CHRONIC KIDNEY DISEASE PREDICTION MODEL

Lalitha Mangalam M

SRI ESHWAR COLLEGE OF ENGINEERING

CHRONIC KIDNEY DISEASE PREDICTION MODEL

1.PROBLEM STATEMENT IDENTIFICATION

Chronic Kidney Disease (CKD) is a significant global health issue affecting millions of people worldwide. Early detection and intervention are crucial for preventing progression to end-stage renal disease and associated complications. Therefore, there is a need to develop an accurate and reliable CKD prediction model that can identify individuals at risk of developing CKD.The goal of this project is to build a predictive model that can assess an individual's risk of developing CKD based on various demographic, clinical, and laboratory features. The model will take input variables such as age, sex, blood pressure, glomerular filtration rate (GFR), serum creatinine levels, albuminuria, diabetes status, hypertension, and other relevant factors that may contribute to CKD development. The final model should be user-friendly, allowing healthcare professionals to input patient information easily and obtain a CKD risk score promptly. Additionally, the model's interpretability should be considered to provide insights into the key features driving the predictions, enhancing clinical decision-making

2.INFORMATION ABOUT DATASET

The dataset provided contains information about diverse set of individuals including

age: Age of the patient (numeric).

bp: Blood pressure (numeric).

sg: Specific gravity of urine (numeric).

al: Albumin level in urine (numeric).

su: Sugar level in urine (numeric).

rbc: Red blood cells (categorical: "normal" or "abnormal").

pc: Pus cell (categorical: "normal" or "abnormal").

pcc: Pus cell clumps (categorical: "present" or "not present").

ba: Bacteria (categorical: "present" or "not present").

bgr: Blood glucose random (numeric).

bu: Blood urea (numeric).

sc: Serum creatinine (numeric).

sod: Sodium (numeric).

pot: Potassium (numeric).

hrmo: Hemoglobin (numeric).

pcv: Packed cell volume (numeric).

wc: White blood cell count (numeric).

rc: Red blood cell count (numeric).

htn: Hypertension (categorical: "yes" or "no").

dm: Diabetes mellitus (categorical: "yes" or "no").

cad: Coronary artery disease (categorical: "yes" or "no").

appet: Appetite (categorical: "good" or "poor").

pe: Pedal edema (categorical: "yes" or "no").

ane: Anemia (categorical: "yes" or "no").

Number of rows: 399

Number of columns: 25

3. DATA PRE-PROCESSING

The dataset has eleven columns with nominal data namely rbc, pc, pcc, ba, htn, dm, cad, appet, pe, ane, classification .To make these data to standard form we preprocess the data(nominal) using one hot encoding strategy

```
[5] dataset=pd.get_dummies(dataset,drop_first=True)
```

```
[8] dataset
```

	age	bp	al	su	bgr	bu	sc	sod	pot	hrmo	...	pc_normal	pcc_present	ba_present	htn_yes	dm_yes	cad_yes	appet_yes	pe_yes	ane_yes	classification_yes
0	2.000000	76.459948	3.0	0.0	148.112676	57.482105	3.077356	137.528754	4.627244	12.518156	...	0	0	0	0	0	0	1	1	0	1
1	3.000000	76.459948	2.0	0.0	148.112676	22.000000	0.700000	137.528754	4.627244	10.700000	...	1	0	0	0	0	0	1	0	0	1
2	4.000000	76.459948	1.0	0.0	99.000000	23.000000	0.600000	138.000000	4.400000	12.000000	...	1	0	0	0	0	0	1	0	0	1
3	5.000000	76.459948	1.0	0.0	148.112676	16.000000	0.700000	138.000000	3.200000	8.100000	...	1	0	0	0	0	0	1	0	1	1
4	5.000000	50.000000	0.0	0.0	148.112676	25.000000	0.600000	137.528754	4.627244	11.800000	...	1	0	0	0	0	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
394	51.492308	70.000000	0.0	0.0	219.000000	36.000000	1.300000	139.000000	3.700000	12.500000	...	1	0	0	0	0	0	1	0	0	1
395	51.492308	70.000000	0.0	2.0	220.000000	68.000000	2.800000	137.528754	4.627244	8.700000	...	1	0	0	1	1	0	1	0	1	1
396	51.492308	70.000000	3.0	0.0	110.000000	115.000000	6.000000	134.000000	2.700000	9.100000	...	1	0	0	1	1	0	0	0	0	1
397	51.492308	90.000000	0.0	0.0	207.000000	80.000000	6.800000	142.000000	5.500000	8.500000	...	1	0	0	1	1	0	1	0	1	1
398	51.492308	80.000000	0.0	0.0	100.000000	49.000000	1.000000	140.000000	5.000000	16.300000	...	1	0	0	0	0	0	1	0	0	0

399 rows × 28 columns

4.MODEL CREATION

4.1 KNN Algorithm

Here the algorithm uses the standard split of dataset as
>80% Train set

>20% Test set

N - Neighbours	F1- score (avg)
1	81%
3	77%
5	75%
7	71%

Conclusion:

For the KNN algorithm the parameter produce the optimal solution as 81% with the parameter N-Neighbours = 1

4.2 Naïve Bayes Algorithm

Here the algorithm uses the standard split of dataset as
>80% Train set

>20% Test set

i) Multinomial NB ->82%

ii) Bernoulli NB ->39%

iii) Categorical NB ->82%

iv) Complement NB -> 51%

Conclusion:

For the multiple Naïve Bayes algorithms the produced the optimal solution is 82% by multinomial and categorical variants

4.3 SVM Algorithm

Here the algorithm uses the standard split of dataset as

>80% Train set

>20% Test set

```
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
```

Conclusion:

For the SVM algorithm the produced the optimal solution is 99% with the parameters C=10, gamma = auto, kernel = sigmoid

```
from sklearn.metrics import f1_score
f1_macro=f1_score(y_test,grid_predictions,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'}: 0.9924946382275899
```

4.4 Decision Tree Algorithm

Here the algorithm uses the standard split of dataset as

>80% Train set

>20% Test set

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

     The report:
                   precision    recall  f1-score   support

                0       0.96      1.00      0.98        51
                1       1.00      0.98      0.99        82

         accuracy                           0.98       133
        macro avg       0.98      0.99      0.98       133
     weighted avg       0.99      0.98      0.99       133
```

Conclusion:

For the decision tree algorithm the produced the optimal solution is 98% with the parameters criterion = gini, max_features = log2, splitter = best

```
[18] from sklearn.metrics import roc_auc_score

     roc_auc_score(y_test,grid.predict_proba(X_test)[:,1])

     0.9878048780487805
```

4.5 Random Forest Algorithm

Here the algorithm uses the standard split of dataset as
>80% Train set

>20% Test set

```
[ ]  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
```

Conclusion:

For the SVM algorithm the produced the optimal solution is 98% with the parameter criterion = gini, max_features = sqrt, n_estimators = 100

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

    The f1_macro value for best parameter {'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators': 100}: 0.9849624060150376
```

4.6 Logistic Regression Algorithm

Here the algorithm uses the standard split of dataset as
>80% Train set

>20% Test set

```
[17]  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
```

Conclusion:

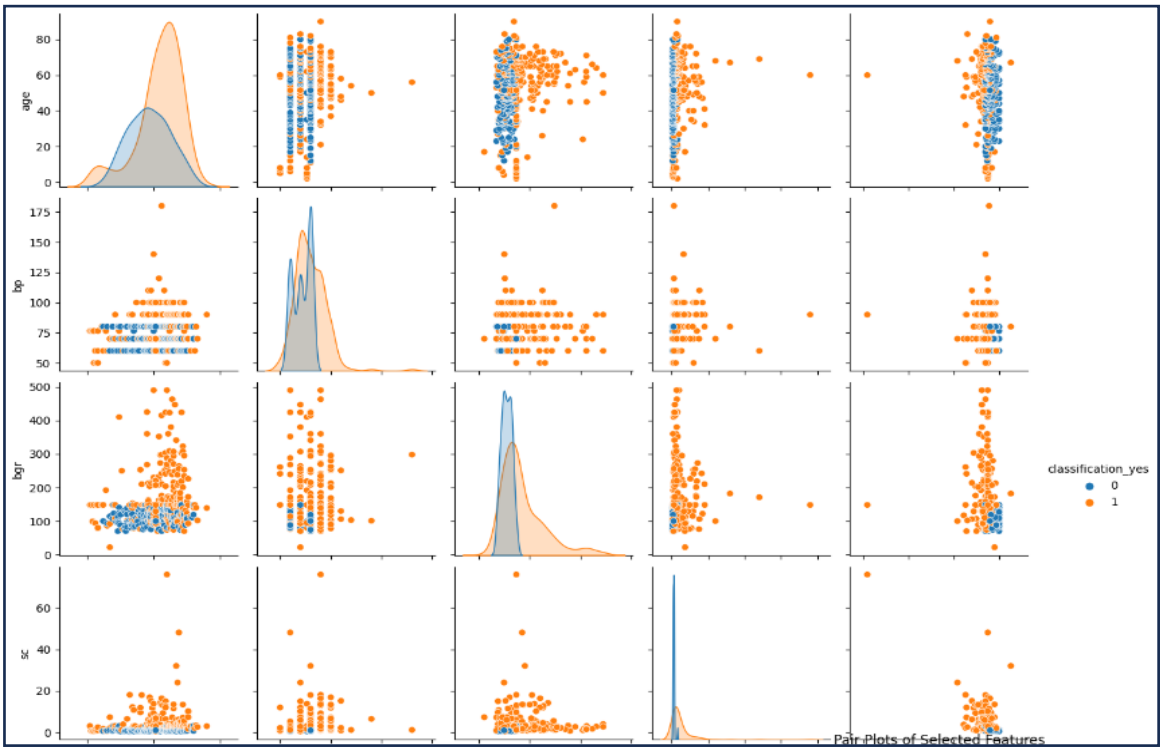
For the SVM algorithm the produced the optimal solution is 99% with the parameter penalty = 12, solver = newton-cg

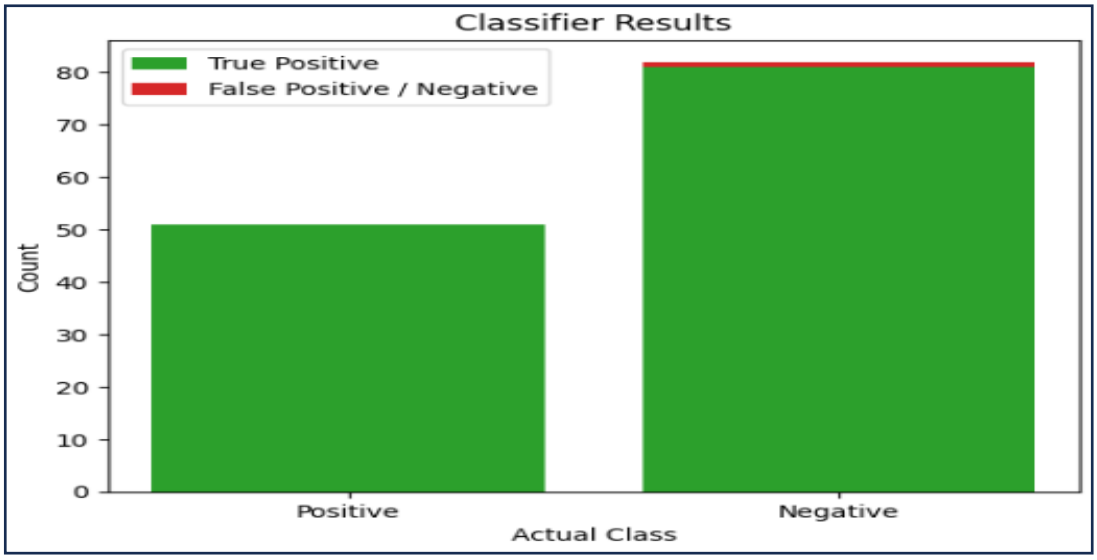
```
[15]
    from sklearn.metrics import f1_score
    f1_macro=f1_score(y_test,grid_predictions,average='weighted')
    print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)

    The f1_macro value for best parameter {'penalty': 'l2', 'solver': 'newton-cg'}: 0.9924946382275899
```

5. FINAL MODEL CONCLUSION

Based on my comprehensive evaluation and comparison of Support Vector Machine (SVM) and Logistic Regression on the CKD dataset, I analysed that Support Vector Machine (SVM) outperforms the other in terms of relevant metrics, e.g., accuracy, sensitivity, specificity, etc.. Therefore, I conclude using Support Vector Machine (SVM) for the task of CKD classification due to its superior performance and SVM try to maximize the margin between the closest support vectors whereas logistic regression maximize the posterior class probability. SVM is deterministic ,while LR is probabilistic.For the kernel space, SVM is faster

RESULTS





Click👇

Github link for all models training