Basic Details:

Name: Samala.Sudhakar
Email: om.sudhakar@gmail.com
Unified Mentor ID: UIMD10042529348
Project : Machine Learning Project

Project Information:

Title: Detect Heart Disease using patient data.

Objective:

Building a machine learning model that can predict if a patient has heart disease.

Dataset:

Attribute	Code given	Unit	Data type
age	Age	in years	Numeric
sex	Sex	0 = female, 1 = male	Binary
chest pain type	chest pain type	1 = typical angina, 2 = atypical angina, 3 = non-anginal pain, 4 = asymptomatic	Nominal
resting blood pressure	resting bp s	in mm Hg	Numeric
serum cholesterol	cholesterol	in mg/dl	Numeric
fasting blood sugar	fasting blood sugar	1 = sugar > 120mg/dL 0 = sugar < 120mg/dL	Binary
resting electrocardiogram results	resting ecg	0 = normal, 1 = ST-T wave abnormality (T wave inversions and/or ST elevation/depression of > 0.05 mV), 2 = Probable or Definite Left Ventricular hypertrophy by Estes' criteria	Nominal
maximum heart rate achieved	max heart rate	71–202	Numeric
exercise induced angina	exercise angina	0 = no, 1 = yes	Binary
oldpeak =ST	oldpeak	depression	Numeric
the slope of the peak exercise ST segment	ST slope	1 = upward 2 = flat, 3 = downward	Nominal
class	target	0 = Normal, 1 = Heart Disease	Binary

Project Link:

https://github.com/AIforeverything/UnifiedMentorInternshipProjects/blob/c86c2928100b9b567ee2361675a7f402cc307a20/Detect%20Heart%20Disease%20using%20patient%20data/Detect_Heart_Disease_using_patient_data.ipynb

https://github.com/AIforeverything/UnifiedMentorInternshipProjects/blob/c86c2928100b9b567ee2361675a7f402cc307a20/categorical/categorical_model.py

Code

Steps Followed:

Step-1: Initially I have created a library for building a categorical machine learning model and used this library for building model.

categorical_model.py

```
# ## Step-1: Common virtual environment was created and activated: myenv
# ## pip install virtualenv
# ## virtualenv myenv
# ## .\myenv\Scripts\activate.ps1

def greet(name):
    return f"good job {name}"

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sys
from pathlib import Path
import zipfile
import warnings
warnings.filterwarnings("ignore")

import sklearn
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import joblib
# import tensorflow as tf
# from tensorflow import keras
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import Dense

class categorical_Model:
    def __init__(self,model, target_column, df):
        """
        Initializes the categoricalTarget class.
```

```
    Parameters:
    model (str): The name of the model to be used.
    target_column (str): The name of the target column.
    df (pd.DataFrame): The DataFrame containing the data.
    """

    self.df = df
    self.target_column = target_column
    self.model = model

# Importing data into a dataframe from csv file in the directory
def readingData():
  #checking the directory for .csv files
    directory = Path('./')
    # List all CSV files
    for csv_file in directory.glob('*.csv'):
        print(csv_file.name)
    df= pd.read_csv(csv_file)
    return df

# # Data extraction from zipfile
def extractingZipFile(zipFilePath, extractTo):
    """
    Extracts the contents of a zip file to a specified directory.

    Parameters:
    zipFilePath (str): The path to the zip file.
    extractTo (str): The directory to extract the contents to.
    """
    with zipfile.ZipFile(zipFilePath, 'r') as zip_ref:
        zip_ref.extractall(extractTo)


# EDA (Exploratory Data Analysis)

# Checking missing values
def checkMissingValues(df):
    """
    Checks for missing values in the DataFrame
    Parameters:
    df (pd.DataFrame): The DataFrame to check for missing values.
    Returns:
    missing values
    """
    return df.isnull().sum()

# Removing duplicates
## function to check for duplicates and remove dupliates
def checkDuplicates(df):
    """
    Checks for duplicate rows in the DataFrame and removes them.
```

```
    Parameters:
    df (pd.DataFrame): The DataFrame to check for duplicates.

    Returns:
    pd.DataFrame: The DataFrame with duplicates removed.
    """
    duplicates = df.duplicated().sum()
    if duplicates > 0:
        df = df.drop_duplicates()
        print(f"Removed {duplicates} duplicate rows.")
    else:
        print("No duplicate rows found.")
    return df

#Function for all columns
def allColumns(df):
    return list(df.columns)

# Function for categorical columns
def catColumns(df):
    catCol=df.select_dtypes(include='object').columns
    return catCol

# Function for Non-categorical columns
def nonCatColumns(df):
    numeric_col=df.select_dtypes(include='number').columns
    return numeric_col

## function to check categorical columns and replacing them with numerical values
def checkCategoricalColumnsAndReplacingWithLE(df):
    """
    Checks for categorical columns in the DataFrame and replaces them with numerical values.

    Parameters:
    df (pd.DataFrame): The DataFrame to check for categorical columns.

    Returns:
    pd.DataFrame: The DataFrame with categorical columns replaced with numerical values.
    """
    categorical_columns = df.select_dtypes(include=['object']).columns
    print(f"Categorical columns: {categorical_columns}")

    for col in categorical_columns:
        print(f"col.unique(): {df[col].unique()}")
        print(f"col.value_counts(): {df[col].value_counts()}")
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
    return df

# function to standardize Non Categorical columns
```

```
def standardizeNonCategoricalColumns(df):
    minMax=MinMaxScaler()
    numeric_col=df.select_dtypes(include='number').columns
    df[numeric_col]=minMax.fit_transform(df[numeric_col])
    return df

## function to removing the missing values
def removeMissingValues(df):
    """
    Removes rows with missing values from the DataFrame.

    Parameters:
    df (pd.DataFrame): The DataFrame to remove missing values from.

    Returns:
    pd.DataFrame: The DataFrame with missing values removed.
    """
    df = df.dropna()
    return df

#function to print the correlation matrix respect to the target column
def printCorrelationMatrix(df, target_column):
    """
    Prints the correlation matrix of the DataFrame with respect to the target column.

    Parameters:
    df (pd.DataFrame): The DataFrame to print the correlation matrix for.
    target_column (str): The name of the target column.

    Returns:
    pd.DataFrame: The correlation matrix.
    """
    # print the correlation matrix with respect to the target column
    print(f"Correlation matrix with respect to {target_column}:")
    print(df.corr()[target_column].sort_values(ascending=False))
    corr_text=df.corr()[target_column].sort_values(ascending=False)
    # .to_string() provides a nicely formatted text version of the DataFrame.
    # This will produce a human-readable file.
    # If we want a machine-readable format instead, consider .to_csv("file.txt", sep='\t').
    with open('correlation.txt', 'w') as f:
        f.write(corr_text.to_string())
    corr = df.corr()
    plt.figure(figsize=(12, 8))
    sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')
    plt.title(f"Correlation Matrix with respect to {target_column}")
    plt.show()
    return corr

#checking missing values of each column
def missing_columns(df):
    return (df.isnull().sum())
```

```
#checking missing values of all columns
def missing_columns_total(df):
    return (df.isnull().sum().sum())

## function to split the data into X,y
def splitDataIntoXy(df, target_column):
    """
    Splits the DataFrame into X and y.
    retuns tuple
    """
    X = df.drop(target_column, axis=1)
    y = df[target_column]
    return X,y

## function to split the data into train and test
def splitData(X,y):
    """
    Splits the DataFrame into training and testing sets.
    Parameters:
    X,y
    Returns:
    tuple: The training and testing sets.
    """
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test

# function to train the model and compare the models and save the best model and the model report and the model performance
def trainModel(X_train, X_test, y_train, y_test):
    """
    Trains the model and compares the models and saves the best model and the model report and the model performance.

    Parameters:
    X_train (pd.DataFrame): The training data.
    X_test (pd.DataFrame): The testing data.
    y_train (pd.Series): The training labels.
    y_test (pd.Series): The testing labels.

    Returns:
    None
    """
    models = {
        "Logistic Regression": LogisticRegressionCV(max_iter=10000),
        "Decision Tree": DecisionTreeClassifier(),
        "RandomForest": RandomForestClassifier(min_samples_split=5),
        "Gradient Boosting": GradientBoostingClassifier(),
        "Naive Bayes" :GaussianNB(),
        "KNN" : KNeighborsClassifier(),
```

```
            "Support Vector Machines" : SVC(),
            "XGBoost": XGBClassifier()
        }

        best_model = None
        best_accuracy = 0

        for name, model in models.items():
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)

            print(f"{name} Accuracy: {accuracy:.4f}")

            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_model = model
                best_model_name = name

        print(f"Best Model: {best_model.__class__.__name__} with accuracy: {best_accuracy:.2f}")

        # Save the best model
        joblib.dump(best_model_name, f'{best_model_name}.pkl')

        # Save the classification report
        report = classification_report(y_test, y_pred)
        with open('classification_report.txt', 'w') as f:
            f.write(f"Model: {best_model_name} \n\n")
            f.write(report)

        # Save the confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        np.savetxt('confusion_matrix.txt', cm, fmt='%d')

    # function to load the model
    def loadModel(model_path):
        """
        Loads the model from the specified path.

        Parameters:
        model_path (str): The path to the model.

        Returns:
        model: The loaded model.
        """
        model = joblib.load(model_path)
        return model

# making an object of the class to use the functions
def main():
    # Unzip the file
```

```
file= categorical_Model.extractingZipFile('./', "./")

# Reading the data
df = categorical_Model.readingData()

# Checking for missing values
missing_values = categorical_Model.checkMissingValues(df)
print(f"Missing values: {missing_values}")

# Checking for duplicates
df = categorical_Model.checkDuplicates(df)

# Checking for categorical columns
df = categorical_Model.checkCategoricalColumns(df)

# Removing missing values
df = categorical_Model.removeMissingValues(df)

# Choosing the target column
target_column = input("Enter the target column name: ")
if target_column not in df.columns:
    print(f"Target column '{target_column}' not found in DataFrame.")
else:
    print(f"Target column '{target_column}' found in DataFrame.")

# Printing the correlation matrix
corr_matrix = categorical_Model.printCorrelationMatrix(df, target_column)

# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = categorical_Model.splitData(df, target_column)

# Training the model and saving the best model
categorical_Model.trainModel(X_train, X_test, y_train, y_test)
```

Step-2 : Code for model:

```
# In[1]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler


# In[2]:


# making a path to get the modules from categorical library
import os
os.chdir('..')
```

In[3]:

```
from categorical.categorical_model import categorical_Model
```

In[4]:

```
categorical_Model.extractingZipFile("./Detect Heart Disease using patient data/heart_disease.zip",'./Detect Heart Disease using patient data/')
```

In[5]:

```
df=pd.read_csv("./Detect Heart Disease using patient data/Heart Disease/dataset.csv")
df.head()
```

EDA

In[6]:

```
df.info()
```

Removing duplicates

In[7]:

```
categorical_Model.checkDuplicates(df)
```

Checking and Removing missing values

In[8]:

```
# Checking missing values for each column
print(categorical_Model.missing_columns(df))
#checking missing values of all columns
print(categorical_Model.missing_columns_total(df))
df.dropna(inplace=True)
df.head()
```

In[9]:

```
def showingUnique(x):
    return x.unique()
```

In[10]:

```
c=list(df.columns)
for i in c:
    if df.dtypes[i]=='object':
        print(i,showingUnique(df[i]))
```

Splitting into X and y before standardize X

In[11]:

```
X,y=categorical_Model.splitDataIntoXy(df,"target")
```

In[12]:

```
df["target"].unique()
```

Standardizing the numerical columns of X

In[13]:

```
categorical_Model.standardizeNonCategoricalColumns(X)
```

converting categorical columns to numerical of X

In[14]:

```
categorical_Model.checkCategoricalColumnsAndReplacingWithLE(X)
```

In[15]:

```
X.info()
```

In[16]:

```
y.info()
```

In[17]:

```
y.unique()
```

In[18]:

X_train, X_test, y_train, y_test=categorical_Model.splitData(X,y)

In[19]:

categorical_Model.trainModel(X_train, X_test, y_train, y_test)

Model Outcomes

Different models are built using the dataset and found

```
categorical_Model.trainModel(X_train, X_test, y_train, y_test)
[19]                                                                    Python

...  Logistic Regression Accuracy: 0.8403
     Decision Tree Accuracy: 0.8824
     RandomForest Accuracy: 0.9538
     Gradient Boosting Accuracy: 0.9160
     Naive Bayes Accuracy: 0.8571
     KNN Accuracy: 0.8739
     Support Vector Machines Accuracy: 0.8824
     XGBoost Accuracy: 0.9286
     Best Model: RandomForestClassifier with accuracy: 0.95
```

Random forest model gave more accuracy.

classification_report :

Model: RandomForest

	precision	recall	f1-score	support
0	0.93	0.91	0.92	107
1	0.93	0.95	0.94	131
accuracy			0.93	238
macro avg	0.93	0.93	0.93	238
weighted avg	0.93	0.93	0.93	238