Arab Republic of Egypt
Ministry of Communications and Information Technology

Digital Egypt Youth

Internet of things (IoT) and artificial Intelligent (AI)

National Telecommunication Institute

شباب مصر الرقمية

Coronary Heart Disease (CHD) Diagnosis and predict Using Traditional machine learning and MLP

Presented By

Fikrya Ahmed Seddik Ahmed

Ahmed Mohie AbdElazeem Younis

Abdelrhman Mohamed Fawzy Ibrahim

Ibrahim Abd-elghany Ibrahim Salem

Presented To

Dr. Mohamed Elzorkany

Abstract

The report will describe a feasibility study to assess the use of neural networks and traditional machine learning algorithms to solve coronary heart disease (CHD) - Diagnosis and also predict a brief description of some algorithms of traditional machine learning we used.

Introduction

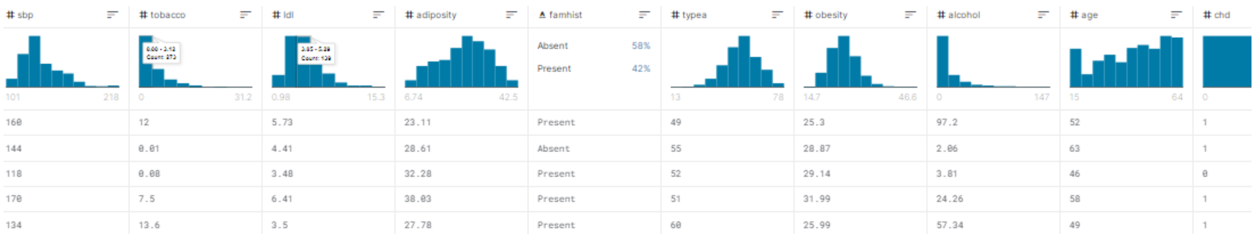
Traditional machine learning is a set of mathematical, statistical and computational methods for developing algorithms that can solve a problem not in a direct way but based on finding patterns in a variety of input data. The solution is calculated not according to a formula, but according to the established dependence of the results on a specific set of features and their values. The neural network concept is to simulate a human's neural system. Its ability to learn using previous experience and thus make fewer errors next time. This is the main feature of neural networks.

Datasets 1

We used two data sets one for diagnosis and other for prediction

First data set

It contains of 9 feature and 1 output



sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
101 – 218	0 – 31.2	0.98 – 15.3	6.74 – 42.5	Absent 58%, Present 42%	13 – 78	14.7 – 46.6	0 – 147	15 – 64	0
160	12	5.73	23.11	Present	49	25.3	97.2	52	1
144	0.01	4.41	28.61	Absent	55	28.87	2.06	63	1
118	0.08	3.48	32.28	Present	52	29.14	3.81	46	0
170	7.5	6.41	38.03	Present	51	31.99	24.26	58	1
134	13.6	3.5	27.78	Present	60	25.99	57.34	49	1

Features

(sbp)---systolic blood pressure

(Tobacco)---yearly tobacco use (in kg)

(Idl)---low density lipoprotein adiposity

(Family history)--- (0 or 1)

(typea)---type A personality score

(Obesity)---(body mass index)

(alcohol use)

(Age)

Output

The diagnosis of CHD (0 or 1).

Models' analysis using Neural Network

```
#import Libraries
import keras
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
#import data
chd_data=pd.read_csv('/content/CHDdata diagnosis.csv')
chd_data
#check for null value
chd_data.isnull().sum()
#check for duplicated
chd_data.duplicated().sum()
#check data types
chd_data.dtypes
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(chd_data.iloc[:,4])
list(le.classes_)
chd_data.iloc[:,4]=le.transform(chd_data.iloc[:,4])
chd_data

#spliting data
x=chd_data.iloc[:,:-1]
y=chd_data.iloc[:,-1]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=54)
x_train.iloc[0:4,:]
mms = StandardScaler()
x_train= mms.fit_transform(x_train)
x_test= mms.transform(x_test)
x_test[0:4,:]
model = keras.Sequential()
model.add(keras.layers.Dense(1, input_shape= (9,),activation = 'sigmoid'))

model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['binar
y_accuracy'])
```

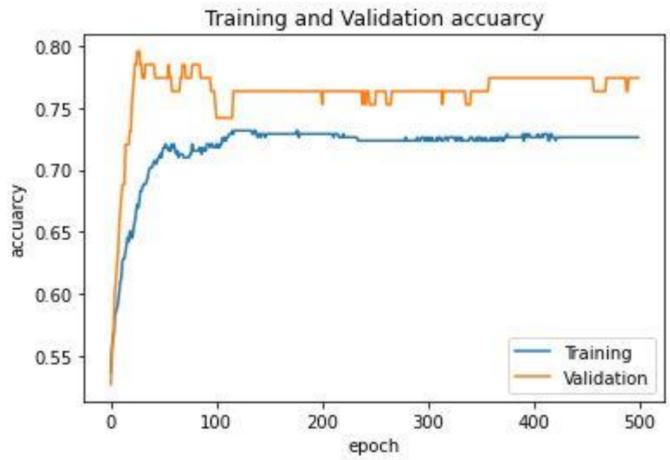
```
results = model.fit( x = x_train, y = y_train,shuffle=True, epochs = 500,
    batch_size =32, validation_data = (x_test, y_test))

eval = model.evaluate(x = x_test, y = y_test)


import matplotlib.pyplot as plt
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation Losses')
plt.xlabel('epoch')
plt.ylabel('Losses')


plt.plot(results.history['binary_accuracy'])
plt.plot(results.history['val_binary_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuarcy')
plt.xlabel('epoch')
plt.ylabel('accuarcy')
```

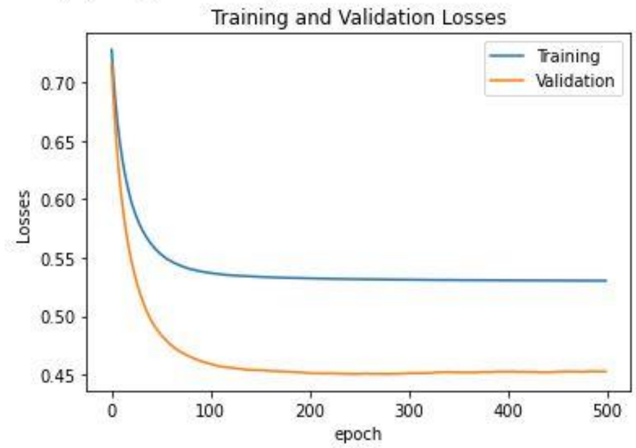
The results: -

```
12/12 [==============================] - 0s 6ms/step - loss: 0.5304 - binary_accuracy: 0.7263 - val_loss: 0.4528 - val_binary_accuracy: 0.7742
Epoch 495/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5304 - binary_accuracy: 0.7263 - val_loss: 0.4529 - val_binary_accuracy: 0.7742
Epoch 496/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5304 - binary_accuracy: 0.7263 - val_loss: 0.4528 - val_binary_accuracy: 0.7742
Epoch 497/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5304 - binary_accuracy: 0.7263 - val_loss: 0.4528 - val_binary_accuracy: 0.7742
Epoch 498/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5304 - binary_accuracy: 0.7263 - val_loss: 0.4528 - val_binary_accuracy: 0.7742
Epoch 499/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5304 - binary_accuracy: 0.7263 - val_loss: 0.4529 - val_binary_accuracy: 0.7742
Epoch 500/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5304 - binary_accuracy: 0.7263 - val_loss: 0.4528 - val_binary_accuracy: 0.7742
```

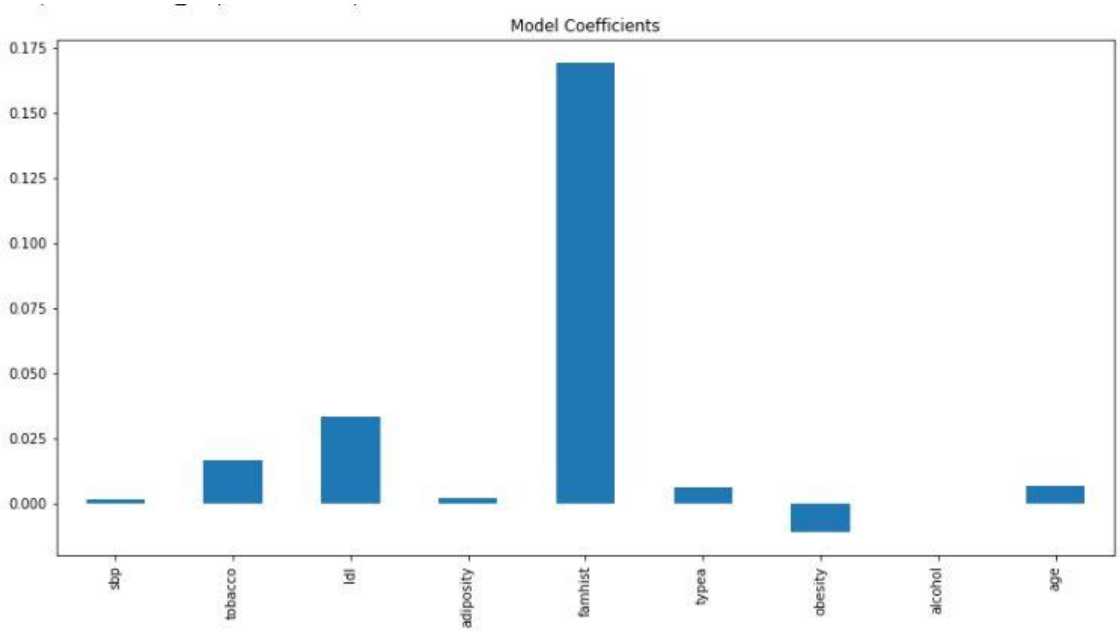
```
[ ] eval = model.evaluate(x = x_test, y = y_test)
```

```
3/3 [==============================] - 0s 6ms/step - loss: 0.4528 - binary_accuracy: 0.7742
```





By using lasso-regression to do automatic feature selection



By dropping 6 low features: (binary_accuracy: 0.7634)

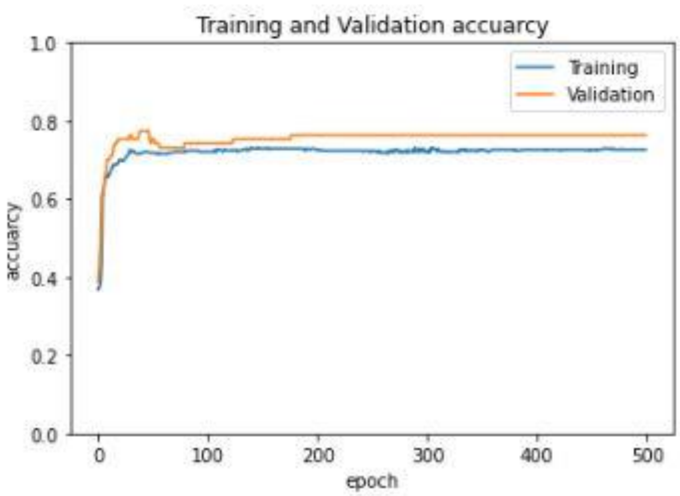
```
x.drop(['sbp','adiposity', 'obesity','ldl','typea', 'alcohol'],inplace=Tru
e,axis=1)
```

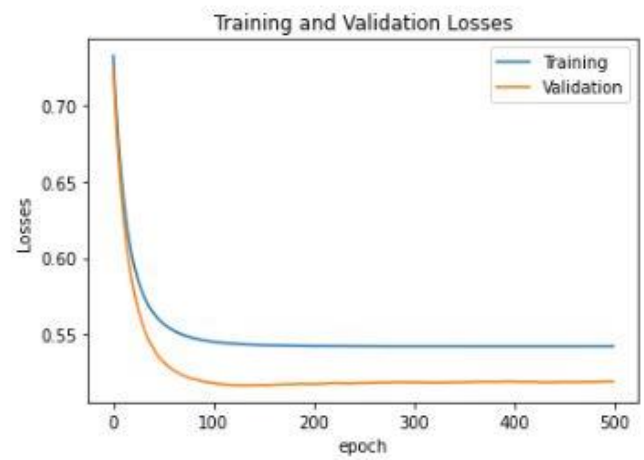
```
model = keras.Sequential()
model.add(keras.layers.Dense(1, input_shape= (3,),  activation = 'sigmoid'
))
```

```
Epoch 498/500
12/12 [==============================] - 0s 6ms/step - loss: 0.5420 - binary_accuracy: 0.7263 - val_loss: 0.5190 - val_binary_accuracy: 0.7634
Epoch 499/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5420 - binary_accuracy: 0.7263 - val_loss: 0.5190 - val_binary_accuracy: 0.7634
Epoch 500/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5420 - binary_accuracy: 0.7263 - val_loss: 0.5190 - val_binary_accuracy: 0.7634
```

```
[21] eval = model.evaluate(x = x_test, y = y_test)
```

```
3/3 [==============================] - 0s 4ms/step - loss: 0.5190 - binary_accuracy: 0.7634
```





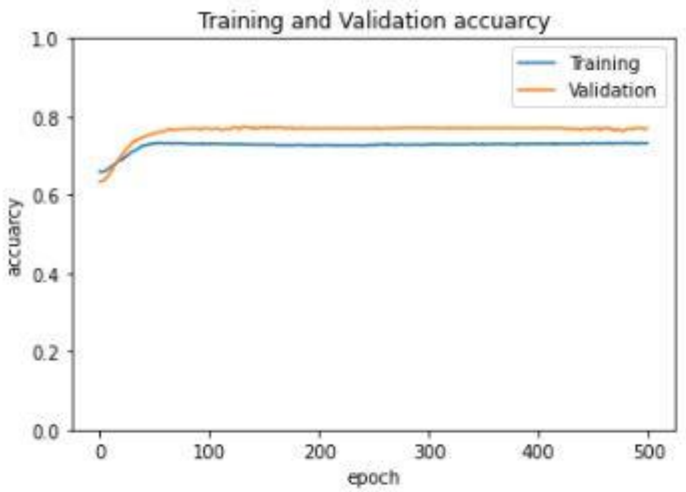
By adding hidden layers (binary_accuracy: 0.7702)

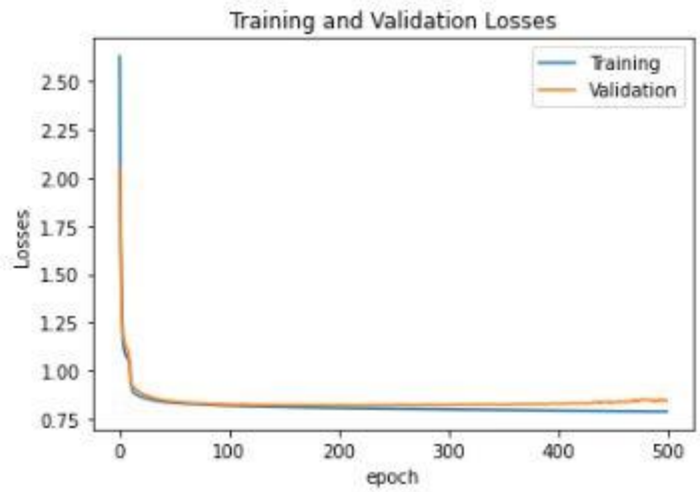
```
model = keras.Sequential()
model.add(keras.layers.Dense(256, input_shape= (4,), activation = 'relu'))
model.add(keras.layers.Dense(128,  activation = 'relu'))
model.add(keras.layers.Dense(64,  activation = 'relu'))
model.add(keras.layers.Dense(32,  activation = 'relu'))
```

```
12/12 [==============================] - 0s 5ms/step - loss: 0.7893 - binary_accuracy: 0.7317 - val_loss: 0.8548 - val_binary_accuracy: 0.7698
Epoch 498/500
12/12 [==============================] - 0s 5ms/step - loss: 0.7895 - binary_accuracy: 0.7315 - val_loss: 0.8504 - val_binary_accuracy: 0.7675
Epoch 499/500
12/12 [==============================] - 0s 5ms/step - loss: 0.7896 - binary_accuracy: 0.7312 - val_loss: 0.8462 - val_binary_accuracy: 0.7695
Epoch 500/500
12/12 [==============================] - 0s 5ms/step - loss: 0.7899 - binary_accuracy: 0.7326 - val_loss: 0.8443 - val_binary_accuracy: 0.7702
```

```
[21] eval = model.evaluate(x = x_test, y = y_test)
```

```
3/3 [==============================] - 0s 4ms/step - loss: 0.8443 - binary_accuracy: 0.7702
```





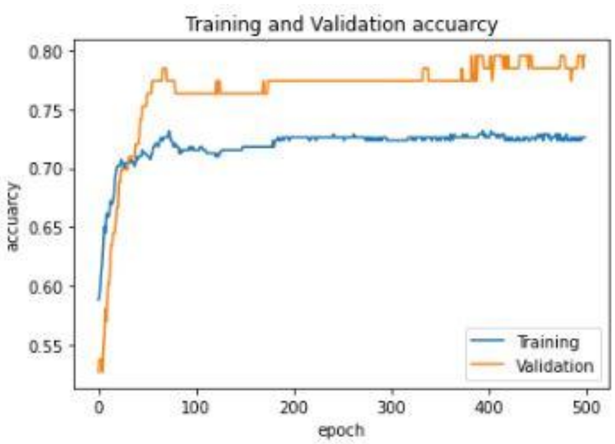
By dropping 5 low features: (binary_accuracy: 0.7957)

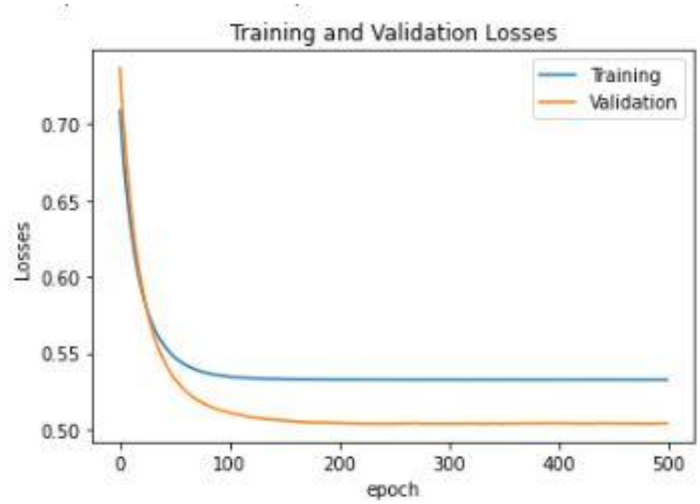
```
x.drop(['sbp','adiposity', 'obesity', 'typea', 'alcohol'],inplace=True,axi
s=1)

model = keras.Sequential()
model.add(keras.layers.Dense(1, input_shape= (4,),  activation = 'sigmoid'
))
```

```
Epoch 498/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5324 - binary_accuracy: 0.7263 - val_loss: 0.5040 - val_binary_accuracy: 0.7849
Epoch 499/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5324 - binary_accuracy: 0.7263 - val_loss: 0.5041 - val_binary_accuracy: 0.7957
Epoch 500/500
12/12 [==============================] - 0s 5ms/step - loss: 0.5325 - binary_accuracy: 0.7263 - val_loss: 0.5040 - val_binary_accuracy: 0.7957
```

```
[21] eval = model.evaluate(x = x_test, y = y_test)
```

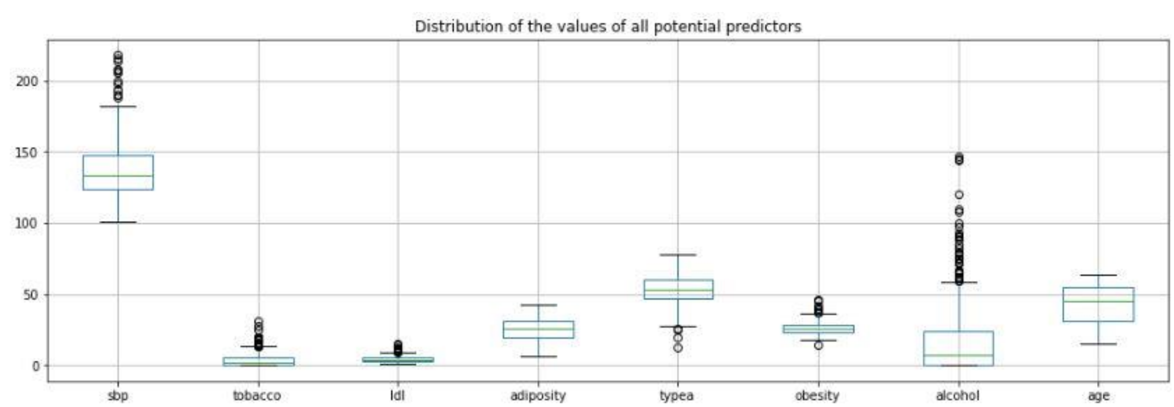
```
3/3 [==============================] - 0s 4ms/step - loss: 0.5040 - binary_accuracy: 0.7957
```



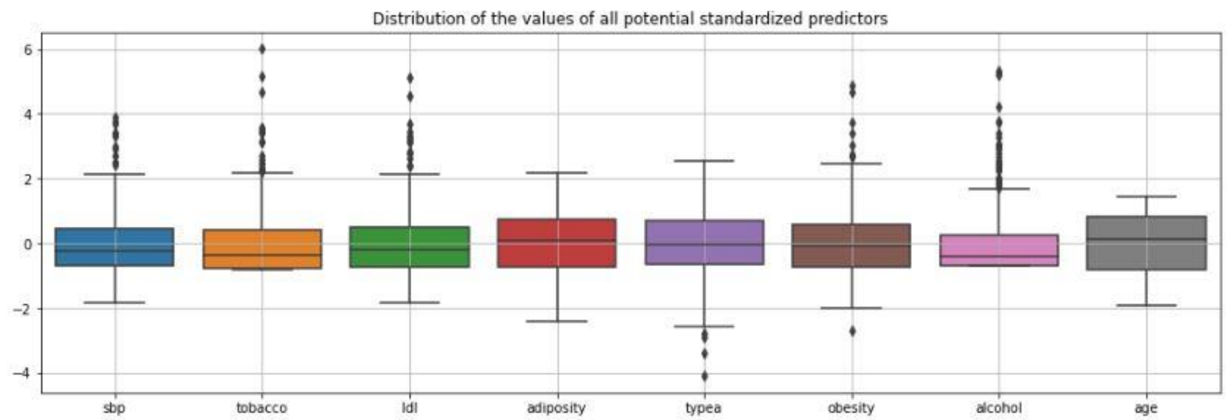


Using the traditional ML

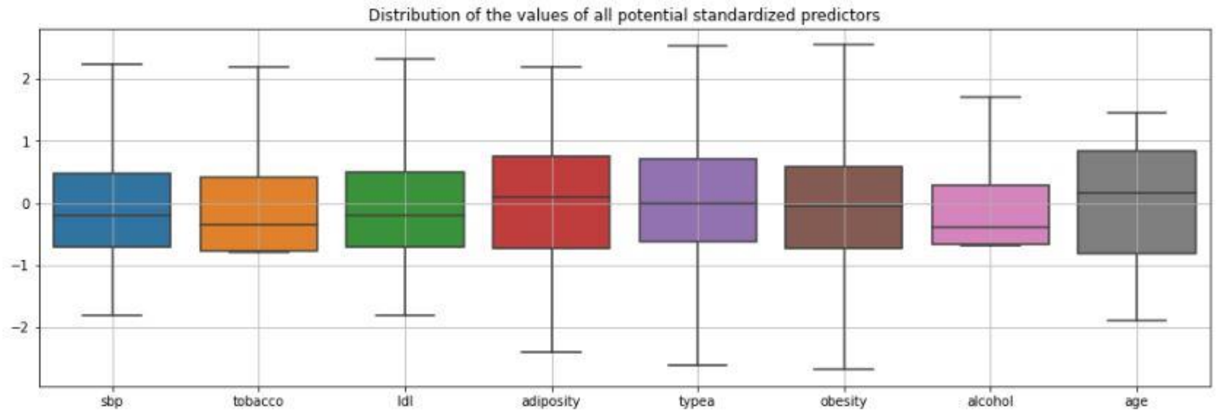
```
#import Libraries
from tensorflow import keras
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler , StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
#import data
chd_data=pd.read_csv('/content/CHDdata diagnosis.csv')
chd_data
#check for null value
chd_data.isnull().sum()
#check data types
chd_data.dtypes
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(chd_data.iloc[:,4])
chd_data.iloc[:,4]=le.transform(chd_data.iloc[:,4])
chd_data

Index = np.r_[0:4,5:9]
plt.figure(figsize=(16,5))
chd_data.iloc[:,Index].boxplot()
plt.title("Distribution of the values of all potential predictors")
plt.show()
```



```
#rescaling
scaler = StandardScaler()
chd_data.iloc[:,Index] = scaler.fit_transform(chd_data.iloc[:,Index])
chd_data.iloc[0:5,:]
plt.figure(figsize=(16,5))
sns.boxplot(data=chd_data.iloc[:,Index])
plt.title("Distribution of the values
of all potential standardized predictors")
plt.grid()
plt.show()
```




```
outliers = ['sbp', 'tobacco', 'ldl', 'typea', 'obesity', 'alcohol']
for column in outliers:
  Q1,Q3 = np.percentile(chd_data[column],[25,75])
  IQR = Q3 - Q1
  lower_fence = Q1 - (1.5*IQR)
  upper_fence = Q3 + (1.5*IQR)
  chd_data[column] = chd_data[column].apply(lambda x: upper_fence if x>upp
er_fence
 else lower_fence if x<lower_fence else x)
plt.figure(figsize=(16,5))
sns.boxplot(data=chd_data.iloc[:,Index])
plt.title("Distribution of the values
of all potential standardized predictors")
plt.grid()
plt.show()
```




```
#spliting data
x=chd_data.iloc[:,:-1]
y=chd_data.iloc[:,-1]


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=23)

x_train.iloc[0,:]

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score


classifiers = []
classifiers.append(("LR",LogisticRegression(random_state=0)))
classifiers.append(("NB",GaussianNB()))
classifiers.append(("DT",DecisionTreeClassifier(random_state = 0)))
classifiers.append(("RF",RandomForestClassifier(random_state = 0)))
classifiers.append(("SVM",SVC()))
classifiers.append(("KNN", KNeighborsClassifier()))
scores = []
clf_names = []
```

```
my_list = [ 0.2, 0.25, 0.33, 0.4]
for clf in classifiers:
  score = 0
  recall = 0
  for p in my_list:
    for i in range(101):
      X_train, X_test, y_train, y_test = train_test_split(x, y, test_size
= p, random_state = i)
      classifier = clf[1]
      classifier.fit(X_train,y_train)
      y_pred = classifier.predict(X_test)
      acc_score = (accuracy_score(y_test,y_pred)*100).round(2)
      rec_score = recall_score(y_test,y_pred)
      if acc_score > score:
          score = acc_score
          recall = rec_score
          paramters = (p, i, score, recall)
  p, i, score, recall = paramters
  print("classifier = {}, P = {}, i = {}, score = {}, recall = {}".format(
classifier, p,i,score, recall))
```

```
classifier = LogisticRegression(random_state=0), P = 0.2, i = 67, score = 81.72, recall = 0.7083333333333334
classifier = GaussianNB(), P = 0.2, i = 33, score = 79.57, recall = 0.7894736842105263
classifier = DecisionTreeClassifier(random_state=0), P = 0.2, i = 0, score = 74.19, recall = 0.625
classifier = RandomForestClassifier(random_state=0), P = 0.2, i = 69, score = 78.49, recall = 0.6176470588235294
classifier = SVC(), P = 0.2, i = 62, score = 81.72, recall = 0.6296296296296297
classifier = KNeighborsClassifier(), P = 0.2, i = 80, score = 75.27, recall = 0.4642857142857143
```

```
my_list = [ 0.2, 0.25, 0.33, 0.4]
for clf in classifiers:
  score = 0
  recall = 0
  for p in my_list:
    for i in range(101):
      X_train, X_test, y_train, y_test = train_test_split(x, y, test_size
= p, random_state = i)
      classifier = clf[1]
      classifier.fit(X_train,y_train)
      y_pred = classifier.predict(X_test)
      acc_score = (accuracy_score(y_test,y_pred)*100).round(2)
      rec_score = recall_score(y_test,y_pred)
      if rec_score > recall:
          score = acc_score
          recall = rec_score
```

```
            paramters = (p, i, score, recall)
    p, i, score, recall = paramters
    print("classifier = {}, P = {}, i = {}, score = {}, recall = {}".format(
classifier, p,i,score, recall))
```

```
classifier = LogisticRegression(random_state=0), P = 0.25, i = 67, score = 79.31, recall = 0.7241379310344828
classifier = GaussianNB(), P = 0.2, i = 62, score = 78.49, recall = 0.8888888888888888
classifier = DecisionTreeClassifier(random_state=0), P = 0.2, i = 43, score = 72.04, recall = 0.7575757575757576
classifier = RandomForestClassifier(random_state=0), P = 0.25, i = 25, score = 73.28, recall = 0.6785714285714286
classifier = SVC(), P = 0.2, i = 54, score = 79.57, recall = 0.7142857142857143
classifier = KNeighborsClassifier(), P = 0.25, i = 78, score = 66.38, recall = 0.6285714285714286
```

```
# Applying GridSearch on the dataset to Check the best paramters for our C
lassification
from sklearn.model_selection import RepeatedStratifiedKFold, GridSearchCV
for classifier_name, classifier in classifiers:
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=0)
    if classifier_name == "LR" :
        parameters = [{'solver' : ['newton-
cg', 'lbfgs'], 'penalty' : ['l2'],
                       'C':[100, 10, 1, 0.1, 0.01],'multi_class':['auto',
'ovr', 'multinomial']},
                      {'solver': ['liblinear'],'penalty' : ['l1', 'l2'],
                       'C':[100, 10, 1, 0.1, 0.01],'multi_class':['auto',
'ovr'],'random_state':[0,1,42]}]
    elif classifier_name == "NB" :
        continue
    elif classifier_name == "DT" :
        parameters = [{'criterion' : ['gini', 'entropy'],'random_state':[
0, 1, 12, 42],
                        'splitter':['best', 'random'], 'max_features': ['s
qrt', 'log2']}]
    elif classifier_name == "RF" :
        parameters = [{'bootstrap':[True], 'criterion' : ['gini', 'entropy
'], 'n_estimators':[10,15,20,25],
                        'max_depth':[110,130,150,170], 'random_state':[ 0,
 1 , 42],
                       'min_samples_leaf':[7,9,11,13],'min_samples_split':
[8,12,14],'max_features': ['sqrt', 'log2']}]
    elif classifier_name == "SVM" :
        parameters = [{'C':[100, 10, 1, 0.1, 0.01], 'gamma' : ['auto', 'sc
ale'],
                       'kernel': ['poly', 'rbf', 'sigmoid']},
                      {'C':[100, 10, 1, 0.1, 0.01], 'kernel': ['linear']}]
```

```
LR  (best score) :  0.7294374318217237
best parameters :  {'C': 0.1, 'multi_class': 'auto', 'penalty': 'l2', 'solver': 'newton-cg'}
DT  (best score) :  0.6495247000155836
best parameters :  {'criterion': 'entropy', 'max_features': 'sqrt', 'random_state': 1, 'splitter': 'best'}
RF  (best score) :  0.7158017765310892
best parameters :  {'bootstrap': True, 'criterion': 'entropy', 'max_depth': 110, 'max_features': 'sqrt', 'min_samples_leaf': 11, 'min_samples_split': 8, 'n_estimators': 25, 'random_s
SVM  (best score) :  0.7251908991740688
best parameters :  {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
KNN  (best score) :  0.6904784167056258
best parameters :  {'algorithm': 'auto', 'n_jobs': -1, 'n_neighbors': 13, 'weights': 'distance'}
```

```
classifiers = []
classifiers.append(("LR",LogisticRegression(C = 0.1, multi_class =  'auto'
, penalty = 'l2', solver =  'newton-cg')))
classifiers.append(("NB",GaussianNB()))
classifiers.append(("DT",DecisionTreeClassifier(criterion = 'entropy', max
_features ='sqrt', random_state = 1, splitter = 'best')))
classifiers.append(("RF",RandomForestClassifier(bootstrap = True, criterio
n = 'entropy', max_depth = 110, max_features = 'sqrt',
                                                min_samples_leaf = 11, min
_samples_split = 8, n_estimators = 25, random_state = 42)))
classifiers.append(("SVM",SVC(C = 1, gamma = 'auto', kernel = 'rbf')))
classifiers.append(("KNN", KNeighborsClassifier(algorithm = 'auto', n_jobs
 = -1, n_neighbors = 13, weights = 'distance')))


my_list = [ 0.2, 0.25, 0.33, 0.4]
for clf in classifiers:
  score = 0
  recall = 0
  for p in my_list:
    for i in range(101):
      X_train, X_test, y_train, y_test = train_test_split(x, y, test_size
= p, random_state = i)
      classifier = clf[1]
      classifier.fit(X_train,y_train)
      y_pred = classifier.predict(X_test)
      acc_score = (accuracy_score(y_test,y_pred)*100).round(2)
      rec_score = recall_score(y_test,y_pred)
      if acc_score > score:
          score = acc_score
          recall = rec_score
          paramters = (p, i, score, recall)
  p, i, score, recall = paramters
  print("classifier = {}, P = {}, i = {}, score = {}, recall = {}".format(
classifier, p, i, score, recall))
```

```
classifier = LogisticRegression(C=0.1, solver='newton-cg'), P = 0.25, i = 67, score = 81.9, recall = 0.7241379310344828
classifier = GaussianNB(), P = 0.2, i = 33, score = 79.57, recall = 0.7894736842105263
classifier = DecisionTreeClassifier(criterion='entropy', max_features='sqrt', random_state=1), P = 0.2, i = 1, score = 76.34, recall = 0.5925925925925926
classifier = RandomForestClassifier(criterion='entropy', max_depth=110, max_features='sqrt',
                       min_samples_leaf=11, min_samples_split=8,
                       n_estimators=25, random_state=42), P = 0.25, i = 9, score = 81.03, recall = 0.5135135135135135
classifier = SVC(C=1, gamma='auto'), P = 0.2, i = 62, score = 82.8, recall = 0.6666666666666666
classifier = KNeighborsClassifier(n_jobs=-1, n_neighbors=13, weights='distance'), P = 0.2, i = 62, score = 83.87, recall = 0.5555555555555556
```

```
my_list = [ 0.2, 0.25, 0.33, 0.4]
for clf in classifiers:
  score = 0
  recall = 0
  for p in my_list:
    for i in range(101):
      X_train, X_test, y_train, y_test = train_test_split(x, y, test_size
= p, random_state = i)
      classifier = clf[1]
      classifier.fit(X_train,y_train)
      y_pred = classifier.predict(X_test)
      acc_score = (accuracy_score(y_test,y_pred)*100).round(2)
      rec_score = recall_score(y_test,y_pred)
      if rec_score > recall:
          score = acc_score
          recall = rec_score
          paramters = (p, i, score, recall)
  p, i, score, recall = paramters
  print("classifier = {}, P = {}, i = {}, score = {}, recall = {}".format(
classifier, p,i,score, recall))
```

```
classifier = LogisticRegression(C=0.1, solver='newton-cg'), P = 0.25, i = 67, score = 81.9, recall = 0.7241379310344828
classifier = GaussianNB(), P = 0.2, i = 62, score = 78.49, recall = 0.8888888888888888
classifier = DecisionTreeClassifier(criterion='entropy', max_features='sqrt', random_state=1), P = 0.2, i = 28, score = 70.97, recall = 0.7
classifier = RandomForestClassifier(criterion='entropy', max_depth=110, max_features='sqrt',
                       min_samples_leaf=11, min_samples_split=8,
                       n_estimators=25, random_state=42), P = 0.25, i = 43, score = 68.97, recall = 0.6904761904761905
classifier = SVC(C=1, gamma='auto'), P = 0.2, i = 54, score = 80.65, recall = 0.7142857142857143
classifier = KNeighborsClassifier(n_jobs=-1, n_neighbors=13, weights='distance'), P = 0.25, i = 78, score = 72.41, recall = 0.6
```

Datasets 2

CHD Prediction”

Its content 15 Features and 1 output

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	0
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	95.0	76.0	0
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	75.0	70.0	0
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	65.0	103.0	1
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	85.0	85.0	0

Features

(male)

(age)

(education)

(currentsmoker)---(0 or 1)

(cigsperday) ---(cigarettes per day)

(BPMeds)---(Blood pressure medications)

(prevalentstroke)---(prevalence stroke, 0 or 1)

(prevalenthyp)---(prevalent hypertension, 0 or 1)

(diabetes)---(0 or 1)

(totchol)---(Total cholesterol)

(SysBP)---(Systolic Blood Pressure)

(diaBP)---(Diastolic blood pressure)
(BMI)---(body mass index)
(heartrate)
(glucose)|

Output

Coronary Heart Disease Prediction (0 or 1).

Models' analysis using Neural Network

```
# import pakcages and libraries needed for the project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('darkgrid')
```

Feature scaling

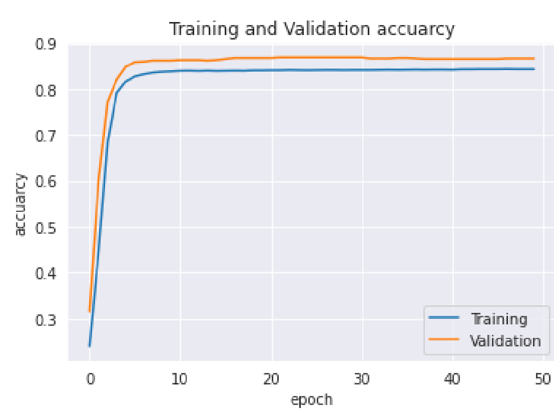
```
# Scaling for making close variables values from each other
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
# choose index for only scaling numerical data
index = np.r_[1,4,9:15]
X_train[:, index] = sc_X.fit_transform(X_train[:, index]) # apply fit() on
 X_train and transform fit on X_train
X_test[:, index] = sc_X.transform(X_test[:, index]) # because we already f
it() on x_train we only transform fit on X_test
```

Model Building

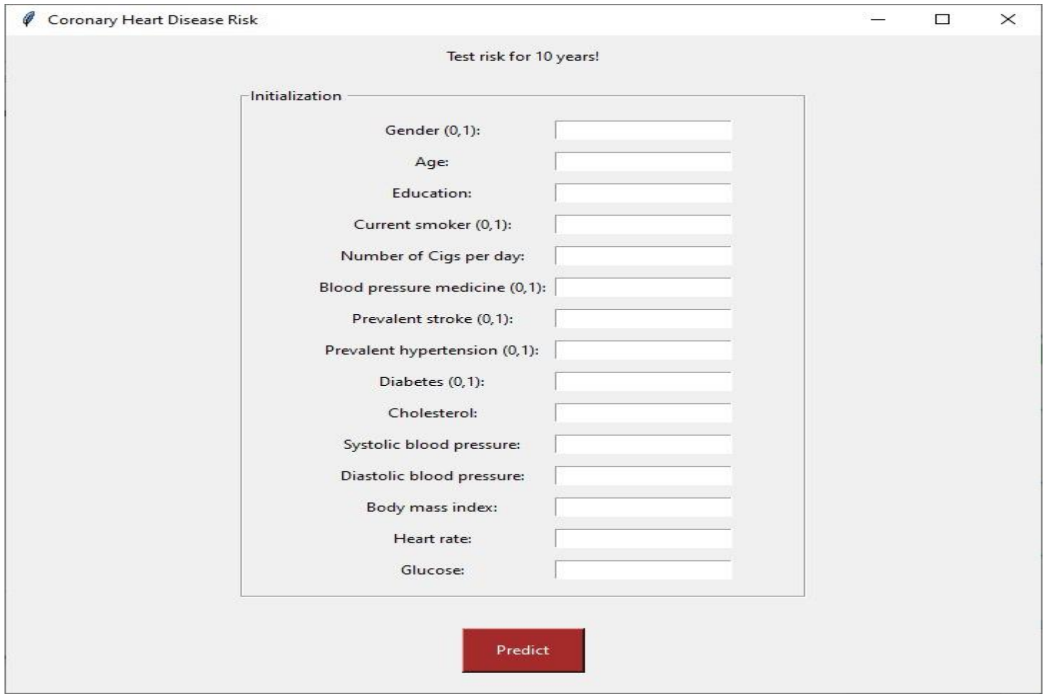
```
from tensorflow import keras
model = keras.Sequential()

# # First Hidden Layer
#model.add(keras.layers.Dense(32, input_shape= (15,), activation ='relu'))

model.add(keras.layers.Dense(1, input_shape= (15,), activation = 'sigmoid'
))
```

The results:-

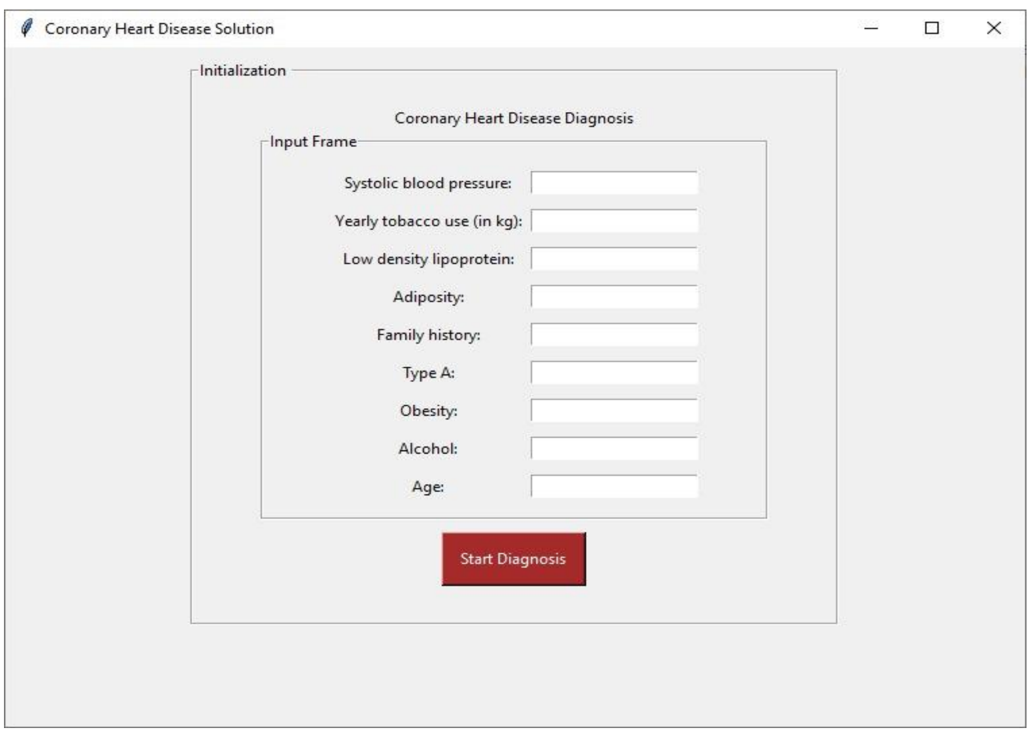
```
27/27 [==============================] - 0s 9ms/step - loss: 0.4241 - binary_accuracy: 0.8410 - val_loss: 0.3889 - val_binary_accuracy: 0.8679
Epoch 22/50
27/27 [==============================] - 0s 7ms/step - loss: 0.4232 - binary_accuracy: 0.8410 - val_loss: 0.3879 - val_binary_accuracy: 0.8691
Epoch 23/50
27/27 [==============================] - 0s 8ms/step - loss: 0.4224 - binary_accuracy: 0.8416 - val_loss: 0.3871 - val_binary_accuracy: 0.8691
Epoch 24/50
27/27 [==============================] - 0s 7ms/step - loss: 0.4216 - binary_accuracy: 0.8413 - val_loss: 0.3863 - val_binary_accuracy: 0.8691
Epoch 25/50
27/27 [==============================] - 0s 8ms/step - loss: 0.4208 - binary_accuracy: 0.8410 - val_loss: 0.3856 - val_binary_accuracy: 0.8691
Epoch 26/50
27/27 [==============================] - 0s 7ms/step - loss: 0.4202 - binary_accuracy: 0.8413 - val_loss: 0.3849 - val_binary_accuracy: 0.8691
Epoch 27/50
27/27 [==============================] - 0s 9ms/step - loss: 0.4195 - binary_accuracy: 0.8416 - val_loss: 0.3842 - val_binary_accuracy: 0.8691
Epoch 28/50
27/27 [==============================] - 0s 7ms/step - loss: 0.4189 - binary_accuracy: 0.8416 - val_loss: 0.3836 - val_binary_accuracy: 0.8691
Epoch 29/50
27/27 [==============================] - 0s 9ms/step - loss: 0.4183 - binary_accuracy: 0.8413 - val_loss: 0.3830 - val_binary_accuracy: 0.8691
```



CUI



Coronary Heart Disease Risk
Test risk for 10 years!
Initialization
Gender (0,1):
Age:
Education:
Current smoker (0,1):
Number of Cigs per day:
Blood pressure medicine (0,1):
Prevalent stroke (0,1):
Prevalent hypertension (0,1):
Diabetes (0,1):
Cholesterol:
Systolic blood pressure:
Diastolic blood pressure:
Body mass index:
Heart rate:
Glucose:
Predict



Coronary Heart Disease Solution
Initialization
Coronary Heart Disease Diagnosis
Input Frame
Systolic blood pressure:
Yearly tobacco use (in kg):
Low density lipoprotein:
Adiposity:
Family history:
Type A:
Obesity:
Alcohol:
Age:
Start Diagnosis

Conclusion

In this project we used two datasets for CHD to diagnosis and prediction for 10 years. First the previous projects maximum accuracy was 69%using traditional Machine Learning models. we managed to increase it using (SVM) traditional Machine Learning to 82.8, and by applying neural network to the model the accuracy did not exceed 79% that mean that the dataset is not complicated enough to apply MLP.

The second dataset we used to get higher percentage, by using traditional Machine Learning models the (SVM) got high score of 87.45% and by applying neural network to the model the accuracy did not exceed 86.6

References

https://www.kaggle.com/code/camillevleonard/bayesian-model-averaging-logistic-regression

https://www.kaggle.com/code/bmfeciura/bayesian-model-averaging-logistic-regression

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3477961/

https://www.kaggle.com/code/yassinehamdaoui1/heart-attack-ensembles-dimensionality-reduction/notebook

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html

https://keras.io/api/optimizers/