**Name: Adithya M SRN: PES1UG20CS621 Section K**

Code:

import numpy as np

from decimal import Decimal

from math import \*

class KNN:

"""

K Nearest Neighbours model

Args:

k\_neigh: Number of neighbours to take for prediction

weighted: Boolean flag to indicate if the nieghbours contribution

is weighted as an inverse of the distance measure

p: Parameter of Minkowski distance

"""

def \_\_init\_\_(self, k\_neigh, weighted=False, p=2):

self.weighted = weighted

self.k\_neigh = k\_neigh

self.p = p

def fit(self, data, target):

"""

Fit the model to the training dataset.

Args:

data: M x D Matrix( M data points with D attributes each)(float)

target: Vector of length M (Target class for all the data points as int)

Returns:

The object itself

"""

self.data = data

self.target = target.astype(np.int64)

return self

def my\_p\_root(self, value, root):

my\_root\_value = 1 / float(root)

return round(Decimal(value) \*\* Decimal(my\_root\_value), 3)

def my\_minkowski\_distance(self, x, y, p\_value):

return float(

self.my\_p\_root(sum(pow(abs(m - n), p\_value) for m, n in zip(x, y)), p\_value)

)

def find\_distance(self, x):

"""

Find the Minkowski distance to all the points in the train dataset x

Args:

x: N x D Matrix (N inputs with D attributes each)(float)

Returns:

Distance between each input to every data point in the train dataset

(N x M) Matrix (N Number of inputs, M number of samples in the train dataset)(float)

"""

# TODO

r = []

for i in range(x.shape[0]):

m = x[i]

lni = []

for j in range(self.data.shape[0]):

n = self.data[j]

lni.append(self.my\_minkowski\_distance(m, n, self.p))

r.append(lni)

return r

def k\_neighbours(self, x):

"""

Find K nearest neighbours of each point in train dataset x

Note that the point itself is not to be included in the set of k Nearest Neighbours

Args:

x: N x D Matrix( N inputs with D attributes each)(float)

Returns:

k nearest neighbours as a list of (neigh\_dists, idx\_of\_neigh)

neigh\_dists -> N x k Matrix(float) - Dist of all input points to its k closest neighbours.

idx\_of\_neigh -> N x k Matrix(int) - The (row index in the dataset) of the k closest neighbours of each input

Note that each row of both neigh\_dists and idx\_of\_neigh must be SORTED in increasing order of distance

"""

# TODO

lni = self.find\_distance(x)

r = [[], []]

for i in range(len(lni)):

indices = [i for i in range(self.data.shape[0])]

d = list(list(zip(\*list(sorted(zip(lni[i], indices)))))[0])

e = list(list(zip(\*list(sorted(zip(lni[i], indices)))))[1])

r[0].append(d[0 : self.k\_neigh])

r[1].append(e[0 : self.k\_neigh])

return r

def predict(self, x):

"""

Predict the target value of the inputs.

Args:

x: N x D Matrix( N inputs with D attributes each)(float)

Returns:

pred: Vector of length N (Predicted target value for each input)(int)

"""

# TODO

indices = self.k\_neighbours(x)[1]

r = []

for i in range(len(indices)):

f = {}

for j in range(len(indices[i])):

if self.target[indices[i][j]] in f:

f[self.target[indices[i][j]]] += 1

else:

f[self.target[indices[i][j]]] = 1

maxF = 0

maxK = None

for i in range(min(f), max(f) + 1):

if f[i] > maxF:

maxF = f[i]

maxK = i

r.append(maxK)

return r

def evaluate(self, x, y):

"""

Evaluate Model on test data using

classification: accuracy metric

Args:

x: Test data (N x D) matrix(float)

y: True target of test data(int)

Returns:

accuracy : (float.)

"""

# TODO

pred = self.predict(x)

right = np.sum(pred == y)

return 100 \* (right) / len(y)

pass

Output:

