**The k-nearest neighbour algorithm in Python**

**Supervised Learning :**   
It is the learning where the value or result that we want to predict is within the training data (labeled data) and the value which is in data that we want to study is known as Target or Dependent Variable or *Response Variable*.

 All the other columns in the dataset are known as the Feature or Predictor Variable or Independent Variable.

Supervised Learning is classified into two categories:

1. **Classification**: Here our target variable consists of the categories.
2. **Regression**: Here our target variable is continuous and we usually try to find out the line of the curve.

As we have understood that to carry out supervised learning we need labeled data. How we can get labeled data? There are various ways to get labeled data:

1. Historical labeled Data
2. Experiment to get data: We can perform experiments to generate labeled data like A/B Testing.
3. Crowd-sourcing

Now it’s time to understand algorithms that can be used to solve supervised machine learning problem. In this post, we will be using popular **scikit-learn** package.

***Note:****There are few other packages as well like TensorFlow, Keras etc to perform supervised learning.*

## **k-nearest neighbour algorithm:**

This algorithm is used to solve classification model problems. K-nearest neighbor or K-NN algorithm basically creates an imaginary boundary to classify the data. When new data points come in, the algorithm will try to predict that to the nearest of the boundary line.

Therefore, larger k value means smother curves of separation resulting in less complex models. Whereas, smaller k value tends to overfit the data and resulting in complex models.

**Note:**It’s very important to have the right k-value when analyzing the dataset to avoid overfitting and underfitting of the dataset.

Using the k-nearest neighbor algorithm we fit the historical data (or train the model) and predict the future.

#### **Example of the k-nearest neighbor algorithm**

* Python3

|  |
| --- |
| # Import necessary modules  from sklearn.neighbors import KNeighborsClassifier  from sklearn.model\_selection import train\_test\_split  from sklearn.datasets import load\_iris    # Loading data  irisData = load\_iris()    # Create feature and target arrays  X = irisData.data  y = irisData.target    # Split into training and test set  X\_train, X\_test, y\_train, y\_test = train\_test\_split(               X, y, test\_size = 0.2, random\_state=42)    knn = KNeighborsClassifier(n\_neighbors=7)    knn.fit(X\_train, y\_train)    # Predict on dataset which model has not seen before  print(knn.predict(X\_test)) |

In the example shown above following steps are performed:

1. The k-nearest neighbor algorithm is imported from the scikit-learn package.
2. Create feature and target variables.
3. Split data into training and test data.
4. Generate a k-NN model using neighbors value.
5. Train or fit the data into the model.
6. Predict the future.

We have seen how we can use K-NN algorithm to solve the supervised machine learning problem. But how to measure the accuracy of the model?

Consider an example shown below where we predicted the performance of the above model:

* Python3

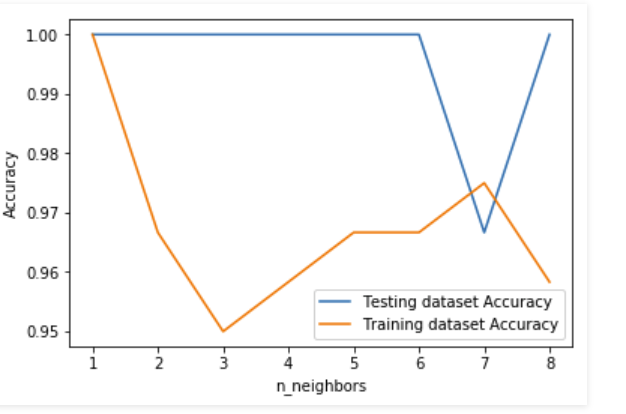
|  |
| --- |
| # Import necessary modules  from sklearn.neighbors import KNeighborsClassifier  from sklearn.model\_selection import train\_test\_split  from sklearn.datasets import load\_iris    # Loading data  irisData = load\_iris()    # Create feature and target arrays  X = irisData.data  y = irisData.target    # Split into training and test set  X\_train, X\_test, y\_train, y\_test = train\_test\_split(               X, y, test\_size = 0.2, random\_state=42)    knn = KNeighborsClassifier(n\_neighbors=7)    knn.fit(X\_train, y\_train)    # Calculate the accuracy of the model  print(knn.score(X\_test, y\_test)) |

**Model Accuracy:**  
So far so good. But how to decide the right k-value for the dataset? Obviously, we need to be familiar to data to get the range of expected k-value, but to get the exact k-value we need to test the model for each and every expected k-value. Refer to the example shown below.

* Python3

|  |
| --- |
| # Import necessary modules  from sklearn.neighbors import KNeighborsClassifier  from sklearn.model\_selection import train\_test\_split  from sklearn.datasets import load\_iris  import numpy as np  import matplotlib.pyplot as plt    irisData = load\_iris()    # Create feature and target arrays  X = irisData.data  y = irisData.target    # Split into training and test set  X\_train, X\_test, y\_train, y\_test = train\_test\_split(               X, y, test\_size = 0.2, random\_state=42)    neighbors = np.arange(1, 9)  train\_accuracy = np.empty(len(neighbors))  test\_accuracy = np.empty(len(neighbors))    # Loop over K values  for i, k in enumerate(neighbors):      knn = KNeighborsClassifier(n\_neighbors=k)      knn.fit(X\_train, y\_train)        # Compute training and test data accuracy      train\_accuracy[i] = knn.score(X\_train, y\_train)      test\_accuracy[i] = knn.score(X\_test, y\_test)    # Generate plot  plt.plot(neighbors, test\_accuracy, label = 'Testing dataset Accuracy')  plt.plot(neighbors, train\_accuracy, label = 'Training dataset Accuracy')    plt.legend()  plt.xlabel('n\_neighbors')  plt.ylabel('Accuracy')  plt.show() |

**Output:** 



Here in the example shown above, we are creating a plot to see the k-value for which we have high accuracy.

**Note:**This is a technique which is not used industry-wide to choose the correct value of n\_neighbors. Instead, we do hyperparameter tuning to choose the value that gives the best performance.