keras_classifier07.02.20

February 7, 2020

0.1 Data preparation

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

0.2 import dataset

[2]:
```
dataset = pd.read_csv("./cardio_train.csv", sep=';')
```

[3]:
```
dataset.head(2)
```

[3]:
```
   id    age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  \
0   0  18393       2     168    62.0    110     80            1     1      0
1   1  20228       1     156    85.0    140     90            3     1      0

   alco  active  cardio
0     0       1       0
1     0       1       1
```

[4]:
```
#get all types of dataset
dataset.describe(include='all')
```

[4]:
```
                 id           age        gender        height        weight  \
count  70000.000000  70000.000000  70000.000000  70000.000000  70000.000000
mean   49972.419900  19468.865814      1.349571    164.359229     74.205690
std    28851.302323   2467.251667      0.476838      8.210126     14.395757
min        0.000000  10798.000000      1.000000     55.000000     10.000000
25%    25006.750000  17664.000000      1.000000    159.000000     65.000000
50%    50001.500000  19703.000000      1.000000    165.000000     72.000000
75%    74889.250000  21327.000000      2.000000    170.000000     82.000000
max    99999.000000  23713.000000      2.000000    250.000000    200.000000

              ap_hi         ap_lo   cholesterol          gluc         smoke  \
count  70000.000000  70000.000000  70000.000000  70000.000000  70000.000000
```

```
mean      128.817286     96.630414      1.366871      1.226457      0.088129
std       154.011419    188.472530      0.680250      0.572270      0.283484
min      -150.000000    -70.000000      1.000000      1.000000      0.000000
25%       120.000000     80.000000      1.000000      1.000000      0.000000
50%       120.000000     80.000000      1.000000      1.000000      0.000000
75%       140.000000     90.000000      2.000000      1.000000      0.000000
max     16020.000000  11000.000000      3.000000      3.000000      1.000000

               alco        active        cardio
count  70000.000000  70000.000000  70000.000000
mean       0.053771      0.803729      0.499700
std        0.225568      0.397179      0.500003
min        0.000000      0.000000      0.000000
25%        0.000000      1.000000      0.000000
50%        0.000000      1.000000      0.000000
75%        0.000000      1.000000      1.000000
max        1.000000      1.000000      1.000000
```

0.3 calculate bmi and years of age, delete unneeded columns

[31]:
```
dataset['years'] = (dataset['age'] / 365).round().astype('int')
dataset['BMI'] = dataset['weight']/((dataset['height']/100)**2)
dataset.isnull().values.any()
dataset.drop(['id', 'age', 'weight', 'height'], axis=1)
```

[31]:
```
       gender  ap_hi  ap_lo  cholesterol  gluc  smoke  alco  active  cardio  \
0           2    110     80            1     1      0     0       1       0
1           1    140     90            3     1      0     0       1       1
2           1    130     70            3     1      0     0       0       1
3           2    150    100            1     1      0     0       1       1
4           1    100     60            1     1      0     0       0       0
5           1    120     80            2     2      0     0       0       0
6           1    130     80            3     1      0     0       1       0
7           2    130     90            3     3      0     0       1       1
8           1    110     70            1     1      0     0       1       0
9           1    110     60            1     1      0     0       0       0
10          1    120     80            1     1      0     0       1       0
11          2    120     80            1     1      0     0       1       0
12          2    120     80            1     1      0     0       0       0
13          1    110     70            1     1      0     0       1       0
14          2    130     90            1     1      1     1       1       0
15          2    120     80            1     1      0     0       0       1
16          1    130     70            1     1      0     0       0       0
17          1    110     70            1     3      0     0       1       0
18          1    100     70            1     1      0     0       0       0
19          2    120     70            1     1      1     0       1       0
20          2    120     80            1     1      0     0       1       0
```

21	1	130	80	1	1	0	0	1	0
22	1	145	85	2	2	0	0	1	1
23	2	110	60	1	1	0	0	1	0
24	1	150	90	3	1	0	0	1	1
25	1	130	100	2	1	0	0	1	0
26	1	130	90	1	1	0	0	1	0
27	1	120	80	1	1	0	0	1	0
28	2	120	80	1	1	0	0	1	0
29	2	130	70	1	3	0	0	0	0
…	…	…	…	…	…	…	…	…	…
69970	2	140	80	3	1	1	1	0	1
69971	2	130	80	1	1	0	0	1	0
69972	1	140	90	1	1	0	0	1	1
69973	2	130	80	1	1	0	0	1	0
69974	1	120	80	1	1	0	0	1	0
69975	2	120	80	1	1	0	0	1	1
69976	1	120	80	2	2	0	0	1	0
69977	1	120	79	1	1	0	0	1	0
69978	1	90	60	1	1	0	0	1	1
69979	1	160	100	2	2	0	0	1	1
69980	2	110	80	1	1	0	1	0	0
69981	2	130	90	2	2	0	0	1	1
69982	1	130	90	1	2	0	0	1	1
69983	1	120	80	1	1	0	0	1	0
69984	2	120	80	1	1	0	0	1	1
69985	1	130	80	1	1	0	1	0	1
69986	2	120	80	1	1	0	0	1	0
69987	1	120	80	1	1	0	0	1	0
69988	1	110	70	1	1	0	0	1	0
69989	1	120	70	1	1	0	0	1	1
69990	1	110	70	1	1	0	0	1	1
69991	1	130	90	2	2	0	0	1	0
69992	1	170	90	1	1	0	0	1	1
69993	1	130	90	1	1	0	0	1	1
69994	1	150	80	1	1	0	0	1	1
69995	2	120	80	1	1	1	0	1	0
69996	1	140	90	2	2	0	0	1	1
69997	2	180	90	3	1	0	1	0	1
69998	1	135	80	1	2	0	0	0	1
69999	1	120	80	2	1	0	0	1	0

	years	BMI
0	50	21.967120
1	55	34.927679
2	52	23.507805
3	48	28.710479
4	48	23.011177

5	60	29.384676
6	61	37.729725
7	62	29.983588
8	48	28.440955
9	54	25.282570
10	62	28.010224
11	52	20.047446
12	41	22.038567
13	54	31.244993
14	40	28.997894
15	46	37.858302
16	58	25.951557
17	46	20.829995
18	48	28.672626
19	60	21.338211
20	54	31.239414
21	59	27.993022
22	63	36.051915
23	64	18.491124
24	46	23.529412
25	40	27.767098
26	54	24.243918
27	50	30.853210
28	40	23.951227
29	58	25.909457
…	…	…
69970	62	34.414782
69971	55	25.535446
69972	47	27.915519
69973	61	23.510204
69974	50	26.573129
69975	58	30.189591
69976	59	24.464602
69977	46	26.573129
69978	52	29.357522
69979	61	27.852008
69980	49	24.740937
69981	48	33.208550
69982	52	36.738007
69983	54	26.446281
69984	49	28.344671
69985	50	41.913215
69986	50	24.074074
69987	52	21.490286
69988	60	23.046875
69989	58	33.672766
69990	41	25.510204

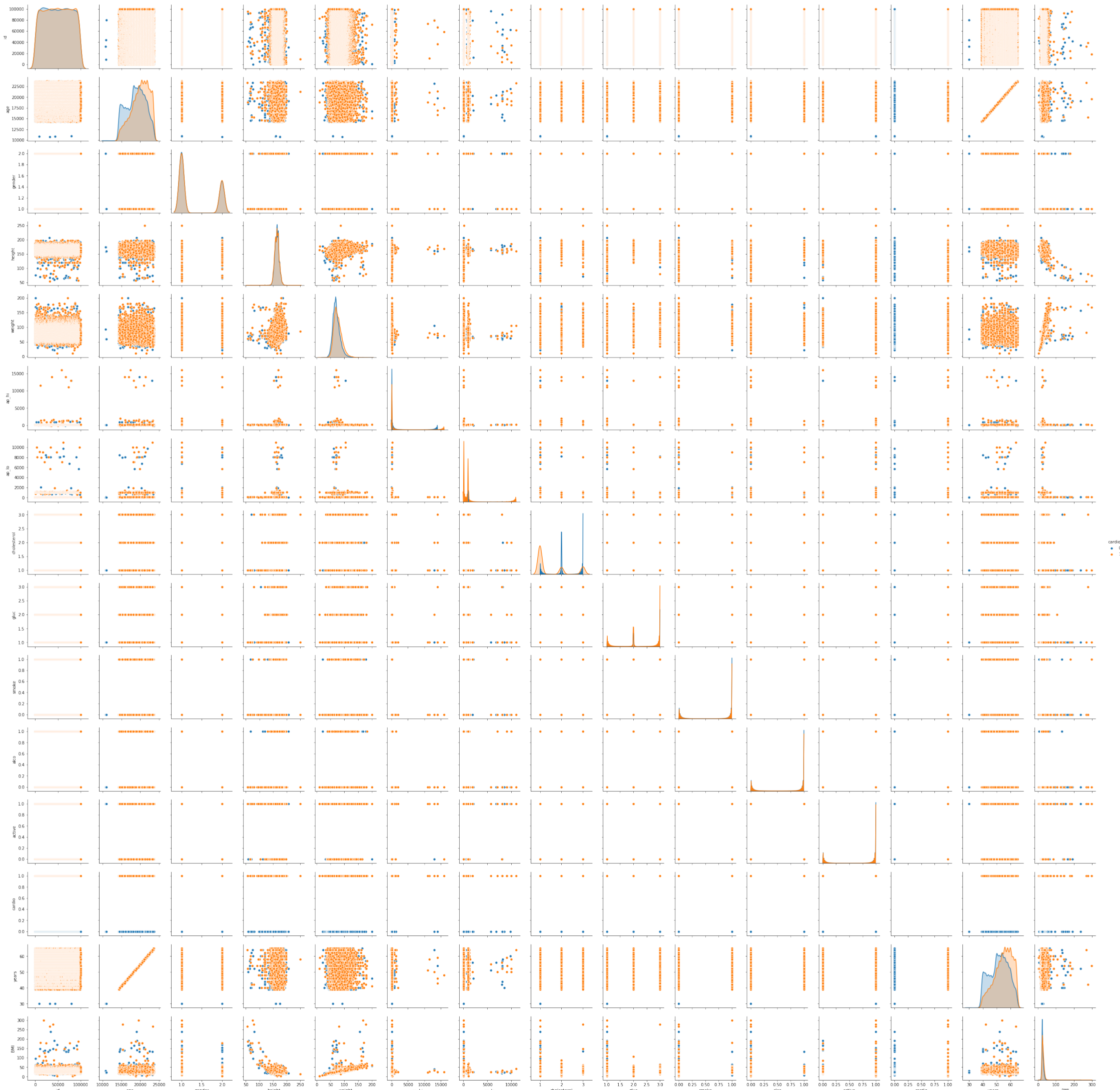
```
69991      56  28.479886
69992      51  21.604105
69993      54  23.661439
69994      58  29.384757
69995      53  26.927438
69996      62  50.472681
69997      52  31.353579
69998      61  27.099251
69999      56  24.913495

[70000 rows x 11 columns]
```

0.4 plot data (pairplot and heatmap for correlation)

[32]:
```
sns.pairplot(dataset, hue='cardio')
```

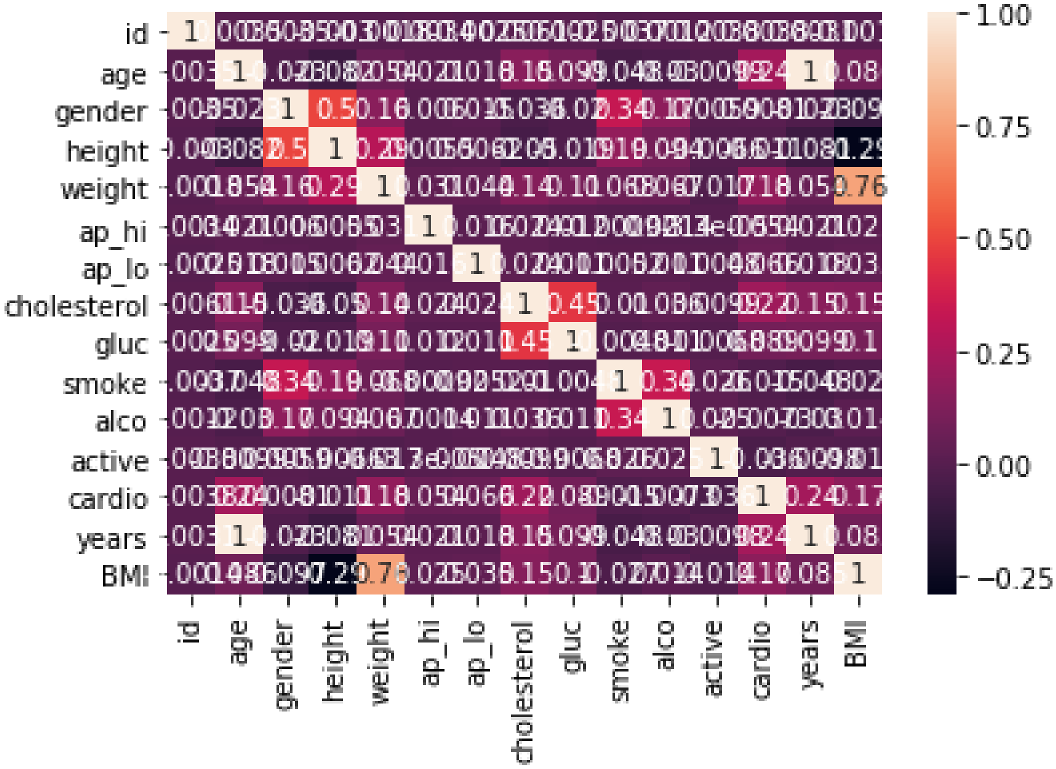
```
/anaconda3/lib/python3.7/site-packages/statsmodels/nonparametric/kde.py:488:
RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/anaconda3/lib/python3.7/site-packages/statsmodels/nonparametric/kdetools.py:34:
RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

[32]:
```
<seaborn.axisgrid.PairGrid at 0x1a1ba7d780>
```



[33]:
```
sns.heatmap(dataset.corr(), annot=True)
```

[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1a7f12f978>



0.5 create input features and target variables for neural network

[50]:
```
# creating input features and target variables
X= dataset.drop(['cardio', 'id','age', 'height', 'weight'], axis=1)
y= dataset.
 ↪drop(['id','height','weight','age','gender','ap_hi','ap_lo','cholesterol','gluc','smoke','a
 ↪axis=1)
```

[51]:
```
X.head(2)
```

[51]:
```
   gender  ap_hi  ap_lo  cholesterol  gluc  smoke  alco  active  years  \
0       2    110     80            1     1      0     0       1     50
1       1    140     90            3     1      0     0       1     55

         BMI
0  21.967120
1  34.927679
```

0.6 normalization of input features

[56]:
```
#standardizing the input feature
#Since our input features are at different scales we need to standardize the␣
↪input.
# (Normalization)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
X
```

[56]:
```
array([[ 1.36405487, -0.12218198, -0.0882385 , …,  0.49416711,
        -0.49350546, -0.91757729],
       [-0.73310834,  0.07261016, -0.03517999, …,  0.49416711,
         0.24556599,  1.21008057],
       [-0.73310834,  0.00767945, -0.14129701, …, -2.02360695,
        -0.19787688, -0.66465218],
       …,
       [ 1.36405487,  0.33233302, -0.03517999, …, -2.02360695,
        -0.19787688,  0.62334178],
       [-0.73310834,  0.04014481, -0.0882385 , …, -2.02360695,
         1.13245175, -0.07506591],
       [-0.73310834, -0.05725127, -0.0882385 , …,  0.49416711,
         0.39338029, -0.4338885 ]])
```

0.7 Model Building

[57]:
```
# split the input features and target variables into training dataset and test␣
↪dataset.
# test dataset will be 30% of our entire dataset.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

[58]:
```
from keras import Sequential
from keras.layers import Dense
```

[59]:
```
classifier = Sequential()
#First Hidden Layer
classifier.add(Dense(5, activation='relu', kernel_initializer='random_normal',␣
↪input_dim=10))#Second  Hidden Layer
classifier.add(Dense(5, activation='relu',␣
↪kernel_initializer='random_normal'))#Output Layer
classifier.add(Dense(1, activation='sigmoid',␣
↪kernel_initializer='random_normal'))
```

```
[60]: #Compiling the neural network
classifier.compile(optimizer ='adam',loss='binary_crossentropy', metrics␣
 ↪=['accuracy'])
```

0.8 Model training

```
[62]: #Fitting the data to the training dataset
classifier.fit(X_train,y_train, batch_size=10, epochs=100)
```

```
Epoch 1/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5421 -
acc: 0.7324
Epoch 2/100
49000/49000 [==============================] - 4s 81us/step - loss: 0.5423 -
acc: 0.7334
Epoch 3/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5422 -
acc: 0.7319
Epoch 4/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5420 -
acc: 0.7328
Epoch 5/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5421 -
acc: 0.7330
Epoch 6/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5421 -
acc: 0.7328
Epoch 7/100
49000/49000 [==============================] - 4s 83us/step - loss: 0.5420 -
acc: 0.7335
Epoch 8/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5421 -
acc: 0.7331
Epoch 9/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5421 -
acc: 0.7327
Epoch 10/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5420 -
acc: 0.7327
Epoch 11/100
49000/49000 [==============================] - 4s 87us/step - loss: 0.5421 -
acc: 0.7326
Epoch 12/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5422 -
acc: 0.7330
Epoch 13/100
49000/49000 [==============================] - 4s 85us/step - loss: 0.5418 -
```

```
acc: 0.7328
Epoch 14/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5421 -
acc: 0.7330
Epoch 15/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5419 -
acc: 0.7336
Epoch 16/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5418 -
acc: 0.7321
Epoch 17/100
49000/49000 [==============================] - 4s 85us/step - loss: 0.5418 -
acc: 0.7332
Epoch 18/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5420 -
acc: 0.7324
Epoch 19/100
49000/49000 [==============================] - 4s 89us/step - loss: 0.5418 -
acc: 0.7336
Epoch 20/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5418 -
acc: 0.7336
Epoch 21/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5416 -
acc: 0.7338
Epoch 22/100
49000/49000 [==============================] - 4s 85us/step - loss: 0.5418 -
acc: 0.7330
Epoch 23/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5417 -
acc: 0.7340
Epoch 24/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5417 -
acc: 0.7336
Epoch 25/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5418 -
acc: 0.7324
Epoch 26/100
49000/49000 [==============================] - 4s 87us/step - loss: 0.5416 -
acc: 0.7327
Epoch 27/100
49000/49000 [==============================] - 4s 87us/step - loss: 0.5419 -
acc: 0.7329
Epoch 28/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5415 -
acc: 0.7337
Epoch 29/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5415 -
```

```
acc: 0.7324
Epoch 30/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5416 -
acc: 0.7325
Epoch 31/100
49000/49000 [==============================] - 4s 89us/step - loss: 0.5418 -
acc: 0.7339
Epoch 32/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5418 -
acc: 0.7320
Epoch 33/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5418 -
acc: 0.7336
Epoch 34/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5416 -
acc: 0.7324
Epoch 35/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5417 -
acc: 0.7339
Epoch 36/100
49000/49000 [==============================] - 4s 87us/step - loss: 0.5417 -
acc: 0.7324
Epoch 37/100
49000/49000 [==============================] - 4s 90us/step - loss: 0.5416 -
acc: 0.7334
Epoch 38/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5418 -
acc: 0.7332
Epoch 39/100
49000/49000 [==============================] - 4s 90us/step - loss: 0.5416 -
acc: 0.7340
Epoch 40/100
49000/49000 [==============================] - 4s 89us/step - loss: 0.5414 -
acc: 0.7339
Epoch 41/100
49000/49000 [==============================] - 4s 88us/step - loss: 0.5415 -
acc: 0.7331
Epoch 42/100
49000/49000 [==============================] - 4s 90us/step - loss: 0.5415 -
acc: 0.7326
Epoch 43/100
49000/49000 [==============================] - 4s 90us/step - loss: 0.5415 -
acc: 0.7328
Epoch 44/100
49000/49000 [==============================] - 4s 91us/step - loss: 0.5416 -
acc: 0.7334
Epoch 45/100
49000/49000 [==============================] - 4s 90us/step - loss: 0.5414 -
```

```
acc: 0.7330
Epoch 46/100
49000/49000 [==============================] - 4s 91us/step - loss: 0.5416 -
acc: 0.7336
Epoch 47/100
49000/49000 [==============================] - 4s 91us/step - loss: 0.5414 -
acc: 0.7332
Epoch 48/100
49000/49000 [==============================] - 4s 91us/step - loss: 0.5416 -
acc: 0.7336
Epoch 49/100
49000/49000 [==============================] - 5s 92us/step - loss: 0.5416 -
acc: 0.7337
Epoch 50/100
49000/49000 [==============================] - 4s 91us/step - loss: 0.5413 -
acc: 0.7331
Epoch 51/100
49000/49000 [==============================] - 5s 95us/step - loss: 0.5415 -
acc: 0.7331
Epoch 52/100
49000/49000 [==============================] - 5s 93us/step - loss: 0.5413 -
acc: 0.7332
Epoch 53/100
49000/49000 [==============================] - 5s 93us/step - loss: 0.5414 -
acc: 0.7334
Epoch 54/100
49000/49000 [==============================] - 5s 92us/step - loss: 0.5415 -
acc: 0.7333
Epoch 55/100
49000/49000 [==============================] - 5s 93us/step - loss: 0.5410 -
acc: 0.7337
Epoch 56/100
49000/49000 [==============================] - 5s 95us/step - loss: 0.5414 -
acc: 0.7332
Epoch 57/100
49000/49000 [==============================] - 5s 103us/step - loss: 0.5414 -
acc: 0.7324
Epoch 58/100
49000/49000 [==============================] - 5s 95us/step - loss: 0.5415 -
acc: 0.7326
Epoch 59/100
49000/49000 [==============================] - 5s 103us/step - loss: 0.5413 -
acc: 0.7333
Epoch 60/100
49000/49000 [==============================] - 5s 93us/step - loss: 0.5413 -
acc: 0.7337
Epoch 61/100
49000/49000 [==============================] - 5s 99us/step - loss: 0.5415 -
```

```
acc: 0.7335
Epoch 62/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5414 -
acc: 0.7329
Epoch 63/100
49000/49000 [==============================] - 4s 85us/step - loss: 0.5413 -
acc: 0.7323
Epoch 64/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5414 -
acc: 0.7336
Epoch 65/100
49000/49000 [==============================] - 4s 81us/step - loss: 0.5414 -
acc: 0.7331
Epoch 66/100
49000/49000 [==============================] - 4s 81us/step - loss: 0.5413 -
acc: 0.7326
Epoch 67/100
49000/49000 [==============================] - 5s 94us/step - loss: 0.5412 -
acc: 0.7344
Epoch 68/100
49000/49000 [==============================] - 5s 106us/step - loss: 0.5413 -
acc: 0.7330
Epoch 69/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5410 -
acc: 0.7334
Epoch 70/100
49000/49000 [==============================] - 5s 98us/step - loss: 0.5412 -
acc: 0.7336
Epoch 71/100
49000/49000 [==============================] - 5s 95us/step - loss: 0.5412 -
acc: 0.7334
Epoch 72/100
49000/49000 [==============================] - 5s 102us/step - loss: 0.5411 -
acc: 0.7346
Epoch 73/100
49000/49000 [==============================] - 5s 102us/step - loss: 0.5410 -
acc: 0.7327
Epoch 74/100
49000/49000 [==============================] - 5s 105us/step - loss: 0.5414 -
acc: 0.7328
Epoch 75/100
49000/49000 [==============================] - 4s 86us/step - loss: 0.5411 -
acc: 0.7328
Epoch 76/100
49000/49000 [==============================] - 5s 101us/step - loss: 0.5413 -
acc: 0.7333
Epoch 77/100
49000/49000 [==============================] - 5s 96us/step - loss: 0.5412 -
```

```
acc: 0.7328
Epoch 78/100
49000/49000 [==============================] - 5s 105us/step - loss: 0.5410 -
acc: 0.7332
Epoch 79/100
49000/49000 [==============================] - 5s 92us/step - loss: 0.5410 -
acc: 0.7345
Epoch 80/100
49000/49000 [==============================] - 4s 91us/step - loss: 0.5413 -
acc: 0.7329
Epoch 81/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5411 -
acc: 0.7325
Epoch 82/100
49000/49000 [==============================] - 4s 83us/step - loss: 0.5411 -
acc: 0.7333
Epoch 83/100
49000/49000 [==============================] - 5s 104us/step - loss: 0.5412 -
acc: 0.7342
Epoch 84/100
49000/49000 [==============================] - 4s 89us/step - loss: 0.5409 -
acc: 0.7341
Epoch 85/100
49000/49000 [==============================] - 4s 89us/step - loss: 0.5413 -
acc: 0.7331
Epoch 86/100
49000/49000 [==============================] - 6s 119us/step - loss: 0.5410 -
acc: 0.7343
Epoch 87/100
49000/49000 [==============================] - 5s 106us/step - loss: 0.5409 -
acc: 0.7342
Epoch 88/100
49000/49000 [==============================] - 6s 112us/step - loss: 0.5410 -
acc: 0.7341
Epoch 89/100
49000/49000 [==============================] - 6s 112us/step - loss: 0.5413 -
acc: 0.7339
Epoch 90/100
49000/49000 [==============================] - 4s 83us/step - loss: 0.5412 -
acc: 0.7327
Epoch 91/100
49000/49000 [==============================] - 4s 83us/step - loss: 0.5411 -
acc: 0.7332
Epoch 92/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5411 -
acc: 0.7340
Epoch 93/100
49000/49000 [==============================] - 4s 83us/step - loss: 0.5410 -
```

```
acc: 0.7345
Epoch 94/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5410 -
acc: 0.7336
Epoch 95/100
49000/49000 [==============================] - 4s 84us/step - loss: 0.5409 -
acc: 0.7333
Epoch 96/100
49000/49000 [==============================] - 4s 83us/step - loss: 0.5410 -
acc: 0.7331
Epoch 97/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5410 -
acc: 0.7339
Epoch 98/100
49000/49000 [==============================] - 4s 83us/step - loss: 0.5410 -
acc: 0.7334
Epoch 99/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5410 -
acc: 0.7354
Epoch 100/100
49000/49000 [==============================] - 4s 82us/step - loss: 0.5410 -
acc: 0.7344
```

[62]:
```
<keras.callbacks.History at 0x1a879c7b70>
```

0.9 Model evaluation

[63]:
```
eval_model=classifier.evaluate(X_train, y_train)
eval_model
```

```
49000/49000 [==============================] - 0s 9us/step
```

[63]:
```
[0.5403157654392476, 0.7336938775510204]
```

0.10 Predict cardiovascular disease

[64]:
```
y_pred=classifier.predict(X_test)
y_pred =(y_pred>0.5)
```

[65]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[8056 2524]
 [3072 7348]]
```

0.11 total richtig/positiv falsch/negativ: 8056 + 7348 = 15404

0.12 insgesamt: 21000

0.13 accuracy: 100 / 21000 * 15404 = 73,35 %

0.14 With the given inputs we can predict with a 73% accuracy if the person will suffer from cardiovascular disease or not