Heart Rate Modelling

December 16, 2021

Model building to predict heart disease using logistic Regression

[1]:
```
#Import modules
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss
from sklearn.metrics import jaccard_score
from scipy import stats
%matplotlib inline
import matplotlib.pyplot as plt
```

[2]:
```
#Load Data
heart_df=pd.read_csv(r'C:\Users\sanus\Documents\Programming\Python\Python data␣
↪analysis portfolio\Data\heart.csv')
heart_df.head()
```

[2]:
```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

[3]:
```
#Explore Data types
heart_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

[4]:
```
#optional: seperate categorical variables from numerical variables
#def get_numerical_var(dframe):
#    lis=[]
#    for col in heart_df:
#        if heart_df[col].dtype!=object:
#            header=''+col
#            lis.append(header)
#    return lis

#def get_cat_var(dframe):
#    lis=[]
#   for col in heart_df:
#        if heart_df[col].dtype==object:
#           header=''+col
#           lis.append(header)
#   return lis

#print(get_cat_var(heart_df))
```

[5]:
```
#Convert categorical independent variables to dummy variables
le = LabelEncoder()
heart_df['Sex']=le.fit_transform(heart_df['Sex'])
heart_df['ChestPainType']=le.fit_transform(heart_df['ChestPainType'])
heart_df['RestingECG']=le.fit_transform(heart_df['RestingECG'])
heart_df['ExerciseAngina']=le.fit_transform(heart_df['ExerciseAngina'])
```

```
heart_df['ST_Slope']=le.fit_transform(heart_df['ST_Slope'])
```

[6]:
```
#Visualize using boxplots
# fig=plt.figure()
# ax0=fig.add_subplot(1,2,1)
# heart_df.plot(kind='box',
#                  figsize=(30,10),
#                  ax=ax0)
# ax0.set_title('Boc plots ')
# plt.show
```

[7]:
```
#Calculate pearson coefficient-optional
heart_df.corr()
```

[7]:
```
                     Age       Sex  ChestPainType  RestingBP  Cholesterol  \
Age             1.000000  0.055750      -0.077150   0.254399    -0.095282
Sex             0.055750  1.000000      -0.126559   0.005133    -0.200092
ChestPainType  -0.077150 -0.126559       1.000000  -0.020647     0.067880
RestingBP       0.254399  0.005133      -0.020647   1.000000     0.100893
Cholesterol    -0.095282 -0.200092       0.067880   0.100893     1.000000
FastingBS       0.198039  0.120076      -0.073151   0.070193    -0.260974
RestingECG     -0.007484  0.071552      -0.072537   0.022656    -0.196544
MaxHR          -0.382045 -0.189186       0.289123  -0.112135     0.235792
ExerciseAngina  0.215793  0.190664      -0.354727   0.155101    -0.034166
Oldpeak         0.258612  0.105734      -0.177377   0.164803     0.050148
ST_Slope       -0.268264 -0.150693       0.213521  -0.075162     0.111471
HeartDisease    0.282039  0.305445      -0.386828   0.107589    -0.232741

                FastingBS  RestingECG     MaxHR  ExerciseAngina   Oldpeak  \
Age              0.198039   -0.007484 -0.382045        0.215793  0.258612
Sex              0.120076    0.071552 -0.189186        0.190664  0.105734
ChestPainType   -0.073151   -0.072537  0.289123       -0.354727 -0.177377
RestingBP        0.070193    0.022656 -0.112135        0.155101  0.164803
Cholesterol     -0.260974   -0.196544  0.235792       -0.034166  0.050148
FastingBS        1.000000    0.087050 -0.131438        0.060451  0.052698
RestingECG       0.087050    1.000000 -0.179276        0.077500 -0.020438
MaxHR           -0.131438   -0.179276  1.000000       -0.370425 -0.160691
ExerciseAngina   0.060451    0.077500 -0.370425        1.000000  0.408752
Oldpeak          0.052698   -0.020438 -0.160691        0.408752  1.000000
ST_Slope        -0.175774   -0.006778  0.343419       -0.428706 -0.501921
HeartDisease     0.267291    0.057384 -0.400421        0.494282  0.403951

                ST_Slope  HeartDisease
Age            -0.268264      0.282039
Sex            -0.150693      0.305445
ChestPainType   0.213521     -0.386828
RestingBP      -0.075162      0.107589
```

```
Cholesterol     0.111471    -0.232741
FastingBS      -0.175774     0.267291
RestingECG     -0.006778     0.057384
MaxHR           0.343419    -0.400421
ExerciseAngina -0.428706     0.494282
Oldpeak        -0.501921     0.403951
ST_Slope        1.000000    -0.558771
HeartDisease   -0.558771     1.000000
```

[8]:
```
#Calculate p-value
for col in heart_df.columns:
    pearson_coef, p_value = stats.pearsonr(heart_df[col],␣
 ↪heart_df['HeartDisease'])
    print("For", col, "The Pearson Correlation Coefficient is", pearson_coef, "␣
 ↪with a P-value of P =", p_value)
```

```
For Age The Pearson Correlation Coefficient is 0.28203850581899736  with a
P-value of P = 3.007953240047123e-18
For Sex The Pearson Correlation Coefficient is 0.30544491596314066  with a
P-value of P = 2.821897823681047e-21
For ChestPainType The Pearson Correlation Coefficient is -0.38682769426256447
with a P-value of P = 3.8887950200777145e-34
For RestingBP The Pearson Correlation Coefficient is 0.10758898037140399  with a
P-value of P = 0.0010953145851714478
For Cholesterol The Pearson Correlation Coefficient is -0.2327406389270114  with
a P-value of P = 9.308308883525767e-13
For FastingBS The Pearson Correlation Coefficient is 0.267291186110298  with a
P-value of P = 1.7535980103286795e-16
For RestingECG The Pearson Correlation Coefficient is 0.05738435701345111  with
a P-value of P = 0.08225947154724081
For MaxHR The Pearson Correlation Coefficient is -0.4004207694631902  with a
P-value of P = 1.137785984026953e-36
For ExerciseAngina The Pearson Correlation Coefficient is 0.49428199182426846
with a P-value of P = 1.0130182683908042e-57
For Oldpeak The Pearson Correlation Coefficient is 0.4039507220628864  with a
P-value of P = 2.3907724240568936e-37
For ST_Slope The Pearson Correlation Coefficient is -0.5587707148497059  with a
P-value of P = 1.6715991289946643e-76
For HeartDisease The Pearson Correlation Coefficient is 0.9999999999999998  with
a P-value of P = 0.0
```

[9]:
```
for col in heart_df.columns:
    strong_corr=''
    if p_value <0.001:
```

```
        strong_corr + ','+ col
        print(strong_corr)
```

[10]:
```
#The p_value for all variables show their statistical relationship is␣
↪significant
```

[11]:
```
#Define X and convert to numpy array
X = np.asarray(heart_df[['Age', 'Sex', 'ChestPainType', 'RestingBP',␣
↪'Cholesterol', 'FastingBS',␣
↪'RestingECG','MaxHR','ExerciseAngina','Oldpeak','ST_Slope']])
X[0:5]
```

[11]:
```
array([[ 40. ,   1. ,   1. , 140. , 289. ,   0. ,   1. , 172. ,   0. ,
          0. ,   2. ],
       [ 49. ,   0. ,   2. , 160. , 180. ,   0. ,   1. , 156. ,   0. ,
          1. ,   1. ],
       [ 37. ,   1. ,   1. , 130. , 283. ,   0. ,   2. ,  98. ,   0. ,
          0. ,   2. ],
       [ 48. ,   0. ,   0. , 138. , 214. ,   0. ,   1. , 108. ,   1. ,
          1.5,   1. ],
       [ 54. ,   1. ,   2. , 150. , 195. ,   0. ,   1. , 122. ,   0. ,
          0. ,   2. ]])
```

[12]:
```
#Define Y and convert to numpy array
y = np.asarray(heart_df['HeartDisease'])
y [0:5]
```

[12]:
```
array([0, 1, 0, 1, 0], dtype=int64)
```

[13]:
```
#Normalize
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

[13]:
```
array([[-1.4331398 ,  0.51595242,  0.22903206,  0.41090889,  0.82507026,
        -0.55134134,  0.01725451,  1.38292822, -0.8235563 , -0.83243239,
```

```
  1.05211381],
[-0.47848359, -1.93816322,  1.27505906,  1.49175234, -0.17196105,
 -0.55134134,  0.01725451,  0.75415714, -0.8235563 ,  0.10566353,
 -0.59607813],
[-1.75135854,  0.51595242,  0.22903206, -0.12951283,  0.7701878 ,
 -0.55134134,  1.60121899, -1.52513802, -0.8235563 , -0.83243239,
  1.05211381],
[-0.5845565 , -1.93816322, -0.81699495,  0.30282455,  0.13903954,
 -0.55134134,  0.01725451, -1.13215609,  1.21424608,  0.57471149,
 -0.59607813],
[ 0.05188098,  0.51595242,  1.27505906,  0.95133062, -0.0347549 ,
 -0.55134134,  0.01725451, -0.5819814 , -0.8235563 , -0.83243239,
  1.05211381]])
```

Train and Predict

[14]:
```
#Train Model
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2,␣
 ↪random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (734, 11) (734,)
Test set: (184, 11) (184,)
```

[15]:
```
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

[15]:
```
LogisticRegression(C=0.01, solver='liblinear')
```

[16]:
```
#Predict
yhat = LR.predict(X_test)
yhat
```

[16]:
```
array([0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 1], dtype=int64)
```

[17]:
```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

[17]:
```
array([[0.5156815 , 0.4843185 ],
       [0.21000747, 0.78999253],
```

```
[0.56400433, 0.43599567],
[0.75538125, 0.24461875],
[0.67403947, 0.32596053],
[0.18052657, 0.81947343],
[0.21522349, 0.78477651],
[0.38227614, 0.61772386],
[0.54362217, 0.45637783],
[0.67546032, 0.32453968],
[0.89131343, 0.10868657],
[0.51782803, 0.48217197],
[0.80825933, 0.19174067],
[0.77862184, 0.22137816],
[0.64098133, 0.35901867],
[0.07528364, 0.92471636],
[0.84393956, 0.15606044],
[0.79067292, 0.20932708],
[0.83189162, 0.16810838],
[0.25299   , 0.74701   ],
[0.19278601, 0.80721399],
[0.71229155, 0.28770845],
[0.80218114, 0.19781886],
[0.78344398, 0.21655602],
[0.08958179, 0.91041821],
[0.26834287, 0.73165713],
[0.89460813, 0.10539187],
[0.50407228, 0.49592772],
[0.20848928, 0.79151072],
[0.54787334, 0.45212666],
[0.1516258 , 0.8483742 ],
[0.24292488, 0.75707512],
[0.05299495, 0.94700505],
[0.21291484, 0.78708516],
[0.44016292, 0.55983708],
[0.78987214, 0.21012786],
[0.15123149, 0.84876851],
[0.04791338, 0.95208662],
[0.10859109, 0.89140891],
[0.60106079, 0.39893921],
[0.11901581, 0.88098419],
[0.84462885, 0.15537115],
[0.32267183, 0.67732817],
[0.40499055, 0.59500945],
[0.25152373, 0.74847627],
[0.17098681, 0.82901319],
[0.20328074, 0.79671926],
[0.19043815, 0.80956185],
[0.66150182, 0.33849818],
```

```
[0.66226393, 0.33773607],
[0.40262852, 0.59737148],
[0.66933417, 0.33066583],
[0.8348269 , 0.1651731 ],
[0.88946666, 0.11053334],
[0.86068682, 0.13931318],
[0.68396416, 0.31603584],
[0.10240867, 0.89759133],
[0.53492133, 0.46507867],
[0.85390378, 0.14609622],
[0.14393647, 0.85606353],
[0.29518045, 0.70481955],
[0.89662348, 0.10337652],
[0.82326528, 0.17673472],
[0.46494034, 0.53505966],
[0.05511084, 0.94488916],
[0.13321716, 0.86678284],
[0.50561048, 0.49438952],
[0.16822714, 0.83177286],
[0.85267776, 0.14732224],
[0.88173901, 0.11826099],
[0.18824383, 0.81175617],
[0.50137339, 0.49862661],
[0.82584709, 0.17415291],
[0.7507718 , 0.2492282 ],
[0.75475291, 0.24524709],
[0.56378538, 0.43621462],
[0.67369807, 0.32630193],
[0.71952348, 0.28047652],
[0.78445745, 0.21554255],
[0.16807562, 0.83192438],
[0.60481667, 0.39518333],
[0.22984594, 0.77015406],
[0.64659397, 0.35340603],
[0.27594705, 0.72405295],
[0.02430098, 0.97569902],
[0.77855616, 0.22144384],
[0.82007122, 0.17992878],
[0.86843899, 0.13156101],
[0.7418594 , 0.2581406 ],
[0.12691496, 0.87308504],
[0.09177076, 0.90822924],
[0.49643833, 0.50356167],
[0.77766977, 0.22233023],
[0.62953108, 0.37046892],
[0.44925408, 0.55074592],
[0.2371203 , 0.7628797 ],
```

```
[0.63163649, 0.36836351],
[0.22654474, 0.77345526],
[0.20027087, 0.79972913],
[0.40354267, 0.59645733],
[0.83328723, 0.16671277],
[0.82784663, 0.17215337],
[0.07569148, 0.92430852],
[0.04257631, 0.95742369],
[0.14131135, 0.85868865],
[0.41661621, 0.58338379],
[0.94640812, 0.05359188],
[0.68222178, 0.31777822],
[0.35717521, 0.64282479],
[0.21316846, 0.78683154],
[0.45110218, 0.54889782],
[0.69151796, 0.30848204],
[0.37879249, 0.62120751],
[0.81933271, 0.18066729],
[0.81976397, 0.18023603],
[0.28519955, 0.71480045],
[0.13852824, 0.86147176],
[0.25483103, 0.74516897],
[0.83169544, 0.16830456],
[0.27601039, 0.72398961],
[0.82124094, 0.17875906],
[0.21147787, 0.78852213],
[0.09115008, 0.90884992],
[0.80103946, 0.19896054],
[0.07444636, 0.92555364],
[0.92777234, 0.07222766],
[0.55824038, 0.44175962],
[0.43736313, 0.56263687],
[0.76361355, 0.23638645],
[0.34972024, 0.65027976],
[0.67635119, 0.32364881],
[0.74389765, 0.25610235],
[0.27165166, 0.72834834],
[0.173697  , 0.826303  ],
[0.43329233, 0.56670767],
[0.37085734, 0.62914266],
[0.36404874, 0.63595126],
[0.23263862, 0.76736138],
[0.41459585, 0.58540415],
[0.11978213, 0.88021787],
[0.60370614, 0.39629386],
[0.14005645, 0.85994355],
[0.83727335, 0.16272665],
```

```
       [0.77506364, 0.22493636],
       [0.08960527, 0.91039473],
       [0.25834229, 0.74165771],
       [0.19470285, 0.80529715],
       [0.73340696, 0.26659304],
       [0.1442758 , 0.8557242 ],
       [0.4660422 , 0.5339578 ],
       [0.25845861, 0.74154139],
       [0.43792758, 0.56207242],
       [0.81870215, 0.18129785],
       [0.34347672, 0.65652328],
       [0.09296722, 0.90703278],
       [0.68809507, 0.31190493],
       [0.20150081, 0.79849919],
       [0.49617001, 0.50382999],
       [0.7304135 , 0.2695865 ],
       [0.0974097 , 0.9025903 ],
       [0.81323138, 0.18676862],
       [0.09195634, 0.90804366],
       [0.25791424, 0.74208576],
       [0.83826492, 0.16173508],
       [0.21704828, 0.78295172],
       [0.32954349, 0.67045651],
       [0.21515663, 0.78484337],
       [0.12536697, 0.87463303],
       [0.43843534, 0.56156466],
       [0.75150608, 0.24849392],
       [0.35584798, 0.64415202],
       [0.85524862, 0.14475138],
       [0.48364278, 0.51635722],
       [0.42790088, 0.57209912],
       [0.23741146, 0.76258854],
       [0.60601205, 0.39398795],
       [0.87690889, 0.12309111],
       [0.08098681, 0.91901319],
       [0.38488163, 0.61511837],
       [0.33923496, 0.66076504],
       [0.40438148, 0.59561852],
       [0.7826983 , 0.2173017 ],
       [0.44250076, 0.55749924],
       [0.39789302, 0.60210698]])
```

0.0.1 Evaluation

[18]:
```
jaccard_score(y_test, yhat,pos_label=0)
```

[18]:
```
0.7628865979381443
```

[19]:
```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

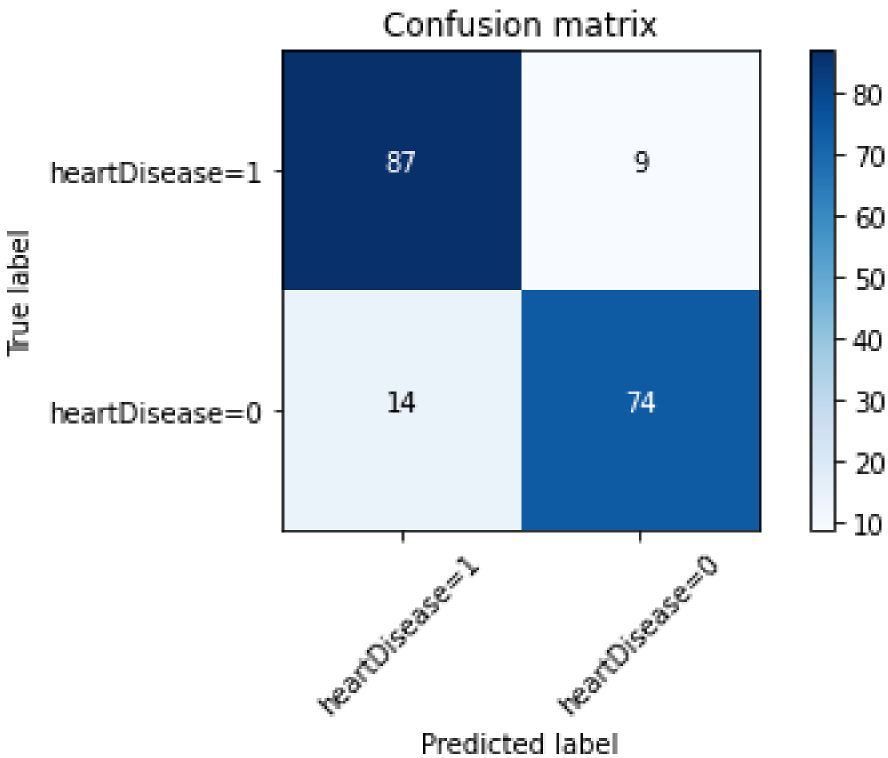
```
[[87  9]
 [14 74]]
```

[24]:
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
```

```
plot_confusion_matrix(cnf_matrix,␣
↪classes=['heartDisease=1','heartDisease=0'],normalize= False, ␣
↪title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[87  9]
 [14 74]]
```



[21]:
```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       0.89      0.84      0.87        88
           1       0.86      0.91      0.88        96

    accuracy                           0.88       184
   macro avg       0.88      0.87      0.87       184
weighted avg       0.88      0.88      0.87       184
```

[22]:
```
#Calculate log_loss
log_loss(y_test, yhat_prob)
```

[22]: 0.38787602959104506

```
[ ]:
```