K-Nearest Neighbor (KNN)

By Sunil Ghimire - “**Code to learn don't learn to code** ”

For the corporate landscape rotating completing around Data Science, it has been one of the most sought areas of nature. You will learn how the KNN algorithm operates and how it can be applied using Python in this article on the KNN algorithm.

KNN concepts can hardly be described in a simpler way. It is an ancient expression that can be used in dozens of languages and traditions. In other terms, it is also said in the ***Bible:*** “He who walks with wise men will be wise, but the compassion of fools will suffer harm.”. It implies the idea of k-nearest neighbor classifiers is part of our everyday existence and judging.

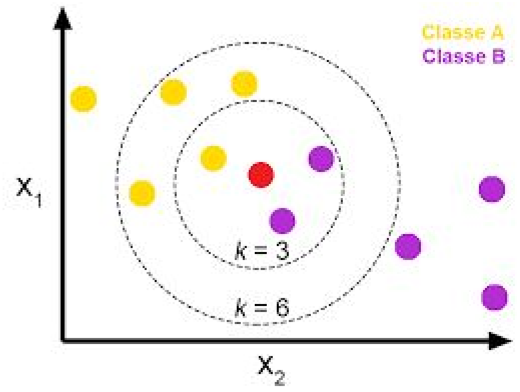
What is KNN Algorithm?

K nearest neighbors or KNN algorithm is a straightforward algorithm that uses the whole dataset in its training dataset. Whenever a prediction is made for an unknown data instance, it looks for the k-most similar across the entire testing dataset, and eventually returns the data with the most similar instances as the predictions. KNN is often used when searching for similar items, such as finding items similar to this one.The Algorithm suggests that you are one of them because you are close to your neighbors.

How does a KNN algorithm work?

To conduct grouping, the KNN algorithm uses a very basic method to perform classification. When a new example is tested, it searches at the training data and seeks the k training examples which are similar to the new example. It then assigns to the test example of the most similar class label.

What does ‘K’ in KNN algorithm represent?



K in KNN algorithm represents the number of nearest neighboring points that vote for a new class of test data.

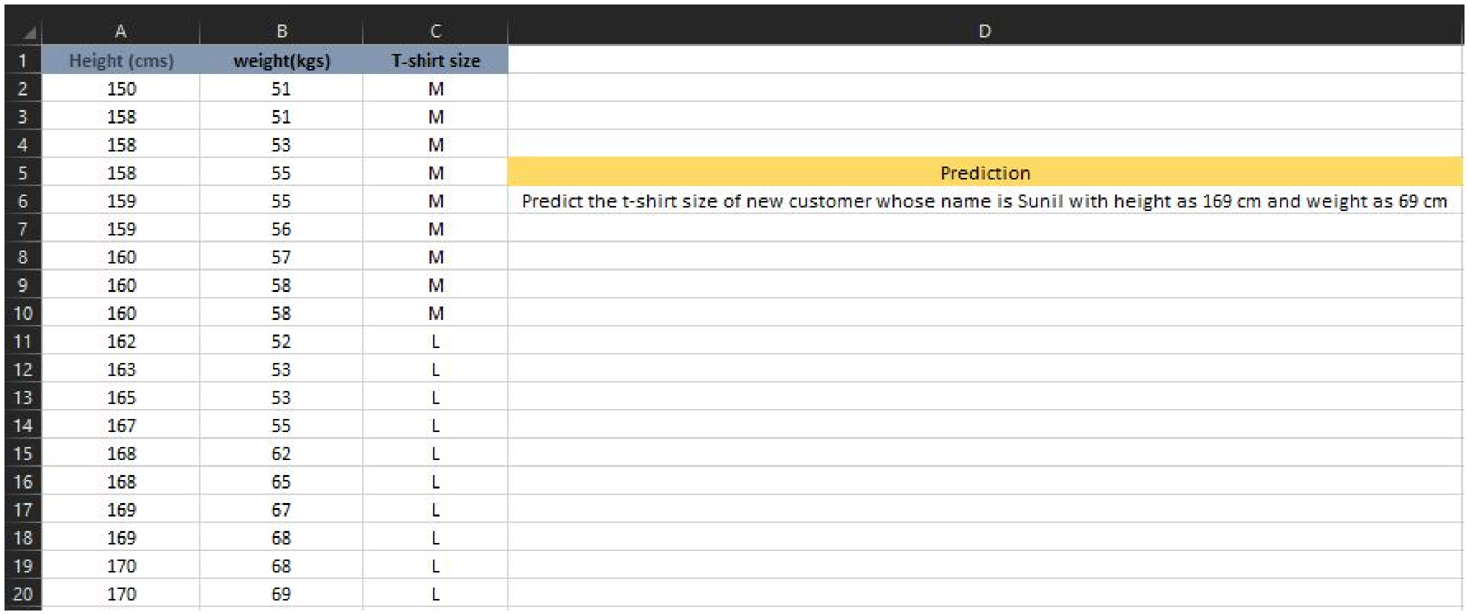
If k = 1, then test examples in the training set will be given the same label as the nearest example.

If k = 3 is checked for the labels of the three closest classes and the most common i.e. occurring at least twice, the label is assigned for larger k’s and so on.

Manual Implementation of KNN algorithm

Let's consider an example of height and weight

The below dataset is about the height and weight of the customer with the corresponding t-shirt size where M represents the medium size and L represents the large size. Now your task is to predict the t-shirt size for the new customer whose name is Sunil with a height of 169 cm and weight of 69 kg.

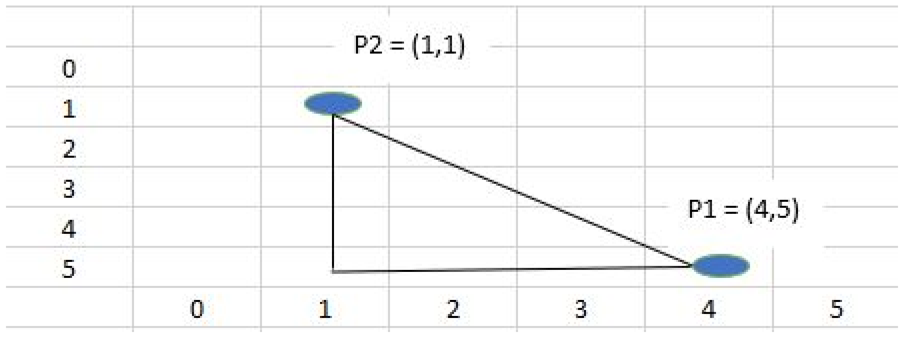


	A	B	C	D
1	Height (cms)	weight(kgs)	T-shirt size	
2	150	51	M	
3	158	51	M	
4	158	53	M	
5	158	55	M	Prediction
6	159	55	M	Predict the t-shirt size of new customer whose name is Sunil with height as 169 cm and weight as 69 cm
7	159	56	M	
8	160	57	M	
9	160	58	M	
10	160	58	M	
11	162	52	L	
12	163	53	L	
13	165	53	L	
14	167	55	L	
15	168	62	L	
16	168	65	L	
17	169	67	L	
18	169	68	L	
19	170	68	L	
20	170	69	L	

Step 1: The initial step is to calculate the Euclidean distance between the existing points and new points. For example, the existing point is (4,5) and the new point is (1 , 1).

So, P1 = (4,5) where x1 = 4 and y1 = 5

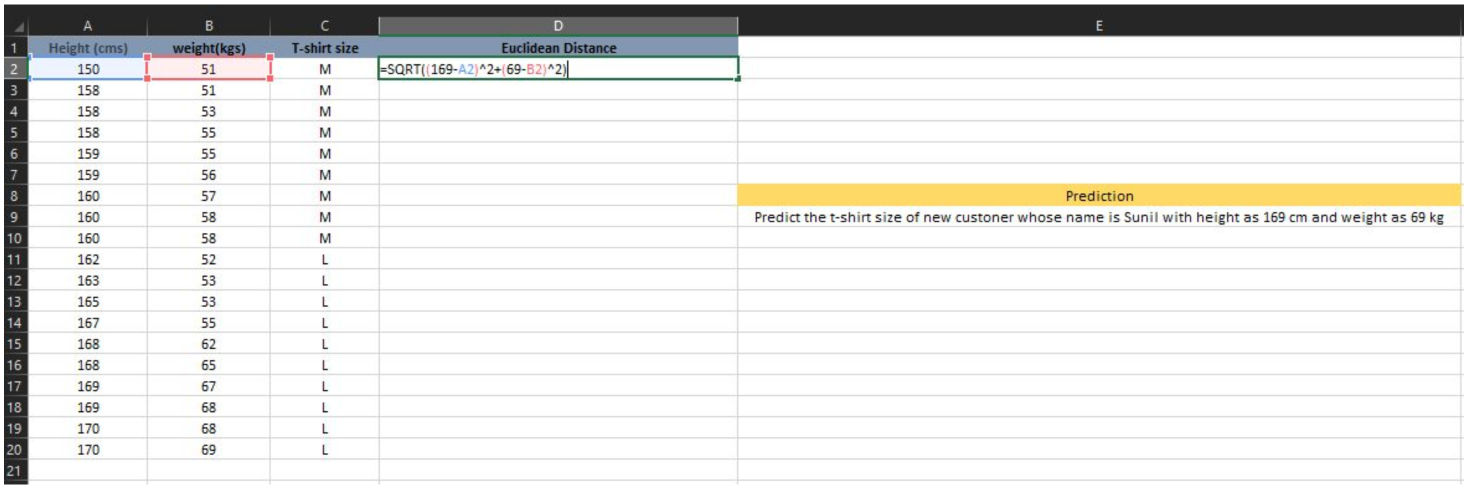
P2 = (1,1) where x2 = 1 and y2 = 1



Now Euclidean distance $= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

$$= \sqrt{(1 - 4)^2 + (1 - 5)^2}$$

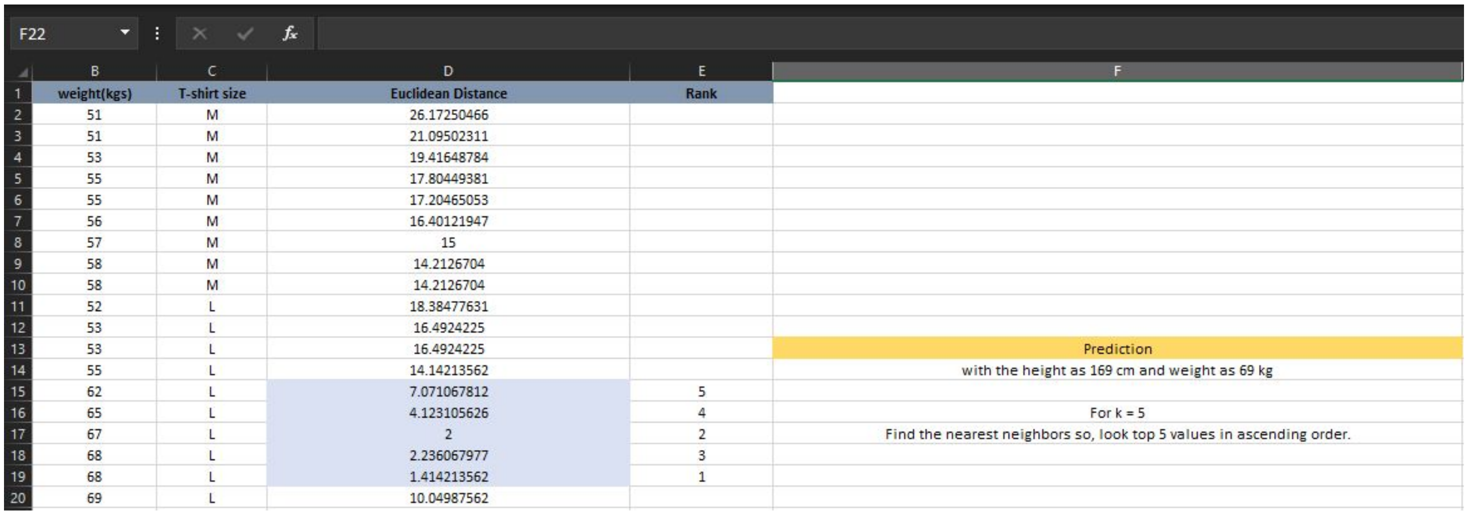
$$= 5$$



	A	B	C	D	E
1	Height (cms)	weight(kgs)	T-shirt size	Euclidean Distance	
2	150	51	M	=SQRT((169-A2)^2+(69-B2)^2)	
3	158	51	M		
4	158	53	M		
5	158	55	M		
6	159	55	M		
7	159	56	M		
8	160	57	M		Prediction
9	160	58	M		Predict the t-shirt size of new custoner whose name is Sunil with height as 169 cm and weight as 69 kg
10	160	58	M		
11	162	52	L		
12	163	53	L		
13	165	53	L		
14	167	55	L		
15	168	62	L		
16	168	65	L		
17	169	67	L		
18	169	68	L		
19	170	68	L		
20	170	69	L		
21					

Step 2: Second step is to choose the k value and select the closest k neighbors to the new item

So, in our case, 5 elements have the least Euclidean distance as compared with others.



F22

	B	C	D	E	F
1	weight(kgs)	T-shirt size	Euclidean Distance	Rank	
2	51	M	26.17250466		
3	51	M	21.09502311		
4	53	M	19.41648784		
5	55	M	17.80449381		
6	55	M	17.20465053		
7	56	M	16.40121947		
8	57	M	15		
9	58	M	14.2126704		
10	58	M	14.2126704		
11	52	L	18.38477631		
12	53	L	16.4924225		
13	53	L	16.4924225		Prediction
14	55	L	14.14213562		with the height as 169 cm and weight as 69 kg
15	62	L	7.071067812	5	
16	65	L	4.123105626	4	For k = 5
17	67	L	2	2	Find the nearest neighbors so, look top 5 values in ascending order.
18	68	L	2.236067977	3	
19	68	L	1.414213562	1	
20	69	L	10.04987562		

Step 3: Count the votes of least distance i.e. Euclidean distance of the predicting values to calculate k neighbors

SInce, K = 5, we have 5 t-shirts of size L. So according to this reason, a new customer name Sunil with a height of 169 cm and a weight of 69 kg will fit into t-shirts of L size.

PYTHON:

Some of the major steps are listed below while implementing the KNN algorithm using Python are listed below:

1. Data handling
2. Distance Calculation
3. Finding K nearest point
4. Predict the class
5. Check the accuracy

Step 1: **Data handling**

The "iris" dataset is handled in the very first phase where the open function opens the data collection and the reader function is used to read the data lines available under the CSV module.

CODE:

```
>>> import csv
>>> with open('../dataset/iris.data') as csvfile:
...     lines = csv.reader(csvfile)
...     for row in lines:
...        print(' , '.join(row))
```

Now for making the prediction and evaluating the accuracy of the model your task is to separate the data into the training set.

The last column or class of iris data is in categorical form. So it needs to be converted into numerical form and separate the data into the train set and test set using the *handleDataset* which load the CSV file along with filename function where standard ratio i.e. 67 percent of data is used in the train set and 33 percent of data is used in the test set.

CODE:

```
>>> import csv
>>> import random
>>> def handleDataset(filename, split, trainSets = [], testSets = []):
...     with open(filename, 'r') as csvfile:
...         lines = csv.reader(csvfile)
...         dataset = list(lines)
...         for x in range(len(dataset) - 1):
...             for y in range(4):
...                 dataset[x][y] = float(dataset[x][y])
...                 if random.random() < split:
...                     trainSets.append(dataset[x])
...                 else:
...                    testSets.append(dataset[x])
```

Let's check the handleDataset function with the proper amount of train set and test set.

CODE:

```
>>> trainSets =[]
>>> testSets=[]
>>> handleDataset('../dataset/iris.data.', 0.66, trainSets, testSets)
>>> print ('Train Set for KNN: ' + repr(len(trainSets)))
>>> print ('Test set for KNN: ' + repr(len(testSets)))

OUTPUT:

Train Set for KNN: 394
Test set for KNN: 206
```

Step 2: **Distance Calculation**

To make some calculations you will need k nearest point for this the distance between the existing point and new point must be determined So, as mentioned on above general introduction Euclidean Distance is needed for calculating the distance which is known as the square root of the total of the square difference between the two numerical arrays.

The iris data contain total 5 columns where column from 1 to 4 is the attributes of the dataset and the last column is a class label which consists of different types of irises like Setosa, Versicolour and Verginica. So, for the distance calculation we need columns from 1 to 4 for distance calculation

CODE:

```
>>> import math
>>> def euclidean_distance(insta_one, insta_two, length):
...     distance = 0
...     for x in range(length):
...         distance += pow((insta_one[x] - insta_two[x]), 2)
...     return math.sqrt(distance)
```

Testing distance of two points using eucludean_function

CODE:

```
>>> data_one = [5, 5, 5, 'a']
>>> data_two = [6, 6, 6, 'b']
>>> distance = euclidean_distance(data_one, data_two, 3)
>>> print ('The distance between two points is: ' + repr(distance))

OUTPUT:

The distance between two points is: 1.7320508075688772
```

Step 3: **Finding K nearest Point**

Now we have measured the distance between two points. Now our task is to create ***getKNeighbors*** function which returns K nearest neighbors from the training set for a given test instance.

CODE:

```
>>> import operator
>>> def getKNeighbors(trainSet, testInstance, K):
...     distance = []
...     length = len(testInstance) - 1
...     for x in range(len(trainSet)):
...         dist = euclidean_distance(testInstance, trainSet[x], length)
...         distance.append((trainSet[x], dist))
...     distance.sort(key = operator.itemgetter(1))
...     neighbors = []
...     for x in range(k):
...         neighbors.append(distance[x][0])
...     return neighbors
```

Now, testing ***getKNeighbors*** function to return closest neighbors

CODE:

```
>>> trainSet = [[5, 5, 5, 'a'], [6, 6, 6, 'b']]
>>> testInstance = [7, 7, 7]
>>> k = 1
>>> neighbors = getKNeighbors(trainSet, testInstance, 1)
>>> print('The nearest neighbors is:', neighbors)

OUTPUT:

The nearest neighbors is: [[6, 6, 6, 'b']]
```

Step 4:**Predict the class**

Now, the next step is to predict the neighbor's response. For this we are going to create a getResponse function to vote their class attribute where the majority vote is taken as prediction.

CODE:

```
>>> import operator
>>> def getResponse(neighbors):
...     vote_class = {}
...     for x in range(len(neighbors)):
...         response = neighbors[x][-1]
...         if response in vote_class:
...             vote_class[response] += 1
...         else:
...             vote_class[response] = 1
...     sortedVotes = sorted(vote_class.items(), key=operator.itemgetter(1), 
reverse=True)
...     return sortedVotes[0][0]
```

Testing ***getResponse*** for prediction

CODE:

```
>>> neighbors = [[5,5,5,'a'], [6,6,6,'a'], [7,7,7,'b']]
>>> print('Majority Vote Response: ',getResponse(neighbors))

OUTPUT:

Majority Vote Response:  a
```

Step 5: **Check the accuracy**

After getting the majority vote response, our next task is to check the accuracy of the majority vote. For this we are going to create a ***getAccuracy*** function to calculate the measure of the percentage of the overall predictions from all the predictions made.

CODE:

```
>>> def getAccuracy(testSet, predictions):
...     correct = 0
...     for x in range(len(testSet)):
...         if testSet[x][-1] is predictions[x]:
...             correct += 1
...     return (correct/float(len(testSet))) * 100.0
```

Testing **getAccuracy** function to calculate the accuracy of the prediction.

CODE:

```
>>> testSet = [[5,5,5,'a'], [6,6,6,'a'], [7,7,7,'b']]
>>> predictions = ['a', 'a', 'a']
>>> accuracy = getAccuracy(testSet, predictions)
>>> print('Accuracy of predictions out of predictions made',
... round((accuracy) , 2 ) , '%')


OUTPUT:

Accuracy of predictions out of predictions made 66.67 %
```

SUMMARY

In the research of KNN, we have learned about the scratch implementation of KNN using example clothes and Python which focused on below points:

1. Implement KNN algorithm step-by-step
2. Evaluate KNN on real dataset
3. Prediction and finding accuracy of new data.

KNN using Scikit-learn

The example below demonstrates KNN implementation on iris dataset using scikit-learn library where iris dataset has petal length, width and sepal length, width with species class/label. Our task is to build a KNN model based on sepal and petal measurements which classifies the new species. We have already downloaded the iris dataset. Now we can make use of it to bulls our KNN model.

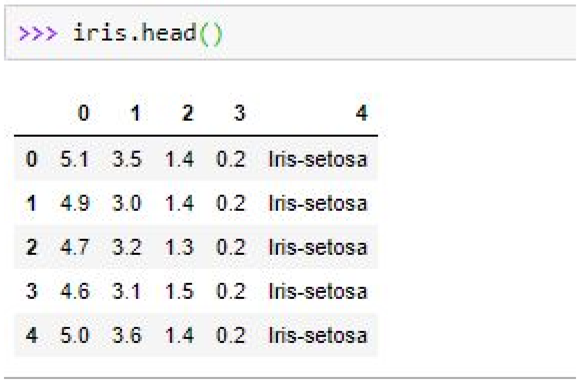
Step 1: **Import the downloaded data and check its features.**

CODE:

```
# load the iris data and check its features
>>> import pandas as pd
>>> iris = pd.read_csv('../dataset/iris.data', header = None)
```

```
# print the iris data
>>> iris.head()
```

OUTPUT:



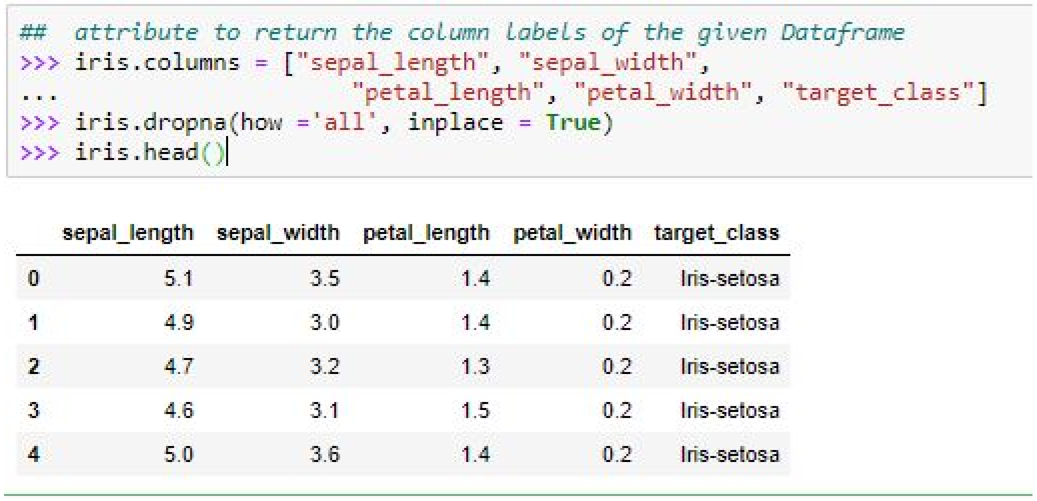
```
>>> iris.head()
```

```
     0    1    2    3            4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa
```

CODE:

```
##  attribute to return the column labels of the given Dataframe
>>> iris.columns = ["sepal_length", "sepal_width",
...                         "petal_length", "petal_width", "target_class"]
>>> iris.dropna(how ='all', inplace = True)
>>> iris.head()
```

OUTPUT:



```
##  attribute to return the column labels of the given Dataframe
>>> iris.columns = ["sepal_length", "sepal_width",
...                         "petal_length", "petal_width", "target_class"]
>>> iris.dropna(how ='all', inplace = True)
>>> iris.head()
```

```
   sepal_length  sepal_width  petal_length  petal_width  target_class
0           5.1          3.5           1.4          0.2   Iris-setosa
1           4.9          3.0           1.4          0.2   Iris-setosa
2           4.7          3.2           1.3          0.2   Iris-setosa
3           4.6          3.1           1.5          0.2   Iris-setosa
4           5.0          3.6           1.4          0.2   Iris-setosa
```

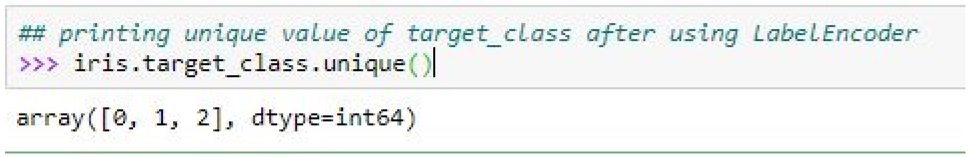
From the above observation it is clear that each observation represents one flower with four columns representing four measurements. And target_class is in categorical form which can not be handled by machine learning algorithms like KNN. So feature and response should be numeric i.e. NumPy arrays which have a specific shape. For this we have implemented a **LabelEncoder** for target_class which are encoded as 0 for iris_setosa, 1 for iris_versicolor and 2 for iris_verginica.

CODE:

```
# using LabeEncoder for target_Class
>>> from sklearn.preprocessing import LabelEncoder
>>> labelencoder = LabelEncoder()
>>> iris['target_class'] = labelencoder.fit_transform(iris['target_class'])
```

CODE:

```
## printing unique value of target_class after using LabelEncoder
>>> iris.target_class.unique()

OUTPUT:
```

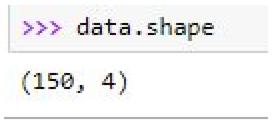


```
## printing unique value of target_class after using LabelEncoder
>>> iris.target_class.unique()

array([0, 1, 2], dtype=int64)
```

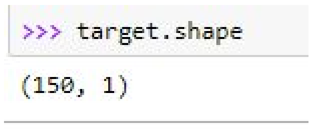
Now, the next step is to step feature class and target class using the iloc function.

CODE:

```
# separating feature class and target class
>>> data = iris.iloc[:, iris.columns != 'target_class']
>>> target  = iris.iloc[:, iris.columns == 'target_class' ]

OUTPUT:
```



```
>>> data.shape

(150, 4)
```



```
>>> target.shape

(150, 1)
```

Step 2: **Split the data into train set and test set and train the KNN model**

It is not an optimal approach for training and testing on the same data, so we need to divide the data into two parts, training set and testing test. For this function called 'train_test_split' provided by Sklearn helps to split the data where the parameter like 'test_size' split the percentage of train data and test data. 'Random_state' is another parameter which helps to give the same result every time when we run our model means split the data in the same way every time. As we are training and testing on various datasets, the subsequent quality of the tests should be a stronger approximation of how well the model would do on unknown data.

CODE:

```
## splitting the data into training and test sets
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(data, target,
...                                               test_size = 0.3, random_state = 524)
```

Shape of train and test objects

CODE:

```
## shape of train and test for X objects
>>> print(X_train.shape)
>>> print(X_test.shape)

OUTPUT:

(105, 4)
(45, 4)
```

```
## shape of y objects
>>> print(y_train.shape)
>>> print(y_test.shape)

OUTPUT:

(105, 1)
(45, 1)
```

Step 3: **Import 'KNeighborsClassifier' class from sklearn**

Sklearn-learn is carefully structured into packages, so that we can quickly import the classes easily. Now we are going to import the 'KNeighborsClassifier' from the 'neighbors' module of Sklearn. The model provider by Sklearn is known as an estimator because the primary role of a model is to estimate the unknown values. In our example we are creating the object called 'knn' which is also known as we are creating the instance 'knn' of the class 'KNeighborsClassifier' which knows how to do KNN classification once we provide the data. The 'n_neighbors' in our model is the hyperparameter which is the number of the neighbors to use by default for neighbor's queries.

We have used the fit method which is used to train the model i.e. X_train and Y_train and similarly we have used the predict method which is used to perform testing on X_test. It is important to select the appropriate value of k, so we use a loop to fit and test the model for various values for K (for 1 - 25) and measure the test accuracy of the KNN in a vector(scores).

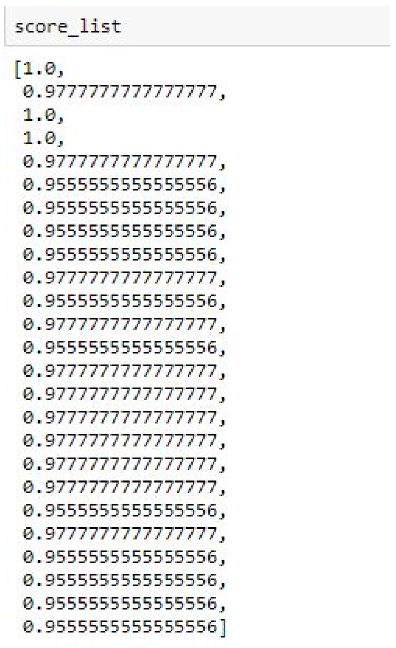
CODE:

```
## Import ‘KNeighborsClassifier’ class from sklearn
>>> from sklearn.neighbors import KNeighborsClassifier
## import metrics model to check the accuracy
>>> from sklearn import metrics
## using loop from k = 1 to k = 25 and record testing accuracy
>>> k_range = range(1,26)
>>> scores = {}
>>> score_list = []
>>> for k in k_range:
...     knn = KNeighborsClassifier(n_neighbors=k)
...     knn.fit(X_train, y_train)
...     y_pred = knn.predict(X_test)
...     scores[k] = metrics.accuracy_score(y_test, y_pred)
...     score_list.append(metrics.accuracy_score(y_test, y_pred))
```

Checking score after using loop from k = 1 to k = 25

CODE:

```
## checking the score_list
>>> score_list

OUTPUT:
```



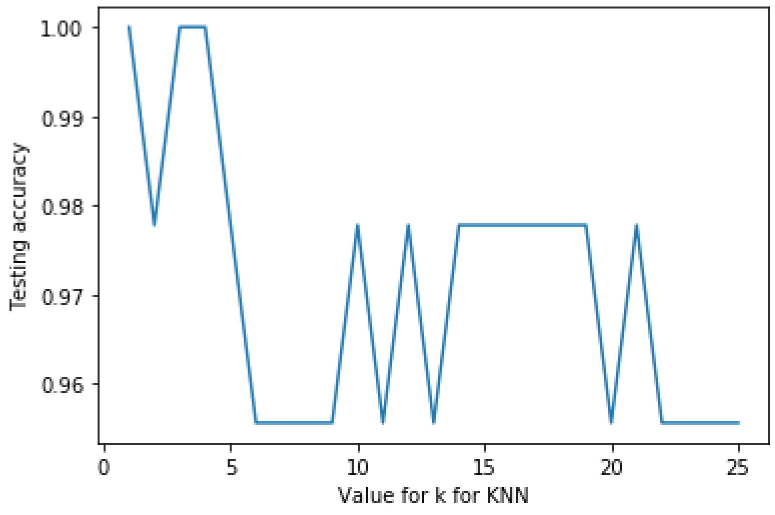
```
score_list

[1.0,
 0.9777777777777777,
 1.0,
 1.0,
 0.9777777777777777,
 0.9555555555555556,
 0.9555555555555556,
 0.9555555555555556,
 0.9555555555555556,
 0.9777777777777777,
 0.9555555555555556,
 0.9777777777777777,
 0.9555555555555556,
 0.9777777777777777,
 0.9777777777777777,
 0.9777777777777777,
 0.9777777777777777,
 0.9777777777777777,
 0.9777777777777777,
 0.9555555555555556,
 0.9777777777777777,
 0.9555555555555556,
 0.9555555555555556,
 0.9555555555555556,
 0.9555555555555556]
```

Step 4: **Plot relationship between k and and the testing accuracy**

After measuring the test_accuracy and recording it in the score_list variable. Now, using the matplotlib library provided by Python is used to plot the relation between the K values and the corresponding testing accuracy. As we can see that it is quite typical when analyzing the model complexity with accuracy because there is a raise and fall of accuracy. So from the graph we can see that when the value of K increases there is the raise of the accuracy and it again falls. Training accuracy typically decreases as the difficulty of the process rises, for the KNN the difficulty of the process is calculated by K's value. The large K value means the less complex model which leads to the smoother complex boundary. The small K value leads to overfitting of the model which is known to be a more complex model. Accuracy testing penalizes configurations that are either too complicated i.e. overfitting or not robust enough i.e. underfitting. In our KNN model we get the maximum testing accuracy from K = 3 to k = 21 and when K = 25 then our testing accuracy of the model is 0.95% .

CODE:

```
>>> %matplotlib inline
>>> import matplotlib.pyplot as plt
>>> plt.plot(k_range, score_list)
>>> plt.xlabel('Value for k for KNN')
>>> plt.ylabel('Testing accuracy')

OUTPUT:
```



Step 5: **Make Predictions**

Now we are going to choose an appropriate value of K as 5 for our model which lies between K = 3 to K = 21. Then retrain the model using all the data available. So this is going to be our final model which is going to make predictions.

CODE:

```
>>> knn = KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(data,target)
>>> target_Classes = {0:'iris_setosa', 1:'iris_versicolor', 2:'iris_verginica'}
>>> x_new = [[4,3,1,2],
...          [5,4,1,3]]
>>> y_predict = knn.predict(x_new)
>>> print('First Predictions  -> ',target_Classes[y_predict[1]])
>>> print('Second Predictions -> ',target_Classes[y_predict[0]])


OUTPUT:

First Predictions  ->  iris_setosa
Second Predictions ->  iris_setosa
```

PROS & CONS

PROS

1. KNN classifier algorithm is used to solve both regression, classification and multi-classification problem
2. KNN classifier algorithms can adapt easily to changes in real-time inputs.
3. We do not have to follow any special requirements before applying KNN.

CONS

1. KNN performs well in a limited number of input variables. So, it’s really challenging to estimate the performance of new data as the number of the variables increases. Thus, it is called the curse of dimensionality. In the modern scientific era, increasing quantities of data are being produced and collected. How, for target_class in machine learning, too much data can be a bad thing. At a certain level, additional features or dimensions will decrease the precision of a model, because more data has to be generalized. Thus, this is recognized as the "Curse of dimensionality".
2. KNN requires data that is normalized and also the KNN algorithm cannot deal with the missing value problem.
3. The biggest problem with the KNN is finding the correct neighbor number.

Do you have any suggestions regarding scratch of KNN and implementing KNN using Sklearn ?? I would be pleased to hear that.