PHW 9 / 201835534 차원우

[DecisionTree.py]

import numpy as np

import pandas as pd

import math

# read dataset

ds = pd.read\_csv('marketingRespond.csv')

print(ds)

all = len(ds)

# calculate entropy of the root node

a = len(ds[ds['Outcome'] == 'Responded'])

b = len(ds[ds['Outcome'] == 'Not responded'])

root = -(a/all \* math.log2(a/all) + b/all \* math.log2(b/all))

print("\nEntropy of the root node: ", round(root,2))

print()

# declare a dictionary that can contain each nodes' entropies (for root node)

entropies = {}

# calculate entropy of the "District" root node

ds1 = ds[ds['District'] == 'Suburban']

ds2 = ds[ds['District'] == 'Rural']

ds3 = ds[ds['District'] == 'Urban']

arr = [ds1, ds2, ds3]

val = 0

entropies['District'] = {}

for dataset in arr:

    # count data by 'outcome' for each dataset

    a = len(dataset[dataset['Outcome'] == 'Responded'])

    b = len(dataset[dataset['Outcome'] == 'Not responded'])

    # in case value 'a' or 'b' is 0

    if a != 0 and b != 0:

        node = -(a/len(dataset) \* math.log2(a/len(dataset)) + b/len(dataset) \* math.log2(b/len(dataset)))

    elif a != 0 and b == 0:

        node = -(a/len(dataset) \* math.log2(a/len(dataset)))

    elif a == 0 and b != 0:

        node = -(b/len(dataset) \* math.log2(b/len(dataset)))

    elif a == 0 and b == 0:

        node = 0

    if dataset is ds1:

        print("Entropy of the suburban node: ", round(node,2))

        entropies['District']['Suburban'] = round(node,2)

        val += len(ds1) \* node

    elif dataset is ds2:

        print("Entropy of the rural node: ", round(node,2))

        entropies['District']['Rural'] = round(node,2)

        val += len(ds2) \* node

    elif dataset is ds3:

        print("Entropy of the urban node: ", round(node,2))

        entropies['District']['Urban'] = round(node,2)

        val += len(ds3) \* node

# information gain

gain1 = root - val / all

print("Information gain of the 'District' root node: ", round(gain1,2))

print()

# calculate entropy of the "House Type" root node

ds1 = ds[ds['House Type'] == 'Detached']

ds2 = ds[ds['House Type'] == 'Semi-detached']

ds3 = ds[ds['House Type'] == 'Terrace']

arr = [ds1, ds2, ds3]

val = 0

entropies['House Type'] = {}

for dataset in arr:

    # count data by 'outcome' for each dataset

    a = len(dataset[dataset['Outcome'] == 'Responded'])

    b = len(dataset[dataset['Outcome'] == 'Not responded'])

    # in case value 'a' or 'b' is 0

    if a != 0 and b != 0:

        node = -(a/len(dataset) \* math.log2(a/len(dataset)) + b/len(dataset) \* math.log2(b/len(dataset)))

    elif a != 0 and b == 0:

        node = -(a/len(dataset) \* math.log2(a/len(dataset)))

    elif a == 0 and b != 0:

        node = -(b/len(dataset) \* math.log2(b/len(dataset)))

    elif a == 0 and b == 0:

        node = 0

    if dataset is ds1:

        print("Entropy of the detached node: ", round(node,2))

        entropies['House Type']['Detached'] = round(node,2)

        val += len(ds1) \* node

    elif dataset is ds2:

        print("Entropy of the semi-detached node: ", round(node,2))

        entropies['House Type']['Semi-detached'] = round(node,2)

        val += len(ds2) \* node

    elif dataset is ds3:

        print("Entropy of the terrace node: ", round(node,2))

        entropies['House Type']['Terrace'] = round(node,2)

        val += len(ds3) \* node

# information gain

gain2 = root - val / all

print("Information gain of the 'House Type' root node: ", round(gain2,2))

print()

# calculate entropy of the "Income" root node

ds1 = ds[ds['Income'] == 'High']

ds2 = ds[ds['Income'] == 'Low']

arr = [ds1, ds2]

val = 0

entropies['Income'] = {}

for dataset in arr:

    # count data by 'outcome' for each dataset

    a = len(dataset[dataset['Outcome'] == 'Responded'])

    b = len(dataset[dataset['Outcome'] == 'Not responded'])

    # in case value 'a' or 'b' is 0

    if a != 0 and b != 0:

        node = -(a/len(dataset) \* math.log2(a/len(dataset)) + b/len(dataset) \* math.log2(b/len(dataset)))

    elif a != 0 and b == 0:

        node = -(a/len(dataset) \* math.log2(a/len(dataset)))

    elif a == 0 and b != 0:

        node = -(b/len(dataset) \* math.log2(b/len(dataset)))

    elif a == 0 and b == 0:

        node = 0

    if dataset is ds1:

        print("Entropy of the high node: ", round(node,2))

        entropies['Income']['High'] = round(node,2)

        val += len(ds1) \* node

    elif dataset is ds2:

        print("Entropy of the low node: ", round(node,2))

        entropies['Income']['Low'] = round(node,2)

        val += len(ds2) \* node

# information gain

gain3 = root - val / all

print("Information gain of the 'Income' root node: ", round(gain3,2))

print()

# calculate entropy of the "Previous Customer" root node

ds1 = ds[ds['Previous Customer'] == 'Yes']

ds2 = ds[ds['Previous Customer'] == 'No']

arr = [ds1, ds2]

val = 0

entropies['Previous Customer'] = {}

for dataset in arr:

    # count data by 'outcome' for each dataset

    a = len(dataset[dataset['Outcome'] == 'Responded'])

    b = len(dataset[dataset['Outcome'] == 'Not responded'])

    # in case value 'a' or 'b' is 0

    if a != 0 and b != 0:

        node = -(a/len(dataset) \* math.log2(a/len(dataset)) + b/len(dataset) \* math.log2(b/len(dataset)))

    elif a != 0 and b == 0:

        node = -(a/len(dataset) \* math.log2(a/len(dataset)))

    elif a == 0 and b != 0:

        node = -(b/len(dataset) \* math.log2(b/len(dataset)))

    elif a == 0 and b == 0:

        node = 0

    if dataset is ds1:

        print("Entropy of the yes node: ", round(node,2))

        entropies['Previous Customer']['Yes'] = round(node,2)

        val += len(ds1) \* node

    elif dataset is ds2:

        print("Entropy of the no node: ", round(node,2))

        entropies['Previous Customer']['No'] = round(node,2)

        val += len(ds2) \* node

# information gain

gain4 = root - val / all

print("Information gain of the 'Previous Customer' root node: ", round(gain4,2))

# Root node selection

root = ''

val = max(gain1, gain2, gain3, gain4)

arr = [gain1, gain2, gain3, gain4]

D = dict(zip(ds.columns.values, arr))

for key in D:

    if D[key] == val:

        print("\nThe selected attribute for root node is: ", key)

        root = key

# select leaf nodes to split

leaf = []

for key in entropies[root]:

    if entropies[root][key] != 0:

        leaf.append(key)

print("\n--------------------------------------------------------\n")

branch = []

for i in range(len(leaf)):

    print("Entropy of the " + leaf[i] + " node: ", entropies[root][leaf[i]])

    print()

    new\_ds = ds[ds[root] == leaf[i]]

    all = len(new\_ds)

    # calculate entropy of the "House Type" root node

    ds1 = new\_ds[new\_ds['House Type'] == 'Detached']

    ds2 = new\_ds[new\_ds['House Type'] == 'Semi-detached']

    ds3 = new\_ds[new\_ds['House Type'] == 'Terrace']

    arr = [ds1, ds2, ds3]

    val = 0

    for dataset in arr:

        # count data by 'outcome' for each dataset

        a = len(dataset[dataset['Outcome'] == 'Responded'])

        b = len(dataset[dataset['Outcome'] == 'Not responded'])

        # in case value 'a' or 'b' is 0

        if a != 0 and b != 0:

            node = -(a/len(dataset) \* math.log2(a/len(dataset)) + b/len(dataset) \* math.log2(b/len(dataset)))

        elif a != 0 and b == 0:

            node = -(a/len(dataset) \* math.log2(a/len(dataset)))

        elif a == 0 and b != 0:

            node = -(b/len(dataset) \* math.log2(b/len(dataset)))

        elif a == 0 and b == 0:

            node = 0

        if dataset is ds1:

            print("Entropy of the detached node: ", round(node,2))

            val += len(ds1) \* node

        elif dataset is ds2:

            print("Entropy of the semi-detached node: ", round(node,2))

            val += len(ds2) \* node

        elif dataset is ds3:

            print("Entropy of the terrace node: ", round(node,2))

            val += len(ds3) \* node

    # information gain

    gain2 = entropies[root][leaf[i]] - val / all

    print("Information gain of the 'House Type' root node: ", round(gain2,2))

    print()

    # calculate entropy of the "Income" root node

    ds1 = new\_ds[new\_ds['Income'] == 'High']

    ds2 = new\_ds[new\_ds['Income'] == 'Low']

    arr = [ds1, ds2]

    val = 0

    for dataset in arr:

        # count data by 'outcome' for each dataset

        a = len(dataset[dataset['Outcome'] == 'Responded'])

        b = len(dataset[dataset['Outcome'] == 'Not responded'])

        # in case value 'a' or 'b' is 0

        if a != 0 and b != 0:

            node = -(a/len(dataset) \* math.log2(a/len(dataset)) + b/len(dataset) \* math.log2(b/len(dataset)))

        elif a != 0 and b == 0:

            node = -(a/len(dataset) \* math.log2(a/len(dataset)))

        elif a == 0 and b != 0:

            node = -(b/len(dataset) \* math.log2(b/len(dataset)))

        elif a == 0 and b == 0:

            node = 0

        if dataset is ds1:

            print("Entropy of the high node: ", round(node,2))

            val += len(ds1) \* node

        elif dataset is ds2:

            print("Entropy of the low node: ", round(node,2))

            val += len(ds2) \* node

    # information gain

    gain3 = entropies[root][leaf[i]] - val / all

    print("Information gain of the 'Income' root node: ", round(gain3,2))

    print()

    # calculate entropy of the "Previous Customer" root node

    ds1 = new\_ds[new\_ds['Previous Customer'] == 'Yes']

    ds2 = new\_ds[new\_ds['Previous Customer'] == 'No']

    arr = [ds1, ds2]

    val = 0

    for dataset in arr:

        # count data by 'outcome' for each dataset

        a = len(dataset[dataset['Outcome'] == 'Responded'])

        b = len(dataset[dataset['Outcome'] == 'Not responded'])

        # in case value 'a' or 'b' is 0

        if a != 0 and b != 0:

            node = -(a/len(dataset) \* math.log2(a/len(dataset)) + b/len(dataset) \* math.log2(b/len(dataset)))

        elif a != 0 and b == 0:

            node = -(a/len(dataset) \* math.log2(a/len(dataset)))

        elif a == 0 and b != 0:

            node = -(b/len(dataset) \* math.log2(b/len(dataset)))

        elif a == 0 and b == 0:

            node = 0

        if dataset is ds1:

            print("Entropy of the yes node: ", round(node,2))

            val += len(ds1) \* node

        elif dataset is ds2:

            print("Entropy of the no node: ", round(node,2))

            val += len(ds2) \* node

    # information gain

    gain4 = entropies[root][leaf[i]] - val / all

    print("Information gain of the 'Previous Customer' root node: ", round(gain4,2))

    print()

    # Branch node selection

    val = max(gain2, gain3, gain4)

    arr = [gain2, gain3, gain4]

    D = dict(zip(new\_ds.columns.values[1:], arr))

    for key in D:

        if D[key] == val:

            print("\nThe selected attribute for branch node is: ", key)

            branch.append(key)

    print("\n--------------------------------------------------------\n")

# Tree Configuration

print("<< DecisionTree >>")

print(root)

i = 0

for key in entropies[root]:

    print("- " + key)

    if key == leaf[i]:

        print("-- " + branch[i])

        i += 1

- output –

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

[KNN.py]

import numpy as np

import pandas as pd

from sklearn import preprocessing

import math

# read dataset

ds = pd.read\_csv('T\_Shirt\_Size.csv')

# scaling data using Standard Scaler

# (I just give it a try)

scaler = preprocessing.StandardScaler()

scaled\_df = scaler.fit\_transform(ds.iloc[:, 0:2])

scaled\_df = pd.DataFrame(scaled\_df, columns=['HEIGHT(cm)','WEIGHT(kg)'])

# add 'T SHIRT SIZE' column

tmp = ds['T SHIRT SIZE']

scaled\_df['T SHIRT SIZE'] = tmp

# create new data for distances

new\_height = (161 - ds['HEIGHT(cm)'].mean(axis=0)) / ds['HEIGHT(cm)'].std(axis=0)

new\_weight = (61 - ds['WEIGHT(kg)'].mean(axis=0)) / ds['WEIGHT(kg)'].std(axis=0)

# compute distances between new value and existing values

dist = []

for i in range(len(scaled\_df)):

    val = math.sqrt(math.pow(scaled\_df['HEIGHT(cm)'][i] - new\_height, 2) + math.pow(scaled\_df['WEIGHT(kg)'][i] - new\_weight, 2))

    dist.append(val)

# add new feature 'distance' in the dataset

new = pd.Series(dist)

scaled\_df['distance'] = new

print(scaled\_df)

print()

# according to value 'k', select top k data from sorted dataframe

# and estimate the 'T SHIRT SIZE' of new data

sorted\_df = scaled\_df.sort\_values(by='distance', ascending=True)

sorted\_df = sorted\_df.reset\_index(drop=True)

print(sorted\_df)

# selecting k nearset neighbors' 'T SHIRT SIZE' in array

neighbors = {}

k = 5 # in case k=5

for i in range(k):

    key = sorted\_df['T SHIRT SIZE'][i]

    if key in neighbors.keys():

        neighbors[key] += 1

    else:

        neighbors[key] = 1

print(neighbors)

max\_key = ''

max\_val = 0

for key, value in neighbors.items():

    if value >= max\_val:

        max\_key = key

        max\_val = val

print("The estimated T SHIRT SIZE for the new customer is: ", max\_key)

- output –

텍스트, 명판이(가) 표시된 사진

자동 생성된 설명