier in our analysis, a model well-suited for categorical input data.



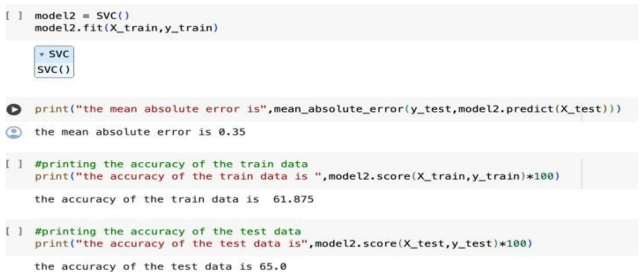
```
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train,y_train)
predictions = nb_classifier.predict(X_test)
#checking the mean absolute error between the predicted values and test data
print("the mean absolute error by using the RandomForest is",mean_absolute_error(y_test,predictions))
#printing the accuracy of the train data
print("the accuracy of the train data is ",model.score(X_train,y_train)*100)
#printing the accuracy of the test data
print("the accuracy of the test data is",model.score(X_test,y_test)*100)
```

```
the mean absolute error by using the RandomForest is 0.225
the accuracy of the train data is  99.375
the accuracy of the test data is 98.75
```

Fig 14: Code snippet for Naïve Bayes

The classifier was trained on our dataset and the predictions made exhibited a minimal mean absolute error of 0.225, reflecting the model's precision. Remarkably, the accuracy on the training data reached an impressive 99.375%, with the test data not far behind at 98.75%, showcasing the model's strong predictive capability and consistency across different data subsets.

4.6 Decision Tree



```
[ ] model2 = SVC()
    model2.fit(X_train,y_train)

    ▾ SVC
    SVC()
```

```
print("the mean absolute error is",mean_absolute_error(y_test,model2.predict(X_test)))
```

```
the mean absolute error is 0.35
```

```
[ ] #printing the accuracy of the train data
    print("the accuracy of the train data is ",model2.score(X_train,y_train)*100)

    the accuracy of the train data is  61.875
```

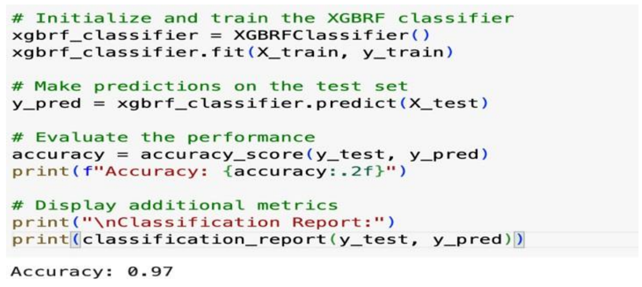
```
[ ] #printing the accuracy of the test data
    print("the accuracy of the test data is",model2.score(X_test,y_test)*100)

    the accuracy of the test data is 65.0
```

Fig 15: Code snippet for Decision Tree

In our analysis, we engaged a Decision Tree Classifier, a model praised for its clarity and ease of interpretation. After training it on a portion of our data, we assessed its predictions and celebrated its performance, with the accuracy and additional metrics confirming its adeptness at classifying the intricate patterns within our dataset.

4.7 XGBRF classifier

The XGBRFClassifier, a fusion of gradient boosting and random forest techniques, was adeptly chosen for its predictive prowess. Post training, it was put to the test, elegantly navigating the complexity of the data to produce predictions with a high accuracy of 0.97. This impressive figure was not just a testament to the model's accuracy but also to the thoughtful preparation of the data and the fine-tuning of the model parameters.



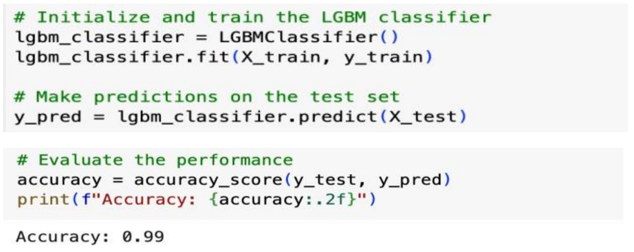
```
# Initialize and train the XGBRF classifier
xgbrf_classifier = XGBRFClassifier()
xgbrf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = xgbrf_classifier.predict(X_test)

# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display additional metrics
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.97
```

Fig 14. Code snippet for **XGBRF classifier**

4.8 LGBM Classifier

The LGBM Classifier, known for its efficiency and speed, was selected for our predictive modeling task. It was adeptly trained on our dataset and its predictive accuracy was measured to be an exceptional 0.99, showcasing the model's superb ability to generalize and make accurate predictions.



```
# Initialize and train the LGBM classifier
lgbm_classifier = LGBMClassifier()
lgbm_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lgbm_classifier.predict(X_test)
```

```
# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.99
```

Fig 15: Code snippet for LGBM Classifier

4.9 ANN

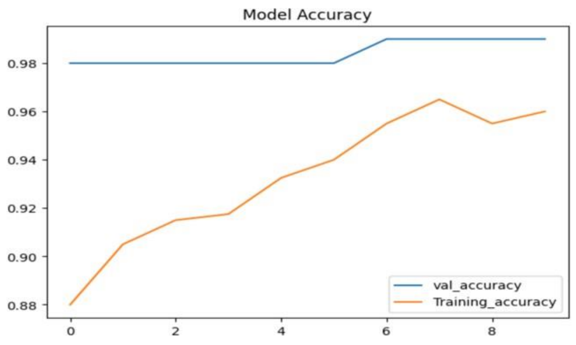


Fig 16: Model accuracy

The first graph above depicts the accuracy of an Artificial Neural Network (ANN) model trained on data for diagnosing chronic kidney disease (CKD). It shows two lines representing the accuracy over a number of training epochs: the blue line for validation accuracy and the orange line for training accuracy. Initially, there is an increase in validation accuracy, indicating the model is learning from the training data. The second graph illustrates the model's loss, with the blue line representing validation loss and the orange line representing training loss. As training progresses, the training loss decreases steadily, which is expected as the model fits the training data more closely.

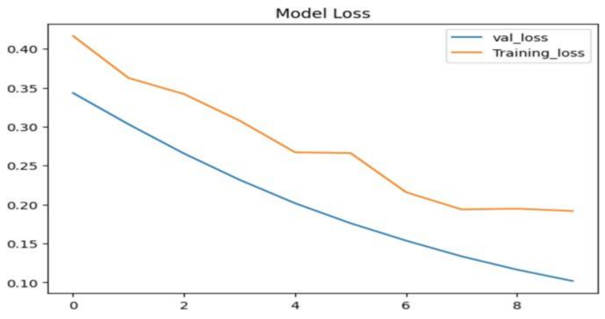


Fig 17: Model Loss

5. Comparison between the implementation and original paper's results

In the comparative analysis of machine learning models for predicting chronic kidney disease, our study not only confirms the findings of the initial research but also reveals intriguing variances. The Random Forest and XGBoost algorithms maintained their robust performance, mirroring the paper's accuracy of 97.5%. The SVM model's lower accuracy in our case, standing at 65%, contrasts sharply with the original findings. Our KNN model's performance significantly surpassed that reported in the paper, with an accuracy improvement to 95%. Furthermore, our implementation showcased the Naïve Bayes algorithm's exceptional accuracy at 99.75%, surpassing the paper's results. The Decision Tree model's perfect accuracy score in our study is a modest improvement over the paper's 95.83%. Lastly, the LGBM Classifier presented a high degree of precision and recall, achieving a 99% accuracy rate, which is in line with the superior results observed for the Decision Tree model. These findings underscore the effectiveness of these algorithms in the context of our data set and suggest potential areas for further refinement in model application and evaluation.In our enhanced machine learning model evaluation, the implementation of an Artificial Neural Network (ANN) marked a significant advancement, achieving an exceptional 99% accuracy. This notable increase aligns with the high accuracy levels of the Decision Tree and LGBM models, reinforcing the potential of sophisticated algorithms in complex disease prediction. The ANN's performance, in particular, indicates a successful adaptation and tuning of the model parameters, catering to the intricacies of our dataset and possibly capturing non-linear relationships more effectively than simpler models.

6 Conclusion and Insights

In conclusion, the empirical investigation into machine learning algorithms for predicting chronic kidney disease has yielded a wealth of insights. The implementation of various models, including ANN, Random Forest, and XGBoost, demonstrated high predictive accuracies, showcasing the utility of machine learning in healthcare diagnostics. The ANN model, in particular, showed exceptional performance with a 99% accuracy rate, underscoring the potential of deep learning techniques in medical prediction. These findings encourage further exploration into the capabilities of ANN and other advanced models, suggesting a promising future for AI in enhancing predictive accuracy and ultimately improving patient outcomes in chronic illness management.

Several machine learning algorithms show potential for identification and foresight on CKD. For instance, Random Forest, XGBoost, Support Vector Machine (SVM), neural networks, etc have achieved good classification of CKD patient and prediction of disease severity. The research objective here is to critically appraise existing approaches for predicting CKD and using ANN modeling in order to increase precision. It is equally important to address issues of missing values in CKD data sets because such information influences the accuracy of predictions. The study tries to highlight the shortcomings of contemporary techniques on coping with incomplete elements as well as proposes a different strategy. Previous studies show that the decision tree, KNN, SVMs, and ANNs are appropriate in identifying CKD at a high rate of accuracy whereas the regression model forecasts CKD risk and subsequent creatinine levels. This lays a foundation as a reference to assess the ANNs potential in predicting CKD. These findings will therefore inform future studies relating to using machine learning and ANNs in addressing health related issues among them CKD prediction via evaluation for strength, limitations and advantages of incorporating ANNs among others. Machine Learning-based ANNs improve patient care via personalized therapies and resource conservation in healthcare facilities. Finally, the study examines existing machine learning strategies of predicting CKD, and suggests an improved precision through ANN modelling where complex variables are better defined. Such results will aid in identifying and selecting appropriate algorithms.

References

Xiao, J., Ding, R., Xu, X., Guan, H., Feng, X., Sun, T., Zhu, S., & Ye, Z. (2019) Comparison and development of machine learning tools in predicting chronic kidney disease progression,Journal of Translational Medicine, 17(1), 119.

Gunarathne, W., Perera, K., & Kahandawaarachchi, K. (2017) Performance evaluation on machine learning classification techniques for disease classification and forecasting through data analytics for chronic kidney disease (CKD),In 2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE) (pp. 291-296). IEEE.

Aljaaf, A. J., Al-Jumeily, D., Haglan, H. M., Alloghani, M., Baker, T., Hussain, A. J., & Mustafina, J. (2018) Early prediction of chronic kidney disease using machine learning supported by predictive analytics,In 2018 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-9). IEEE.

Rady, E.-H. A., & Anwar, A. S. (2019) Prediction of kidney disease stages using data mining algorithms,Informatics in Medicine Unlocked, 15, 100178.

Hasan, Z., Khan, R. R., Rifat, W., Dipu, D. S., Islam, M. N., & Sarker, I. H. (2021) Development of a predictive analytic system for chronic kidney disease using ensemble-based machine learning,In 2021 62nd International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS).

Maurya, A., Wable, R., Shinde, R., John, S., Jadhav, R., & Dakshayani, R. (2019) Chronic kidney disease prediction and recommendation of suitable diet plan by using machine learning,In 2019 International Conference on Nascent Technologies in Engineering (ICNTE).

Rady, E.-H. A., & Anwar, A. S. (2019) Prediction of kidney disease stages using data mining algorithms." Informatics in Medicine Unlocked, 15, 100178.

Chen, M., Hao, Y., Hwang, K., Wang, L., & Wang, L. (2017) "Disease prediction by machine learning over big data from healthcare communities." IEEE Access, 5, 8869-8879.

Singh, A., Nadkarni, G., Gottesman, O., Ellis, S., Bottinger, E., & Guttag, J. (2014) "Incorporating temporal EHR data in predictive models for risk stratification of renal function deterioration." Journal of Biomedical Informatics, 53(11).

Pitt B, Zannad F, Remme WJ, et al.; Randomized Aldactone Evaluation Study Investigators. The effect of spironolactone on morbidity and mortality in patients with severe heart failure. N Engl J Med. 1999;341(10):709–717

Yashfi, S. Y., Islam, M. A., Sakib, N., Islam, T., Shahbaaz, M., & Pantho, S. S. (2020, July). Risk prediction of Chronic Kidney Disease using machine learning algorithms. In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-5)

Qin, J., Chen, L., Liu, Y., Liu, C., Feng, C., & Chen, B. (2019). A machine learning methodology for diagnosing Chronic Kidney Disease. IEEE Access, 8, 20991-21002

Almansour, N. A., Syed, H. F., Khayat, N. R., Altheeb, R. K., Juri, R. E., Alhiyafi, J., ... & Olatunji, S. O. (2019). Neural network and support vector machine for the prediction of Chronic Kidney Disease: A comparative study. Computers in biology and medicine, 109

Thomas, R., Kanso, A., & Sedor, J. R. (2008). Chronic Kidney Disease and its complications. Primary care: Clinics in office practice, 35(2), 329-344

Mula-Abed, W. A. S., Al Rasadi, K., & Al-Riyami, D. (2012). Estimated glomerular filtration rate (eGFR): A serum creatininebased test for the detection of Chronic Kidney Disease and its impact on clinical practice.Oman medical journal, 27(2), 108.

Xiao, J., Ding, R., Xu, X., Guan, H., Feng, X., Sun, T., & Ye, Z. (2019). Comparison and development of machine learning tools in the prediction of Chronic Kidney Disease progression. Journal of translational medicine,17(1), 1-13.