**HealthCare**

**Introduction**

* In this assignment, I am building a Machine Learning model to predict the medical condition of patient. To build this model, I am using Kaggle dataset HealthCare\_Dataset and targeted column is Medical Condition.

**Dataset Information**

Each column provides specific information about the patient, their admission, and the healthcare services provided, making this dataset suitable for various data analysis and modeling tasks in the healthcare domain. Here's a brief explanation of each column in the dataset -

* **Name:** This column represents the name of the patient associated with the healthcare record.
* **Age:** The age of the patient at the time of admission, expressed in years.
* **Gender:** Indicates the gender of the patient, either "Male" or "Female."
* **Blood Type:** The patient's blood type, which can be one of the common blood types (e.g., "A+", "O-", etc.).
* **Medical Condition:** This column specifies the primary medical condition or diagnosis associated with the patient, such as "Diabetes," "Hypertension," "Asthma," and more.
* **Date of Admission:** The date on which the patient was admitted to the healthcare facility.
* **Doctor:** The name of the doctor responsible for the patient's care during their admission.
* **Hospital:** Identifies the healthcare facility or hospital where the patient was admitted.
* **Insurance Provider:** This column indicates the patient's insurance provider, which can be one of several options, including "Aetna," "Blue Cross," "Cigna," "UnitedHealthcare," and "Medicare."
* **Billing Amount:** The amount of money billed for the patient's healthcare services during their admission. This is expressed as a floating-point number.
* **Room Number:** The room number where the patient was accommodated during their admission.
* **Admission Type:** Specifies the type of admission, which can be "Emergency," "Elective," or "Urgent," reflecting the circumstances of the admission.
* **Discharge Date:** The date on which the patient was discharged from the healthcare facility, based on the admission date and a random number of days within a realistic range.
* **Medication:** Identifies a medication prescribed or administered to the patient during their admission. Examples include "Aspirin," "Ibuprofen," "Penicillin," "Paracetamol," and "Lipitor."
* **Test Results:** Describes the results of a medical test conducted during the patient's admission. Possible values include "Normal," "Abnormal," or "Inconclusive," indicating the outcome of the test.

**Contents of This Assignment**

1. Data Collection and Preprocessing
2. Exploratory Data Analysis
3. Model Selection And Training
4. Model Evaluation
5. Model Optimization

**1. Data Collection and Preprocessing**

* **Importing all necessary libraries**

import warnings

warnings.filterwarnings('ignore')

import pandas as pd

import numpy as np

from datetime import datetime

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from sklearn.model\_selection import GridSearchCV

* **Load the Dataset**

data = pd.read\_csv('healthcare\_dataset.csv')

* **Preprocessing**

data.isnull().sum() 🡪 Checking Missing Value in the Dataset

data.info() 🡪 Summary of DataFrame

data.describe() 🡪 Generates Descriptive Statistics of Dataset

data.shape 🡪 Show dimension of Dataset

data.columns 🡪 Print all Columns name

data.drop(columns = ['Name', 'Room Number'], axis=1) 🡪 Drop column from dataset

data.head() 🡪 Show top five rows of Dataset

data['Discharge Date'] = pd.to\_datetime(data['Discharge Date']) 🡪 Convert date column to datetime

**2. Exploratory Data Analysis**

* First filter numeric columns than plot boxplot for the numeric columns to check the outliers in the dataset. In our dataset, outliers is not present and also ploting the heatmap to check the correlation between features.
* For categorical column ploting the count plot and pie chart for each column to find out the percentage of each category in dataset.
* **Code for BoxPlot**

for column in numeric\_cols:

plt.figure(figsize=(8, 4))

sns.boxplot(x=data[column],palette='rainbow')

plt.title(f'Boxplot of {column}')

plt.show()

* **Code of HeatMap**

plt.figure(figsize=(10, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='rainbow', fmt=".2f", linewidths=0.5)

plt.title('Correlation Matrix')

* **Code for Count plot and Pie Chart**

for col in categorical\_cols:

plt.figure(figsize=(20,10))

fig, ax = plt.subplots(1, 2, figsize=(16, 8))

fig.suptitle('\* ' + col + ' \*', fontsize=18)

plt.style.use('seaborn')

ax[0].set\_title('Bar Plot')

data[col].value\_counts().plot(kind='bar', color=sns.color\_palette("rainbow"), ax=ax[0])

ax[1].set\_title('Pie Chart')

data[col].value\_counts().plot(kind='pie', autopct="%.2f%%", colors=sns.color\_palette("rainbow"),ax=ax[1])

plt.show()

* **Ordinal Encoding on Independent Column**

def encode\_features\_target\_guided(df,feature,target):

ordinal\_labels=df.groupby([feature])[target].mean().sort\_values().index

encode = {k:i for i,k in enumerate(ordinal\_labels,0)}

df[feature+'\_encoded']=df[feature].map(encode)

# Drop the original feature

df.drop(columns=[feature], inplace=True)

* **Label Encoding on Dependent Column**

from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder

encoder = LabelEncoder()

data['Medical Condition'] = encoder.fit\_transform(data['Medical Condition'])

* **MinMaxScaler to Scale the Dataset**

from sklearn.preprocessing import MinMaxScaler

scale = MinMaxScaler()

for col in num\_col:

data[[col]] = scale.fit\_transform(data[[col]])

**3. Model Selection and Training**

* **Split the Dataset into Train and Test**

X = data.drop(columns=['Medical Condition'])

y = data['Medical Condition']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

* **Train the Model**

1. **Random Forest**

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

**2. KNN**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train,y\_train)

**3. Decision Tree**

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt.fit(X\_train,y\_train)

**4. AdaBoost Classifier**

from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier

ada = AdaBoostClassifier()

ada.fit(X\_train,y\_train)

**5. Gradient Boosting Classifier**

gb = GradientBoostingClassifier()

gb.fit(X\_train,y\_train)

**6. SVM**

from sklearn.svm import SVC

sr = SVC()

sr.fit(X\_train, y\_train)

**4. Model Evaluation**

* For model evaluation, I am checking accuracy\_score, precision\_score, recall\_score and f1\_score.
* In this model, best accuracy is coming after train the random forest model on some hyperparameter and Classimfication metrices such that are:
* Accuracy: 0.7990990990990992
* Precision: 0.8168838473346955
* Recall: 0.7990990990990992
* F1\_score: 0.7990990990990992

**Code:**

accuracy = accuracy\_score(y\_test, y\_pred\_best)

precision = precision\_score(y\_test, y\_pred\_best,average='weighted')

recall = recall\_score(y\_test, y\_pred\_best,average='weighted')

f1\_Score = recall\_score(y\_test, y\_pred\_best,average='weighted')

print(f'Accuracy: {accuracy}')

print(f'Precision: {precision}')

print(f'Recall: {recall}')

print(f'F1\_score: {f1\_Score}')

**5. Model Optimization**

* Model Optimization is done to increase accuracy of model performance and here I am using gridsearchCv for model optimization for this we tune some parameter and than retrain the model on tuned parameter for better performance.

**Code:**

param\_grid = {

'n\_estimators': [100,200,250],

'max\_depth' : [10,15,20],

'min\_samples\_split' : [2,4,8],

'min\_samples\_leaf' : [2,4,6],

'max\_features': ['auto', 'sqrt', 'log2']

}

grid\_search = GridSearchCV(estimator=model, param\_grid=param\_grid, cv=5, n\_jobs=-1, verbose=2)

grid\_search.fit(X\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

**Conclusion**

In this healthcare dataset analysis, we utilized a Random Forest classifier to predict medical conditions based on various categorical features. After pre-processing the data through ordinal encoding and splitting it into training and testing sets, we trained the Random Forest model. The performance of the model was evaluated using key metrics: accuracy, precision, recall, and F1 score.

The model achieved an accuracy of **0.79**, a precision of **0.81**, a recall of **0.79**, and an F1 score of **0.79**. These results indicate that the Random Forest model is fairly effective at predicting medical conditions from the given features, demonstrating a balanced performance across different evaluation metrics.