МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ЗВІТ

З ЛАБОРАТОРНОЇ РОБОТИ №3

З ДИСЦИПЛІНИ: « ПРОГРАМУВАННЯ ІНТЕЛЕКТУАЛЬНИХ ІНФОРМАЦІЙНИХ СИСТЕМ»

|  |  |
| --- | --- |
| Виконав:  ІП-15 Мєшков А.І. | Перевірив:  Курченко О.А. |

Київ 2023

ЗАВДАННЯ

1. Dataset1: /kaggle/input/adult-dataset/adult.csv'

Зробити предікшн Bayesian Classification + Support Vector Machine алгоритмами. Порівняти наступні метрики: Recall, f1-score, Confusion matrix, accuracy score. Порівняти з нуль-гіпотезою і перевірити на оверфітинг. Пояснити результати.

1. Dataset2: <https://www.kaggle.com/code/stieranka/k-nearest-neighbors>

K nearest neighbours.

Те саме що і в 1 завданні, але порівнюємо між собою метрики. Euclidean, Manhattan, Minkowski. Кластери потрібно візуалізувати. Метрики аналогічно п.1

1. Dataset3: <https://www.kaggle.com/code/nuhashafnan/cluster-analysis-kmeans-kmediod-agnes-birch-dbscan>

Agnes,Birch,DBSCAN

Інші методи можна ігнорувати. Зняти метрики (Silhouette Coefficient, ARI, NMI. Можна з п.1-2), пояснити.

1. Dataset4: https://www.kaggle.com/code/datark1/customers-clustering-k-means-dbscan-and-ap

Affinity propagation.

 Порівняти з k-means. Метрики - Silhouette Coefficient, ARI, NMI

У звіті до кожної задачі:

1 Візуалізувати кластери

2 Вивести метрики. Для кластерів - Silhouette Coefficient, ARI, NMI

3 Порівняння з нулем і перевірка на оверфіт.

4 Висновок.

ХІД РОБОТИ

1. **Bayesian Classification + Support Vector Machine**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import LabelEncoder, RobustScaler

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.metrics import recall\_score, f1\_score, confusion\_matrix, accuracy\_score

from scipy import stats

import warnings

warnings.simplefilter("ignore")

data = pd.read\_csv('adult.csv', names=['age', 'workclass', 'fnlwgt', 'education', 'education\_num', 'marital\_status', 'occupation', 'relationship', 'race', 'sex', 'capital\_gain', 'capital\_loss', 'hours\_per\_week', 'native\_country', 'income'])

data['workclass'].replace('?', np.NaN, inplace=True)

data['occupation'].replace('?', np.NaN, inplace=True)

data['native\_country'].replace('?', np.NaN, inplace=True)

data['workclass'].fillna(data['workclass'].mode()[0], inplace=True)

data['occupation'].fillna(data['occupation'].mode()[0], inplace=True)

data['native\_country'].fillna(data['native\_country'].mode()[0], inplace=True)

columns\_to\_encode = ['workclass', 'education', 'marital\_status', 'occupation', 'relationship', 'race', 'sex', 'native\_country']

label\_encoder = LabelEncoder()

for column in columns\_to\_encode:

data[column] = label\_encoder.fit\_transform(data[column])

X = data.drop('income', axis=1)

y = data['income']

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

cols = X\_train.columns

scaler = RobustScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

X\_train = pd.DataFrame(X\_train, columns=[cols])

X\_test = pd.DataFrame(X\_test, columns=[cols])

# Bayesian Classification

bayes\_classifier = GaussianNB()

bayes\_classifier.fit(X\_train, y\_train)

bayes\_predictions = bayes\_classifier.predict(X\_test)

# Support Vector Machine

svm\_classifier = SVC()

svm\_classifier.fit(X\_train, y\_train)

svm\_predictions = svm\_classifier.predict(X\_test)

def evaluate\_model(predictions, model\_name):

recall1 = recall\_score(y\_test, predictions, pos\_label=' >50K')

f1 = f1\_score(y\_test, predictions, pos\_label=' >50K')

c\_matrix = confusion\_matrix(y\_test, predictions)

accuracy = accuracy\_score(y\_test, predictions)

print(f"\nEvaluation metrics for {model\_name}:")

print("Recall: ", recall1)

print("f1\_score: ", f1)

print("Accuracy Score: ", accuracy)

print("Confusion Matrix:")

conf\_matrix = pd.DataFrame(data=c\_matrix, columns=['Actual Positive:1', 'Actual Negative:0'], index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='YlGnBu')

evaluate\_model(bayes\_predictions, "Bayesian Classification")

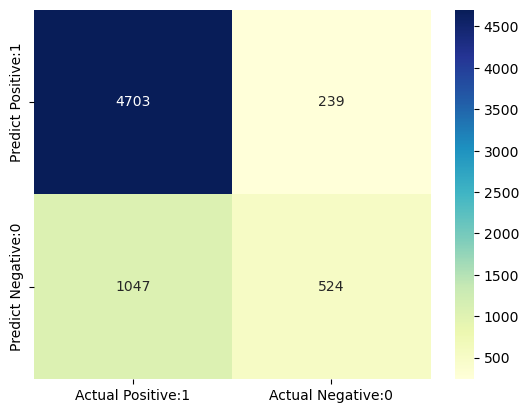
**Output:  
Evaluation metrics for Bayesian Classification:**

**Recall: 0.33354551241247615**

**f1\_score: 0.44901456726649536**

**Accuracy Score: 0.8025487486565331**

**Confusion Matrix:**



evaluate\_model(svm\_predictions, "Support Vector Machine")

**Output:**

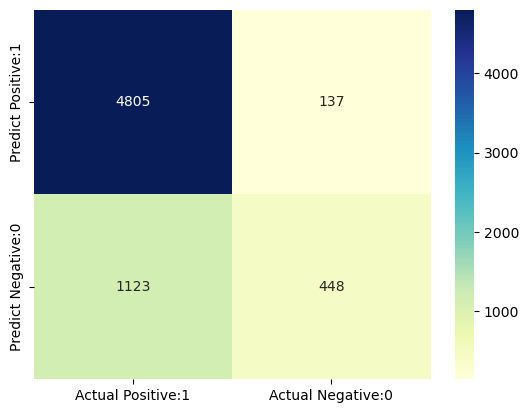
**Evaluation metrics for Support Vector Machine:**

**Recall: 0.28516868236791854**

**f1\_score: 0.4155844155844156**

**Accuracy Score: 0.806540764624597**

**Confusion Matrix:**

****

train\_accuracy\_bayes = accuracy\_score(y\_train, bayes\_classifier.predict(X\_train))

test\_accuracy\_bayes = accuracy\_score(y\_test, bayes\_predictions)

train\_accuracy\_svm = accuracy\_score(y\_train, svm\_classifier.predict(X\_train))

test\_accuracy\_svm = accuracy\_score(y\_test, svm\_predictions)

print(f"Train Bayesian Classification accuracy score: {train\_accuracy\_bayes}")

print(f"Test Bayesian Classification accuracy score: {test\_accuracy\_bayes}")

print(f"Train SVM accuracy score: {train\_accuracy\_svm}")

print(f"Test SVM accuracy score: {test\_accuracy\_svm}")

if train\_accuracy\_bayes > test\_accuracy\_bayes:

print('Bayesian Classification model may be overfitting')

else:

print('Bayesian Classification model is not overfitting')

if train\_accuracy\_svm > test\_accuracy\_svm:

print('SVM model may be overfitting')

else:

print('SVM model is not overfitting')

**Output:**

**Train Bayesian Classification accuracy score: 0.7991784398034398**

**Test Bayesian Classification accuracy score: 0.8025487486565331**

**Train SVM accuracy score: 0.8012899262899262**

**Test SVM accuracy score: 0.806540764624597**

**Bayesian Classification model is not overfitting**

**Bayesian Classification model is not overfitting**

t\_statistic, p\_value = stats.ttest\_rel([test\_accuracy\_bayes], [test\_accuracy\_svm])

alpha = 0.05

if p\_value < alpha:

print('Різниця між моделями статистично значуща - відхиляємо Н0')

else:

print('Різниця між моделями не статистично значуща - не відкидаємо Н0')

**Output:**

**Різниця між моделями не статистично значуща - не відкидаємо Н0**

Висновок:

Так як виявлено, що результат обох моделей не сильно відрізняється, ми не відхиляємо нульову гіпотезу. Bayesian Classification показує кращі результати: recall, f1\_score. SVM показує трохи більшу точність.

1. **K nearest neighbours**

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

def train\_and\_evaluate\_knn(X\_train, y\_train, X\_test, y\_test, metric='euclidean', k=5):

knn\_model = KNeighborsClassifier(n\_neighbors=k, metric=metric)

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

knn\_predictions = knn\_model.predict(X\_test)

recall = recall\_score(y\_test, knn\_predictions, average='weighted')

f1 = f1\_score(y\_test, knn\_predictions, average='weighted')

confusion = confusion\_matrix(y\_test, knn\_predictions)

accuracy = accuracy\_score(y\_test, knn\_predictions)

return recall, f1, confusion, accuracy, knn\_model

def visualize(X, y, model, metric, k):

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=model.predict(X), cmap='viridis')

plt.title(f'KNN Clusters with {metric} Metric (k={k})')

plt.show()

k\_neighbors = 6

metrics = ['euclidean', 'manhattan', 'minkowski']

data2 = pd.read\_csv('teleCust1000t.csv')

X = data2.drop('custcat', axis=1)

y = data2['custcat']

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

cols = X\_train.columns

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

X\_train = pd.DataFrame(X\_train, columns=[cols])

X\_test = pd.DataFrame(X\_test, columns=[cols])

results = []

for metric in metrics:

recall, f1, confusion, accuracy, knn\_model = train\_and\_evaluate\_knn(X\_train, y\_train, X\_test, y\_test, metric, k\_neighbors)

results.append({'Metric': metric, 'Recall': recall, 'F1 Score': f1, 'Confusion Matrix': confusion, 'Accuracy': accuracy, 'KNN model':knn\_model})

visualize(X\_test, y\_test, knn\_model, metric, k\_neighbors)

for result in results:

print(f"""Metric: {result['Metric']}

Recall: {result['Recall']}

F1 Score: {result['F1 Score']}

Accuracy: {result['Accuracy']}""")

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

sns.heatmap(result['Confusion Matrix'], annot=True, fmt='d', cmap='YlGnBu')

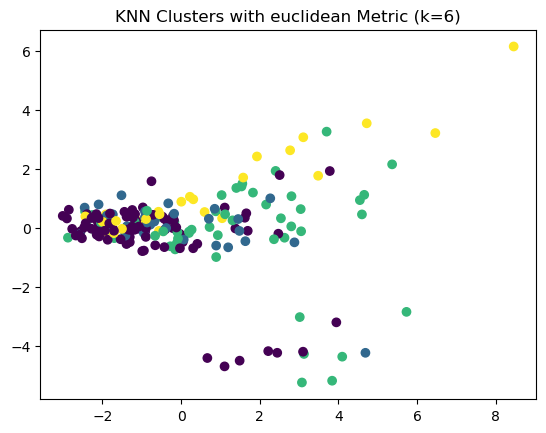
plt.title(f"Confusion matrix ({result['Metric']} Metric)")

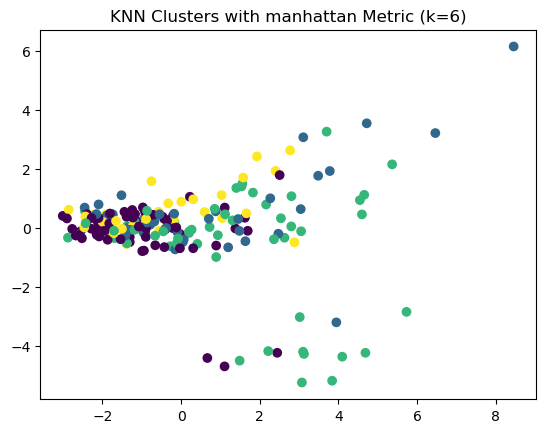
plt.xlabel('Predicted')

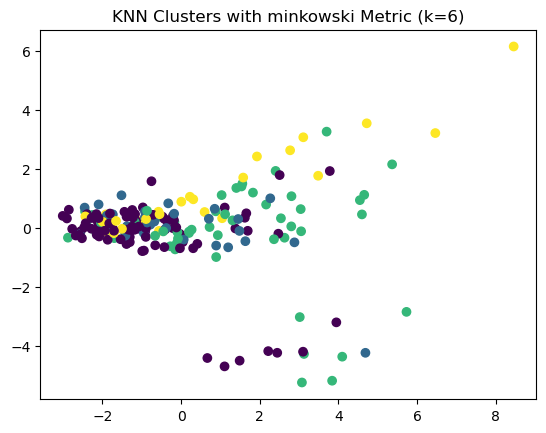
plt.ylabel('Actual')

plt.show()

**Output:**

****

****

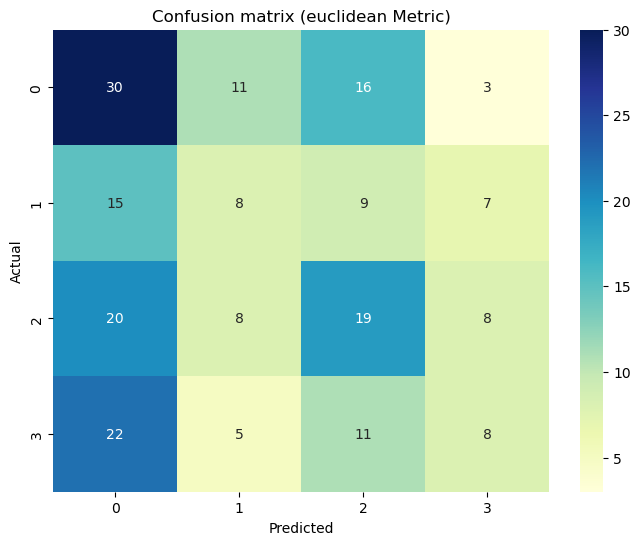
****

**Metric: euclidean**

**Recall: 0.325**

**F1 Score: 0.31250375267477887**

**Accuracy: 0.325**

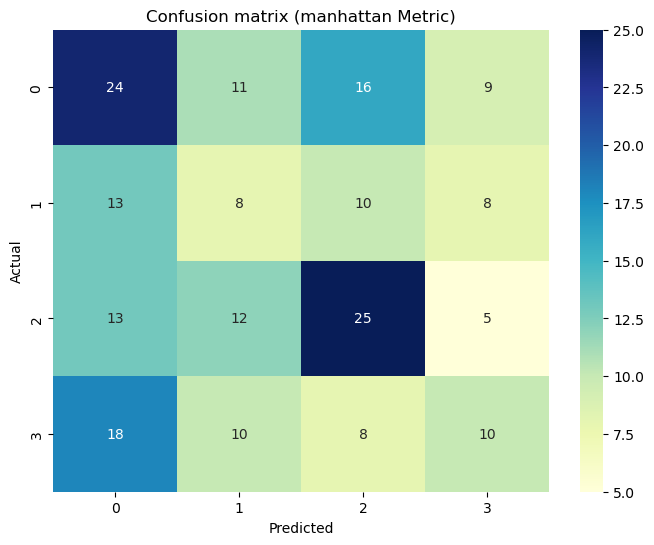
****

**Metric: manhattan**

**Recall: 0.335**

**F1 Score: 0.3310883940620783**

**Accuracy: 0.335**

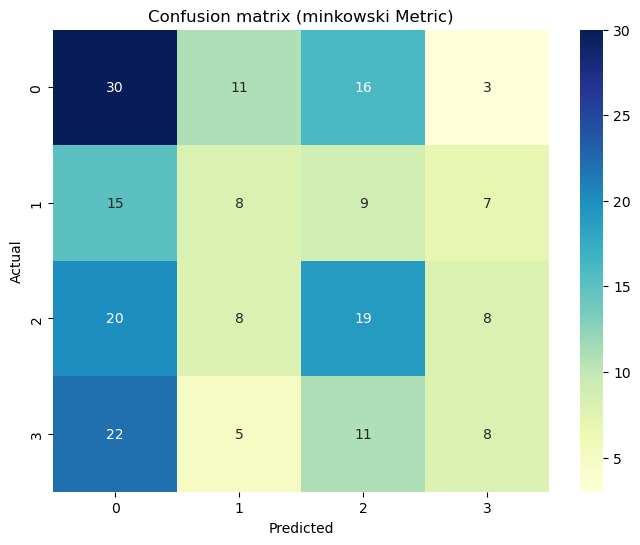
****

**Metric: minkowski**

**Recall: 0.325**

**F1 Score: 0.31250375267477887**

**Accuracy: 0.325**

****

Висновок:

Усі метрики показали схожу продуктивність. Метрики мають низькі значення, що звідчить про те, що модель KNN не дуже добре впоралася з класифікацією даних на цьому датасеті, що може бути результатом неоптимізованого вибору параметрів.

1. **Agnes, Birch, DBSCAN**

from sklearn import cluster, datasets, mixture

from sklearn.datasets import make\_blobs, make\_moons

from sklearn.cluster import AgglomerativeClustering, Birch, DBSCAN

from sklearn.metrics import silhouette\_score, adjusted\_rand\_score, normalized\_mutual\_info\_score

X1, Y1 = make\_moons(n\_samples=2000, noise=.09, random\_state=10)

plt.scatter(X1[:, 0], X1[:, 1], s=10, c=Y1)

plt.title('DATASET 1')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.show()

X2, Y2 = make\_blobs(n\_samples=2000, cluster\_std=3.5, centers=2, n\_features=2, random\_state=10)

plt.scatter(X2[:, 0], X2[:, 1], s=10, c=Y2)

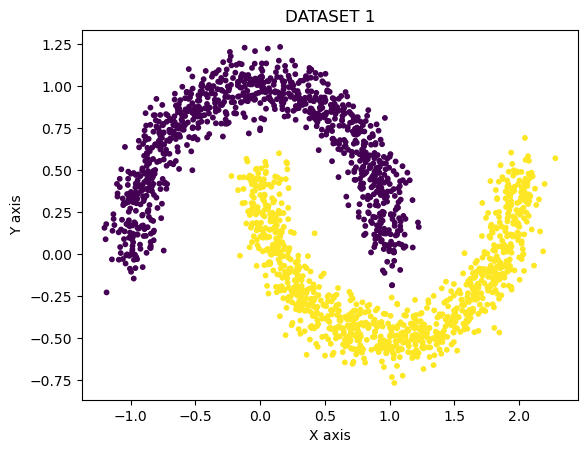
plt.title('DATASET 2')

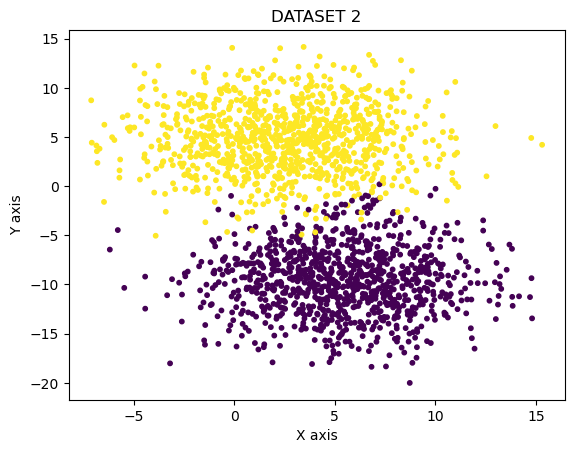
plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.show()

**Output:**

****

****

agglomerative = AgglomerativeClustering(n\_clusters=2)

birch = Birch(n\_clusters=2)

dbscan = DBSCAN(eps=.2, min\_samples=5)

datasets = [(X1,Y1, 'Dataset 1'), (X2,Y2, 'Dataset 2')]

cluster\_models = [agglomerative, birch, dbscan]

results = []

for X,Y,dataset\_name in datasets:

for model in cluster\_models:

labels = model.fit\_predict(X)

silhouette = silhouette\_score(X, labels)

ari = adjusted\_rand\_score(Y, labels)

nmi = normalized\_mutual\_info\_score(Y, labels)

results.append(f"""

{dataset\_name}, {model.\_\_class\_\_.\_\_name\_\_}

Silhouette Coefficient: {silhouette}

ARI: {ari}

NMI: {nmi}

""")

for res in results:

print(res)

**Output:**

**Dataset 1, AgglomerativeClustering**

**Silhouette Coefficient: 0.4062161591537881**

**ARI: 0.7155769186783432**

**NMI: 0.6713586477684496**

**Dataset 1, Birch**

**Silhouette Coefficient: 0.4583503187056948**

**ARI: 0.3767076067566142**

**NMI: 0.341366173543779**

**Dataset 1, DBSCAN**

**Silhouette Coefficient: 0.2565169878472582**

**ARI: 0.0**

**NMI: 0.0009941928229785931**

**Dataset 2, AgglomerativeClustering**

**Silhouette Coefficient: 0.5878339188420266**

**ARI: 0.90816307882887**

**NMI: 0.8427393441408568**

**Dataset 2, Birch**

**Silhouette Coefficient: 0.5431524473083246**

**ARI: 0.7533026115565626**

**NMI: 0.7031895934229431**

**Dataset 2, DBSCAN**

**Silhouette Coefficient: -0.6345258021400355**

**ARI: 0.0009284826724144531**

**NMI: 0.05429587739656559**

Висновок:

Dataset1:

* Agglomerative Clustering має результати метрик, які показують, що кластери мають певну ступінь відокремленості, але загалом кластерізація непогана.
* Birch показує гірші результати, гірша кластерізація.
* DBSCAN не зміг якісно розділити дані на кластери, що свідчить про низьку ефективність методу для цього набору даних.

Dataset2:

* Agglomerative Clustering показує гарні результати. Це вказує на високу якість кластерізації.
* Birch показує гірші результати, але вище середнього.
* DBSCAN не зміг якісно розділити дані на кластери, що свідчить про низьку ефективність методу для цього набору даних.

Загалом Agglomerative Clustering та Birch мають вищі показники якості кластерізації порівняно з DBSCAN.

1. **Affinity propagation**

from sklearn.cluster import AffinityPropagation, KMeans

from sklearn.manifold import TSNE

data4 = pd.read\_csv('Mall\_Customers.csv')

X = data4[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

scaler = StandardScaler()

X = scaler.fit\_transform(X)

affinity\_propagation = AffinityPropagation(damping=0.9)

affinity\_labels = affinity\_propagation.fit\_predict(X)

kmeans = KMeans(n\_clusters=3)

kmeans\_labels = kmeans.fit\_predict(X)

silhouette\_affinity = silhouette\_score(X, affinity\_labels)

ari\_affinity = adjusted\_rand\_score(data4['CustomerID'], affinity\_labels)

nmi\_affinity = normalized\_mutual\_info\_score(data4['CustomerID'], affinity\_labels)

silhouette\_kmeans = silhouette\_score(X, kmeans\_labels)

ari\_kmeans = adjusted\_rand\_score(data4['CustomerID'], kmeans\_labels)

nmi\_kmeans = normalized\_mutual\_info\_score(data4['CustomerID'], kmeans\_labels)

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

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

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)

plt.scatter(X\_tsne[:, 0], X\_tsne[:, 1], c=affinity\_labels, cmap='viridis', s=50)

plt.title('Affinity Propagation Clusters')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.subplot(1,2,2)

plt.scatter(X\_tsne[:, 0], X\_tsne[:, 1], c=kmeans\_labels, cmap='viridis', s=50)

plt.title('K-means Clusters')

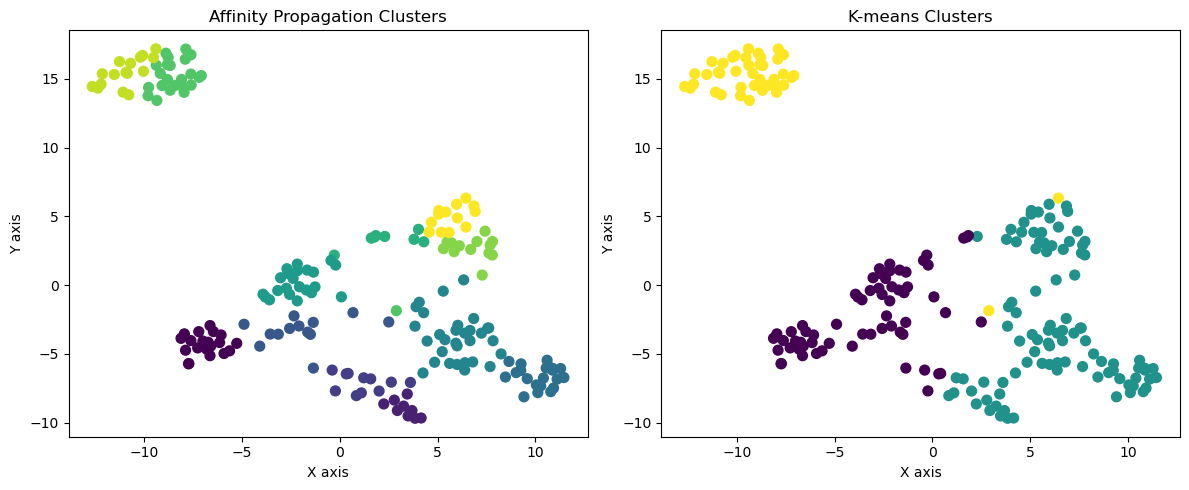
plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.tight\_layout()

plt.show()

**Output:**



print(f"""

Affinity Propagation

Silhouette Coefficient: {silhouette\_affinity}

ARI: {ari\_affinity}

NMI: {nmi\_affinity}""")

print(f"""

K-Means

Silhouette Coefficient: {silhouette\_kmeans}

ARI: {ari\_kmeans}

NMI: {nmi\_kmeans}""")

**Output:**

**Affinity Propagation**

**Silhouette Coefficient: 0.36667189318854604**

**ARI: 0.0**

**NMI: 0.6252294886222641**

**K-Means**

**Silhouette Coefficient: 0.3579234303882264**

**ARI: 0.0**

**NMI: 0.3284424877118733**

Висновок:

Affinity Propagation показава кращу внутрішню схожість кластерів та схожість з істинними кластерами в порівнянні з K-Means, на основі Silhouette Coefficient та NMI. Однак обидва методи мають нульове значення ARI, що вказує на несхожість результатів з істинними мітками, це може свідчити про необхідність додаткового налаштування параметрів методів кластерізації.