**Basic Details:**

**Name:** Samala.Sudhakar

**Email:** [om.sudhakar@gmail.com](mailto: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\_cancer.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]:

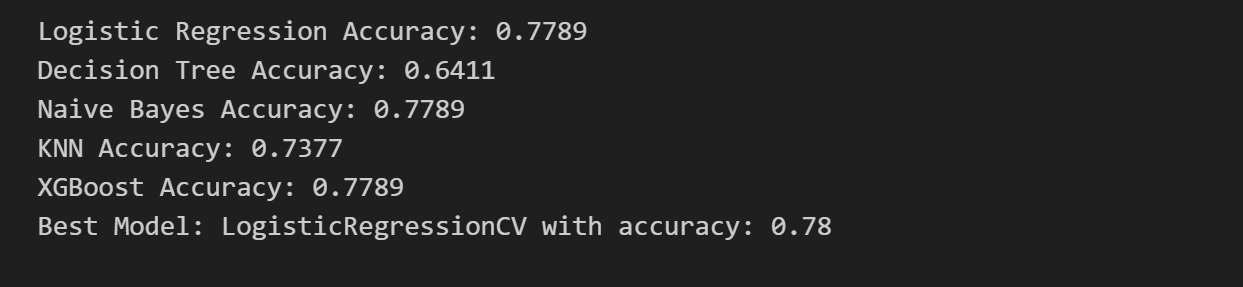
X\_train, X\_test, y\_train, y\_test=categorical\_Model.splitData(df,"survived")

# In[29]:

categorical\_Model.trainModel(X\_train, X\_test, y\_train, y\_test)

**Model Outcomes**

Different models are built using the dataset and found



**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