#### Decision Tree

import pandas as pd

import math

from collections import Counter

import tkinter as tk

from tkinter import ttk

import os

# Tính entropy của tập dữ liệu

def entropy(data, target\_attr):

    values = data[target\_attr].unique()

    entropy\_val = 0

    for val in values:

        p = len(data[data[target\_attr] == val]) / len(data)

        entropy\_val -= p \* math.log2(p)

    return entropy\_val

# Tính thông tin thu được sau khi chia theo thuộc tính

def info\_gain(data, attr, target\_attr):

    total\_entropy = entropy(data, target\_attr)

    values = data[attr].unique()

    weighted\_entropy = 0

    for val in values:

        subset = data[data[attr] == val]

        weighted\_entropy += (len(subset) / len(data)) \* entropy(subset, target\_attr)

    return total\_entropy - weighted\_entropy

# Thuật toán ID3

def id3(data, target\_attr, attributes, default\_class=None):

    if len(data) == 0:

        return default\_class

    if len(data[target\_attr].unique()) == 1:

        return data[target\_attr].iloc[0]

    if len(attributes) == 0:

        return data[target\_attr].mode()[0]

    default\_class = data[target\_attr].mode()[0]

    gains = [(attr, info\_gain(data, attr, target\_attr)) for attr in attributes]

    best\_attr = max(gains, key=lambda x: x[1])[0]

    tree = {best\_attr: {}}

    for val in data[best\_attr].unique():

        subset = data[data[best\_attr] == val]

        remaining\_attrs = [a for a in attributes if a != best\_attr]

        subtree = id3(subset, target\_attr, remaining\_attrs, default\_class)

        tree[best\_attr][val] = subtree

    return tree

# Dự đoán mẫu mới

def predict(tree, sample):

    if not isinstance(tree, dict):

        return tree

    root = next(iter(tree))

    branch = sample.get(root)

    if branch in tree[root]:

        return predict(tree[root][branch], sample)

    else:

        return None

# Hàm vẽ cây quyết định

# Hàm vẽ cây (fix chồng trùng node)

def draw\_tree(tree, canvas, x, y, dx, dy, level=0):

    if not isinstance(tree, dict):

        canvas.create\_rectangle(x - 30, y - 20, x + 30, y + 20, fill="#90ee90", outline="black")

        canvas.create\_text(x, y, text=tree, font=("Arial", 10, "bold"))

        return

    attr = next(iter(tree))

    children = tree[attr]

    n = len(children)

    total\_width = dx \* (n - 1)

    start\_x = x - total\_width // 2

    # Vẽ node hiện tại

    canvas.create\_rectangle(x - 50, y - 20, x + 50, y + 20, fill="#87cefa", outline="black")

    canvas.create\_text(x, y, text=attr + "?", font=("Arial", 11, "bold"))

    i = 0

    for val, subtree in children.items():

        child\_x = start\_x + i \* dx

        child\_y = y + dy

        canvas.create\_line(x, y + 20, child\_x, child\_y - 20)

        canvas.create\_text((x + child\_x) // 2, (y + child\_y) // 2 - 10, text=str(val), font=("Arial", 9))

        draw\_tree(subtree, canvas, child\_x, child\_y, dx=max(dx // 2, 60), dy=dy, level=level + 1)

        i += 1

# Đọc dữ liệu và xây cây

def run\_decision\_tree(csv\_path, target\_attr):

    if not os.path.exists(csv\_path):

        raise FileNotFoundError(f"Không tìm thấy file: {csv\_path}")

    df = pd.read\_csv(csv\_path)

    if target\_attr not in df.columns:

        raise ValueError(f"Cột mục tiêu '{target\_attr}' không tồn tại trong dữ liệu.")

    attributes = list(df.columns)

    attributes.remove(target\_attr)

    tree = id3(df, target\_attr, attributes)

    return tree, df

# Chạy chương trình chính

if \_\_name\_\_ == "\_\_main\_\_":

    csv\_path = "data.csv"

    target\_attr = "buys\_computer"

    # Huấn luyện cây quyết định

    tree, df = run\_decision\_tree(csv\_path, target\_attr)

    print("\nCây quyết định:")

    print(tree)

    # Dự đoán mẫu đầu tiên

    sample = df.iloc[0].to\_dict()

    true\_label = sample[target\_attr]

    del sample[target\_attr]

    prediction = predict(tree, sample)

    print("\nMẫu dự đoán:")

    print("Input:", sample)

    print("Kết quả dự đoán:", prediction, "| Nhãn thực tế:", true\_label)

    # Vẽ giao diện

    # Vẽ giao diện có thanh cuộn

    root = tk.Tk()

    root.title("Decision Tree Visualization")

    root.geometry("1000x1200")

    # Tạo frame chứa canvas và thanh cuộn

    frame = tk.Frame(root)

    frame.pack(fill=tk.BOTH, expand=True)

    # Tạo canvas với scrollregion rộng để chứa cây lớn

    canvas = tk.Canvas(frame, bg="white", scrollregion=(0, 0, 1500, 1200))

    canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

    # Thanh cuộn dọc

    v\_scroll = tk.Scrollbar(frame, orient=tk.VERTICAL, command=canvas.yview)

    v\_scroll.pack(side=tk.RIGHT, fill=tk.Y)

    canvas.config(yscrollcommand=v\_scroll.set)

    # Thanh cuộn ngang

    h\_scroll = tk.Scrollbar(root, orient=tk.HORIZONTAL, command=canvas.xview)

    h\_scroll.pack(side=tk.BOTTOM, fill=tk.X)

    canvas.config(xscrollcommand=h\_scroll.set)

    # Vẽ cây bắt đầu từ giữa canvas (tránh dính sát viền)

    draw\_tree(tree, canvas, x=700, y=50, dx=400, dy=120)

    root.mainloop()

#### Bayes

import tkinter as tk

from tkinter import ttk

import pandas as pd

import os

# ===== XỬ LÝ NAIVE BAYES =====

def tinh\_xac\_suat(df, feature, value, target\_value):

subset = df[df["buys\_computer"] == target\_value]

count = len(subset[subset[feature] == value])

return count / len(subset) if len(subset) > 0 else 0

def phan\_loai():

global ket\_qua\_du\_doan # lưu kết quả để dùng cho nút thêm

try:

df = pd.read\_csv("data.csv")

except Exception as e:

output\_text.set(f"❌ Không thể đọc file dữ liệu: {e}")

result\_label.config(fg="#c62828")

return

age = age\_combobox.get()

income = income\_combobox.get()

student = student\_combobox.get()

credit = credit\_combobox.get()

# Tính xác suất P(yes) và P(no)

total = len(df)

count\_yes = len(df[df["buys\_computer"] == "yes"])

count\_no = len(df[df["buys\_computer"] == "no"])

p\_yes = count\_yes / total

p\_no = count\_no / total

# Tính P(x|yes) và P(x|no)

p\_age\_yes = tinh\_xac\_suat(df, "age", age, "yes")

p\_income\_yes = tinh\_xac\_suat(df, "income", income, "yes")

p\_student\_yes = tinh\_xac\_suat(df, "student", student, "yes")

p\_credit\_yes = tinh\_xac\_suat(df, "credit\_rating", credit, "yes")

p\_age\_no = tinh\_xac\_suat(df, "age", age, "no")

p\_income\_no = tinh\_xac\_suat(df, "income", income, "no")

p\_student\_no = tinh\_xac\_suat(df, "student", student, "no")

p\_credit\_no = tinh\_xac\_suat(df, "credit\_rating", credit, "no")

# P(x|c) \* P(c)

prob\_yes = p\_yes \* p\_age\_yes \* p\_income\_yes \* p\_student\_yes \* p\_credit\_yes

prob\_no = p\_no \* p\_age\_no \* p\_income\_no \* p\_student\_no \* p\_credit\_no

# Normalize

total\_prob = prob\_yes + prob\_no

norm\_yes = prob\_yes / total\_prob if total\_prob > 0 else 0

norm\_no = prob\_no / total\_prob if total\_prob > 0 else 0

if norm\_yes > norm\_no:

ket\_qua\_du\_doan = "yes"

ket\_luan = "✅ Kết luận: SẼ mua máy tính"

color = "#2e7d32"

else:

ket\_qua\_du\_doan = "no"

ket\_luan = "❌ Kết luận: KHÔNG mua máy tính"

color = "#c62828"

result = (

f"P(yes) = {norm\_yes:.6f}\n"

f"P(no) = {norm\_no:.6f}\n\n"

f"{ket\_luan}"

)

output\_text.set(result)

result\_label.config(fg=color)

# ===== THÊM DÒNG VÀO FILE =====

def them\_vao\_file():

global ket\_qua\_du\_doan

try:

if ket\_qua\_du\_doan not in ["yes", "no"]:

output\_text.set("⚠️ Hãy phân loại trước khi thêm dữ liệu.")

result\_label.config(fg="#c62828")

return

row = [age\_combobox.get(), income\_combobox.get(), student\_combobox.get(), credit\_combobox.get(), ket\_qua\_du\_doan]

file\_path = "data.csv"

is\_exist = os.path.isfile(file\_path)

with open(file\_path, "a") as f:

if not is\_exist:

f.write("age,income,student,credit\_rating,buys\_computer\n")

f.write(",".join(row) + "\n")

output\_text.set("✅ Dòng dữ liệu đã được thêm vào 'data.csv'")

result\_label.config(fg="#2e7d32")

ket\_qua\_du\_doan = None # reset sau khi thêm

except Exception as e:

output\_text.set(f"❌ Lỗi khi thêm dữ liệu: {e}")

result\_label.config(fg="#c62828")

# ===== GIAO DIỆN =====

root = tk.Tk()

root.title("🧠 Naive Bayes - Dự đoán mua máy tính")

root.geometry("520x520")

root.configure(bg="#f1f9fc")

input\_frame = tk.LabelFrame(root, text="📥 Nhập dữ liệu thuộc tính", padx=15, pady=10, bg="#eaeaea", font=("Segoe UI", 10, "bold"))

input\_frame.pack(padx=20, pady=15, fill="x")

tk.Label(input\_frame, text="Tuổi:", bg="#eaeaea").grid(row=0, column=0, sticky="w")

age\_combobox = ttk.Combobox(input\_frame, values=["<=30", "31..40", ">40"], state="readonly")

age\_combobox.grid(row=0, column=1, padx=10, pady=5)

age\_combobox.current(0)

tk.Label(input\_frame, text="Thu nhập:", bg="#eaeaea").grid(row=1, column=0, sticky="w")

income\_combobox = ttk.Combobox(input\_frame, values=["low", "medium", "high"], state="readonly")

income\_combobox.grid(row=1, column=1, padx=10, pady=5)

income\_combobox.current(0)

tk.Label(input\_frame, text="Sinh viên:", bg="#eaeaea").grid(row=2, column=0, sticky="w")

student\_combobox = ttk.Combobox(input\_frame, values=["yes", "no"], state="readonly")

student\_combobox.grid(row=2, column=1, padx=10, pady=5)

student\_combobox.current(0)

tk.Label(input\_frame, text="Xếp hạng tín dụng:", bg="#eaeaea").grid(row=3, column=0, sticky="w")

credit\_combobox = ttk.Combobox(input\_frame, values=["fair", "excellent"], state="readonly")

credit\_combobox.grid(row=3, column=1, padx=10, pady=5)

credit\_combobox.current(0)

# Nút PHÂN LOẠI

btn\_phan\_loai = tk.Button(

root,

text="🛠️ PHÂN LOẠI",

font=("Segoe UI", 12, "bold"),

bg="#4CAF50",

fg="white",

activebackground="#45a049",

padx=10,

pady=5,

command=phan\_loai

)

btn\_phan\_loai.pack(pady=5)

# Nút THÊM VÀO FILE

btn\_them\_file = tk.Button(

root,

text="📥 THÊM VÀO DỮ LIỆU",

font=("Segoe UI", 11, "bold"),

bg="#2196F3",

fg="white",

activebackground="#1976D2",

padx=10,

pady=5,

command=them\_vao\_file

)

btn\_them\_file.pack(pady=5)

# Khung kết quả

result\_frame = tk.LabelFrame(root, text="📊 Kết quả", padx=10, pady=10, bg="#eaeaea", font=("Segoe UI", 10, "bold"))

result\_frame.pack(padx=20, pady=10, fill="both", expand=True)

output\_text = tk.StringVar()

result\_label = tk.Label(

result\_frame,

textvariable=output\_text,

font=("Consolas", 12),

justify="left",

anchor="nw",

bg="white",

wraplength=450

)

result\_label.pack(fill="both", expand=True, padx=5, pady=5)

ket\_qua\_du\_doan = None # Biến lưu trạng thái dự đoán cuối cùng

root.mainloop()

#### Kmean

import numpy as np

import matplotlib.pyplot as plt

# ==== Nhập k và các tâm cụm khởi tạo ====

k = int(input("Nhập số cụm k: "))

centers = []

for i in range(k):

x = float(input(f"Nhập hoành độ tâm cụm C{i+1}: "))

y = float(input(f"Nhập tung độ tâm cụm C{i+1}: "))

centers.append(np.array([x, y]))

# Gán nhãn và màu cho các cụm

labels = [f'C{i+1}' for i in range(k)]

colors = ['red', 'green', 'blue', 'orange', 'purple', 'cyan', 'brown', 'pink', 'gray', 'olive']

# ==== Dữ liệu ====

X = np.array([

[1, 5], [3, 5], [2, 7], [3, 4], [5, 5], [2, 6],

[4, 1], [4, 3], [4, 7], [4, 6], [7, 1], [6, 5],

[8, 1], [4, 8], [8, 3], [12, 4], [3, 16], [7, 14]

])

# ==== Hàm tính khoảng cách Euclidean ====

def euclidean\_distance(point, center):

return np.linalg.norm(point - center)

# ==== Thuật toán K-means ====

max\_iters = 100

tolerance = 1e-4

for iteration in range(1, max\_iters + 1):

cluster\_assignment = []

print(f"\n🔁 Vòng lặp {iteration}")

header = "STT\tX=(x1,x2)" + "".join([f"\td({lbl})" for lbl in labels]) + "\t→ Cụm"

print(header)

print("-" \* len(header.expandtabs()))

for i, x in enumerate(X):

dists = [euclidean\_distance(x, c) for c in centers]

assigned\_cluster = np.argmin(dists)

cluster\_assignment.append(assigned\_cluster)

dist\_str = "\t".join([f"{d:.2f}" for d in dists])

print(f"{i:02}\t({x[0]},{x[1]})\t{dist\_str}\t→ {labels[assigned\_cluster]}")

# Tính lại tâm cụm

new\_centers = []

for i in range(k):

points\_in\_cluster = X[np.array(cluster\_assignment) == i]

if len(points\_in\_cluster) > 0:

new\_center = np.mean(points\_in\_cluster, axis=0)

else:

new\_center = centers[i]

new\_centers.append(new\_center)

# In ra tọa độ tâm cụm mới

print("\n📍 Tọa độ tâm cụm mới:")

for i in range(k):

print(f" - {labels[i]}: ({new\_centers[i][0]:.2f}, {new\_centers[i][1]:.2f})")

# Kiểm tra điều kiện hội tụ

converged = all(np.allclose(new\_centers[i], centers[i], atol=1e-4) for i in range(k))

if converged:

print(f"\n✅ Thuật toán dừng ở vòng lặp {iteration} vì tâm cụm không đổi.")

break

centers = new\_centers

# ==== Vẽ kết quả cuối cùng ====

plt.figure(figsize=(8, 6))

for i, x in enumerate(X):

cluster\_idx = cluster\_assignment[i]

plt.scatter(x[0], x[1], color=colors[cluster\_idx], label=labels[cluster\_idx] if i == cluster\_assignment.index(cluster\_idx) else '', s=50)

for idx, c in enumerate(centers):

plt.scatter(c[0], c[1], color=colors[idx], marker='x', s=150, label=f'{labels[idx]} Center')

for i in range(len(X)):

plt.annotate(str(i), (X[i][0], X[i][1]), textcoords="offset points", xytext=(5, 5), ha='center')

plt.title(f'Phân cụm dữ liệu với K-means (k={k}) - Dừng sau vòng lặp {iteration}')

plt.xlabel('X1')

plt.ylabel('X2')

plt.grid(True)

plt.legend()

plt.show()