**Write a program to implement k-Nearest Neighbour algorithm for a iris dataset(Download the dataset link: https://archive.ics.uci.edu/ml/datasets/Iris ). Compute the accuracy of the classifier, considering few test data sets.**

# **K-Nearest Neighbor (KNN)**

* KNN is simple supervised learning algorithm used for both regression and classification problems.
* KNN is basically store all available cases and classify new cases based on similarities with stored cases.

**Concept:** So the concept that KNN works on is Basically similarities measurements, for example, if you look at Mango,it is more similar to Apple then dog or cat, then what KNN will do is put it in the category of fruits not in the category of animals.

## **What is K in KNN**

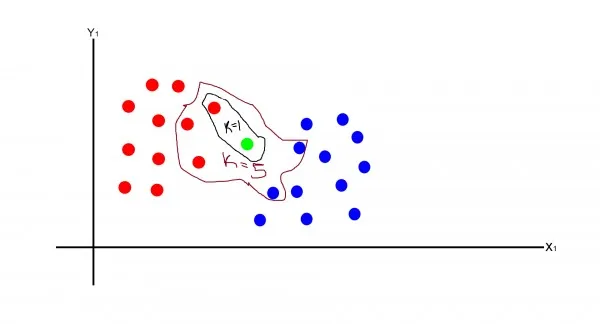
What happens in KNN,we trained the model and after that we want to test our model , means we want to classify our new data (test-data),for that we will check **some (K)** classes around it and assign the most common class to the test-data.

**K- Number of nearest neighbors**

K=1 means the testing data are given the same level as the closet example in training set.

K=4 means the labels of the four closet classes are check and most common class is assign to the testing data.

## **How does KNN is work ?**

****

Let's understand it with the above given diagram

1. In this diagram we have 2 classes one blue class one red class
2. Now we have a new green point, we have to find out whether this point is in class red or blue
3. For this, we will define the value of K
4. At K= 1, we will see the distance from the green point to the nearest points, and select the point with lowest distance and classify the green point in that class, here it red.
5. At K=5 We will calculate the distance from the green point to the nearest points and select the five points with the lowest distance and classify the green point to the most common class, that is red here.
6. **How to choose the value of K?** The value of k is not defined, it depends on the cases.

## **Lazy Learner**

1. KNN is simple algorithm for classification but that's not the reason
2. KNN is lazy learner because it doesn't learn a discriminative function from the training data but **memorizes** the training dataset instead.

## **KNN Algorithm**

let's understand the concept of KNN algorithm with **iris flower problem**

**Data:** This data consist of total 150 instances (samples) , 4 features , and three classes (targets).

**Problem:** Using four features we have to classify which flower belongs to which category.

## **Importing Data-set**

| **import sklearn**  **import pandas as pd**  **from sklearn.datasets import load\_iris**  **iris=load\_iris()**  **iris.keys()**  **df=pd.DataFrame(iris['data'])**  **print(df)**  **print(iris['target\_names'])**  **iris['feature\_names']** |
| --- |

**Note:**

1. Now we need a target and data so that we can train the model
2. As we know that we have to find out the class from the features we have
3. With this logic,our target is classes (0,1,2) and data is in df.

| **X=df**  **y=iris['target']** |
| --- |

## **Splitting Data**

1. The data is split so that with some data we can train the model and from the remaining data we can test the model and can check how well our model is
2. To do this we have an inbuilt function in sklearn

| **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)** |
| --- |

**Note:** It will split our 33% data into testing data and remaining data is our training data

## **KNN Classifier and Training of the Model**

| **from sklearn.neighbors import KNeighborsClassifier**  **knn=KNeighborsClassifier(n\_neighbors=3)** |
| --- |

**Note:**

1. It implements the concepts of KNN. Here we have taken number of neighbors (K)= 3.
2. First, it will calculate the distance with all the training points to the test point and then select the three lowest distance points.
3. And test data point is classify to the class most common in among three.

| **knn.fit(X\_train,y\_train)** |
| --- |

**Note:-** Training the model with features values (data) and target values (target)

## **Prediction and Accuracy**

**Demo:**

1. Here I want to show you just by taking one data point
2. we have a data point x\_new

*import numpy as np*

*x\_new=np.array([[5,2.9,1,0.2]])*

Now we want to see the class or category of this point

*prediction=knn.predict(x\_new)*

*iris['target\_names'][prediction]*

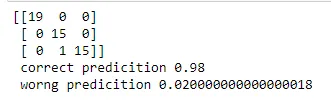
Output



**Note:** As we can see that our point belongs to class (0 or setosa class), this demo is just for understanding

| **from sklearn.metrics import confusion\_matrix**  **from sklearn.metrics import accuracy\_score**  **from sklearn.metrics import classification\_report**  **y\_pred=knn.predict(X\_test)**  **cm=confusion\_matrix(y\_test,y\_pred)**  **print(cm)**  **print(" correct predicition",accuracy\_score(y\_test,y\_pred))**  **print(" worng predicition",(1-accuracy\_score(y\_test,y\_pred)))** |
| --- |

**Output**

****

**Note:** As you can see in confusion matrix only one prediction is wrong , and also our accuracy is 0.98 (98%).

**Implement the CSP :TOM + NAG = GOAT**

**Clearly, G = 1 (Max carry is 1)**

* **O + A = O**
* **This is only possible in two ways**
  + **If O is zero**
  + **Or O is 9 and there is 1 carry from M + G = T step**
* **However, T + N = O it generates 1 carry to result G**
* **Thus, clearly T + N results 10 and O is zero and 1 carry to result G**

### Step 2

* **T + N should be 10**
* **T and N can be (8,2), (7,3), (6,4), (3,7), (2,8)**
* **Lets take (8,2)**
* **In this case A can take any values 3, 4, 5, 6, 7**
* **There are too many possibilities**
* **In this case unfortunately, the answer can not be determined.**

def solutions():

# letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')

all\_solutions = list()

for s in range(9, -1, -1):

for e in range(9, -1, -1):

for n in range(9, -1, -1):

for d in range(9, -1, -1):

for m in range(9, 0, -1):

for o in range(9, -1, -1):

for r in range(9, -1, -1):

for y in range(9, -1, -1):

if len(set([s, e, n, d, m, o, r, y])) == 8:

send = 1000 \* s + 100 \* e + 10 \* n + d

more = 1000 \* m + 100 \* o + 10 \* r + e

money = 10000 \* m + 1000 \* o + 100 \* n + 10 \* e + y

if send + more == money:

all\_solutions.append((send, more, money))

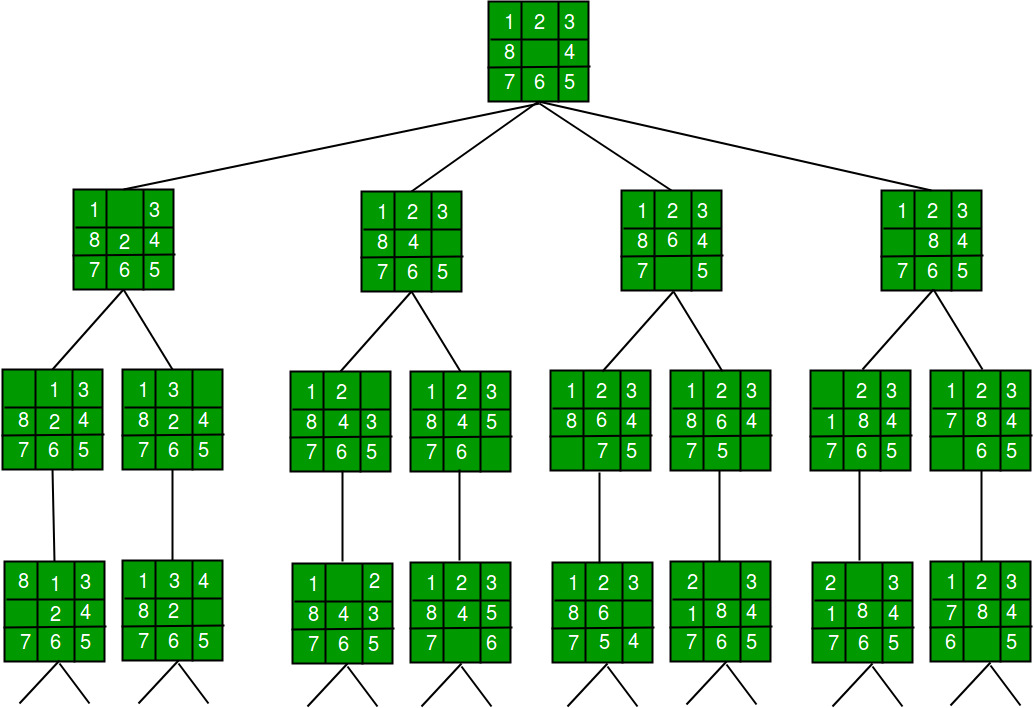
return all\_solutions

print(solutions())

**Implementation and Analysis of DFS and BFS for an application" - 8-Puzzle problem**

DFS (Brute-Force)

We can perform a depth-first search on state-space (Set of all configurations of a given problem i.e. all states that can be reached from the initial state) tree.



In this solution, successive moves can take us away from the goal rather than bringing us closer. The search of state-space tree follows the leftmost path from the root regardless of the initial state. An answer node may never be found in this approach.

**2. BFS (Brute-Force)**

We can perform a Breadth-first search on the state space tree. This always finds a goal state nearest to the root. But no matter what the initial state is, the algorithm attempts the same sequence of moves like DFS.

**Find the shortest path from node ‘S’ to ‘E’ using A\* search**

#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> pi;

vector<vector<pi>> graph;

vector<vector<pi>> g;

vector<vector<pi>> realcost;

vector<string> places;

vector<int> estimated;

vector<int> realtill;

void best\_first\_search(int source, int target, int n)

{

vector<bool> visited(n, false);

priority\_queue<pi, vector<pi>, greater<pi> > pq;

int total = 0;

pq.push(make\_pair(0, source));

int s = source;

visited[s] = true;

int prev = -1;

while (!pq.empty())

{

int x = pq.top().second;

cout << places[x] << " ";

if(prev != -1)

{

for (int i = 0; i < realcost[x].size(); i++)

{

if(realcost[x][i].second == prev)

{

total += realcost[x][i].first;

}

}

}

prev = x;

pq.pop();

if (x == target)

break;

for (int i = 0; i < g[x].size(); i++)

{

if (!visited[g[x][i].second])

{

visited[g[x][i].second] = true;

pq.push(make\_pair(g[x][i].first,g[x][i].second));

}

}

}

cout << "\nTotal cost = " << total << "\n";

}

int main()

{

cout << "Enter total number of vertices: \n";

int v;

cin >> v;

graph.resize(v);

g.resize(v);

realcost.resize(v);

realtill.resize(v, 0);

cout << "Enter total number of edges: \n";

int e;

cin >> e;

cout << "Enter the names of the places in order alomg with estimated cost: \n";

for(int i=0; i<v; i++)

{

string x;

int d;

cin >> x >> d;

places.push\_back(x);

estimated.push\_back(d);

}

realtill.push\_back(0);

cout << "Enter source destination cost:\n";

for(int i=0; i<e; i++)

{

int x,y,cost;

cin >> x >> y >> cost;

int es = estimated[y];

realtill[y] = realtill[x]+cost;

graph[x].push\_back(make\_pair(realtill[y]+es, y));

graph[y].push\_back(make\_pair(realtill[y]+es, x));

g[x].push\_back(make\_pair(es, y));

g[y].push\_back(make\_pair(es, x));

realcost[x].push\_back(make\_pair(cost, y));

realcost[y].push\_back(make\_pair(cost, x));

}

cout << "Enter the source and target: \n";

int source, target;

cin >> source >> target;

cout << "The best\_first\_search is: \n";

best\_first\_search(source, target, v);

cout << "\n";

return 0;

}

**Write a program to implement Support vector machine algorithm to classify the iris data set (Download the dataset link: https://archive.ics.uci.edu/ml/datasets/Iris ).. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem**

**import os**

**print(os.listdir("../input"))’**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**%matplotlib inline**

**# Import the dataset using Seaborn library**

**iris=pd.read\_csv('../input/IRIS.csv')**

**# Checking the dataset**

**iris.head()**

**sns.pairplot(data=iris, hue='species', palette='Set2')**

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

**# Separating the independent variables from dependent variables**

**x=iris.iloc[:,:-1]**

**y=iris.iloc[:,4]**

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

**from sklearn.svm import SVC**

**model=SVC()**

**model.fit(x\_train, y\_train)**

**pred=model.predict(x\_test)**

**from sklearn.metrics import classification\_report, confusion\_matrix**

**print(confusion\_matrix(y\_test,pred))**

**print(classification\_report(y\_test, pred))**

**House Price prediction using SVM:**

[**https://www.enjoyalgorithms.com/blog/boston-house-price-prediction-using-machine-learning**](https://www.enjoyalgorithms.com/blog/boston-house-price-prediction-using-machine-learning)

**Stock Price Prediction**

[**https://www.geeksforgeeks.org/predicting-stock-price-direction-using-support-vector-machines/?ref=rp**](https://www.geeksforgeeks.org/predicting-stock-price-direction-using-support-vector-machines/?ref=rp)

**Heart disease using svm:**

[**https://www.kaggle.com/code/cdabakoglu/heart-disease-classifications-machine-learning/notebook**](https://www.kaggle.com/code/cdabakoglu/heart-disease-classifications-machine-learning/notebook)