Published in DataDrivenInvestor

Adipta Martulandi Follow
Oct 22, 2019 · 7 min read · Listen

K-Nearest Neighbors in Python + Hyperparameters Tuning

Photo by Christian Stahl on Unsplash

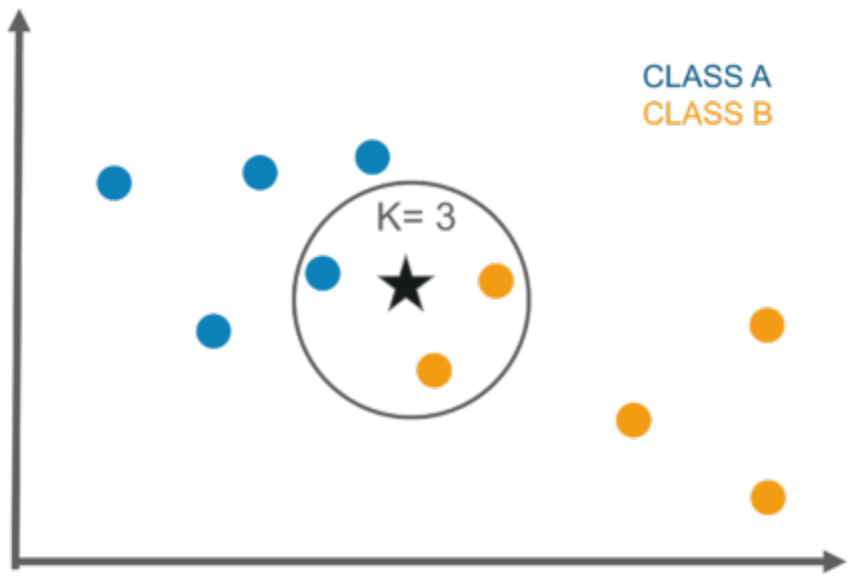
So actually **KNN** can be used for **Classification** or **Regression** problem, but in general, KNN is used for Classification Problems. Some applications of KNN are in **Handwriting Recognition**, **Satellite Image Recognition**, and **ECG Pattern Recognition**. This algorithm is very **simple** but is often used by Data Scientists.

Which is More Promising: Data Science or Software Engineering? | Data Driven Investor

About a month back, while I was sitting at a café and working on developing a website for a client, I found this woman...

www.datadriveninvestor.com

In Overview, the KNN algorithm works to **classify** new data based on its proximity to **K-neighbors** (training data). So if the new data is surrounded by training data that has Class 1, it can be concluded that the new data is included in Class 1. To make it easier to understand, see the **illustration below**.



Sources Edureka

can be said that star data is included in Class B. In more detail, how KNN works is as follows:

1. Determine the value of K.
The first step is to determine the value of K. The determination of the K value varies greatly depending on the case. If using the **Scikit-Learn** Library the default value of K is 5.

2. Calculate the distance of new data with training data.
To calculate distances, 3 distance metrics that are often used are **Euclidean** Distance, **Manhattan** Distance, and **Minkowski** Distance.

$$d_{euclidean} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \qquad d_{manhattan} = \sum_{i=1}^{n} |x_i - y_i| \qquad d_{minkowski} = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

Distances Formula

When you use Scikit-Learn, the default distance used is Euclidean. It can be seen in the Minkowski distance formula that there is a Hyperparameter p, if set p = 1 then it will use the Manhattan distance and p = 2 to be Euclidean.

3. Find the closest K-neighbors from the new data.
After calculating the distance, then look for K-Neighbors that are closest to the new data. If using K = 3, look for 3 training data that is closest to the new data.

4. New Data Class Prediction.
To determine the class of new data, select the class of training data that closest to the new data and have the highest quantity.

5. Evaluation.
Calculate the **accuracy** of the model, if the accuracy is still low, then this process can be

After knowing how KNN works, the next step is **implemented** in Python. I will use Python Scikit-Learn Library. The dataset I will use is a **heart dataset** in which this dataset contains characteristics of the patient whether the patient has heart disease or not.

> *To follow this tutorial, you should at least know about:*
> *1. Basic programming in **Python**.*
> *2. **Pandas** and **Numpy** libraries for data analysis tools.*
> *3. **Matplotlib** and **Seaborn** libraries for data visualization.*
> *4. **Scikit-Learn** Library for Machine Learning.*
> *5. **Jupyter Notebook**.*

The steps in solving the Classification Problem using KNN are as follows:
1. Load the **library**
2. Load the **dataset**
3. **Sneak peak** data
4. Handling **missing values**
5. Exploratory Data Analysis (EDA)
6. Modeling
7. Tuning Hyperparameters

> *Dataset and Full code can be downloaded at my **Github** and all work is done on Jupyter Notebook.*

1 Loading several libraries that will be used to do the analysis in this tutorial. I assume that you have already installed the library.

```
import pandas as pd
```

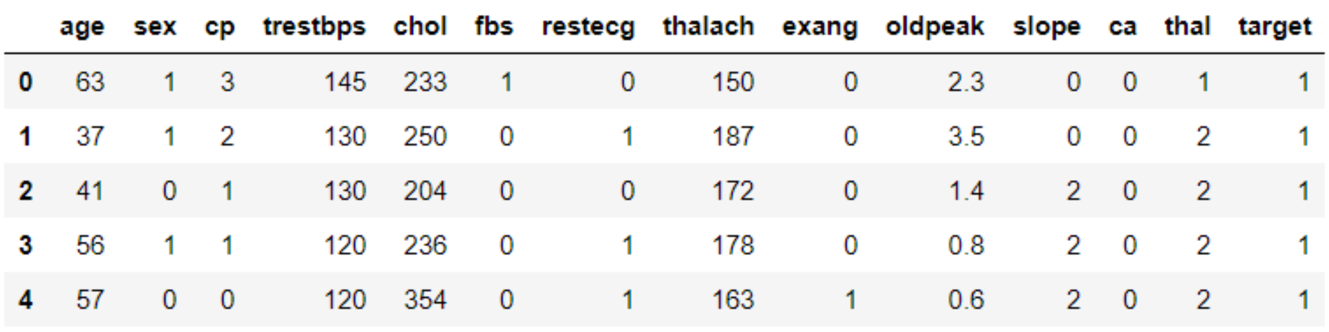
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
```

2 Load the dataset to be used, dataset contains historical data from patients who have been examined for heart disease.

```
df = pd.read_csv('heart.csv')
```

3 Let's see some general information from the data to be more familiar with our data.

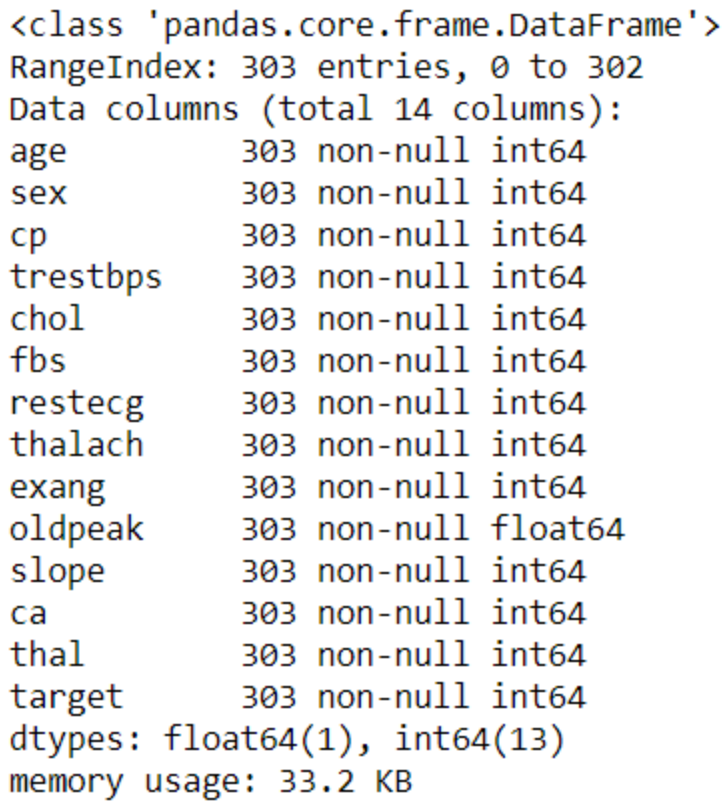
```
df.head()
```



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Top 5 Data

```
df.shape
df.info()
```

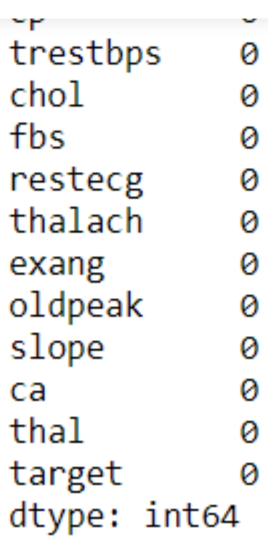


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     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
```

General Information of Data

4 Checking whether there is missing data or not, if not then it can proceed to the Exploratory Data Analysis (EDA) stage.

```
df.isnull().sum()
```

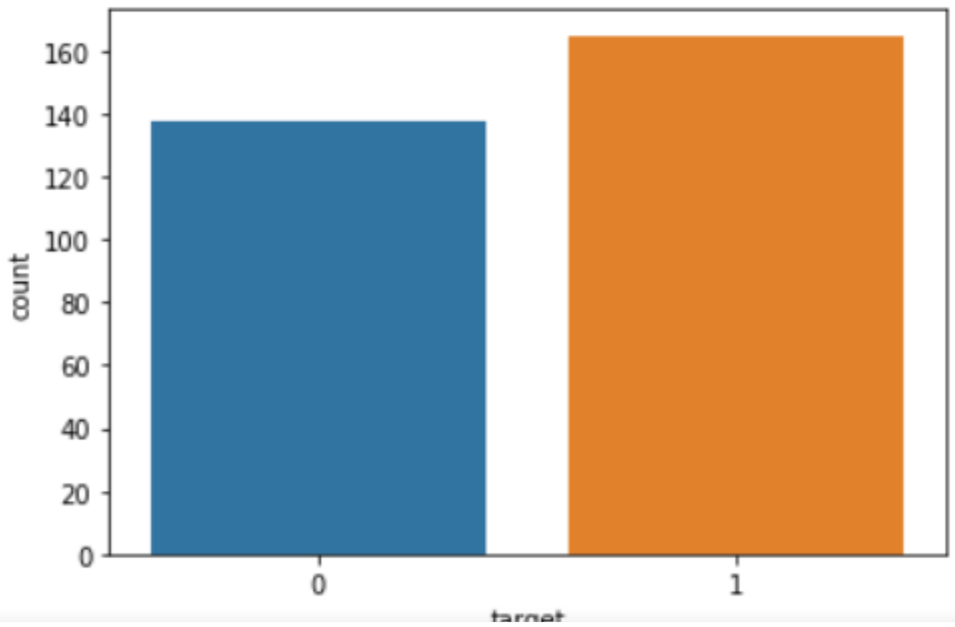


```
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Sum of Missing Values

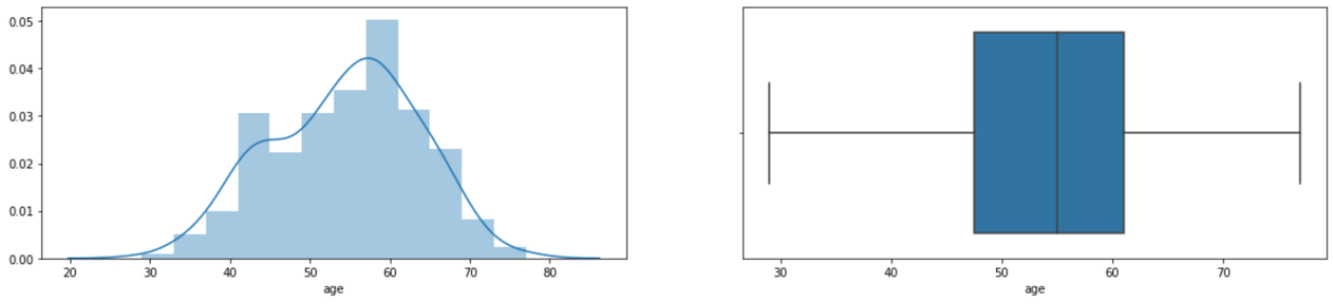
5 Conducting Exploratory Data Analysis (EDA) to understand our data better (only a few are displayed, complete on Github).

```
#Univariate analysis target.
sns.countplot(df['target'])
```



- Value 0 for Heart Disease.
- Value 1 for No Heart Disease.

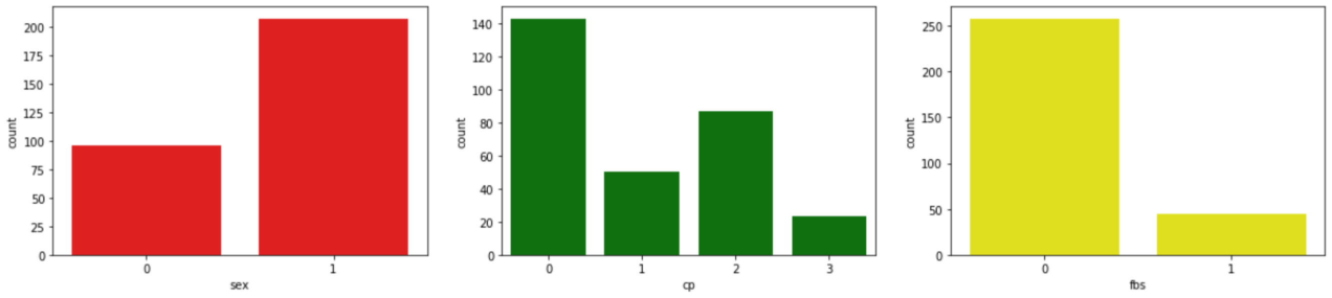
```
#Univariate analysis age.
f = plt.figure(figsize=(20,4))
f.add_subplot(1,2,1)
sns.distplot(df['age'])
f.add_subplot(1,2,2)
sns.boxplot(df['age'])
```



Distplot and Boxplot Feature Age

- From the distplot, it can be seen that the density of the data lies in the range of 50–60 years and very rarely patients aged 30 years or below or 80 years and above.
- The boxplot shows that the data has no outliers.

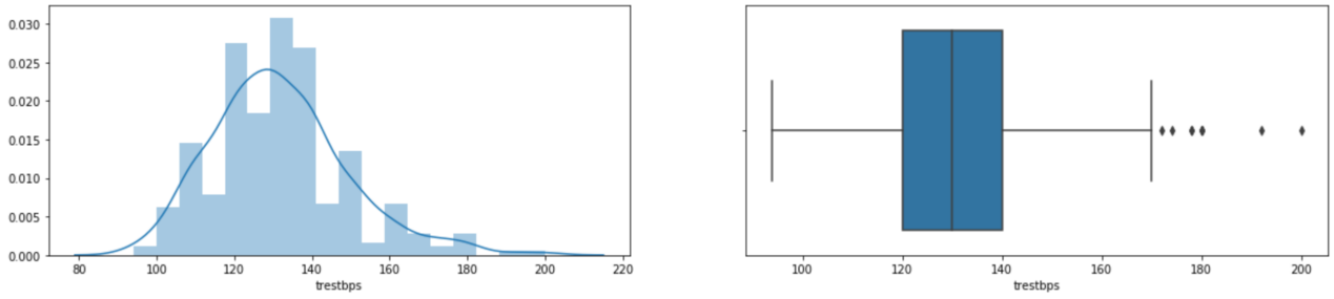
```
#Univariate analysis sex: 1=male; 0=female.
#Univariate analysis chest pain type (cp): 0=typical angina;
1=atypical angine; 2=non-anginal pain; 3=asymptomatic
#Univariate analysis fasting blood sugar: 1 if > 120 mg/dl; 0
otherwise.

f = plt.figure(figsize=(20,4))
f.add_subplot(1,3,1)
df['sex'].value_counts().plot('bar', color='red')
```



Countplot Feature Sex, CP, FBS.

- Male patients turns out to have more numbers or even 2 times the number of female patients.
- Most patients have type CP 0, which is typical angine and the least type is 3, which is asymptomatic.
- The plot above shows that there are many fasting blood sugar values below 120 or 0.

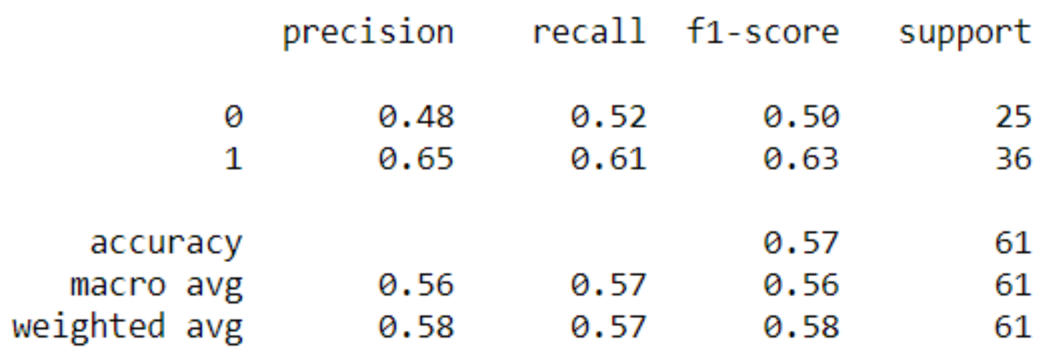
```
#Univariate analysis resting blood pressure (mm Hg) atau trestbps.

f = plt.figure(figsize=(20,4))
f.add_subplot(1,2,1)
sns.distplot(df['trestbps'])
f.add_subplot(1,2,2)
sns.boxplot(df['trestbps'])
```



- The trestbps feature has several outliers.

6 The next stage is to make a model that will be used to predict patients. In this case, we will use KNN algorithm.

```
#Create KNN Object.
knn = KNeighborsClassifier()

#Create x and y variables.
x = df.drop(columns=['target'])
y = df['target']

#Split data into training and testing.
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=4)

#Training the model.
knn.fit(x_train, y_train)

#Predict test data set.
y_pred = log_reg.predict(x_test)

#Checking performance our model with classification report.
print(classification_report(y_test, y_pred))

#Checking performance our model with ROC Score.
roc_auc_score(y_test, y_pred)
```



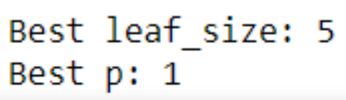
```
              precision    recall  f1-score   support

           0       0.48      0.52      0.50        25
           1       0.65      0.61      0.63        36

    accuracy                           0.57        61
   macro avg       0.56      0.57      0.56        61
weighted avg       0.58      0.57      0.58        61
```

- From the classification report, it can be seen that the model has an average performance of around 57% ranging from precision, recall, f1-score, and support. Accuracy also shows in value of 57%.

- Then for the AUC score, it can be seen that the value is around 56.5%.

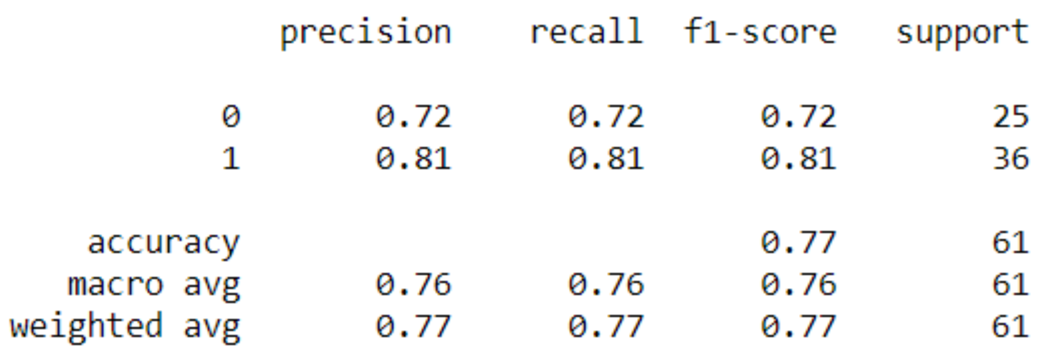
7 Because the performance is still low, Let's try to use Hyperparameter Tuning to Improve Model Performance.

```
#List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]

#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, 
p=p)

#Create new KNN object
knn_2 = KNeighborsClassifier()

#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)

#Fit the model
best_model = clf.fit(x,y)

#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()
['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()
['n_neighbors'])
```



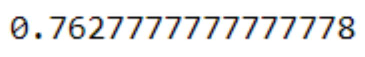
```
Best leaf_size: 5
Best p: 1
```

- From GridSearch, it can be seen that the best number of leaf_size is 5 while the optimal distance method is Manhattan or p = 1.
- Then the most optimal number of K is 7.



```
              precision    recall  f1-score   support

           0       0.72      0.72      0.72        25
           1       0.81      0.81      0.81        36

    accuracy                           0.77        61
   macro avg       0.76      0.76      0.76        61
weighted avg       0.77      0.77      0.77        61
```

New Classification Report



```
0.7627777777777778
```

New ROC Score

- Using Hyperparameters Tuning can improve model performance by about 20% to a range of 77% for all evaluation matrices.
- The ROC value also increased to 76%.

But even though the performance has improved at 77%, I am not sure about my model and will make some modifications that I will share next week. One of the things I will do is rescaling using StandardScalar.

Thank you for reading this story until the end, if there are criticisms or suggestions you can immediately comment and if the story is useful you can clap or share it!