CLASSIFICATION ALGORITHM

Assignment

Abstract

To predict the Chronic Kidney Disease [CKD] based on the patient data

1st September 2025

Ilango P

pailango@gmail.com

Objective

To build a predictive model that identifies the likelihood of Chronic Kidney Disease (CKD) based on various patient attributes.

Dataset Overview

The client has provided a dataset containing 399 records with 24 parameters with the target fields of "Classification".

age
bp
sg
al
su
rbc
pc
pcc
ba
bgr
bu
sc
sod
pot
hrmo
pcv
wc
rc
htn
dm
cad
appet
pe
ane
classification

The goal is to predict the **classification** using the other fields as input features. The dataset size and feature set are considered sufficient for model development.

Model Development Approach

1. Domain Selection

- The dataset consists of numerical and categorical data, making it suitable for **Machine Learning**.

2. Learning Type

- Since both input features and the target variable are available and clearly defined, this falls under **Supervised Learning**.

3. Learning Task

- Since the target variable represents categorical outcomes, this constitutes a classification problem

---

Modelling Phases

Data Preparation

- **Data Source**: The dataset is provided in a file named CKD.csv.
- **Feature Types**:
  - **Numerical**: Thirteen fields are of numerical type – no preprocessing required.
  - **Categorical**: Eleven input fields are categorical type – need to be converted to numerical format.
  - Some are binary and some ordinal (e.g., normal/abnormal, yes/no, a/b/c/d etc.).
  - Encoding options: **Label Encoding** or **One-Hot Encoding** (both yield similar results).

Train-Test Split

- The dataset will be split into training and testing sets in a **70:30 ratio**.

Model Training and Evaluation

The following regression algorithms were used to identify the best model:

Training Parameters applied:

1. Support Vector Regression (SVR)

{'kernel': ['linear'],'C': [10,100,1000,2000,3000]},

{'kernel': ['poly','rbf','sigmoid'], 'C': [10,100,1000,2000,3000], 'gamma': ['scale', 'auto']}

2. Decision Tree Regressor

{'criterion': ['gini', 'entropy', 'log_loss'],

'max_features': ['log2', 'sqrt', None],

'splitter': ['best', 'random']}

3. Random Forest Regressor

{'criterion':['gini', 'entropy'],

'max_features':['log2','sqrt',None]}Parameters: n_estimators=50, random_state=10, criterion=None

4. Logistic Regressor

```
# Parameters for 'l1' penalty (compatible with liblinear and saga)
{'penalty': ['l1'],
 'C': [0.001, 0.01, 0.1, 1, 10, 100],
 'solver': ['liblinear', 'saga']},

# Parameters for 'l2' penalty (compatible with multiple solvers)
{'penalty': ['l2'],
 'C': [0.001, 0.01, 0.1, 1, 10, 100],
 'solver': ['liblinear', 'lbfgs', 'saga']},

# Parameters for 'elasticnet' penalty (only with saga)
{'penalty': ['elasticnet'],
 'l1_ratio': [0.5], # You can add more values to tune this parameter
 'C': [0.001, 0.01, 0.1, 1, 10, 100],
 'solver': ['saga']},

# Parameters for no penalty (compatible with lbfgs, newton-cg, sag)
{'penalty': [None], # Use None (the Python object), not the string 'None'
 'C': [0.001, 0.01, 0.1, 1, 10, 100],
 'solver': ['lbfgs']} # saga is not compatible with no penalty
```

5. K Neighbor

```
{'n_neighbors': [3, 5, 7, 9, 11],
'weights': ['uniform', 'distance'],
'algorithm': ['auto', 'ball_tree', 'kd_tree'],
```

'leaf_size': [20, 30, 40, 50],

'p': [1, 2]}

6. Naïve bayes

Parameters for Gaussian Naive Bayes

{'classifier': [GaussianNB()],

'classifier__var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]},

Parameters for Multinomial Naive Bayes

{'classifier': [MultinomialNB()],

'classifier__alpha': [0.1, 0.5, 1.0],

'classifier__fit_prior': [True, False]},

Parameters for Bernoulli Naive Bayes

{'classifier': [BernoulliNB()],

'classifier__alpha': [0.1, 0.5, 1.0],

'classifier__fit_prior': [True, False]}

Evaluation:

Best model and scores:

Model	Parameter	f1_value	roc_auc_score
SVM	'C': 10, 'gamma': 'scale', 'kernel': 'sigmoid'	0.98340188	0.999703704
Decision Tree	'criterion': 'log_loss', 'max_features': 'log2', 'splitter': 'random'	0.991647444	0.988888889
Random Forest	'criterion': 'gini', 'max_features': 'sqrt'	0.983333333	0.999703704
Logistic regression	**'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'**	**0.993288591**	**1.00000000**
Kneighbor	'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 5, 'p': 1, 'weights': 'uniform'	0.958333333	0.999555556
Naïve bayes	'classifier': GaussianNB(), 'classifier__var_smoothing': 1e-09	0.985680145	1.00000000

Model Selection

Based on the evaluation data provided, **Logistic Regression** is the best performing model overall. It achieved the highest scores in both key metrics, with a perfect **ROC AUC score** of **1.000** and the highest **F1 score** of **0.993**.

🎯 Metric Breakdown

- **F1 Score**: This metric is a balance of precision and recall. A high F1 score indicates that the model is excellent at identifying positive cases (**high recall**) while also avoiding false alarms (**high precision**). The Logistic Regression model had the highest F1 score, making it the most reliable choice when both types of errors are costly.
- **ROC AUC Score**: The Area Under the Receiver Operating Characteristic curve measures a model's ability to distinguish between the positive and negative classes. A score of **1.0** indicates a perfect model that can completely separate the two classes. The Logistic Regression and Naïve Bayes models both achieved this perfect score, showing their superior ability to rank predictions correctly.

🏆 Conclusion

While multiple models performed well, **Logistic Regression** stands out because it excels across the board. It not only achieves a perfect ROC AUC score, indicating flawless class separation, but it also has the highest F1 score, demonstrating superior balanced performance. For most applications, this combination makes it the most robust and dependable choice.

It will be saved as: final_model_randomforest.sav

Deployment Steps

Load

- Load Model and Scaler: Use the pickle library to load both the saved GridSearchCV model (final_model_logisticRegression.sav) and the StandardScaler object (scaler_LR.sav). Both are essential because the model was trained on scaled data, and new inputs must be transformed identically.

Input Collection

- Collect User Inputs: Collect 27 numerical feature values from the user. These include physiological measurements (e.g., age, bp, al, su, etc.) and the one-hot encoded dummy variables for categorical features (e.g., rbc_normal, pc_normal, etc.). The order of these values must precisely match the order of the features used to train the model.

Preprocessing

- Scale the Input: Use the loaded StandardScaler (loaded_sc) to transform the collected user input. This step is critical for ensuring the data is in the correct format for the trained model to make an accurate prediction.

Prediction

- Make Prediction: Use the loaded model's (loaded_model) predict() function on the preprocessed (scaled) input to get the predicted outcome. The output will be a single value, either 0 (for 'no CKD') or 1 (for 'yes CKD').

Action

- Interpret Result: Interpret the predicted value to provide a clear result to the user. A prediction of 1 indicates the model believes the patient has a high probability of having CKD, while a prediction of 0 indicates a low probability. This information can be used to inform a diagnosis or recommend further medical evaluation.
