International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Heart Disease Prediction: Artificial Intelligence / Machine Learning

Dr Yasin Bouanani

Abstract: ***In this article I will discuss the use of k-Nearest Neighbors (k-NN) algorithms for prediction of heart disease using a medical data set and which will constitute our basic essential data for machine learning in this study. In order to make the machine learn to be able to determine whether the patient will be prone to a heart disease or not.***

Keywords: Machine learning, Artificial Intelligence, Heart Disease, Prediction algorithms, K-NN, SVM KERNEL, Scikit-Learn

1. Introduction

K-NN is a very simple algorithm, easy to understand, versatile and one of the most advanced in machine learning. KNN is used in various applications such as finance, health, political science, handwriting detection, image recognition and video recognition.

In this article we will discuss the prediction of cardiac attacks with the Scikit Learn library which is a Python free library for machine learning, this library that includes functions is used for different algorithms such as Simple Linear Regresions, Multiple, Random Forets, knn, svm kernel, svm linear and Decision tree. Also we will use the Python language because this library is written in python which will allow us a performance optimization.

In data mining, classification is a supervised learning that can be used to design models describing important data classes.

In our case, KNN is a simple classifier, in which samples are ranked according to the class of their nearest neighbor. Medical databases are large volumes. If the dataset contains redundant and irrelevant attributes, the classification may produce less accurate results. Heart disease is the leading cause of death in some countries.

Therefore, it is necessary to define a decision support system that helps clinicians decide to take precautionary measures. In this article, I propose the K-NN algorithm for efficient classification; this algorithm will improve the accuracy of diagnosis of heart disease. The application of this K-NN algorithm will be done on a dataset recovered and offered free of charge by the Kaggle website.

K-NN is one of the simplest of all supervised machine learning algorithms. It simply calculates the distance of a new data point to all other learning data points. In this article we will discuss some essential points and we will rely mainly on the application of this algorithm on the given game in a practical way to provide an effective solution for the prediction of heart disease that some countries are very seriously affected.

2. Theory

The KNN algorithm that we will use in this article, simply calculates the distance of a new data point to all other learning data points. The distance can be of any type, for example Euclidean or Manhattan, etc. It then selects the nearest K data points, K being any integer. Finally, it assigns the data point to the class to which most of the K data points belong.

Let's see this algorithm in action using a simple example. Suppose you have a dataset with two variables that, when plotted, looks like the following figure.

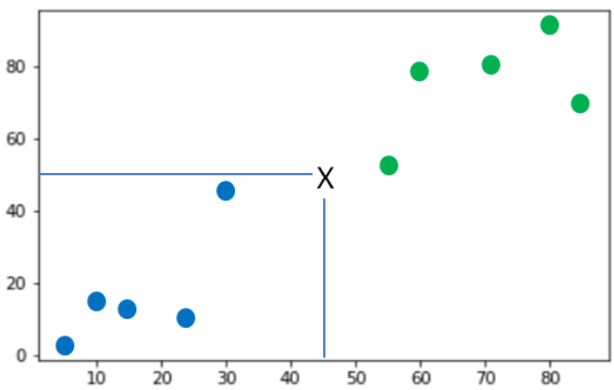


Figure 1: Two-variable data set

Our task is to classify a new data point with 'X' in the "blue" class or the "green" class. The coordinates of the data point are x = 45 and y = 50. Suppose that the value of K is 3. The KNN algorithm begins by calculating the distance of point X from all points. It then finds the 3 closest points with the least distance at point X. This is illustrated in the figure below. The three closest points were circled.

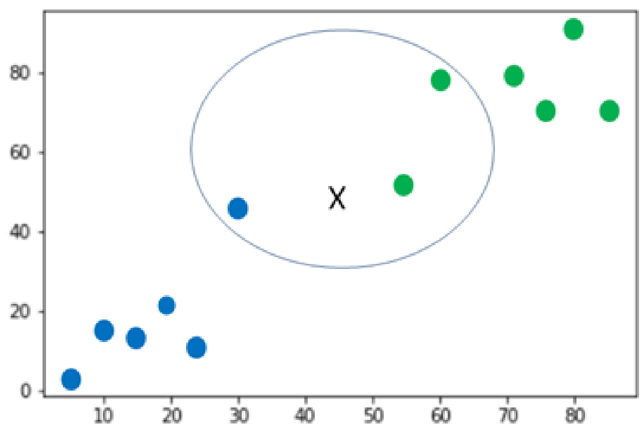


Figure 2.The three closest points

The last step of the KNN algorithm is to assign a new point to the class to which most of the three closest points belong. In the figure above, we can see that the two of the three closest points belong to the "green" class, while one belongs to the "Blue" class. Therefore, the new data point will be classified as "green".

3. Application

Using the Scikit-Learn library and implementation of the K-NN

Algorithm

In this section, we'll see how the Python Scikit-Learn library can be used to implement the K-NN algorithm in our dataset.

The Dataset:

We will use the famous dataset uploaded on the Kaggle website (https: //www.kaggle.com/ronitf/heart-disease-uci) for our KNN algorithm example. This dataset has 14 attributes: 1.**age**, 2.**sex**, 3.**cp**: type of chest pain (4 values), 4.**trestbps**: resting blood pressure, 5. **Chol**: serum cholesterol in mg/dl, 6.**fbs:** fasting glucose in mg / dl, 7. **restecg**: electrocardiographic results at rest (valeurs 0, 1, 2), 8. **Thalach:** Maximum heart frequency reached, 9.**exang:** Exercise-induced angina pectoris, 10. **oldpeak** = ST depression induced by exercise versus rest, 11. **Slope:** slope of the maximum exercise segment ST, 12. **ca:** number of main vessels (0-3) stained by fluorescence, 13. **thal**: 3 = normal; 6 = fixed fault; 7 = reversible defect, 14. **Target**: (1: Heart Disease Prediction, 0: No heart Disease prediction)

Our dataset looks like this:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
147	70	1	1	156	245	0	0	143	0	0	2	0	2	1
148	44	0	2	118	242	0	1	149	0	0.3	1	1	2	1
149	60	0	3	150	240	0	1	171	0	0.9	2	0	2	1
150	44	1	2	120	226	0	1	169	0	0	2	0	2	1
151	42	1	2	130	180	0	1	150	0	0	2	0	2	1
152	66	1	0	160	228	0	0	138	0	2.3	2	0	1	1
153	71	0	0	112	149	0	1	125	0	1.6	1	0	2	1
154	64	1	3	170	227	0	0	155	0	0.6	1	0	3	1
155	66	0	2	146	278	0	0	152	0	0	1	1	2	1
156	39	0	2	138	220	0	1	152	0	0	1	0	2	1
157	58	0	0	130	197	0	1	131	0	0.6	1	0	2	1
158	47	1	2	130	253	0	1	179	0	0	2	0	2	1
159	35	1	1	122	192	0	1	174	0	0	2	0	2	1
160	58	1	1	125	220	0	1	144	0	0.4	1	4	3	1
161	56	1	1	130	221	0	0	163	0	0	2	0	3	1
162	56	1	1	120	240	0	1	169	0	0	0	0	2	1
163	55	0	1	132	342	0	1	166	0	1.2	2	0	2	1
164	41	1	1	120	157	0	1	182	0	0	2	0	2	1
165	38	1	2	138	175	0	1	173	0	0	2	4	2	1
166	38	1	2	138	175	0	1	173	0	0	2	4	2	1
167	67	1	0	160	286	0	0	108	1	1.5	1	3	2	0
168	67	1	0	120	229	0	0	129	1	2.6	1	2	3	0
169	62	0	0	140	268	0	0	160	0	3.6	0	2	2	0
170	63	1	0	130	254	0	0	147	0	1.4	1	1	3	0
171	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
172	56	1	2	130	256	1	0	142	1	0.6	1	1	1	0
173	48	1	1	110	229	0	1	168	0	1	0	0	3	0
174	58	1	1	120	284	0	0	160	0	1.8	1	0	2	0
175	58	1	2	132	224	0	0	173	0	3.2	2	2	3	0
176	60	1	0	130	206	0	0	132	1	2.4	1	2	3	0
177	40	1	0	110	167	0	0	114	1	2	1	0	3	0
178	60	1	0	117	230	1	1	160	1	1.4	2	2	3	0
179	64	1	2	140	335	0	1	158	0	0	2	0	2	0

Figure 3: Dataset of 303 rows

Before exploiting this k-NN algorithm, it is essential to import some modules needed for its implementation:

3.1 Modules import

I proceed by importing these modules which are as follows:

```
# Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
from sklearn.neighbors import KNeighborsClassifier
```

Figure 4: Modules import

Some definitions:

import numpyasnp:
Allows to work with all functions present in the module.

import pandas aspd: Pandas has two data structures for Data Storage (Series, Dataframe)

import matplotlib.pyplotasplt:
Is a set of functions of command style that allows matplotlib to function as MATLAB, for example, create a figure, create a plot area in a figure, draws lines in a plot area, decorates the layout with labels

frommatplotlibimportrcParams:
An instance of RcParams to manage the default values of matplotlib.

from matplotlib.cm import rainbow:
supports a wide range of color tables, each of whichtranslates numeric data values into visible colors in a path

fromsklearn.neighborsimportKNeighborsClassifier:

Paper ID: ART20201082 DOI: 10.21275/ART20201082

import of classifier implementing the vote of the nearest k-neighbors.

3.2Import dataset and change to variable:

In this part we import our dataset in CSV format and put it in memory in a variable df.

df = pd.read_csv ('dataset4.csv')

We obtain the following result:

Nom	Type	Taille	Valeur
df	DataFrame	(303, 14)	Column names: age, sex, cp, trestbps, chol, fbs, restecg, thalach, exa ...

Figure 5: Passing the dataset in memory in the variable df

df - DataFrame

Index	age	sex	cp	trestbps	chol	fbs	res
0	63	1	3	145	233	1	0
1	37	1	2	130	250	0	1
2	41	0	1	130	204	0	0
3	56	1	1	120	236	0	1
4	57	0	0	120	354	0	1
5	57	1	0	140	192	0	1
6	56	0	1	140	294	0	0
7	44	1	1	120	263	0	1
8	52	1	2	172	199	1	1
9	57	1	2	150	168	0	1
10	54	1	0	140	239	0	1
11	48	0	2	130	275	0	1
12	49	1	1	130	266	0	1

Format | Ajuster | Couleur de fond | Colonne min/max | Enregistrer et Fermer | Fermer

Figure 6: Display of data and appearance of all attributes in dataframe: df

3.3 Checking the rows of the dataset

We check if the lines are not empty by the function: df.info ().

We get the following results in the terminal of spider environment:

```
In [2]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
target      303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.2 KB
```

Figure 7: Display of the results data

Analyzing this result, we find that there are 303 lines of data, as we can see it in the output, the summary includes the list of all the columns with their data types and the number of non-null values in each column we also have the range index value provided for the index axis.

3.4Summary statistics of the dataset:

Typing the method: df.describe () we get the following results:

describe (): is a method used to display some basic statistical details such as percentages, mean, standard values, etc.

```
In [3]: df.describe()
Out[3]:
              age         sex          cp  ...          ca        thal      target
count  303.000000  303.000000  303.000000  ...  303.000000  303.000000  303.000000
mean    54.366337    0.683168    0.966997  ...    0.729373    2.313531    0.544554
std      9.082101    0.466011    1.032052  ...    1.022606    0.612277    0.498835
min     29.000000    0.000000    0.000000  ...    0.000000    0.000000    0.000000
25%     47.500000    0.000000    0.000000  ...    0.000000    2.000000    0.000000
50%     55.000000    1.000000    1.000000  ...    0.000000    2.000000    1.000000
75%     61.000000    1.000000    2.000000  ...    1.000000    3.000000    1.000000
max     77.000000    1.000000    3.000000  ...    4.000000    3.000000    1.000000

[8 rows x 14 columns]
```

Figure 8: Display of statistical data

3.5 Determination of correlations between data

In this part we will use Seaborn which is a visualization library of Python data based on matplotlib. Execute code below to display the correlation matrix to analyze which attributes have more influence on having a heart disease.

```
#get correlations of each features in dataset
import seaborn as sns
corrmat = df.corr ()
top_corr_features = corrmat.index
plt.figure (figsize= (20, 20))
#plot heat map
g=sns.heatmap (df[top_corr_features].corr (), annot=True, cmap="RdYlGn")
```

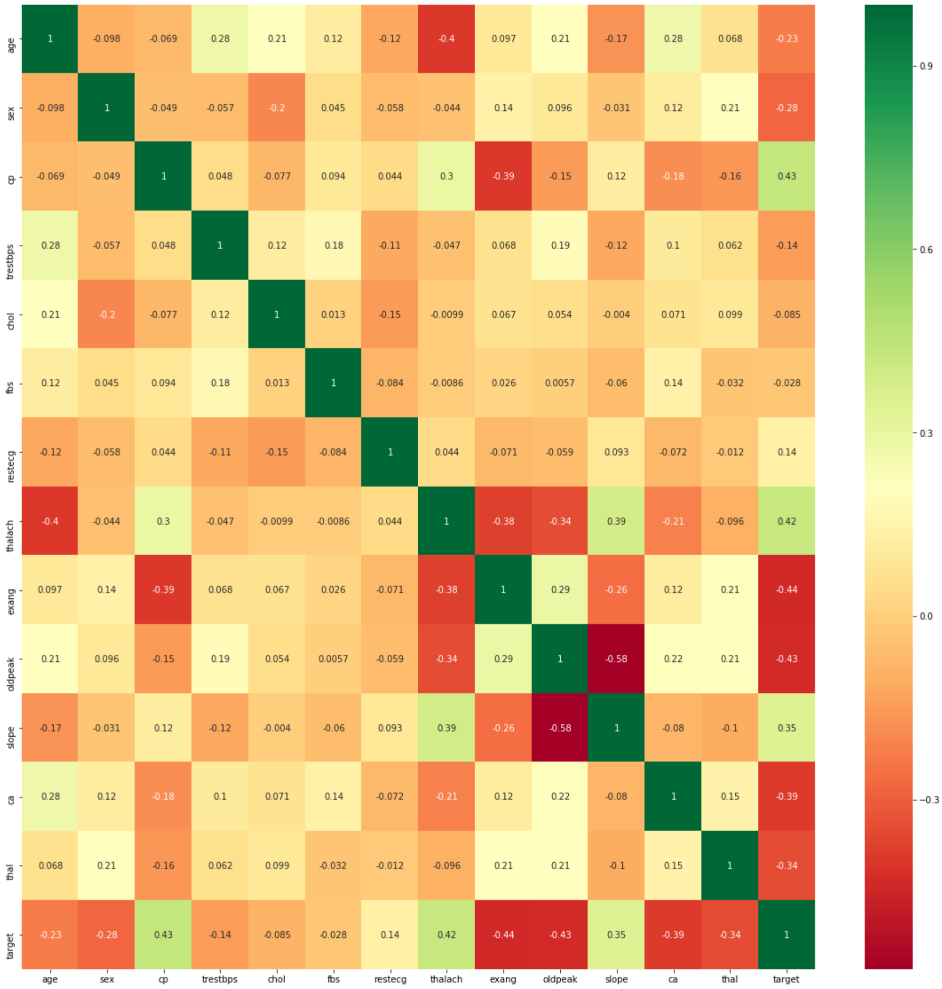


Figure 9: Correlation Matrix

You can also type df.corr () and we get the followingresult:

```
In [13]: df.corr()
Out[13]:
               age       sex        cp  ...        ca      thal    target
age       1.000000 -0.098447 -0.068653  ...  0.276326  0.068001 -0.225439
sex      -0.098447  1.000000 -0.049353  ...  0.118261  0.210041 -0.280937
cp       -0.068653 -0.049353  1.000000  ... -0.181053 -0.161736  0.433798
trestbps  0.279351 -0.056769  0.047608  ...  0.101389  0.062210 -0.144931
chol      0.213678 -0.197912 -0.076904  ...  0.070511  0.098803 -0.085239
fbs       0.121308  0.045032  0.094444  ...  0.137979 -0.032019 -0.028046
restecg  -0.116211 -0.058196  0.044421  ... -0.072042 -0.011981  0.137230
thalach  -0.398522 -0.044020  0.295762  ... -0.213177 -0.096439  0.421741
exang     0.096801  0.141664 -0.394280  ...  0.115739  0.206754 -0.436757
oldpeak   0.210013  0.096093 -0.149230  ...  0.222682  0.210244 -0.430696
slope    -0.168814 -0.030711  0.119717  ... -0.080155 -0.104764  0.345877
ca        0.276326  0.118261 -0.181053  ...  1.000000  0.151832 -0.391724
thal      0.068001  0.210041 -0.161736  ...  0.151832  1.000000 -0.344029
target   -0.225439 -0.280937  0.433798  ... -0.391724 -0.344029  1.000000

[14 rows x 14 columns]
```

Figure 10: Correlation Matrix displayed in terminal

Following these results we find that there is a strong correlation between **cp** and **target** because the value is at **0.433798** as the graphically shows the matrix also, there is a strong correlation between **thalach** and **target (0.421741)**

3.6Tracing of histograms

In this part it is possible and interesting to make appear the histograms of our data by typing df.hist ():

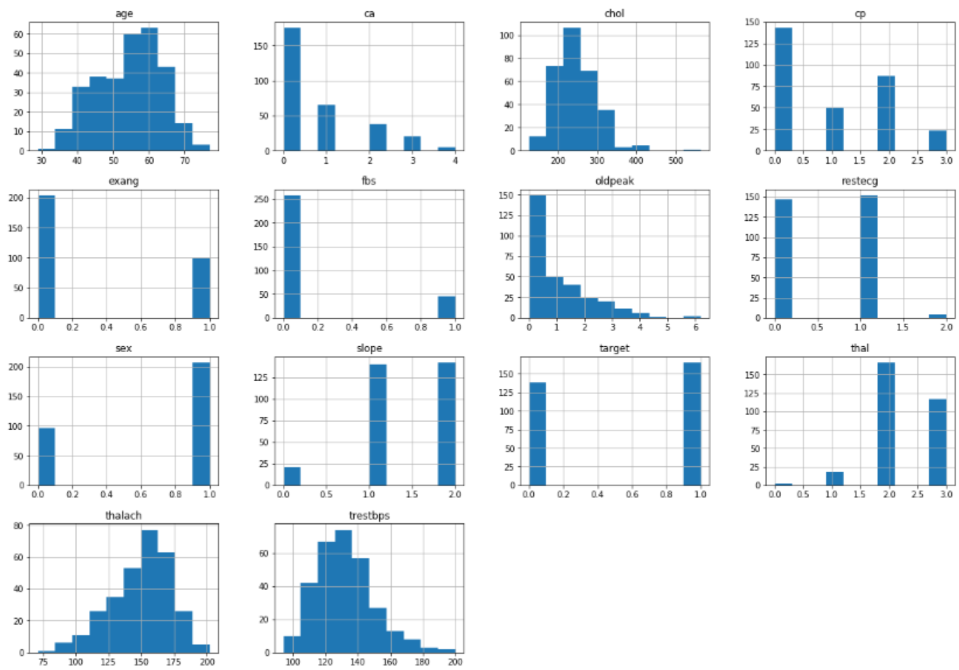


Figure 11: Histograms

From these histograms we find that on the Target graph men are more prone to Hearth disease which the number is more than 160 persons.

Let's execute the following code:
sns.set_style ('whitegrid')
sns.countplot (x='target', data=df, palette='RdBu_r')

Result

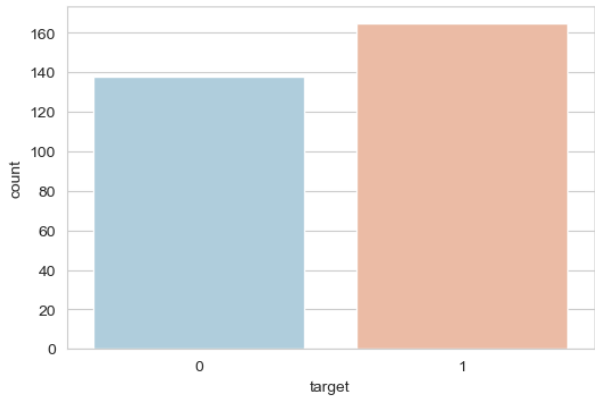


Figure 12: 0: No Heart Disease, **1**: Heart Disease

3.7 Notions of Dummies Variables

In the field of machine learning and to prepare the data set for predictions I will implement this data using Scikit-learn and pyton by creating variable dummies. A dummy variable (or dummy variable) is a numeric variable representing categorical data, such as in our case **sex, cp, fbs, restecg, exang, slope, ca, thal.**

Their range of values is small; they can only take two quantitative values or three. In practice, are easier to interpret when the variables are limited to two specific values, 1 or 0. As a rule, 1 represents the presence of a qualitative attribute and 0, absence.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
2	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
3	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
4	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
5	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
6	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
7	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
8	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
9	44	1	1	120	263	0	1	173	0	0	2	0	3	1
10	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
11	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
12	54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
13	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
14	49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
15	64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
16	58	0	3	150	283	1	0	162	0	1	2	0	2	1
17	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
18	58	0	2	120	340	0	1	172	0	0	2	0	2	1
19	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
20	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
21	69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
22	59	1	0	135	234	0	1	161	0	0.5	1	0	3	1
23	44	1	2	130	233	0	1	179	1	0.4	2	0	2	1

Figure 13: Selection and analysis of dummiesvariable

Let's transform this data into variable dummies and apply this script using scikit-learn and Python:

dataset = pd.get_dummies (df, columns = [**'sex'**, **'cp'**, **'fbs'**, **'restecg'**, **'exang'**, **'slope'**, **'ca'**, **'thal'**])

In this script we create a dataset variable that will transform the above data into variable dummies from the df variable in which the data is stored.

dataset	DataFrame	(303, 31)	Column names: age, trestbps, target, sex_0, se ...

Figure14: Apparition de la variable dataset avec les Dummies variables

On verifie notre dataset s'il contient les dummies variables

dataset - DataFrame

Index	halach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	cp_2	cp_3	fbs_0	fb
0		2.3	1	0	1	0	0	0	1	0	1
1		3.5	1	0	1	0	0	1	0	1	0
2		1.4	1	1	0	0	1	0	0	1	0
3		0.8	1	0	1	0	1	0	0	1	0
4		0.6	1	1	0	1	0	0	0	1	0
5		0.4	1	0	1	1	0	0	0	1	0
6		1.3	1	1	0	0	1	0	0	1	0
7		0	1	0	1	0	1	0	0	1	0
8		0.5	1	0	1	0	0	1	0	0	1
9		1.6	1	0	1	0	0	1	0	1	0
10		1.2	1	0	1	1	0	0	0	1	0
11		0.2	1	1	0	0	0	1	0	1	0
12		0.6	1	0	1	0	1	0	0	1	0
13		1.8	1	0	1	0	0	0	1	1	0

Format | Ajuster | Couleur de fond | Colonne min/max | Enregistrer et Fermer | Fermer

Figure 15: Appearance of Variable Dummies

We find that the variables that had 3 values turn into three possibilities, for example the attribute **cp_0, cp_1, cp_2**, when one is at 1 the others are at 0.

4. Dataset Scaling (Normalization)

The normalizaion (Scaling) of a dataset is a common requirement for many machine learning estimators: they may behave incorrectly if the individual entities do not resemble more or less standard data normally distributed.

Normalization will be applied to the **age**, **trestbps**, **chol**, **thalach**, and **oldpeak** columns because their values are large enough for the dataset data scale.

Apply the normalization:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler ()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[columns_to_scale] = standardScaler.fit_transform (dataset[columns_to_scale])
Resultat: valeurssontnormalisées
```

dataset - DataFrame

Index	age	trestbps	chol	thalach	oldpeak
0	0.952197	0.763956	-0.256334	0.0154428	1.08734
1	-1.91531	-0.0927378	0.0721995	1.63347	2.12257
2	-1.47416	-0.0927378	-0.816773	0.977514	0.310912
3	0.180175	-0.663867	-0.198357	1.2399	-0.206705
4	0.290464	-0.663867	2.08205	0.583939	-0.379244
5	0.290464	0.478391	-1.04868	-0.0720182	-0.551783
6	0.180175	0.478391	0.922521	0.146634	0.224643
7	-1.14329	-0.663867	0.323431	1.02124	-0.896862
8	-0.26098	2.306	-0.9134	0.540209	-0.465514
9	0.290464	1.04952	-1.51249	1.06497	0.483451
10	-0.0404028	0.478391	-0.140381	0.452748	0.138373
11	-0.702136	-0.0927378	0.555337	-0.465593	-0.724323
12	-0.591847	-0.0927378	0.381407	0.933783	-0.379244
13	1.06249	-1.235	-0.681494	-0.24694	0.65599
14	0.400752	1.04952	0.70994	0.540209	-0.0341662
15	-0.481558	-0.663867	-0.52689	0.365287	0.483451
16	0.400752	-0.663867	1.81149	0.977514	-0.896862
17	1.28306	1.04952	-0.391612	-1.55886	1.34615
18	-1.25358	1.04952	0.014223	0.933783	0.397182
19	1.61393	0.478391	-0.140381	0.0591733	0.65599

Format | Ajuster | ☑ Couleur de fond | ☑ Colonne min/max | Enregistrer et Fermer | Fermer

Figure16: Normalization of values

4.1 Création des variables X et y

In this dataset to teach our Machine we have to divide these data into two variables X and y, where X is the matrix that contains all the fields except the column of 'Target' and will represent the variable of the field column ' Target 'which is the vector, the objective is to distribute the variable X in two data sets in training_set and test_set with a percentage for the training _set of 80% and 20% for the test set, it means that the machine will do a learning with 80% of data and will be tested with the 20%. For Variable y, it will also be divided into two part training_set to 80% and the other which is the dataset test_set to 20%.

Creation of X and y variables:

```
y = dataset['target']
X = dataset.drop (['target'], axis = 1)
```

After creating the variables X and y we get these results:

X	DataFrame	(303, 30)	Column names: age, trestbps, chol, thalach, oldpeak, sex_0, sex_1, cp_ ...
dataset	DataFrame	(303, 31)	Column names: age, trestbps, chol, thalach, oldpeak, target, sex_0, se ...
y	Series	(303,)	Series object of pandas.core.series module

Figure 17: Création of y and X variables

y - Series

Index	target
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1

Figure 18: Variable y which represents the column 'Target'

X - DataFrame

Index	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	
0	0.952197	0.763956	-0.256334	0.0154428	1.08734	0	1	0	0
1	-1.91531	-0.0927378	0.0721995	1.63347	2.12257	0	1	0	0
2	-1.47416	-0.0927378	-0.816773	0.977514	0.310912	1	0	0	1
3	0.180175	-0.663867	-0.198357	1.2399	-0.206705	0	1	0	1
4	0.290464	-0.663867	2.08205	0.583939	-0.379244	1	0	1	0
5	0.290464	0.478391	-1.04868	-0.0720182	-0.551783	0	1	1	0
6	0.180175	0.478391	0.922521	0.146634	0.224643	1	0	0	1
7	-1.14329	-0.663867	0.323431	1.02124	-0.896862	0	1	0	1
8	-0.26098	2.306	-0.9134	0.540209	-0.465514	0	1	0	0
9	0.290464	1.04952	-1.51249	1.06497	0.483451	0	1	0	0
10	-0.0404028	0.478391	-0.140381	0.452748	0.138373	0	1	1	0
11	-0.702136	-0.0927378	0.555337	-0.465593	-0.724323	1	0	0	0
12	-0.591847	-0.0927378	0.381407	0.933783	-0.379244	0	1	0	1
13	1.06249	-1.235	-0.681494	-0.24694	0.65599	0	1	0	0

Figure 19: Variable X that contains all columns except the 'Target' column

4.2 Creating Learning Datasets

The goal of creating these datasets is to divide tablesor matrix in random learning and testing subsets.

In this case, the solution is obviously to divide the dataset you have into two sets, one for training and the other for testing; and you do it before you start training your model.

We will now divide the X dataset into two distinct sets: X_train and X_test. Similarly, we will also split the dataset there into two: y_train and y_test using the sklearn library attached to the code to execute:

```
fromsklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.20)
```

As you can see from the code, we divided the dataset into a ratio of 80 to 20, which is a common practice in data science.

Results obtained after separation of data sets:

Nom	Type	Taille	Valeur
X	DataFrame	(303, 30)	Column names: age, trestbps, chol, thalach, oldpeak, sex_0, sex_1, cp_ ...
X_test	DataFrame	(61, 30)	Column names: age, trestbps, chol, thalach, oldpeak, sex_0, sex_1, cp_ ...
X_train	DataFrame	(242, 30)	Column names: age, trestbps, chol, thalach, oldpeak, sex_0, sex_1, cp_ ...
columns_to_scale	list	5	['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
cormat	DataFrame	(14, 14)	Column names: age, sex, cp, trestbps, chol, fbs, restecg, thalach, exa ...
dataset	DataFrame	(303, 31)	Column names: age, trestbps, chol, thalach, oldpeak, target, sex_0, se ...
df	DataFrame	(303, 14)	Column names: age, sex, cp, trestbps, chol, fbs, restecg, thalach, exa ...
y	Series	(303,)	Series object of pandas.core.series module
y_test	Series	(61,)	Series object of pandas.core.series module
y_train	Series	(242,)	Series object of pandas.core.series module

Figure 20: Variable X that contains all columns except the 'Target' column

Appearance of new variables: X_test, X_train, y_Test, y_train
Showing these variables, in the X_train we have 80% of data and in the X_test

We have 20% of data for learning in the y_test and in X_test.

X_test - DataFrame

Index	age	trestbps	chol	
126	-0.812425	-1.12077	-0.816773	-0.
259	-1.80502	-0.663867	-0.294985	1.4
197	1.39335	-0.378302	0.149501	0.5
209	0.511041	0.478391	-1.33856	0.5
181	1.17277	1.04952	-0.410938	-1.
22	-1.36387	0.478391	-0.391612	1.2
295	0.952197	0.478391	-1.14531	-0.
157	-2.13589	-0.549641	-1.04868	1.6
258	0.841908	1.04952	-0.0437534	0.1
280	-1.36387	0.24994	1.32836	-1.
108	-0.481558	-0.663867	-0.0437534	0.5
158	0.400752	-0.378302	-0.507565	-0.
81	-1.033	-0.206964	1.19308	0.8
122	-1.47416	-1.12077	0.420058	0.9
267	-0.591847	-0.778093	-1.87967	-1.

X_train - DataFrame

Index	age	trestbps	chol	thalach	oldp
85	1.39335	-0.949431	6.1404	0.452748	0.48345
279	0.731619	0.364165	-1.55114	-1.07782	2.20884
36	-0.0404028	0.192827	1.11578	0.890053	-0.8968
30	-1.47416	-1.52056	-0.932726	0.802592	-0.8968
58	-2.24618	-0.778093	-1.24193	1.06497	-0.8968
242	1.06249	0.763956	-0.662169	-0.771706	0.82852
133	-1.47416	-1.235	-0.217683	0.146634	-0.8968
257	-0.481558	0.706843	-0.894075	-1.03409	-0.1204
134	-1.47416	-0.321189	1.15443	0.583939	-0.8968
73	-0.371269	0.478391	0.28478	1.58974	-0.8968
286	0.511041	0.135714	-0.816773	0.540209	-0.2067
236	0.400752	-0.378302	1.03847	0.933783	-0.8968
60	1.83451	-1.235	0.362082	-0.859167	-0.8968
2	-1.47416	-0.0927378	-0.816773	0.977514	0.31091
6	0.180175	0.478391	0.922521	0.146634	0.22464

Figure 21: X_test and X_train datasets created

We also took 80% for y_train and 20% for y_test

y_train - Series

Index	target
34	1
107	1
46	1
116	1
291	0
108	1
267	0
228	0
215	0
142	1
8	1
144	1
1	1
85	1

y_test - Series

Index	target
120	1
172	0
176	0
292	0
122	1
82	1
17	1
63	1
92	1
218	0
91	1
231	0
235	0
15	1

Figure 22: Jeux de données y_test et y_train crées

We can check the dataset to see if the values have been normalized by executing the following method:
dataset.head ()

We obtain the following result:

International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

```
In [21]: dataset.head()
Out[21]:
        age  trestbps      chol   thalach  ...  thal_0  thal_1  thal_2  thal_3
0  0.952197  0.763956 -0.256334  0.015443  ...       0       1       0       0
1 -1.915313 -0.092738  0.072199  1.633471  ...       0       0       1       0
2 -1.474158 -0.092738 -0.816773  0.977514  ...       0       0       1       0
3  0.180175 -0.663867 -0.198357  1.239897  ...       0       0       1       0
4  0.290464 -0.663867  2.082050  0.583939  ...       0       0       1       0

[5 rows x 31 columns]
```

Figure 23: Display of the first 5 rows

It is found that all values have been normalized.

4.3 Determining the K-neighbors with the highest score:

We will import the cross_val_score library and create a score variable

Let's execute this script:

fromsklearn.model_selection import cross_val_score
knn_scores = [ ]
for k in range (1, 21):
knn_classifier = KNeighborsClassifier (n_neighbors = k)
score=cross_val_score (knn_classifier, X, y, cv=10)
knn_scores.append (score.mean ())

Then we draw the K-neighbors classifier curves:
Let's execute this script for the graphical plot and determination of K and the highest score:

plt.plot ([k for k in range (1, 21)], knn_scores, color = 'red')
for i in range (1, 21):
plt.text (i, knn_scores[i-1], (i, knn_scores[i-1]))
plt.xticks ([i for i in range (1, 21)])
plt.xlabel ('Number of Neighbors (K)')
plt.ylabel ('Scores')
plt.title ('K Neighbors Classifier scores for different K values')

Result after executing the script:

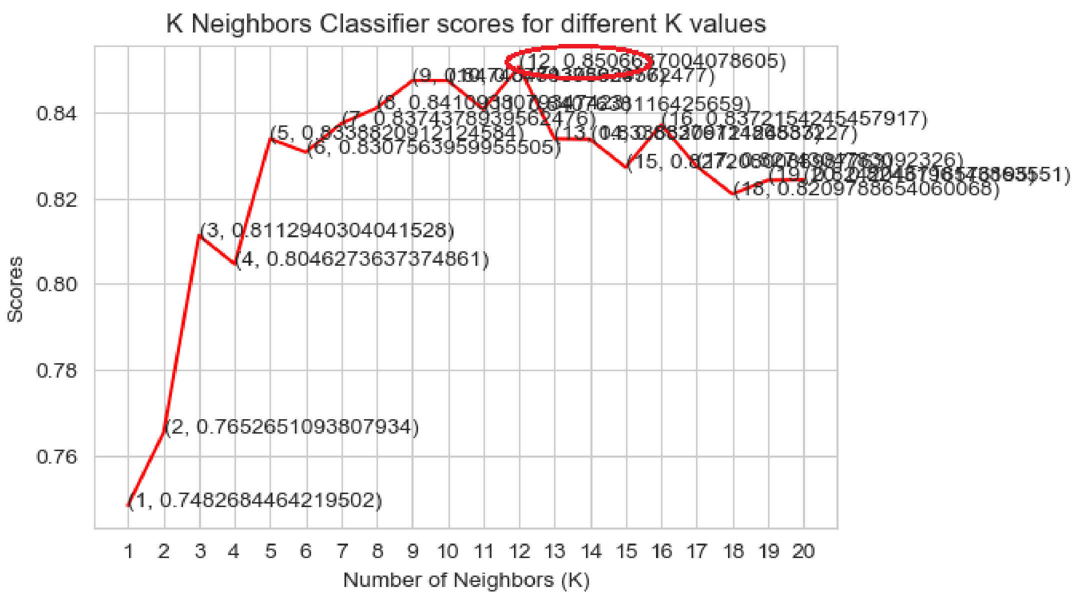


Figure 24: Determination of K which corresponds to the highest score

We get the value 12 for the highest score which is 0.85 so we get ascore of 0.85066 for a k = 12.

4.4 Implement a K-Neigbors classifier for K = 12:

Since we found the K -Neighbors equal to 12 corresponding to the score of 0.85, the **X_train** and **y_train** workout datasets are introduced into the nearest neighbor classifier.

#Application of classifier KNN with K=12
knn_classifier = KNeighborsClassifier (n_neighbors = 12)
knn_classifier.fit (X_train, y_train)

Message obtained in terminal of spider environment meaning that the classifier has been applied:

KNeighborsClassifier (algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=12, p=2, weights='uniform')

4.5 Prediction on X_test dataset

We create ay_pred variable with the following script:
y_pred = knn_classifier.predict (**X_test**)

Let's analyze the variables y_pred, and execute the following script:
print (y_pred)

We obtain the following prediction result:

```
In [33]: print(y_pred)
[1 0 0 0 0 1 0 1 1 0 1 1 1 1 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1
0 1 1 1
 1 1 0 1 0 0 0 1 0 0 1 1 0 0 1 1 1 0 1 0 1 0 0 1]
```

Figure 25: Prediction results after training for X_test

These results are the predictions of the X_test dataset set

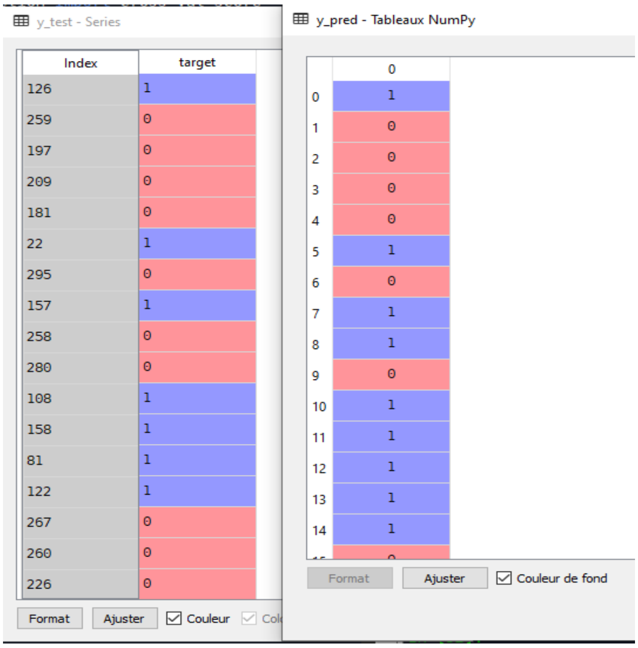


Figure 26: Comparison of y_pred with y_test

These values of 1 and 0 are the predictions that we obtained on the set ofX-test and we can compare the results of y_pred with y_test which is the already the existing dataset

5. Evaluation of our K-NN algorithm

Following our prediction on the test game X_test, we need to evaluate this algorithm, there are several evaluation methods, in this article I opted for the following evaluation method, we will import the report library classification and confusion matrix:

Let's run this script that will display the score and accuracy of our algorithm:

fromsklearn.metrics import classification_report, confusion_matrix
print (confusion_matrix (y_test, y_pred))
print (classification_report (y_test, y_pred))

We obtain the following result:

```
[[25  5]
 [ 4 27]]
              precision    recall  f1-score   support

           0       0.86      0.83      0.85        30
           1       0.84      0.87      0.86        31

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```

Figure 27: The score is 0.85, the precision is 0.86

The results show that our K-NN algorithm was able to classify the X_test set records with 85% accuracy, we can improve these results with a better score, we also have the confusion matrix as you see on the terminal image Figure (27):

$$\begin{bmatrix} 25 & 5 \\ 4 & 27 \end{bmatrix}$$

This confusion matrix gives us a 85.24% success rate

5.1 Error calculation for the different values of K between 1 and 40:

The purpose of the error calculation in this part is to find the K for the lowest error
Let's run this script:

error = [ ]

# Calculating error for K values between 1 and 40
for i in range (1, 40):
knn = KNeighborsClassifier (n_neighbors=i)
knn.fit (X_train, y_train)
pred_i = knn.predict (X_test)
error.append (np.mean (pred_i != y_test))

plt.figure (figsize= (12, 6))
plt.plot (range (1, 40), error, color='red', linestyle='dashed', marker='o', markerfacecolor='blue', markersize=10)
plt.title ('Error Rate K Value')
plt.xlabel ('K Value')
plt.ylabel ('Mean Error')

The result obtained:

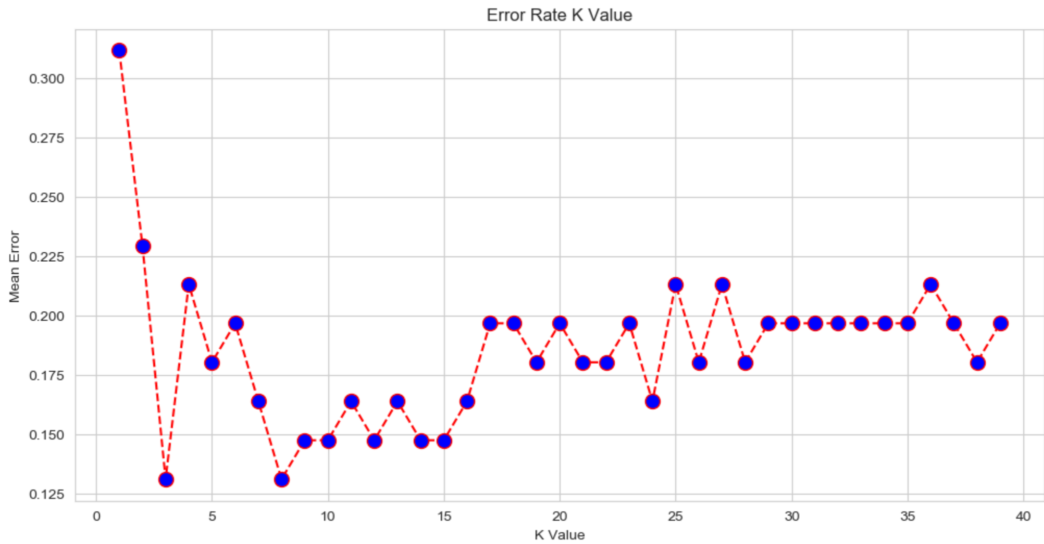


Figure 28: Determination of the lowest mean error that corresponds to K

On this graph we notice that for a low average error value we have a K = 8, it would be interesting to test our algorithm with this new value of K for evaluation.

5.2 Evaluation of the algorithm with K = 8:

We will evaluate our algorithm with K = 8 after classifier application:

```
#application of classifier KNN withK=8
knn_classifier = KNeighborsClassifier (n_neighbors = 8)
knn_classifier.fit (X_train, y_train)
```

Execution of this script also for evaluation:

```
fromsklearn.metrics import classification_report, confusion_matrix
print (confusion_matrix (y_test, y_pred))
print (classification_report (y_test, y_pred))
```

Obtained result:

```
[[26  4]
 [ 4 27]]
              precision    recall  f1-score   support

           0       0.87      0.87      0.87        30
           1       0.87      0.87      0.87        31

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

Figure 29: Matrix of Confusion, precision and score

We find that the confusion matrix is now:

$$\begin{bmatrix} 26 & 5 \\ 4 & 27 \end{bmatrix}$$

This confusion matrix gives us a percentage of 87%of success and accuracy of 87%

6. Predictions for medical dataset received

6.1 Conversion of data toa variable: datamedical

In this part we will receive a medical data set having all the attributes with their values except the 'Target' part which is the value to predict and test our algorithm because the learning has already been done.
Here is the file received to predict the heart disease of these patients:

It is very important that when receiving the data to be predicted, all the categorical variables must be present in the columns because during the transformation phase into Dummies variables no column should be missing.

File received for prediction: **medicalForPredict2.csv**

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
2	53	0	2	128	216	0	0	115	0	0	2	0	0
3	53	0	0	138	234	0	0	160	0	0	2	0	2
4	51	0	2	130	256	0	0	149	0	0.5	2	0	2
5	66	1	0	120	302	0	0	151	0	0.4	1	0	2
6	62	1	2	130	231	0	1	146	0	1.8	1	3	3
7	44	0	2	108	141	0	1	175	0	0.6	1	0	2
8	63	0	2	135	252	0	0	172	0	0	2	0	2
9	52	1	1	134	201	0	1	158	1	0.8	2	1	2
10	48	1	0	122	222	0	0	186	0	0	2	0	2
11	45	1	0	115	260	0	0	185	0	0	2	2	2
12	34	1	3	118	182	0	0	174	0	0	2	0	2
13	57	0	0	128	303	0	0	159	0	0	0	1	2
14	71	0	2	110	265	1	0	130	0	0	2	1	2
15	54	1	1	108	309	0	1	156	0	0	2	0	3
16	52	1	3	118	186	0	0	190	0	0	1	0	1
17	53	0	2	128	216	0	0	115	0	0	2	0	0
18	53	0	0	138	234	0	0	160	0	0	2	0	2
19	51	0	2	130	256	0	0	149	0	0.5	2	0	2
20	66	1	0	120	302	0	0	151	0	0.4	1	0	2
21	62	1	2	130	231	0	1	146	0	1.8	1	3	3
22	57	0	1	130	236	0	0	174	0	0	1	1	2
23	59	1	2	126	218	1	1	134	0	2.2	1	1	1
24	40	1	0	152	223	0	1	181	0	0	2	0	3
25	58	1	0	114	318	0	2	140	0	4.4	0	3	1
26	38	1	2	138	175	0	1	173	0	0	2	4	2
27													

Figure 30: Patient data set for prediction without Target column

In spider environment you type:
datamedical = pd.read_csv (**'medicalForPredict2.csv'**)
This step is used to pass the file into a datamedical variable

Result:

datamedical	DataFrame	(23, 13)	Column names: age, sex, cp, trestbps, chol, fbs, restecg, thalach, exa ...

Figure 31: Variable created: datamedical

datamedical - DataFrame

Index	age	sex	cp	trestbps	chol	fbs
0	59	1	2	126	218	1
1	40	1	0	152	223	0
2	61	1	0	140	207	0
3	46	1	0	140	311	0
4	59	1	3	134	204	0
5	57	1	1	154	232	0
6	57	1	0	110	335	0
7	55	0	0	128	205	0
8	61	1	0	148	203	0
9	58	1	0	114	318	0
10	58	0	0	170	225	1
11	67	1	2	152	212	0
12	44	1	0	120	169	0
13	63	1	0	140	187	0

Format | Ajuster | Couleur de fond | Colonne min/max | Enregistrer et Fermer | Fermer

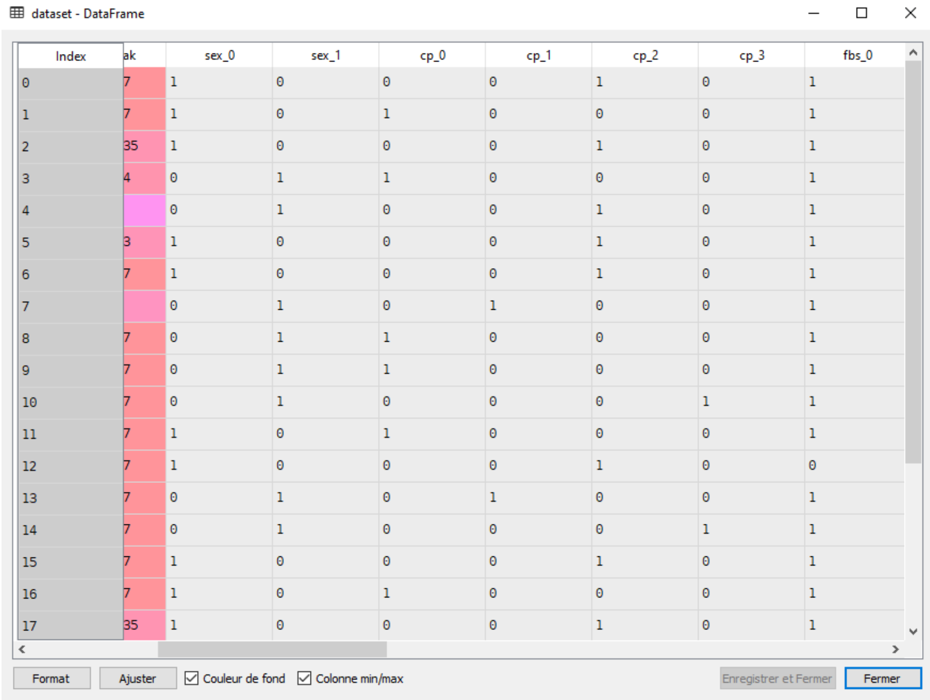
Figure 32: Apparition des nouvelles données de patients dans variable data medical

We display the data received by the medical staff to predict. These data must also be normalized and also pass the values into dummies variable so that the predictions are reliable.

6.2 Passing data into dummies variables

dataset = pd.get_dummies (**datamedical**, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
We obtain a new variable: **dataset** containing the dummies variable

International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426



dataset - DataFrame

Index	ak	sex_0	sex_1	cp_0	cp_1	cp_2	cp_3	fbs_0
0	7	1	0	0	0	1	0	1
1	7	1	0	1	0	0	0	1
2	35	1	0	0	0	1	0	1
3	4	0	1	1	0	0	0	1
4		0	1	0	0	1	0	1
5	3	1	0	0	0	1	0	1
6	7	1	0	0	0	1	0	1
7		0	1	0	1	0	0	1
8	7	0	1	1	0	0	0	1
9	7	0	1	1	0	0	0	1
10	7	0	1	0	0	0	1	1
11	7	1	0	1	0	0	0	1
12	7	1	0	0	0	1	0	0
13	7	0	1	0	1	0	0	1
14	7	0	1	0	0	0	1	1
15	7	1	0	0	0	1	0	1
16	7	1	0	1	0	0	0	1
17	35	1	0	0	0	1	0	1

Format Ajuster Couleur de fond Colonne min/max Enregistrer et Fermer Fermer

Figure 33: Appearance of Dummies variables

6.3 Data normalizations

Let's execute the same script seen before:

```
fromsklearn.model_selection import train_test_split
fromsklearn.preprocessing import StandardScaler
standardScaler = StandardScaler ()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[columns_to_scale] = standardScaler.fit_transform (dataset[columns_to_scale])
```

dataset - DataFrame

Index	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	
0	-0.0771644	0.199051	-0.516896	-2.08587	-0.532627	1	0	0	0
1	-0.0771644	1.15603	-0.108103	0.140509	-0.532627	1	0	1	0
2	-0.304118	0.390447	0.391533	-0.403716	-0.0357735	1	0	0	0
3	1.39804	-0.56653	1.43623	-0.304766	-0.135144	0	1	1	0
4	0.944129	0.390447	-0.176235	-0.552141	1.25605	0	1	0	0
5	-1.09846	-1.7149	-2.2202	0.882634	0.0635973	1	0	0	0
6	1.05761	0.868935	0.30069	0.734209	-0.532627	1	0	0	0
7	-0.190641	0.773237	-0.857557	0.041559	0.262339	0	1	0	1
8	-0.64455	-0.375135	-0.380632	1.42686	-0.532627	0	1	1	0
9	-0.984981	-1.04502	0.482376	1.37738	-0.532627	0	1	1	0
10	-2.23323	-0.757926	-1.28906	0.833159	-0.532627	0	1	0	0
11	0.376744	0.199051	1.45894	0.091034	-0.532627	1	0	1	0
12	1.96542	-1.52351	0.595929	-1.34374	-0.532627	1	0	0	0
13	0.0363127	-1.7149	1.5952	-0.057391	-0.532627	0	1	0	1
14	-0.190641	-0.757926	-1.19822	1.62476	-0.532627	0	1	0	0
15	-0.0771644	0.199051	-0.516896	-2.08587	-0.532627	1	0	0	0
16	-0.0771644	1.15603	-0.108103	0.140509	-0.532627	1	0	1	0
17	-0.304118	0.390447	0.391533	-0.403716	-0.0357735	1	0	0	0

Format Ajuster Couleur de fond Colonne min/max Enregistrer et Fermer Fermer

Figure 34: Data normalization

Now that we have created the variable dummies and normalize the data of the data received by the medical to predict, we can proceed to the prediction:

6.4 Let's apply the KNN classifier with K = 8:

```
knn_classifier = KNeighborsClassifier (n_neighbors = 8)
knn_classifier.fit (X_train, y_train)
```

We will create the variable y_pred with the passage in argument the dataset that interests us to predict:

```
y_pred = knn_classifier.predict (dataset)
```

Let's show the predictions of this dataset of patients received and transmitted by the medical team:

```
print (y_pred)
```

Results for all Patients

Predictions:
[1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1]

```
In [42]: print(y_pred)
[1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1]
```

Figure 35: Prediction results

We find that 20 patients are going to have a heart disease and 5 patients are not going to have heart disease, and our algorithm have 87% of accuracy (20 values = 1 and 5 values = 0)

7. Prediction by patient alone

Let's take the variable dataset and copy the first row of the patient 1

dataset - DataFrame

Index	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	cp_1	
0	-0.0771644	0.199051	-0.516896	-2.08587	-0.532627	1	0	0	0	1
1	-0.0771644	1.15603	-0.108103	0.140509	-0.532627	1	0	1	0	0
2	-0.304118	0.390447	0.391533	-0.403716	-0.0357735	1	0	0	0	1
3	1.39804	-0.56653	1.43623	-0.304766	-0.135144	0	1	1	0	0
4	0.944129	0.390447	-0.176235	-0.552141	1.25605	0	1	0	0	1
5	-1.09846	-1.7149	-2.2202	0.882634	0.0635973	1	0	0	0	1
6	1.05761	0.868935	0.30069	0.734209	-0.532627	1	0	0	0	1
7	-0.190641	0.773237	-0.857557	0.041559	0.262339	0	1	0	1	0
8	-0.64455	-0.375135	-0.380632	1.42686	-0.532627	0	1	1	0	0

Figure 36: Selection and copy of first line of patient 1 (index 0)

For this example, take the first line 0 of the 1st patient and copy the first line and place it as an argument of the following script:

knn_classifier.predict ([[-0.07716439054355188, 0.19905117369814235, -0.5168960856949255, -2.085865430252193, -0.532627428060773, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0]])

Write the following script to display the result for this patient:

print ('**The prediction is**: ', knn_classifier.predict ([[-.07716439054355188, 0.19905117369814235, -0.5168960856949255, -2.085865430252193, -0.532627428060773, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0]]))

8. Result Per Patient

The prediction is: [1]
[1]: means that this patient will have a heart disease and that preventive actions are necessary.

In conclusion, we can have predictions by patient alone or by group of patients.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heart Disease
2	53	0	2	128	216	0	0	115	0	0	2	0	0	1
3	53	0	0	138	234	0	0	160	0	0	2	0	2	1
4	51	0	2	130	256	0	0	149	0	0.5	2	0	2	1
5	66	1	0	120	302	0	0	151	0	0.4	1	0	2	1
6	62	1	2	130	231	0	1	146	0	1.8	1	3	3	0
7	44	0	2	108	141	0	1	175	0	0.6	1	0	2	1
8	63	0	2	135	252	0	0	172	0	0	2	0	2	1
9	52	1	1	134	201	0	1	158	1	0.8	2	1	2	0
10	48	1	0	122	222	0	0	186	0	0	2	0	2	1
11	45	1	0	115	260	0	0	185	0	0	2	2	2	1
12	34	1	3	118	182	0	0	174	0	0	2	0	2	1
13	57	0	0	128	303	0	0	159	0	0	0	1	2	1
14	71	0	2	110	265	1	0	130	0	0	2	1	2	1
15	54	1	1	108	309	0	1	156	0	0	2	0	3	1
16	52	1	3	118	186	0	0	190	0	0	1	0	1	1
17	53	0	2	128	216	0	0	115	0	0	2	0	0	1
18	53	0	0	138	234	0	0	160	0	0	2	0	2	1
19	51	0	2	130	256	0	0	149	0	0.5	2	0	2	1
20	66	1	0	120	302	0	0	151	0	0.4	1	0	2	1
21	62	1	2	130	231	0	1	146	0	1.8	1	3	3	0
22	57	0	1	130	236	0	0	174	0	0	1	1	2	1
23	59	1	2	126	218	1	1	134	0	2.2	1	1	1	0
24	40	1	0	152	223	0	1	181	0	0	2	0	3	1
25	58	1	0	114	318	0	2	140	0	4.4	0	3	1	0
26	38	1	2	138	175	0	1	173	0	0	2	4	2	1

Figure 37: Patient data with results after predictions

9. Conclusion

This K-NN algorithm could be very effective for prediction of heart disease; the diagnosis of heart disease in most cases depends on a complex combination of clinical and pathological data. Because of this complexity, health professionals and researchers are increasingly interested in accurate and accurate prediction of heart disease. In this article, I use this algorithm as a prediction system for heart disease that can help health professionals predict the state of heart disease based on clinical data of patients. The dataset used with the 14 attributes can be used with other attributes that can be added by the medical staff, plus we have relevant data plus our algorithm will be precise.

This algorithm and method can be used for prediction of heart disease and provide preventive actions to patients for their heart disease.

References

[1] Holte, Robert C. "Very simple classification rules perform well on most commonly used datasets." Machine learning 11.1 (1993): 63-90.
[2] A. Blum, J. Hopcroft and R. Kannan .Foundations of Data Science : An Introduction to Computational Learning Theory (M. Kearns and U. Vazirani)
[3] S. Shalev-Shwartz and S. Ben-David. Understanding Machine Learning: From Theory to Algorithms
[4] John D. Kelleher, Brian Mac Nameeet Aoife D'Arcy « Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies » par John D. Kelleher, Brian Mac Nameeet Aoife D'Arcy
[5] Ian H. Witten, Eibe Frank et Mark A. Hall .« Data Mining: Practical Machine Learning Tools and Techniques »
[6] O'Mahony C, Jichi F, Pavlou M, Monserrat L, Anastasakis A, Rapezzi C, Biagini E, Gimeno JR, Limongelli G, McKenna WJ, Omar RZ, Elliott PM. A novel clinical risk prediction model for sudden cardiac death in hypertrophic cardiomyopathy (HCM risk-SCD) Eur Heart J. 2014;35: 2010–2020.
[7] Guyon I, Elisseeff A. An introduction to variable and feature selection. J Mach Learn Res. 2003;3: 1157–1182.
[8] Clarke, B.S., Fokoué, E. & Zhang, H.H. (2009). Principles and Theory for Data Mining and Machine Learning. Springer Verlag.
[9] Giuseppe Bonaccorso .Machine Learning Algorithms

Author Profile

Yasin Bouanani, is a Professor with a Ph.D in Computer Sciences, currently personal Researcher in Morocco. His current researches focuses on 3D immersive Virtual Reality for analysis and learning and Artificial Intelligence in Machine Learning and Deep Learning, he has an MBA of the Conley College London.. His current research focuses on Artificial Intelligence in Machine Learning and Deep Learning