**LAB # 10**

**Introduction to Learning, Nearest Neighbors**

**Introduction**

The nearest neighbor algorithm was one of the first algorithms used to determine a solution to the travelling salesman problem. In it, the salesman starts at a random city and repeatedly visits the nearest city until all have been visited. It quickly yields a short tour, but usually not the optimal one.

**Example with the traveling salesman problem**

Below is the application of nearest neighbor algorithm to the travelling salesman problem. These are the steps of the algorithm:

**1.** start on an arbitrary vertex as current vertex.

**2.** find out the shortest edge connecting current vertex and an unvisited vertex V. **3.** set current vertex to V.

**4.** mark V as visited.

**5.** if all the vertices in domain are visited, then terminate.

**6.** Go to step 2.

The sequence of the visited vertices is the output of the algorithm.

The nearest neighbor algorithm is easy to implement and executes quickly, but it can sometimes miss shorter routes which are easily noticed with human insight, due to its "greedy" nature. As a general guide, if the last few stages of the tour are comparable in length to the first stages, then the tour is reasonable; if they are much greater, then it is likely that there are much better tours. Another check is to use an algorithm such as the lower bound algorithm to estimate if this tour is good enough.

In the worst case, the algorithm results in a tour that is much longer than the optimal tour. To be precise, for every constant r there is an instance of the traveling salesman problem such that the length of the tour computed by the nearest neighbor algorithm is greater than r times the length of the optimal tour. Moreover, for each number of cities there is an assignment of distances between the cities for which the nearest neighbor heuristic produces the unique worst possible tour.

The nearest neighbor algorithm may not find a feasible tour at all, even when one exists.

EXAMPLE CODE:

print(dataset.head())

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20)

classifier = KNeighborsClassifier(n\_neighbors=5)

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

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

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

**Lab Tasks/Practical Work**

1. **Implement machine learning and nearest neighbors algorithms on Traveling Salesman Problem.**

**Code**

import numpy as np

import matplotlib.pyplot as plt

np.random.seed(42)

num\_cities = 10

cities = np.random.rand(num\_cities, 2) \* 100

def calculate\_distance(city1, city2):

return np.sqrt(np.sum((city1 - city2) \*\* 2))

def tsp\_nearest\_neighbor(cities):

num\_cities = len(cities)

visited = [False] \* num\_cities

path = []

current\_city = 0

path.append(current\_city)

visited[current\_city] = True

total\_distance = 0

for \_ in range(num\_cities - 1):

nearest\_city = None

nearest\_distance = float('inf')

for next\_city in range(num\_cities):

if not visited[next\_city]:

distance = calculate\_distance(cities[current\_city], cities[next\_city])

if distance < nearest\_distance:

nearest\_distance = distance

nearest\_city = next\_city

path.append(nearest\_city)

visited[nearest\_city] = True

total\_distance += nearest\_distance

current\_city = nearest\_city

total\_distance += calculate\_distance(cities[current\_city], cities[path[0]])

path.append(path[0])

return path, total\_distance

path, total\_distance = tsp\_nearest\_neighbor(cities)

def plot\_tsp(cities, path):

plt.figure(figsize=(8, 6))

for i in range(len(path) - 1):

city1, city2 = cities[path[i]], cities[path[i + 1]]

plt.plot([city1[0], city2[0]], [city1[1], city2[1]], 'bo-')

plt.scatter(cities[:, 0], cities[:, 1], color='red', s=100, zorder=5)

for i, city in enumerate(cities):

plt.text(city[0] + 1, city[1] + 1, f"{i}", fontsize=10)

plt.title("TSP Solution using Nearest Neighbor")

plt.xlabel("X Coordinate")

plt.ylabel("Y Coordinate")

plt.grid(True)

plt.show()

plot\_tsp(cities, path)

print(f"Path: {path}")

print(f"Total Distance: {total\_distance:.2f}")

**Output**

