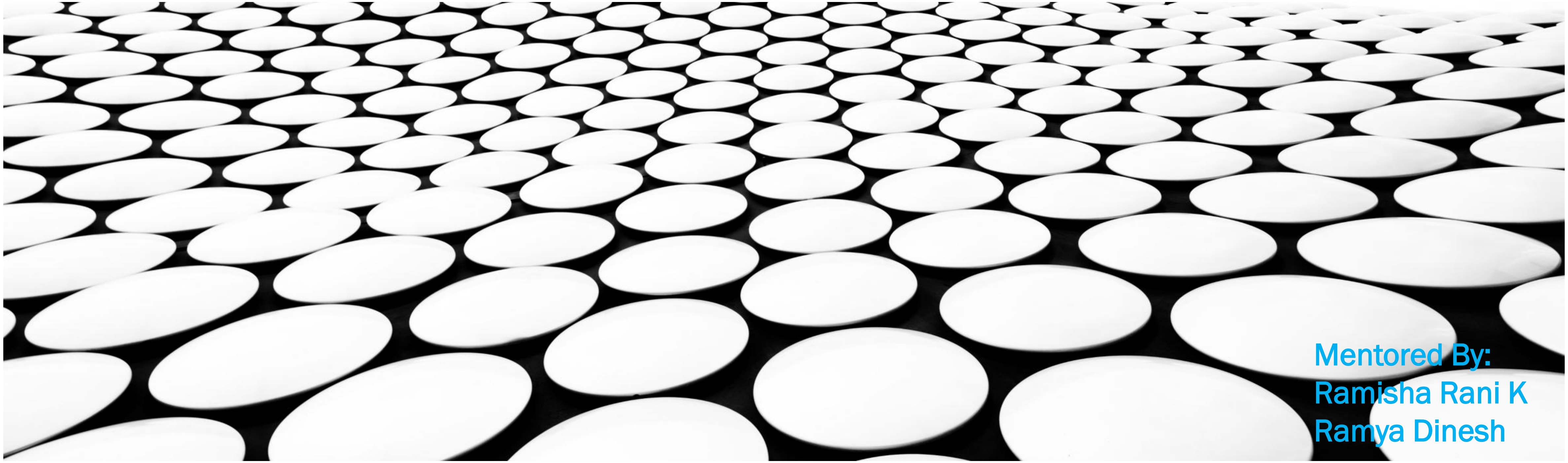
AI – PROJECT

-

AI-POWERED PREDICTION SYSTEM FOR CHRONIC KIDNEY DISEASE

MACHINE LEARNING - CLASSIFICATION ALGORITHM

AUTHOR: SUBRAMANI SURESH



Mentored By:
Ramisha Rani K
Ramya Dinesh

1. PROBLEM STATEMENT / REQUIREMENT

- The hospital management requested a predictive model to **detect** **Chronic Kidney Disease (CKD)** based on several patient parameters.
- A dataset was provided for model development.

2. DATASET

- File: CKD.csv
- Size: 399 rows × 25 columns

Columns include:

```
['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']
```

3. DOMAIN PREDICTION: MACHINE LEARNING

- Predicted Value: Yes/No (1 or 0)
- Input: Numerical & Categorical Data.

4. LEARNING PREDICTION: SUPERVISED LEARNING

- Requirement clear.
- Both input and output data are available

5. ALGORITHM PREDICTION: CLASSIFICATION

- Prediction is Yes/No → Classification Problem
- **Inputs**: 24 features
- **Output**: 1 target

CLASSIFICATION ALGORITHMS USED

1. Logistic Regression
2. Support Vector Classification (SVM)
3. Decision Tree Classification
4. **Random Forest Classification**

   Best Model → Weighted F1-score = 0.9917, ROC-AUC Score = 1.0
5. K-Nearest Neighbour Classification
6. Naive Bayes (GaussianNB)

6. DATA COLLECTION

- **Python Coding**

```
#importing the Libraies
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#data collection
dataset=pd.read_csv("CKD.csv")
print(dataset.columns)
print(dataset)
```

```
Index(['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'],
      dtype='object')
          age         bp  sg   al   su     rbc        pc         pcc  \
0    2.000000  76.459948   c  3.0  0.0  normal  abnormal  notpresent
```

7. DATA PREPROCESSING

- Categorical Data: Nominal - Converted into numbers using **One-Hot Encoding**.

Column	Original Values	Encoded
sg	a, b, c, d, e	1/0
rbc	normal, abnormal	1/0
pc	normal, abnormal	1/0
pcc	present, notpresent	1/0
ba	present, notpresent	1/0
htn	yes, no	1/0
dm	yes, no	1/0
cad	yes, no	1/0
appet	yes, poor	1/0
pe	yes, no	1/0
ane	yes, no	1/0
classification	yes, no	1/0

7. DATA PREPROCESSING

- Python Coding

```
#preprocess string to number  0 or 1
#Nominal: So one hot encoding
dataset=pd.get_dummies(dataset,drop_first=True).astype(int)
print(dataset.columns)
dataset
```

```
Index(['age', 'bp', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hrmo', 'pcv',
       'wc', 'rc', 'sg_b', 'sg_c', 'sg_d', 'sg_e', 'rbc_normal', 'pc_normal',
       'pcc_present', 'ba_present', 'htn_yes', 'dm_yes', 'cad_yes',
       'appet_yes', 'pe_yes', 'ane_yes', 'classification_yes'],
      dtype='object')
```

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
2	76	3	0	148	57	3	137	4	12	...	0	0	0	0	0	0	1	1	0	1
3	76	2	0	148	22	0	137	4	10	...	1	0	0	0	0	0	1	0	0	1
4	76	1	0	99	23	0	138	4	12	...	1	0	0	0	0	0	1	0	0	1

399 rows × 28 columns

7. DATA PREPROCESSING

Split input & output:

- X input: **independent=dataset[[**'age', 'bp', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hrmo', 'pcv','wc', 'rc', 'sg_b', 'sg_c', 'sg_d', 'sg_e', 'rbc_normal','pc_normal','pcc_present', 'ba_present', 'htn_yes', 'dm_yes', 'cad_yes','appet_yes', 'pe_yes', 'ane_yes’**]]**

- y output: **dependent=dataset**[['classification_yes']]

- Python Coding

```
#input output split
independent=dataset[['age', 'bp', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hrmo', 'pcv','wc',
dependent=dataset[['classification_yes']]
print(independent.shape)
print(dependent.shape)
```

```
(399, 27)
(399, 1)
```

8. TRAIN-TEST SPLIT

- **Training set**: 70%
- **Testing set**: 30%

- Python Coding

```
#Train Test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(independent, dependent, test_size=0.30,random_state=0)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(279, 27)
(120, 27)
(279, 1)
(120, 1)
```

9. MODEL CREATION

- Training done using 70% dataset (X_train, y_train) with the following algorithms.
- Hyperparameter tuning performed with **GridSearchCV**

Models Tested:

1. Logistic Regression
2. Support Vector Classification (SVM)
3. Decision Tree Classification
4. **Random Forest Classification**
5. K-Nearest Neighbour Classification
6. Naive Bayes (GaussianNB)

9. MODEL CREATION

1. LOGISTIC REGRESSION

- Python Coding

```
#Model creation
from sklearn.linear_model import LogisticRegression
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {'solver':['newton-cg', 'lbfgs', 'liblinear', 'saga'],
             'penalty':['l2']}
classifier = GridSearchCV(LogisticRegression(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

2. SUPPORT VECTOR CLASSIFICATION(SVM)

- Python Coding

```
#Model creation
from sklearn.svm import SVC
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {'kernel':['rbf','poly','sigmoid','linear'],
              'C':[0.1, 1, 10]}
classifier = GridSearchCV(SVC(probability=True),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

9. MODEL CREATION

3. DECISION TREE CLASSIFICATION

- Python Coding

```
#Model creation
from sklearn.tree import DecisionTreeClassifier

#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_features': [None, 'sqrt', 'log2'],
    'splitter': ['best', 'random']
}
classifier = GridSearchCV(DecisionTreeClassifier(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

4. RANDOM FOREST CLASSIFICATION

- Python Coding

```
#Model creation
from sklearn.ensemble import RandomForestClassifier
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {'criterion':['gini','entropy'],
              'max_features': ['auto','sqrt','log2'],
              'n_estimators':[10,100]}
classifier = GridSearchCV(RandomForestClassifier(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

9. MODEL CREATION

5. K-NEAREST NEIGHBOUR

- Python Coding

```
#Model creation
from sklearn.neighbors import KNeighborsClassifier
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],          # Different K values
    'weights': ['uniform', 'distance'],       # Uniform = equal weight, Di
    'metric': ['euclidean', 'manhattan', 'minkowski']  # Different distan
}
classifier = GridSearchCV(KNeighborsClassifier(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
```

6. NAIVE BAYES (GAUSSIANNB)

- Python Coding

```
#Model creation
from sklearn.naive_bayes import GaussianNB
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}
classifier = GridSearchCV(GaussianNB(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

10. EVALUATION METRICS

- Each model will be evaluated on the 30% test set (X_test, y_test) using the following metrics:

Confusion Matrix (Best Model – Random Forest):

True Positives (TP) = 45 False Negatives (FN) = 0

False Positives (FP) = 1 True Negatives (TN) = 74

Classification Report Metrics:

- **Precision:** Degree of Correctness 0 →0.98 1 → 1.00
- **Recall:** Degree of Completeness 0 → 1.00 1 → 0.99
- **F1-Score:** Balance between Precision and Recall. 0 → 0.99 1 → 0.99
- **Macro Average:** Average metric across all classes equally. F1→ **0.99**
- **Weighted Average:** (better when classes are imbalanced).

  Average metric weighted by class size. F1→ **0.9917**
- **Accuracy Score:** Overall proportion of correct predictions. F1→ **0.99**

- Python Coding

```
#Evaluation Matrix: Confusion_matrix
#-----------------------------------
re=classifier.cv_results_
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
#Evaluation Matrix: calculate P R F
from sklearn.metrics import classification_report
clf_report = classification_report(y_test, y_pred)
print(clf_report)
```

```
[[45  0]
 [ 1 74]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        45
           1       1.00      0.99      0.99        75

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120
```

10. EVALUATION METRICS

- Best Model: Random Forest
- Weighted F1-score: 0.9917
- ROC-AUC Score: 1.0
- Parameters: {criterion: 'entropy', max_features: 'log2', n_estimators: 100}

- Python Coding

```
# print best parameter after tuning
#print(classifier.best_params_)
re=classifier.cv_results_
from sklearn.metrics import f1_score
f1_weighted=f1_score(y_test,y_pred,average='weighted')
print("The weighted F1-score for best parameter {}:".format(classifier.best_params_),f1_weighted)
```

```
The weighted F1-score for best parameter {'criterion': 'entropy', 'max_features': 'log2', 'n_estim
ators': 100}: 0.9916844900066377
```

```
#Roc = ROC (Receiver Operating Characteristic) Curve
#AUC (Area Under the Curve)
from sklearn.metrics import roc_auc_score
print("ROC-AUC Score:",roc_auc_score(y_test,classifier.predict_proba(X_test)[:,1]))
table=pd.DataFrame.from_dict(re)
table
```

```
ROC-AUC Score : 1.0
```

11. MODEL COMPARISON

- Best Model: Random Forest

Algorithm	Best Parameter	Weighted F1-score	ROC-AUC Score
1. Logistic Regression	`{'penalty':'l2','solver':'liblinear'}`	0.9749	0.9979
2. Support Vector Classification (SVM)	`{'C':0.1,'kernel':'linear'}`	0.9916	0.9991
3. Decision Tree	`{'criterion':'entropy','max_features':None,'splitter':'random'}`	0.9750	0.9755
4. Random Forest	**`{'criterion':'entropy','max_features':'log2','n_estimators':100}`**	**0.9917**	**1.0**
5. K Nearest Neighbour	`{'metric':'manhattan','n_neighbors':9,'weights':'distance'}`	0.7863	0.8642
6. Naive Bayes (GaussianNB)	`{'var_smoothing':1e-09}`	0.9834	1.0

12. SAVE THE BEST MODEL

- **Random Forest** achieved the highest Weighted F1-score (0.9917) and ROC-AUC Score (1.0).
- Saved as the final deployment model.

- Python Coding

```
#save Best model
import pickle
filename="final_Sav_Model_RF.sav"
pickle.dump(classifier,open(filename,'wb'))
load_model=pickle.load(open("final_Sav_Model_RF.sav",'rb'))

#Check result
result=load_model.predict([[2,76,3,0,148,57,3,137,4,12,38,8408,4,0,1,0,0,1,0,0,0,0,0,0,1,1,0]])
result
```

```
C:\Anaconda3\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have va
omForestClassifier was fitted with feature names
  warnings.warn(
```

```
array([1])
```

13. DEPLOYMENT / IMPLEMENTATION

- Load the best Random Forest model.
- Accept user inputs to predict **Chronic Kidney Disease**: Yes/No.

- Python Coding

```
# Deployment
#pickle is library for save model
import pickle
#load the model from file.sav :r-read, b binary
load_model=pickle.load(open("final_Sav_Model_RF.sav","rb"))
#user input 1
result=load_model.predict([[17,60,0,0,114,50,1,135,4,14,51,7200,5,1,0,0,0,1,1,0,0,0,0,0,1,0,0]])
print("Chronic Kidney Disease: yes/No 1/0 =",result)
#user input 2
result=load_model.predict([[2,76,3,0,148,57,3,137,4,12,38,8408,4,0,1,0,0,1,0,0,0,0,0,0,1,1,0]])
print("Chronic Kidney Disease: yes/No 1/0 =",result)
```

```
Chronic Kidney Disease: yes/No 1/0 = [0]
Chronic Kidney Disease: yes/No 1/0 = [1]
```

14. CALL TO ACTION

- The trained model can now be used by hospital management to assist in early detection of Chronic Kidney Disease (CKD) and support medical decision-making.