Machine Learning（SS24）

Assignment 01: Preprocessing and K-Nearest Neighbors

Qu Wang

M.Sc, Matriculation Number 3700666, Study Program: Integrative Technologies and Architectural Design Research (ITECH)

st190363@stud.uni-stuttgart.de

Lianhan Huang

M.Sc, Matriculation Number 3700459, Study Program: Integrative Technologies and Architectural Design Research(ITECH)

St188954@stud.uni-stuttgart.de

1. Preprocessing

Final result:

ID	Age	Income	Owns_Car
1.0	25.0	0.3333333333333335	1.0
2.0	33.75	0.0	0.0
3.0	35.0	0.5	1.0
4.0	45.0	1.0000000000000002	1.0
5.0	30.0	0.6666666666666667	0.0

Number of Vehicles	Preferred Transport Mode_Bike	Preferred Transport Mode_Car	Preferred Transport Mode_Public Transport
2.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0
1.0	0.0	1.0	0.0
0.0	0.0	1.0	0.0
0.0	1.0	0.0	0.0

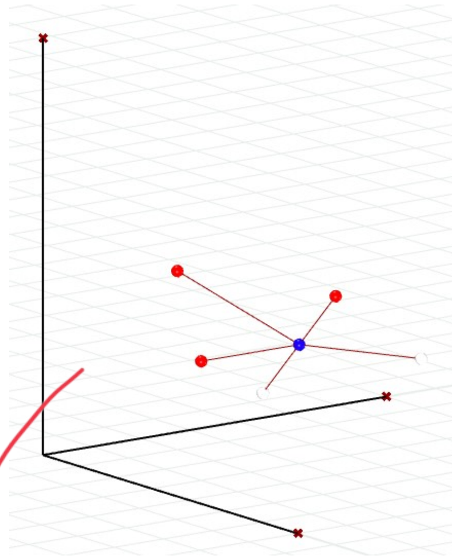
Codes:

assignment1.py

```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt


# Specify the file path
file_path = "D:\\A1_ITECH\\12_ML\Assignments_github\ML_assignment\\assignment1\\transportation_preferance.csv"

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)

print(df.head())

#question_a
# Identify columns with missing values
missing_cols = df.columns[df.isnull().any()]

# Impute missing values
for col in missing_cols:
    if col == 'Age':
        mean_age = df['Age'].mean()
        df[col].fillna(mean_age, inplace=True)
    elif col == 'Income':
        median_income = df['Income'].median()
        df[col].fillna(median_income, inplace=True)
    elif col == 'Number of Vehicles':
        mode_vehicles = df['Number of Vehicles'].mode()[0]
        df[col].fillna(mode_vehicles, inplace=True)

print(df)
```

```python

#question_b
# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Apply Min-Max scaling to the "Income (K$)" column
df['Income'] = scaler.fit_transform(df[['Income']])

print(df['Income'])

#question_c
# Binary encoding for "Owns_Car" column
df['Owns_Car'] = df['Owns_Car'].map({'Yes': 1, 'No': 0})

# One-hot encoding for "Preferred Transport Mode" column
df = pd.get_dummies(df, columns=['Preferred Transport Mode'], dtype=int)

print(df['Preferred Transport Mode_Car'], df['Preferred Transport Mode_Bike'],
      df['Preferred Transport Mode_Public Transport'])

# Plotting the DataFrame
plt.figure(figsize=(40, 30))  # Adjust the figure size as needed
plt.table(cellText=df.values,
          colLabels=df.columns,
          loc='center')
plt.axis('off')  # Turn off the axis
plt.savefig( *args: 'table_image.png', bbox_inches='tight', pad_inches=0.05)  # Save as image
plt.show()

```

2. K-Nearest Neighbors



```
dataset = np.array([
    [1, 2, 3, 0],
    [2, 3, 1, 1],
    [3, 1, 2, 0],
    [4, 5, 1, 1],
    [3, 3, 4, 0]
])
```

(a) **Distance Calculation**

The nearest 3 neighbors for K=3:
Neighbor Index: 1, Distance: 1.4142, Class Label: 1
Neighbor Index: 2, Distance: 2.0000, Class Label: 0
Neighbor Index: 4, Distance: 2.0000, Class Label: 0

Assigned class for K=3: 0

(b) **Impact of K**

Assigned class for K=1: 1
Assigned class for K=5: 0

Small K Value:
-> Benefits: (1) Very sensitive the local variation of the data set;
(2) Can deal with the situation where there are many small group of data.
-> Drawbacks: (1) over-fitting, which means the model will be specific sensitive the training data and can not fit the new data very well;
(2) Will be disturbed by abnormal data, if the new observation is surrounded by some wrong data, the model will make the wrong decision;
(3) The model can not have a very good „understanding“ of the global data set.

Big K Value:
-> Benefits: (1) Will not be disturbed by wrong data, when there is a relative big data set, big K value will lead to a more average result;
(2) Have a better understanding of the whole data set.
-> Drawbacks: (1) lack of precision, which means some small group will be ignored;
(3) Oversimplified.

(c) **Distance Weighting**

Some type of distance-weighted voting:

Inverse Distance Weighting - In this type of voting, the closer the data is, the more impossible for the new observation to be the same. It will reverse the result, which means when K=3, the new observation(X1=3, X2=3, X3=2) will be classified as Class Label: 1.

Gaussian Weighting - In this type, the weight will be calculated based on the distance using a Gaussian function like $w_i = e^{-\alpha \times d_i^2}$ .So when K=3, the nearest three neighbors are at indices 1, 2, and 4 in the data set, their new distance will be:

Neighbor 2 (index 1): New_distance = 1.4142 * 0.1353 = 0.1913

Neighbor 3 (index 2): New_distance = 2 * 0.0183 = 0.1353

Neighbor 5 (index 4): New_distance = 2 * 0.0183 = 0.1353

New class label will 1.

Code

```python
import numpy as np
from scipy.spatial import distance

dataset = np.array([
    [1, 2, 3, 0],
    [2, 3, 1, 1],
    [3, 1, 2, 0],
    [4, 5, 1, 1],
    [3, 3, 4, 0]
])

new_observation = np.array([3, 3, 2])

def knn_classification(K):
    distances = []
    for idx, obs in enumerate(dataset):
        euclidean_distance = distance.euclidean(new_observation, obs[:3])
        distances.append((euclidean_distance, obs[3], idx))

    distances.sort(key=lambda x: x[0])
    nearest_neighbors = distances[:K]

    classes = [neighbor[1] for neighbor in nearest_neighbors]
    majority_class = max(set(classes), key=classes.count)

    return majority_class, nearest_neighbors
def knn_classification_weight(K):
    distances = []
    for idx, obs in enumerate(dataset):
        euclidean_distance = distance.euclidean(new_observation, obs[:3])
        distances.append((euclidean_distance, obs[3], idx))

    distances.sort(key=lambda x: x[0])
    nearest_neighbors = distances[:K]

    classes = [neighbor[1] for neighbor in nearest_neighbors]
    majority_class = max(set(classes), key=classes.count)

    return majority_class, nearest_neighbors


def print_nearest_neighbors(nearest_neighbors, K):
    print(f"\nThe nearest {K} neighbors for K={K}:")
    for neighbor in nearest_neighbors:
        distance, label, index = neighbor
        print(f"Neighbor Index: {index}, Distance: {distance:.4f}, Class Label: {label}")

class_k1, nearest_neighbors_k1 = knn_classification(1)
class_k3, nearest_neighbors_k3 = knn_classification(3)
class_k5, nearest_neighbors_k5 = knn_classification(5)

print(f"Assigned class for K=1: {class_k1}")
print(f"Assigned class for K=3: {class_k3}")
print(f"Assigned class for K=5: {class_k5}")

print_nearest_neighbors(nearest_neighbors_k1, 1)
print_nearest_neighbors(nearest_neighbors_k3, 3)
print_nearest_neighbors(nearest_neighbors_k5, 5)
```