**KNN Classification**

Introduction

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry.

## Working of KNN Algorithm

K-nearest neighbours (KNN) algorithm uses ‘feature similarity’ to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps:

**Step 1** − For implementing any algorithm, we need dataset. So, during the first step of KNN, we must load the training as well as test data.

**Step 2** − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

**Step 3** − For each point in the test data do the following −

* **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
* **3.2** − Now, based on the distance value, sort them in ascending order.
* **3.3** − Next, it will choose the top K rows from the sorted array.
* **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

**Step 4** − End

## Implementation in Python

First, start with importing necessary python packages –

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

Next, download the iris dataset from its weblink as follows –

path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Next, we need to assign column names to the dataset as follows –

headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

Now, we need to read dataset to pandas dataframe as follows −

dataset = pd.read\_csv(path, names = headernames)

dataset.head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Index** | **sepal-length** | **sepal-width** | **petal-length** | **petal-width** | **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 |

Data Preprocessing will be done with the help of following script lines.

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

Next, we will divide the data into train and test split. Following code will split the dataset into 60% training data and 40% of testing data −

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30)

Next, data scaling will be done as follows −

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

Next, train the model with the help of KNeighborsClassifier class of sklearn as follows −

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 8)

classifier.fit(X\_train, y\_train)

At last we need to make prediction. It can be done with the help of following script −

y\_pred = classifier.predict(X\_test)

Next, print the results as follows –

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

result = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(result)

result1 = classification\_report(y\_test, y\_pred)

print("Classification Report:",)

print (result1)

result2 = accuracy\_score(y\_test,y\_pred)

print("Accuracy:",result2)

Output:

Confusion Matrix:

[[14 0 0]

[ 0 15 0]

[ 0 2 14]]

Accuracy: 0.9555555555555556

We got really good accuracy

Deploy model on Web application

# flask\_app.py file (for integrate model with application)

import numpy as np

from flask import request,Flask,jsonify,render\_template

import pickle

app = Flask(\_\_name\_\_) # initialize flask application

model = pickle.load(open('knn\_model.pickle','rb'))#load\_kkn model

@app.route('/') #give url for your homepage

def home(): # call this method when we go to (‘/’) url

return render\_template('index.html') #return template

@app.route('/predict',methods=['POST']) #create predict url

def predict(): # call this method when we go to (‘/predict’) url

input = [float(x) for x in request.form.values()] #get value from the form which is present in index.html

input\_array = np.asarray(input) #convert values into array

prediction = model.predict(input\_array.reshape(1,-1)) #predict value based on the input array from our knn model

return render\_template('index.html',prediction='{}'.format(prediction))

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)