Chronic Kidney Disease Prediction Using Machine Learning Classification Models

Problem Statement:

Chronic Kidney Disease (CKD) is a long-term condition where the kidneys gradually lose function. Detecting it early is important so that patients can get treatment before it becomes serious.

The goal of this project is to build a machine learning model that can predict whether a person has CKD based on their medical information, such as age, blood pressure, blood test results, and health conditions like diabetes or anaemia.

This model can help doctors make faster and better decisions by using data to identify people at risk of kidney disease.

Dataset:

This dataset contains medical information about individuals, used to help predict the presence of chronic kidney disease (CKD).

- Total Records (Rows): 399
- Total Features (Columns): 25 (including the target column)

Features:

The dataset includes 24 input features such as:

- Age, blood pressure, specific gravity, albumin, sugar levels, Red and white blood cell counts, Blood urea, serum creatinine, sodium, potassium, Presence of hypertension, diabetes, anaemia, etc.

[3]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	
0	2.000000	76.459948	c	3.0	0.0	normal	abnormal	notpresent	notpresent	148.112676	...	38
1	3.000000	76.459948	c	2.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	34
2	4.000000	76.459948	a	1.0	0.0	normal	normal	notpresent	notpresent	99.000000	...	34
3	5.000000	76.459948	d	1.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	38
4	5.000000	50.000000	c	0.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	36
...	...	...	...	...	...	...	...	...	...	...	...	...
394	51.492308	70.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	219.000000	...	37
395	51.492308	70.000000	c	0.0	2.0	normal	normal	notpresent	notpresent	220.000000	...	27
396	51.492308	70.000000	c	3.0	0.0	normal	normal	notpresent	notpresent	110.000000	...	26
397	51.492308	90.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	207.000000	...	38
398	51.492308	80.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	100.000000	...	53

399 rows × 25 columns

Pre-Processing Methods:

To prepare the dataset for machine learning classification models, the following preprocessing steps were performed:

1. **Handling Categorical (Nominal) Data:**

Categorical columns were converted into numerical format using One-Hot Encoding with pd.get_dummies(). This creates new binary columns for each category.

- Method Used: pd.get_dummies(dataset, dtype = int, drop_first=True)

Column	Type	Encoding Method	Notes
rbc	Nominal	One-Hot Encoding	abnormal/normal -> binary column created
sg	Nominal	One-Hot Encoding	Sg_b, sg_c etc., (drop_first = True)
pc	Nominal	One-Hot Encoding	abnormal/normal -> binary column created
pcc, ba	Nominal	One-Hot Encoding	present/notpresent
htn, dm, cad, ane	Nominal	One-Hot Encoding	yes/no -> converted to binary
appet, pe	Nominal	One-Hot Encoding	good/poor
classification	Nominal	One-Hot Encoding	Yes/no ->converted to binary

2. No Encoding Needed for Numeric Columns:

Numerical columns were used as-is.

Column	Type	Encoding Method	Notes
age, bp, al, su, bgr, bu, sc, sod, pot, hrmo, pcv, wc, rc	Numeric	-	Used as-is

[5]:

	age	bp	al	su	bgr	bu	sc	sod	pot	hrn
0	2.000000	76.459948	3.0	0.0	148.112676	57.482105	3.077356	137.528754	4.627244	12.5181
1	3.000000	76.459948	2.0	0.0	148.112676	22.000000	0.700000	137.528754	4.627244	10.7000
2	4.000000	76.459948	1.0	0.0	99.000000	23.000000	0.600000	138.000000	4.400000	12.0000
3	5.000000	76.459948	1.0	0.0	148.112676	16.000000	0.700000	138.000000	3.200000	8.1000
4	5.000000	50.000000	0.0	0.0	148.112676	25.000000	0.600000	137.528754	4.627244	11.8000
...	...	...	...	...	...	...	...	...	...	
394	51.492308	70.000000	0.0	0.0	219.000000	36.000000	1.300000	139.000000	3.700000	12.5000
395	51.492308	70.000000	0.0	2.0	220.000000	68.000000	2.800000	137.528754	4.627244	8.7000
396	51.492308	70.000000	3.0	0.0	110.000000	115.000000	6.000000	134.000000	2.700000	9.1000
397	51.492308	90.000000	0.0	0.0	207.000000	80.000000	6.800000	142.000000	5.500000	8.5000
398	51.492308	80.000000	0.0	0.0	100.000000	49.000000	1.000000	140.000000	5.000000	16.3000

399 rows × 28 columns

Model Development:

Support Vector Machine (SVM):

1. **Best Parameters**: {'C': 10, 'gamma': 'auto', 'kernel': 'sigmoid'}
2. **Accuracy**: 0.99
3. **F1 Score (weighted)**: 0.99
4. **ROC AUC Score**: 1.00

```
18]: from sklearn.metrics import f1_score
     f1_macro =  f1_score(y_test, grid_pred, average = 'weighted')
     print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)

     The f1_macro value for best parameter {'C': 10, 'gamma': 'auto', 'kernel': 'sigmoid'}: 0.99
     24946382275899

19]: print("The confusion matrix: \n", cm)

     The confusion matrix:
      [[51  0]
      [ 1 81]]

20]: print("The report: \n", clf_report)

     The report:
                   precision    recall  f1-score   support

                0       0.98      1.00      0.99        51
                1       1.00      0.99      0.99        82

         accuracy                           0.99       133
        macro avg       0.99      0.99      0.99       133
     weighted avg       0.99      0.99      0.99       133

21]: from sklearn.metrics import roc_auc_score
     roc_auc_score(y_test, grid.predict_proba(X_test)[:,1])

21]: np.float64(1.0)
```

Decision Tree:

1. **Best Parameters**: {'criterion': 'entropy', 'max_features': 'log2', 'splitter': 'random'}
2. **Accuracy**: 0.96
3. **F1 Score (weighted)**: 0.96
4. **ROC AUC Score**: 0.965

[17]:
```
from sklearn.metrics import f1_score
f1_macro =  f1_score(y_test, grid_pred, average = 'weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

```
The f1_macro value for best parameter {'criterion': 'entropy', 'max_features': 'log2', 'spl
itter': 'random'}: 0.9625928174473452
```

[18]:
```
print("The confusion matrix: \n", cm)
```

```
The confusion matrix:
 [[50  1]
 [ 4 78]]
```

[19]:
```
print("The report: \n", clf_report)
```

```
The report:
               precision    recall  f1-score   support

           0       0.93      0.98      0.95        51
           1       0.99      0.95      0.97        82

    accuracy                           0.96       133
   macro avg       0.96      0.97      0.96       133
weighted avg       0.96      0.96      0.96       133
```

[20]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, grid.predict_proba(X_test)[:,1])
```

[20]:
```
np.float64(0.9658058345289334)
```

Random Forest:

1. **Best Parameters**: {'criterion': 'entropy', 'max_features': 'log2', 'n_estimators': 100}
2. **Accuracy**: 0.98
3. **F1 Score (weighted)**: 0.98
4. **ROC AUC Score**: 0.998

[17]:
```
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test, grid_pred, average = 'weighted')
print("The f1_macro value for the best parameter {}:".format(grid.best_params_), f1_macro)
```

```
The f1_macro value for the best parameter {'criterion': 'entropy', 'max_features': 'log2',
'n_estimators': 100}: 0.9849624060150376
```

[18]:
```
print("The confusion matrix: \n", cm)
```

```
The confusion matrix:
 [[50  1]
 [ 1 81]]
```

[19]:
```
print("The report:\n", clf_report)
```

```
The report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98        51
           1       0.99      0.99      0.99        82

    accuracy                           0.98       133
   macro avg       0.98      0.98      0.98       133
weighted avg       0.98      0.98      0.98       133
```

[20]:
```
from sklearn.metrics import roc_auc_score             #ROC curve_area under curve
                                                      #true positive and false positive rate
roc_auc_score(y_test, grid.predict_proba(X_test)[:,1])#probability as input
```

[20]:
```
np.float64(0.9997608799617408)
```

Logistic Regression:

1. **Best Parameters**: {'penalty': 'l2', 'solver': 'newton-cg'}
2. **Accuracy**: 0.99
3. **F1 Score (weighted)**: 0.99
4. **ROC AUC Score**: 1.00

[17]:
```
from sklearn.metrics import f1_score
f1_macro =  f1_score(y_test, grid_pred, average = 'weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

```
The f1_macro value for best parameter {'penalty': 'l2', 'solver': 'newton-cg'}: 0.992494638
2275899
```

[18]:
```
print("The confusion matrix: \n", cm)
```

```
The confusion matrix:
 [[51  0]
 [ 1 81]]
```

[19]:
```
print("The report: \n", clf_report)
```

```
The report:
               precision    recall  f1-score   support

           0       0.98      1.00      0.99        51
           1       1.00      0.99      0.99        82

    accuracy                           0.99       133
   macro avg       0.99      0.99      0.99       133
weighted avg       0.99      0.99      0.99       133
```

[20]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, grid.predict_proba(X_test)[:,1])
```

[20]:
```
np.float64(1.0)
```

K-NearestNeighbor:

1. **Best Parameters**: {'metric': 'minkowski', 'n_neighbors': 3, 'p': 1, 'weights': 'uniform'}
2. **Accuracy**: 0.97
3. **F1 Score (weighted)**: 0.97
4. **ROC AUC Score**: 0.98

[17]:
```
from sklearn.metrics import f1_score
f1_macro =  f1_score(y_test, grid_pred, average = 'weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

```
The f1_macro value for best parameter {'metric': 'minkowski', 'n_neighbors': 3, 'p': 1, 'we
ights': 'uniform'}: 0.9701163285572423
```

[18]:
```
print("The confusion matrix: \n", cm)
```

```
The confusion matrix:
 [[51  0]
 [ 4 78]]
```

[19]:
```
print("The report: \n", clf_report)
```

```
The report:
               precision    recall  f1-score   support

           0       0.93      1.00      0.96        51
           1       1.00      0.95      0.97        82

    accuracy                           0.97       133
   macro avg       0.96      0.98      0.97       133
weighted avg       0.97      0.97      0.97       133
```

[20]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, grid.predict_proba(X_test)[:,1])
```

[20]:
```
np.float64(0.9873266379722621)
```

Naïve Bayes:

1. **Best Parameters**: {'var_smoothing': 1e-08}
2. **Accuracy**: 0.98
3. **F1 Score (weighted)**: 0.97
4. **ROC AUC Score**: 1.00

```
from sklearn.metrics import f1_score
f1_macro =  f1_score(y_test, grid_pred, average = 'weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

```
The f1_macro value for best parameter {'var_smoothing': 1e-08}: 0.9775556904684072
```

```
print("The confusion matrix: \n", cm)
```

```
The confusion matrix:
 [[51  0]
 [ 3 79]]
```

```
print("The report: \n", clf_report)
```

```
The report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97        51
           1       1.00      0.96      0.98        82

    accuracy                           0.98       133
   macro avg       0.97      0.98      0.98       133
weighted avg       0.98      0.98      0.98       133
```

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, grid.predict_proba(X_test)[:,1])
```

```
np.float64(1.0)
```

Final Model Selection:

Model	Best Parameters	Accuracy	F1 Score (Weighted)	ROC AUC Score
Support Vector Machine (SVM)	{'C': 10, 'gamma': 'auto', 'kernel': 'sigmoid'}	0.99	0.99	1.00
Decision Tree	{'criterion': 'entropy', 'max_features': 'log2', 'splitter': 'random'}	0.96	0.96	0.965
Random Forest	{'criterion': 'entropy', 'max_features': 'log2', 'n_estimators': 100}	0.98	0.98	0.998
Logistic Regression	{'penalty': 'l2', 'solver': 'newton-cg'}	0.99	0.99	1.00
K-NearestNeighbor	{'metric': 'minkowski', 'n_neighbors': 3, 'p': 1, 'weights': 'uniform'}	0.97	0.97	0.98
Naïve Bayes	{'var_smoothing': 1e-08}	0.98	0.97	1.00

After evaluating all models, **Logistic Regression** and **SVM** both demonstrated the highest performance across all metrics (accuracy, F1 score, and ROC AUC score). However, **Logistic Regression** was selected as the final model because:

- It is simpler and faster to train.
- It is easier to interpret and explain to medical professionals.
- It performs equally well compared to more complex models.
- It reduces the risk of overfitting and generalizes well on new data.

Thus, Logistic Regression is considered a reliable and practical choice for predicting CKD in clinical applications.