Artificial Intelligence Lab

CSE-0408 Summer 2021

SANJIDA AKTER
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
Muktaeva11@gmail.com

I. K-Nearest Neighbor Problem

!/usr/bin/env python coding: utf-8

In[14]:

Import necessary modules from sklearn.neighbors import KNeighborsClassifier from sklearn.model$_s electionimporttrain_test_s plitfromsklearn.datasetsimportload_iris$

Loading data irisData = load$_i ris()$

Create feature and target arrays X = irisData.data y = irisData.target

Split into training and test set $X_t rain, X_t est, y_t rain, y_t est = train_t est_s plit(X, y, test_s ize = 0.2, random_s tate = 42)$

knn = KNeighborsClassifier(n$_n eighbors = 7)$

knn.fit(X$_t rain, y_t rain)$

Predict on dataset which model has not seen before print(knn.predict(X$_t est)$)

In[13]:

Import necessary modules from sklearn.neighbors import KNeighborsClassifier from sklearn.model$_s electionimporttrain_test_s plitfromsklearn.datasetsimportload_iris$

Loading data irisData = load$_i ris()$

Create feature and target arrays X = irisData.data y = irisData.target

Split into training and test set $X_t rain, X_t est, y_t rain, y_t est = train_t est_s plit(X, y, test_s ize = 0.2, random_s tate = 42)$

knn = KNeighborsClassifier(n$_n eighbors = 7)$

knn.fit(X$_t rain, y_t rain)$

Calculate the accuracy of the model print(knn.score(X$_t est, y_t est)$)

In[12]:

Import necessary modules from sklearn.neighbors import KNeighborsClassifier from sklearn.model$_s electionimporttrain_test_s plitfromsklearn.datasetsimportload_iris$

irisData = load$_i ris()$

Create feature and target arrays X = irisData.data y = irisData.target

Split into training and test set $X_t rain, X_t est, y_t rain, y_t est = train_t est_s plit(X, y, test_s ize = 0.2, random_s tate = 42)$

neighbors = np.arange(1, 9) train$_a ccuracy = np.empty(len(neighbors))test_a ccuracy = np.empty(len(neighbors))$

Loop over K values for i, k in enumerate(neighbors): knn = KNeighborsClassifier(n$_n eighbors = k)knn.fit(X_t rain, y_t rain)$

Compute training and test data accuracy train$_a ccuracy[i] = knn.score(X_t rain, y_t rain)test_a ccuracy[i] = knn.score(X_t est, y_t est)$

Generate plot plt.plot(neighbors, test$_a ccuracy, label = 'TestingdatasetAccuracy')plt.plot(neighbors, train_a ccuracy, label = 'TrainingdatasetAccuracy')$

plt.legend() plt.xlabel('n$_n eighbors')plt.ylabel('Accuracy')plt.show()$

In[ ]:

II. Decision Tree

import numpy as np import matplotlib.pyplot as plt import pandas as pd import math import copy

dataset = pd.read$_c sv('tennis.csv')X = dataset.iloc[:, 1 :].valuesprint(X)attribute = ['outlook','temp','humidity','wind']$

class Node(object): def $_i nit_(self) : self.value = Noneself.decision = Noneself.childs = None$

def findEntropy(data, rows): yes = 0 no = 0 ans = -1 idx = len(data[0]) - 1 entropy = 0 for i in rows: if data[i][idx] == 'Yes': yes = yes + 1 else: no = no + 1

x = yes/(yes+no) y = no/(yes+no) if x != 0 and y != 0: entropy = -1 * (x*math.log2(x) + y*math.log2(y)) if x == 1: ans = 1 if y == 1: ans = 0 return entropy, ans

def findMaxGain(data, rows, columns): maxGain = 0 retidx = -1 entropy, ans = findEntropy(data, rows) if entropy == 0: """if ans == 1: print("Yes") else: print("No")""" return maxGain, retidx, ans

for j in columns: mydict = idx = j for i in rows: key = data[i][idx] if key not in mydict: mydict[key] = 1 else: mydict[key] = mydict[key] + 1 gain = entropy

print(mydict) for key in mydict: yes = 0 no = 0 for k in rows: if data[k][j] == key: if data[k][-1] == 'Yes': yes = yes + 1 else: no = no + 1 print(yes, no) x = yes/(yes+no) y = no/(yes+no) print(x, y) if x != 0 and y != 0: gain += (mydict[key] * (x*math.log2(x) + y*math.log2(y)))/14 print(gain) if gain ¿ maxGain: print("hello") maxGain = gain retidx = j

return maxGain, retidx, ans

def buildTree(data, rows, columns):

maxGain, idx, ans = findMaxGain(X, rows, columns) root = Node() root.childs = [] print(maxGain) if maxGain == 0:

if ans == 1: root.value = 'Yes' else: root.value = 'No' return root

root.value = attribute[idx] mydict = for i in rows: key = data[i][idx] if key not in mydict: mydict[key] = 1 else: mydict[key] += 1

newcolumns = copy.deepcopy(columns) newcolumns.remove(idx) for key in mydict: newrows = [] for i in rows: if data[i][idx] == key: newrows.append(i) print(newrows) temp = buildTree(data, newrows, newcolumns) temp.decision = key root.childs.append(temp) return root

def traverse(root): print(root.decision) print(root.value)

n = len(root.childs) if n ¿ 0: for i in range(0, n): traverse(root.childs[i])

def calculate(): rows = [i for i in range(0, 14)] columns = [i for i in range(0, 4)] root = buildTree(X, rows, columns) root.decision = 'Start' traverse(root)

calculate()

ACKNOWLEDGMENT

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.