Basic Details:

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

Project Information:

Title: Detect Lung Cancer using patient diagnosis data

Objective
Build a system that can predict the survival of a patient given details of the patient. Explore the data to understand the features and figure out an approach.

Dataset
This dataset contains data about lung cancer Mortality and is a comprehensive collection of patient information, specifically focused on individuals diagnosed with cancer.
Description of columns:
●id: A unique identifier for each patient in the dataset.
●age: The age of the patient at the time of diagnosis.
●gender: The gender of the patient (e.g., male, female).
●country: The country or region where the patient resides.
●diagnosis_date: The date on which the patient was diagnosed with lung cancer.
●cancer_stage: The stage of lung cancer at the time of diagnosis (e.g., Stage I, Stage II, Stage III, Stage IV).
●family_history: Indicates whether there is a family history of cancer (e.g., yes, no).
●smoking_status: The smoking status of the patient (e.g., current smoker, former smoker, never smoked, passive smoker).
●bmi: The Body Mass Index of the patient at the time of diagnosis.
●cholesterol_level: The cholesterol level of the patient (value).
●hypertension: Indicates whether the patient has hypertension (high blood pressure) (e.g., yes, no).
●asthma: Indicates whether the patient has asthma (e.g., yes, no).
●cirrhosis: Indicates whether the patient has cirrhosis of the liver (e.g., yes, no).
●other_cancer: Indicates whether the patient has had any other type of cancer in addition to the primary diagnosis (e.g., yes, no).
●treatment_type: The type of treatment the patient received (e.g., surgery, chemotherapy, radiation, combined).
●end_treatment_date: The date on which the patient completed their cancer treatment or died.
survived: Indicates whether the patient survived (e.g., yes, no).
Project Link:
https://github.com/AIforeverything/UnifiedMentorInternshipProjects/blob/c86c2928100b9b567ee2361675a7f402cc307a20/DetectLungCancerUsingPatientDiagnosisData/lungCancer.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:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# In[4]:


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


# In[5]:


from categorical.categorical_model import categorical_Model


# In[6]:
```

```
categorical_Model.extractingZipFile("./DetectLungCancerUsingPatientDiagnosisData/lung_c
ancer.zip",'./DetectLungCancerUsingPatientDiagnosisData/')


# In[7]:


df=pd.read_csv("./DetectLungCancerUsingPatientDiagnosisData/Lung
Cancer/dataset_med.csv")
df.head()


# ## EDA

# ## Following step by stem and analysing and cleaning columns
# #### Step-1 : df["country] is not useful for model buiding so dropping it

# In[8]:


df.drop(['country'],axis=1,inplace=True)
df.head()


# In[9]:


df.info()


# ### Feature Engineering on the columns diagnosis_date and end_treatment_date

# In[10]:


df["end_treatment_date"]=pd.to_datetime(df["end_treatment_date"])
df["diagnosis_date"]=pd.to_datetime(df["diagnosis_date"])
df.info()


# In[11]:


df["treatment_duration"]=df["end_treatment_date"]-df["diagnosis_date"]
df.head()


# ### extracting days from the duration and making into a fraction of a year

# In[12]:
```

```
df["treatment_duration"]=df["treatment_duration"].astype(str)
df["treatment_duration"]=df["treatment_duration"].str.extract(r"(\d+)").astype(int)
df.head()


# In[13]:


df["treatment_duration_scaled"]=df["treatment_duration"]/(365.0)
df.head()


# In[14]:


df.drop(["diagnosis_date","end_treatment_date","treatment_duration"],axis=1,inplace=True)
df.head()


# In[15]:


df.info()


# In[16]:


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


# In[17]:


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


# ### converting categorical columns to numerical

# In[18]:


categorical_Model.checkCategoricalColumns(df)
```

```
# #### Removing Id column

# In[19]:


df.drop(["id"],axis=1,inplace=True)
df.head()


# ### Removing duplicates

# In[20]:


categorical_Model.checkDuplicates(df)


# ### Removing missing values

# In[21]:


# Checking missing values for each column
print(categorical_Model.missing_columns(df))


# In[22]:


#checking missing values of all columns
print(categorical_Model.missing_columns_total(df))


# In[23]:


df.dropna(inplace=True)


# In[24]:


df.head()


# In[25]:


from sklearn.preprocessing import MinMaxScaler
```

```
minMax=MinMaxScaler()
df["age_scaled"]=minMax.fit_transform(df[["age"]])
df["bmi_scaled"]=minMax.fit_transform(df[["bmi"]])
df["cholesterol_level_scaled"]=minMax.fit_transform(df[["cholesterol_level"]])
df.head()


# In[26]:


df.drop(["age","bmi","cholesterol_level"],axis=1,inplace=True)


# In[27]:


categorical_Model.printCorrelationMatrix(df,"survived")


# In[28]:


X_train, X_test, y_train, y_test=categorical_Model.splitData(df,"survived")


# In[29]:
categorical_Model.trainModel(X_train, X_test, y_train, y_test)
```

Model Outcomes

Different models are built using the dataset and found

```
Logistic Regression Accuracy: 0.7789
Decision Tree Accuracy: 0.6411
Naive Bayes Accuracy: 0.7789
KNN Accuracy: 0.7377
XGBoost Accuracy: 0.7789
Best Model: LogisticRegressionCV with accuracy: 0.78
```

classification_report :

Model: Logistic Regression

	precision	recall	f1-score	support
0	0.78	1.00	0.88	138639
1	0.40	0.00	0.00	39361
accuracy			0.78	178000
macro avg	0.59	0.50	0.44	178000
weighted avg	0.70	0.78	0.68	178000