University of California Irvine – Heart Disease Dataset

This paper uses the UCI heart disease dataset to illustrate various techniques and facets in an AI project.

This document will be regularly updated as new approaches are uncovered.

Data Definition

The original database had 76 attributes. However, 14 attributes were eventually made available, in particular the Cleveland database. The following definitions have been provided:

Attribute	Type	Measurement
age	Integer	Years
Sex	Categorical	0: female 1: male
Cp (chest pain)	Categorical	1: typical angina 2: atypical angina 3: non-anginal pain 4: asymptomatic
trestbps (resting blood pressure (on admission to the hospital)	Integer	mm Hg
Chol (Serum cholesterol)	Integer	mg/dl
Fbs (fasting blood sugar > 120 mg/dl)	Categorical	0: False 1: True
Restecg	Categorical	0: normal 1: having ST-T wave abnormality 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
Thalach (maximum heart rate achieved)	Integer	
Exang (exercise induced angina)	Categorical	0: No 1: Yes
Oldpeak (ST depression induced by exercise relative to rest)	Integer	
Slope (the slope of the peak exercise ST segment)	Categorical	1: upsloping 2: flat 3: down sloping
Ca (number of major vessels (0-3) coloured by fluoroscopy)	Integer	
Thal	Categorical	3: normal 6: fixed defect 7: reversible defect
Num (Number of blocked vessels with blockage > 50%)	Integer	0: No blockage > 50% 1 to 4: Number of blockage's > 50%

Problem Statement

Use the above data to determine if the subject will have one or more blockages greater than 50%. Accordingly, the last attribute ***Num (Number of blocked vessels)*** will be the dependant variable – to be determined based on the first thirteen attributes which are deemed to be features.

It was felt that as a first pass it would perhaps be best to frame this as a binary classification problem i.e.

0: *patient has NO arteries > 50% blockage*

1: patient has one or more arteries > 50% blockage

Accessing the Data

A Jupyter Python notebook with various popular libraries in the python eco-system has been used for this analysis.

UCI have provided a package to access the database. The Python code is displayed below:

```
#
## Import various packages that will be used in the analysis and building of the model
#
# PKGUTIL is a utilities package.
import pkgutil

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

#
## ACCESSING THE DATA (provided by UCI)
# The package UCIMLREPO is required to access the UCI repository
if pkgutil.find_loader('ucimlrepo') is None:
# install package for UCI ML repository if not already installed
    !pip3 install -U ucimlrepo

from ucimlrepo import fetch_ucirepo

# fetch dataset
heart_disease = fetch_ucirepo(id=45)

# Read data (as pandas dataframes)
X = heart_disease.data.features
y = heart_disease.data.targets
```

This creates two data frames:

X which has the features

y which has the labels – in this case, number of blockages

Exploratory Data Analysis

Data read into the Pandas data frames was analysed as illustrated below:

```
#
## EXPLORATORY DATA ANALYSIS
#
# Number of entries
print('Number of Entries')
print(f'There are {X.shape[0]} entries with {X.shape[1]} features')
print(f'There are corresponding {y.shape[0]} labels')

# Data Set Feature Types
print('\n\nFeature Details\n')
print( X.info() )

#Display Statistical Summary
print('\n\nStatistical Summary\n')
print(X.describe(include='all'))

#Display first five records
print('\n\nFirst Few Records from the Feature Dataset')
print(X.head())

#Analysis of Categorical Values
print('\n\nCategorical Values of Various Fields')
print(f'Sex: {X.sex.unique()}' )
print(f'chest pain: {X.cp.unique()}' )
print(f'Fasting Blood Sugar: {X.fbs.unique()}' )
print(f'Rest ecg: {X.restecg.unique()}' )
print(f'slope: {X.slope.unique()}' )
print(f'thal: {X.thal.unique()}' )
print(f'exang: {X.exang.unique()}' )

#Number of Rows with Null Values
print(f'\n\nRows with null values: {X.isnull().any(axis=1).sum()}')

#Number of unique classes (blockages)
print(f'\n\nNumber of blockages: {y.num.unique()}')
```

This gave the following results:

```
Number of Entries
There are 303 entries with 13 features
There are corresponding 303 labels


Feature Details

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        299 non-null    float64
 12  thal      301 non-null    float64
dtypes: float64(3), int64(10)
memory usage: 30.9 KB
None

    Statistical Summary

              age         sex          cp    trestbps        chol         fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean    54.438944    0.679868    3.158416  131.689769  246.693069    0.148515
std      9.038662    0.467299    0.960126   17.599748   51.776918    0.356198
min     29.000000    0.000000    1.000000   94.000000  126.000000    0.000000
25%     48.000000    0.000000    3.000000  120.000000  211.000000    0.000000
50%     56.000000    1.000000    3.000000  130.000000  241.000000    0.000000
75%     61.000000    1.000000    4.000000  140.000000  275.000000    0.000000
max     77.000000    1.000000    4.000000  200.000000  564.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  299.000000
mean     0.990099  149.607261    0.326733    1.039604    1.600660    0.672241
std      0.994971   22.875003    0.469794    1.161075    0.616226    0.937438
min      0.000000   71.000000    0.000000    0.000000    1.000000    0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
50%      1.000000  153.000000    0.000000    0.800000    2.000000    0.000000
75%      2.000000  166.000000    1.000000    1.600000    2.000000    1.000000
max      2.000000  202.000000    1.000000    6.200000    3.000000    3.000000

             thal
count  301.000000
mean     4.734219
std      1.939706
min      3.000000
25%      3.000000
50%      3.000000
75%      7.000000
max      7.000000
```

There are a total of 303 patient records. The attribute ***ca (Chest angina)*** *has four missing entries and the attribute* ***thal*** has two missing entries.

```
First Few Records from the Feature Dataset
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

    ca  thal
0  0.0   6.0
1  3.0   3.0
2  2.0   7.0
3  0.0   3.0
4  0.0   3.0


Categorical Values of Various Fields
Sex: [1 0]
chest pain: [1 4 3 2]
Fasting Blood Sugar: [1 0]
Rest ecg: [2 0 1]
slope: [3 2 1]
thal: [ 6.  3.  7. nan]
exang: [0 1]


Rows with null values: 6


Number of blockages: [0 2 1 3 4]
```

Handling Missing Values

From the above analysis we note that 6 patient records have missing data (*Rows with some null value)*. It was felt that it is best to ignore these records. Accordingly, these records were dropped.

```
## HANDLING MISSING VALUES
# First Recombine features and labels. This will enable us to Drop rows with missing values.
#
X.reset_index(drop=True, inplace=True)
y.reset_index(drop=True, inplace=True)
result = pd.concat([X , y], axis = 1)
result = result.dropna()

#  Get back Features and labels into seperate data frames
features = result[['age','sex' , 'cp' , 'trestbps' , 'chol' , 'fbs' , 'restecg' , 'thalach' , 'exang' , 'oldpeak' ,
                   'slope' , 'ca' ,'thal']]
labels = result[['num']]
```

Check Any dependencies between features

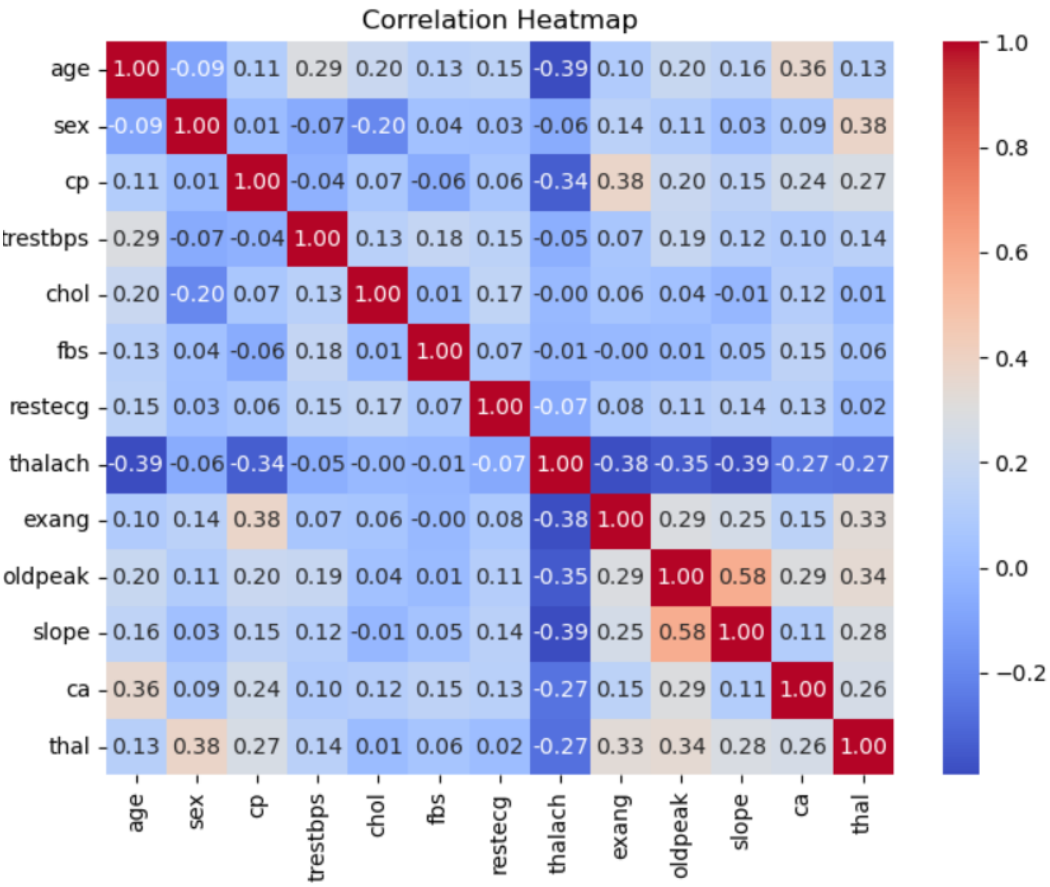
```
#
## Check for Any Correlation between the features
#
corr_matrix = features.corr()
#Set Size
plt.figure(figsize = (8,6))

#Plot the Heatmap
sns.heatmap( corr_matrix, annot = True, cmap = 'coolwarm', fmt = ".2f")

plt.title('Correlation Heatmap')

plt.show()
```

The above code snippet generates a heatmap as seen below:



From the above it is quite evident that the features are independent and do not exhibit any strong collinearity. Accordingly, all of the attributes have been considered in the model.

Model Implementation

Train/Test Split

```
#
## MODEL IMPLEMENTATION
#
# Split into Train/Test Data sets
from sklearn.model_selection import train_test_split


(train_X  , test_X , train_y , test_y) = train_test_split(features, np.ravel(labels) ,
                                                          test_size = 0.20,random_state = 42,shuffle = True)
```

The data set was split into a training and test data set in a 80/20 split.

The random seed ‘42’ ensures that the results are consistent and reproducible.

Also note that 1-d vector of labels has been converted into an array using the np.ravel() function.

Data Normalization

The exploratory data analysis reveals that different features have different measurement scales. Accordingly, all of them have been normalized using the popular MIN/MAX scaler which has the effect of scaling all features to [0,1].

Note that the Normalization is done ***after*** splitting separately on the Train/Test data set.

```
#
## DATA NORMALIZATION
#
column_names = [i for i in features.columns.values]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
train_X[column_names] = scaler.fit_transform(train_X[column_names])
test_X[column_names] = scaler.transform(test_X[column_names])
```

Implementing Logistics Regression

Logistics Regression is a popular choice for classification problems. Accordingly, this was chosen as the initial predictive model.

```
]: #
   ##LOGISTICS REGRESSION
   #
   from sklearn.linear_model import LogisticRegression
   from sklearn.metrics import accuracy_score ## To check accuracy

   # Replace labels as 0,1. All values greater than 0 are converted to 1. This then becomes a binary classification problem!
   test_y[test_y > 0] = 1
   train_y[train_y > 0] = 1

   lrModel = LogisticRegression(random_state=42 )
   lrModel.fit(train_X, train_y)
   lrPredictions = lrModel.predict(test_X)
   lrAccuracy = accuracy_score(test_y , lrPredictions )
   print(f"The accuracy score for LR Model is: {lrAccuracy}")

   num_classes = len(lrModel.classes_)
   print(f"Number of classes: {num_classes}")
```

We have converted the target to 0/1. This then becomes a binary classification problem:

0: No blockages

1: One or more blockages > 50%

```
The accuracy score for LR Model is: 0.9
Number of classes: 2
```

This simple model has an accuracy of 90%!