Часть А.

Разработать SVM-классификатор для набора данных, указанного в таблице 1

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn import svm

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score, classification\_report

import pandas as pd

import re

from sklearn.preprocessing import LabelEncoder

data = []

with open('C:/Users/TO THANH/OneDrive/Máy tính/hôm nay/university.data', 'r') as file:

    university = {}

    for line in file:

        if '(def-instance' in line:

            university = {'name': line.split()[1]}

        elif '))' in line:

            data.append(university)

            university = {}

        else:

            matches = re.findall('\((.\*?)\)', line)

            if matches:

                split\_data = matches[0].split(None, 1)

                if len(split\_data) == 2:

                    attribute, value = split\_data

                    university[attribute] = value

df = pd.DataFrame(data)

le = LabelEncoder()

for column in df.columns:

    df[column] = le.fit\_transform(df[column].astype(str))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('name', axis=1), df['name'], test\_size=0.2, random\_state=42)

parameters = {'kernel':('linear', 'rbf', 'poly', 'sigmoid'), 'C':[1, 10]}

svc = svm.SVC()

clf = GridSearchCV(svc, parameters)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred, average='macro'))

print("Precision:", precision\_score(y\_test, y\_pred, average='macro'))

print("F1 Score:", f1\_score(y\_test, y\_pred, average='macro'))

print("Number of support vectors:", clf.best\_estimator\_.n\_support\_)

print(classification\_report(y\_test, y\_pred))

best\_classifier = clf.best\_estimator\_

print("The best method is:", best\_classifier)

Accuracy: 0.3076923076923077

Recall: 0.05263157894736842

Precision: 0.05263157894736842

F1 Score: 0.05263157894736842

Number of support vectors: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1]

precision recall f1-score support

0 0.00 0.00 0.00 1

4 0.00 0.00 0.00 0

5 0.00 0.00 0.00 0

6 0.00 0.00 0.00 1

7 0.00 0.00 0.00 0

10 0.00 0.00 0.00 0

12 0.00 0.00 0.00 1

13 0.00 0.00 0.00 1

16 0.00 0.00 0.00 1

21 0.00 0.00 0.00 0

26 0.00 0.00 0.00 0

28 0.00 0.00 0.00 0

30 0.00 0.00 0.00 1

32 0.00 0.00 0.00 1

33 0.00 0.00 0.00 0

35 0.00 0.00 0.00 1

36 0.00 0.00 0.00 0

...

macro avg 0.05 0.05 0.05 13

weighted avg 0.31 0.31 0.31 13

The best method is: SVC(C=1, kernel='linear')

import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

import umap

tsne = TSNE(n\_components=2, random\_state=42)

umap\_model = umap.UMAP(n\_neighbors=5, min\_dist=0.3, n\_components=2, random\_state=42)

tsne\_results = tsne.fit\_transform(df.drop('name', axis=1))

umap\_results = umap\_model.fit\_transform(df.drop('name', axis=1))

plt.figure(figsize=(6,5))

plt.scatter(tsne\_results[:, 0], tsne\_results[:, 1], c=df['name'])

plt.title('t-SNE visualization')

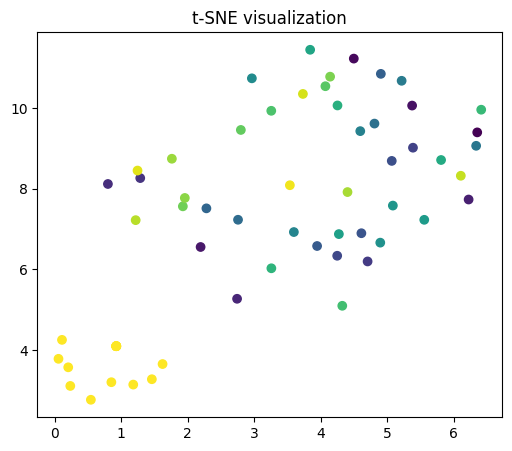
plt.show()

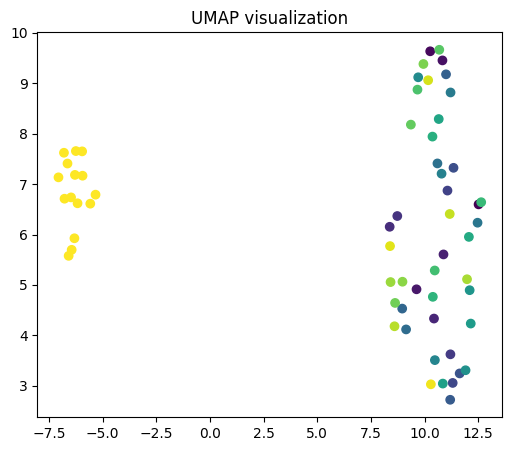
plt.figure(figsize=(6,5))

plt.scatter(umap\_results[:, 0], umap\_results[:, 1], c=df['name'])

plt.title('UMAP visualization')

plt.show()





import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

tsne = TSNE(n\_components=2, random\_state=42)

X\_transformed = tsne.fit\_transform(df.drop('name', axis=1))

X\_train\_transformed = X\_transformed[X\_train.index]

X\_test\_transformed = X\_transformed[X\_test.index]

plt.figure(figsize=(6,5))

plt.scatter(X\_test\_transformed[:, 0], X\_test\_transformed[:, 1], c=y\_test)

plt.title('t-SNE visualization based on actual labels')

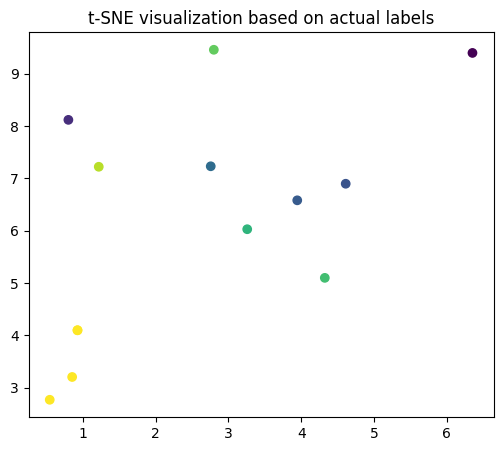
plt.show()

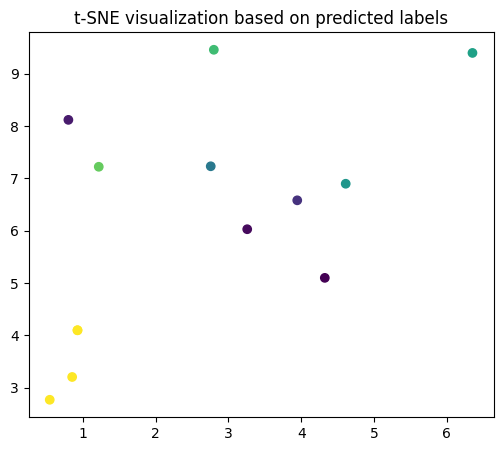
plt.figure(figsize=(6,5))

plt.scatter(X\_test\_transformed[:, 0], X\_test\_transformed[:, 1], c=y\_pred)

plt.title('t-SNE visualization based on predicted labels')

plt.show()





Часть B.

Разработать knn-классификатор для набора данных, указанного в варианте методических указаний (для задания 1)

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn import svm

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score, classification\_report

import pandas as pd

import re

from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier

data = []

with open('C:/Users/TO THANH/OneDrive/Máy tính/hôm nay/university.data', 'r') as file:

    university = {}

    for line in file:

        if '(def-instance' in line:

            university = {'name': line.split()[1]}

        elif '))' in line:

            data.append(university)

            university = {}

        else:

            matches = re.findall('\((.\*?)\)', line)

            if matches:

                split\_data = matches[0].split(None, 1)

                if len(split\_data) == 2:

                    attribute, value = split\_data

                    university[attribute] = value

df = pd.DataFrame(data)

le = LabelEncoder()

for column in df.columns:

    df[column] = le.fit\_transform(df[column].astype(str))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('name', axis=1), df['name'], test\_size=0.2, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred, average='macro'))

print("Precision:", precision\_score(y\_test, y\_pred, average='macro'))

print("F1 Score:", f1\_score(y\_test, y\_pred, average='macro'))

print(classification\_report(y\_test, y\_pred))

parameters = {'n\_neighbors':[3, 5, 7], 'weights':('uniform', 'distance'), 'metric':('euclidean', 'manhattan')}

knn = KNeighborsClassifier()

clf = GridSearchCV(knn, parameters)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

parameters = {'n\_neighbors':[3, 5, 7], 'weights':('uniform', 'distance'), 'metric':('euclidean', 'manhattan')}

knn = KNeighborsClassifier()

clf = GridSearchCV(knn, parameters)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

best\_classifier = clf.best\_estimator\_

print("The best method is:", best\_classifier)

Accuracy: 0.3076923076923077

Recall: 0.0625

Precision: 0.0625

F1 Score: 0.0625

precision recall f1-score support

0 0.00 0.00 0.00 1

2 0.00 0.00 0.00 0

3 0.00 0.00 0.00 0

6 0.00 0.00 0.00 1

7 0.00 0.00 0.00 0

10 0.00 0.00 0.00 0

12 0.00 0.00 0.00 1

13 0.00 0.00 0.00 1

15 0.00 0.00 0.00 0

16 0.00 0.00 0.00 1

30 0.00 0.00 0.00 1

32 0.00 0.00 0.00 1

35 0.00 0.00 0.00 1

36 0.00 0.00 0.00 0

41 0.00 0.00 0.00 1

46 1.00 1.00 1.00 4

accuracy 0.31 13

macro avg 0.06 0.06 0.06 13

weighted avg 0.31 0.31 0.31 13

The best method is: KNeighborsClassifier(metric='euclidean', n\_neighbors=3)

from sklearn.model\_selection import GridSearchCV

from sklearn.manifold import TSNE

import umap

import matplotlib.pyplot as plt

parameters = {'n\_neighbors':[3, 5, 7], 'weights':('uniform', 'distance'), 'metric':('euclidean', 'manhattan')}

knn = KNeighborsClassifier()

clf = GridSearchCV(knn, parameters)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

tsne = TSNE(n\_components=2, random\_state=42)

umap\_model = umap.UMAP(n\_neighbors=5, min\_dist=0.3, n\_components=2, random\_state=42)

tsne\_results = tsne.fit\_transform(df.drop('name', axis=1))

umap\_results = umap\_model.fit\_transform(df.drop('name', axis=1))

plt.figure(figsize=(6,5))

plt.scatter(tsne\_results[:, 0], tsne\_results[:, 1], c=df['name'])

plt.title('t-SNE visualization')

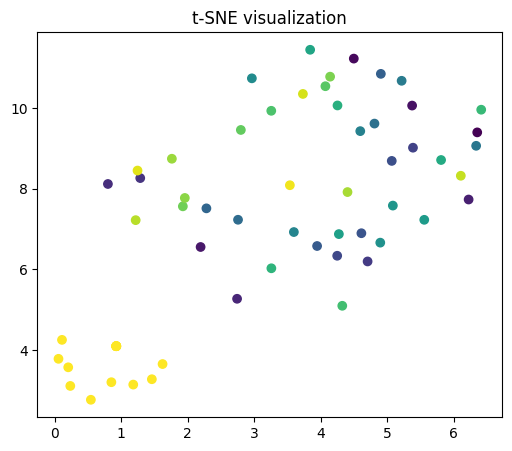
plt.show()

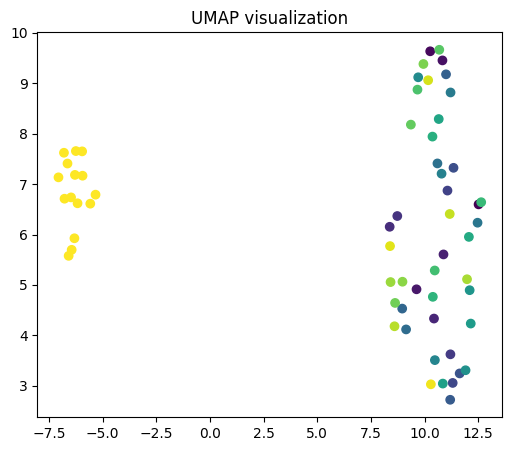
plt.figure(figsize=(6,5))

plt.scatter(umap\_results[:, 0], umap\_results[:, 1], c=df['name'])

plt.title('UMAP visualization')

plt.show()





tsne = TSNE(n\_components=2, random\_state=42, perplexity=10)  # Giả sử số lượng mẫu của bạn là 10

tsne\_results = tsne.fit\_transform(X\_test)

plt.figure(figsize=(6,5))

plt.scatter(tsne\_results[:, 0], tsne\_results[:, 1], c=y\_test)

plt.title('t-SNE visualization based on actual labels')

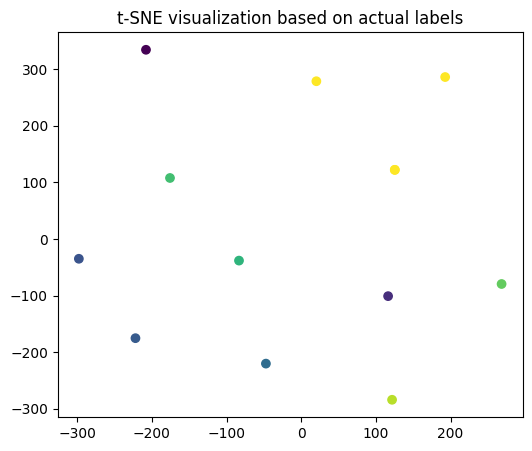
plt.show()

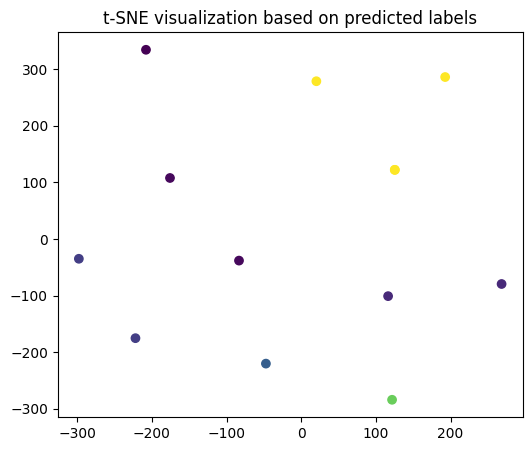
plt.figure(figsize=(6,5))

plt.scatter(tsne\_results[:, 0], tsne\_results[:, 1], c=y\_pred)

plt.title('t-SNE visualization based on predicted labels')

plt.show()





Часть C.

Разработать RF-классификатор (Random Forest) для набора данных, указанного в варианте методических указаний (для задания 1).

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn import svm

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score, classification\_report

import pandas as pd

import re

from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, accuracy\_score, recall\_score, precision\_score, f1\_score

from sklearn.model\_selection import GridSearchCV, train\_test\_split

data = []

with open('C:/Users/TO THANH/OneDrive/Máy tính/hôm nay/university.data', 'r') as file:

    university = {}

    for line in file:

        if '(def-instance' in line:

            university = {'name': line.split()[1]}

        elif '))' in line:

            data.append(university)

            university = {}

        else:

            matches = re.findall('\((.\*?)\)', line)

            if matches:

                split\_data = matches[0].split(None, 1)

                if len(split\_data) == 2:

                    attribute, value = split\_data

                    university[attribute] = value

df = pd.DataFrame(data)

le = LabelEncoder()

for column in df.columns:

    df[column] = le.fit\_transform(df[column].astype(str))

X = df.drop('ACADEMICS', axis=1)

y = df['ACADEMICS']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

rf = RandomForestClassifier(random\_state=42)

rf\_parameters = {'n\_estimators': [100, 200, 300, 400, 500], 'criterion': ['gini', 'entropy'], 'max\_depth': [None, 10, 20, 30, 40, 50]}

knn = KNeighborsClassifier()

knn\_parameters = {'n\_neighbors': [3, 5, 7, 9, 11], 'weights': ['uniform', 'distance'], 'metric': ['euclidean', 'manhattan']}

rf\_clf = GridSearchCV(rf, rf\_parameters)

knn\_clf = GridSearchCV(knn, knn\_parameters)

rf\_clf.fit(X\_train, y\_train)

knn\_clf.fit(X\_train, y\_train)

rf\_y\_pred = rf\_clf.predict(X\_test)

knn\_y\_pred = knn\_clf.predict(X\_test)

print("RF Accuracy:", accuracy\_score(y\_test, rf\_y\_pred))

print("RF Recall:", recall\_score(y\_test, rf\_y\_pred, average='macro'))

print("RF Precision:", precision\_score(y\_test, rf\_y\_pred, average='macro'))

print("RF F1 Score:", f1\_score(y\_test, rf\_y\_pred, average='macro'))

print("KNN Accuracy:", accuracy\_score(y\_test, knn\_y\_pred))

print("KNN Recall:", recall\_score(y\_test, knn\_y\_pred, average='macro'))

print("KNN Precision:", precision\_score(y\_test, knn\_y\_pred, average='macro'))

print("KNN F1 Score:", f1\_score(y\_test, knn\_y\_pred, average='macro'))

print("RF Classification Report:\n", classification\_report(y\_test, rf\_y\_pred))

print("KNN Classification Report:\n", classification\_report(y\_test, knn\_y\_pred))

print("RF Best Parameters: ", rf\_clf.best\_params\_)

print("KNN Best Parameters: ", knn\_clf.best\_params\_)

RF Accuracy: 1.0

RF Recall: 1.0

RF Precision: 1.0

RF F1 Score: 1.0

KNN Accuracy: 0.9230769230769231

KNN Recall: 0.5

KNN Precision: 0.46153846153846156

KNN F1 Score: 0.48000000000000004

RF Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 1

1 1.00 1.00 1.00 12

accuracy 1.00 13

macro avg 1.00 1.00 1.00 13

weighted avg 1.00 1.00 1.00 13

KNN Classification Report:

precision recall f1-score support

0 0.00 0.00 0.00 1

1 0.92 1.00 0.96 12

accuracy 0.92 13

...

weighted avg 0.85 0.92 0.89 13

RF Best Parameters: {'criterion': 'gini', 'max\_depth': None, 'n\_estimators': 100}

KNN Best Parameters: {'metric': 'euclidean', 'n\_neighbors': 3, 'weights': 'uniform'}

import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

import umap

X = df.drop('ACADEMICS', axis=1)

y = df['ACADEMICS']

tsne = TSNE(n\_components=2, random\_state=42)

X\_tsne = tsne.fit\_transform(X)

plt.figure(figsize=(6, 5))

for i, label in enumerate(list(set(y))):

    plt.scatter(X\_tsne[y == label, 0], X\_tsne[y == label, 1], label=label)

plt.legend()

plt.show()

reducer = umap.UMAP(random\_state=42)

X\_umap = reducer.fit\_transform(X)

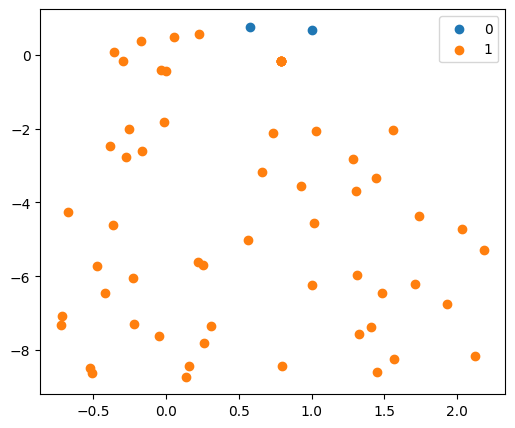
plt.figure(figsize=(6, 5))

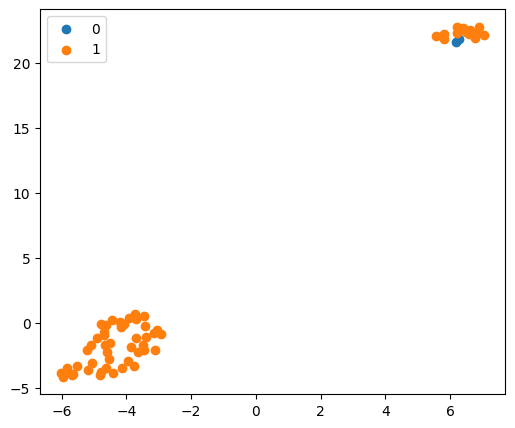
for i, label in enumerate(list(set(y))):

    plt.scatter(X\_umap[y == label, 0], X\_umap[y == label, 1], label=label)

plt.legend()

plt.show()





import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

import umap

X = df.drop('ACADEMICS', axis=1)

y = df['ACADEMICS']

tsne = TSNE(n\_components=2, random\_state=42)

X\_tsne = tsne.fit\_transform(X)

plt.figure(figsize=(6, 5))

for i, label in enumerate(list(set(y))):

    plt.scatter(X\_tsne[y == label, 0], X\_tsne[y == label, 1], label=label)

plt.legend()

plt.title("t-SNE visualization based on actual labels")

plt.show()

ed = rf\_clf.predict(X)

plt.figure(figsize=(6, 5))

for i, label in enumerate(list(set(y\_pred))):

    plt.scatter(X\_tsne[y\_pred == label, 0], X\_tsne[y\_pred == label, 1], label=label)

plt.legend()

plt.title("t-SNE visualization based on predicted labels")

plt.show()

