# k-Nearest Neighbors Algorithm (k-NN)

Name: Debatreya Das Roll No. 12212070 CS A4 ML Lab 14

## Loading Dataset

import pandas as pd  
# Load the climate dataset  
data = pd.read\_csv('kNN.csv')  
  
# Inspect the data (first few rows and columns)  
print(data.info())

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 2096 entries, 0 to 2095  
Data columns (total 30 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Year 2096 non-null int64   
 1 Mo 2096 non-null int64   
 2 Globe 2096 non-null float64  
 3 Land 2096 non-null float64  
 4 Ocean 2096 non-null float64  
 5 NH 2096 non-null float64  
 6 Land.1 2096 non-null float64  
 7 Ocean.1 2096 non-null float64  
 8 SH 2096 non-null float64  
 9 Land.2 2096 non-null float64  
 10 Ocean.2 2096 non-null float64  
 11 Trpcs 2096 non-null float64  
 12 Land.3 2096 non-null float64  
 13 Ocean.3 2096 non-null float64  
 14 NoExt 2096 non-null float64  
 15 Land.4 2096 non-null float64  
 16 Ocean.4 2096 non-null float64  
 17 SoExt 2096 non-null float64  
 18 Land.5 2096 non-null float64  
 19 Ocean.5 2096 non-null float64  
 20 NoPol 2096 non-null float64  
 21 Land.6 2096 non-null float64  
 22 Ocean.6 2096 non-null float64  
 23 SoPol 2096 non-null float64  
 24 Land.7 2096 non-null float64  
 25 Ocean.7 2096 non-null float64  
 26 USA48 2096 non-null float64  
 27 USA49 2096 non-null float64  
 28 AUST 2094 non-null float64  
 29 Troposphere 2096 non-null object   
dtypes: float64(27), int64(2), object(1)  
memory usage: 491.4+ KB  
None

data.head()

Year Mo Globe Land Ocean NH Land.1 Ocean.1 SH Land.2 ... \  
0 1978 12 -0.48 -0.51 -0.47 -0.44 -0.46 -0.42 -0.52 -0.62 ...   
1 1979 1 -0.47 -0.64 -0.41 -0.64 -0.86 -0.50 -0.31 -0.13 ...   
2 1979 2 -0.43 -0.56 -0.39 -0.47 -0.57 -0.41 -0.39 -0.53 ...   
3 1979 3 -0.38 -0.51 -0.33 -0.46 -0.51 -0.44 -0.30 -0.53 ...   
4 1979 4 -0.40 -0.57 -0.34 -0.47 -0.62 -0.37 -0.34 -0.46 ...   
  
 NoPol Land.6 Ocean.6 SoPol Land.7 Ocean.7 USA48 USA49 AUST \  
0 -0.39 -0.68 -0.06 -0.45 -0.38 -0.49 -1.29 -1.15 -1.29   
1 -0.46 -0.95 0.10 -0.16 -0.15 -0.16 -3.22 -2.42 0.92   
2 -2.01 -2.30 -1.66 -0.80 -1.25 -0.58 -1.76 -1.84 -0.30   
3 -0.56 -0.47 -0.65 -0.52 -1.25 -0.18 -0.70 -0.39 0.23   
4 -0.84 -0.81 -0.88 -0.26 0.26 -0.51 -0.72 -0.46 -1.12   
  
 Troposphere   
0 Lower   
1 Lower   
2 Lower   
3 Lower   
4 Lower   
  
[5 rows x 30 columns]

# Standard Scaling the data except the last column  
  
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
data.iloc[:, :-1] = scaler.fit\_transform(data.iloc[:, :-1])

C:\Users\debat\AppData\Local\Temp\ipykernel\_16556\2898124973.py:5: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise in a future error of pandas. Value '[-1.76483148 -1.68552002 -1.68552002 ... 1.72487303 1.72487303  
 1.72487303]' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.  
 data.iloc[:, :-1] = scaler.fit\_transform(data.iloc[:, :-1])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_16556\2898124973.py:5: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise in a future error of pandas. Value '[ 1.59907075 -1.58580963 -1.29627505 ... -0.42767131 -0.13813673  
 0.15139785]' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.  
 data.iloc[:, :-1] = scaler.fit\_transform(data.iloc[:, :-1])

## Split the Data

from sklearn.model\_selection import train\_test\_split  
  
# Separate features (X) and labels (y)  
X = data.drop(columns=['Troposphere'])  
y = data['Troposphere'] # 'Troposphere' is the target column  
  
# Split the data into 80% training and 20% testing  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
print('Training set size:', X\_train.shape)  
print('Testing set size:', X\_test.shape)

Training set size: (1676, 29)  
Testing set size: (420, 29)

## Define the Distance Metric (Euclidean Distance)

import numpy as np  
  
def euclidean\_distance(point1, point2):  
 # Ensure both points are numeric and align them by their indices  
 point1 = point1.astype(np.float64)  
 point2 = point2.astype(np.float64)  
 return np.sqrt(np.sum((point1 - point2) \*\* 2))

## Implement the k-NN Algorithm

from collections import Counter  
  
def knn\_predict(X\_train, y\_train, test\_point, k=3):  
 distances = []  
  
 # Make sure test\_point is numeric  
 test\_point = test\_point.astype(np.float64)  
  
 # Calculate the distance from the test point to all training points  
 for i in range(len(X\_train)):  
 train\_point = X\_train.iloc[i]  
  
 # Calculate the Euclidean distance between the test point and the training point  
 distance = euclidean\_distance(train\_point, test\_point)  
 distances.append((distance, y\_train.iloc[i]))  
  
 # Sort distances in ascending order and select the top k  
 distances = sorted(distances)[:k]  
  
 # Extract the labels of the k closest neighbors  
 neighbors\_labels = [label for \_, label in distances]  
  
 # Return the most common label among the neighbors  
 most\_common\_label = Counter(neighbors\_labels).most\_common(1)[0][0]  
 return most\_common\_label

## Make Predictions and Evaluate

# Set k for k-NN  
k = 3  
  
# Lists to store correct and incorrect predictions  
correct\_predictions = []  
incorrect\_predictions = []  
  
# Iterate over each test point and make predictions  
for i in range(len(X\_test)):  
 test\_point = X\_test.iloc[i]  
 true\_label = y\_test.iloc[i]  
 predicted\_label = knn\_predict(X\_train, y\_train, test\_point, k=k)  
   
 # Check if prediction is correct  
 if predicted\_label == true\_label:  
 correct\_predictions.append((test\_point.values, true\_label, predicted\_label))  
 else:  
 incorrect\_predictions.append((test\_point.values, true\_label, predicted\_label))  
  
# Print correct predictions  
print('\nCorrect Predictions:')  
for test\_point, true\_label, predicted\_label in correct\_predictions:  
 print(f'Test Point: {test\_point}, True Label: {true\_label}, Predicted Label: {predicted\_label}')  
  
# Print incorrect predictions  
print('\nIncorrect Predictions:')  
for test\_point, true\_label, predicted\_label in incorrect\_predictions:  
 print(f'Test Point: {test\_point}, True Label: {true\_label}, Predicted Label: {predicted\_label}')

! pandoc kNN.ipynb -o kNN\_12212070.docx