

ЛАБОРАТОРНА РОБОТА № 4

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВО- РЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні та створити рекомендаційні системи.

Хід роботи:

Завдання 1

Лістинг програми:

```
import argparse
import numpy as np
import matplotlib.pyplot as plt

from sklearn.metrics import classification_report
from sklearn import model_selection

from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

from sklearn.metrics import classification_report
from utilities import visualize_classifier
#Парсер аргументів
import argparse

def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify data using Ensemble Learning techniques')

    parser.add_argument('--classifier-type',
                        dest='classifier_type',
                        required=True,
                        choices=['rf', 'erf'],
                        help="Type of classifier to use; can be either 'rf' or 'erf'")
```

					ДУ «Житомирська політехніка».22.122.4.000 – Лр4			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Дяченко В.В.			Звіт з лабораторної роботи		Лім.	Арк.
Перевір.								1
Керівник							ФІКТ Гр. КН-20-1(1)	
Н. контр.								
Зав. каф.								

```

    return parser
if __name__ == "__main__":
    import argparse
    import sys

    sys.argv = [sys.argv[0], '--classifier-type', 'rf']
    # Вилучення вхідних аргументів
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type
# Завантаження вхідних даних
import numpy as np

input_file = 'data_random_forests.txt'

data = np.loadtxt(input_file, delimiter=',')

X, y = data[:, :-1], data[:, -1]

# Розбиття вхідних даних на три класи
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])
# Візуалізація вхідних даних
import matplotlib.pyplot as plt

plt.figure()

plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
            facecolors='white', edgecolors='black',
            linewidth=1, marker='s')

plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
            facecolors='white', edgecolors='black',
            linewidth=1, marker='o')

plt.scatter(class_2[:, 0], class_2[:, 1], s=75,
            facecolors='white', edgecolors='black',
            linewidth=1, marker='*')

plt.title('Вхідні дані')
plt.show()

# Розбивка даних на навчальний та тестовий набори
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)
# Класифікатор на основі ансамблевого навчання

```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		2

```

params = {'n_estimators': 100, 'max_depth': 10, 'random_state': 0}
if classifier_type == 'rf':
    classifier = RandomForestClassifier(**params)
else:
    classifier = ExtraTreesClassifier(**params)

classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train)
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test)

# Перевірка роботи класифікатора
class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train, classifier.predict(X_train), tar-
get_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred, target_names=class_names))
print("#" * 40 + "\n")

```

Створення класифікатора на основі випадкового лісу за допомогою прапорця `rf` вхідного аргументу:

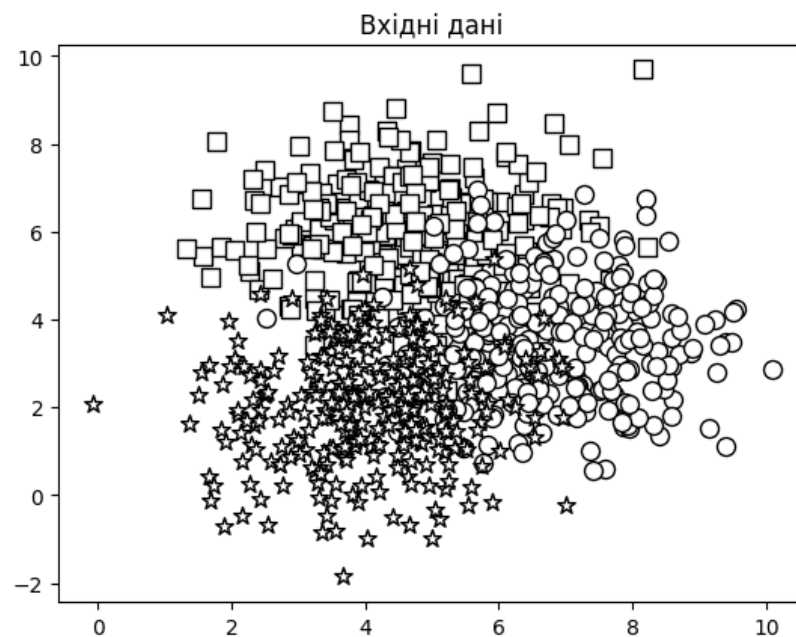


Рис.1. Графік вхідних даних

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		3

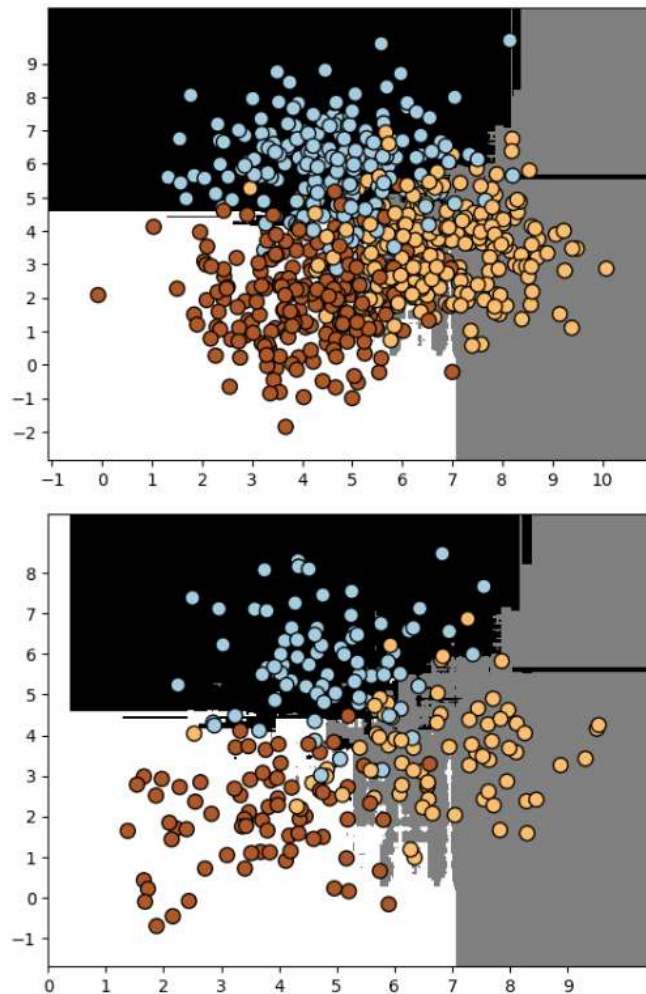


Рис.2. Візуалізація класифікатора на тренувальному та тестовому наборах.

```
#####
Classifier performance on training dataset
      precision    recall  f1-score   support

   Class-0       1.00      0.98      0.99        221
   Class-1       0.97      1.00      0.99        230
   Class-2       1.00      0.99      1.00        224

 accuracy              0.99        675
 macro avg              0.99      0.99        675
 weighted avg           0.99      0.99      0.99        675

#####
#####

Classifier performance on test dataset
      precision    recall  f1-score   support

   Class-0       0.90      0.82      0.86         79
   Class-1       0.84      0.81      0.83         70
   Class-2       0.81      0.91      0.86         76

 accuracy              0.85        225
 macro avg              0.85      0.85        225
 weighted avg           0.85      0.85      0.85        225

#####
```

Рис.3. Звіт із результатами класифікації.

Створення класифікатора на основі гранично випадкового лісу за допомогою прапорця `erf` вхідного аргументу:

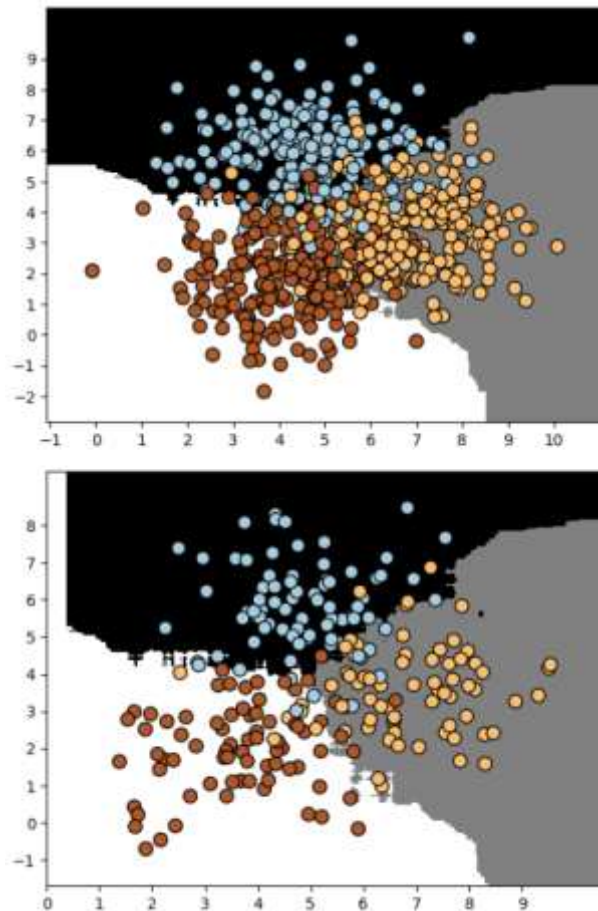


Рис.4. Візуалізація класифікатора на тренувальному та тестовому наборах.

```
#####
Classifier performance on training dataset
      precision    recall  f1-score   support

   Class-0       0.93      0.91      0.92        221
   Class-1       0.86      0.90      0.88        230
   Class-2       0.91      0.89      0.90        224

 accuracy          0.90          675
  macro avg       0.90      0.90      0.90          675
  weighted avg    0.90      0.90      0.90          675

#####
#####
Classifier performance on test dataset
      precision    recall  f1-score   support

   Class-0       0.92      0.86      0.89         79
   Class-1       0.86      0.84      0.85         70
   Class-2       0.84      0.91      0.87         76

 accuracy          0.87          225
  macro avg       0.87      0.87      0.87          225
  weighted avg    0.87      0.87      0.87          225

#####
```

Рис.5. Звіт із результатами класифікації.

Лістинг програми:

```
# Обчислення параметрів довірливості
test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 11]])
print("\nConfidence measure:")
for datapoint in test_datapoints:
    probabilities = classifier.predict_proba([datapoint])[0]
    predicted_class = 'Class-' + str(np.argmax(probabilities))
    print('\nDatapoint:', datapoint)
    print('Predicted class:', predicted_class)
# Візуалізація точок даних
visualize_classifier(classifier, test_datapoints, [0]*len(test_datapoints))
plt.show()
```

Прапорець rf:

```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2

Datapoint: [ 5 11]
Predicted class: Class-0
```

Рис.6. Класифікація точок та рівень довірливості.

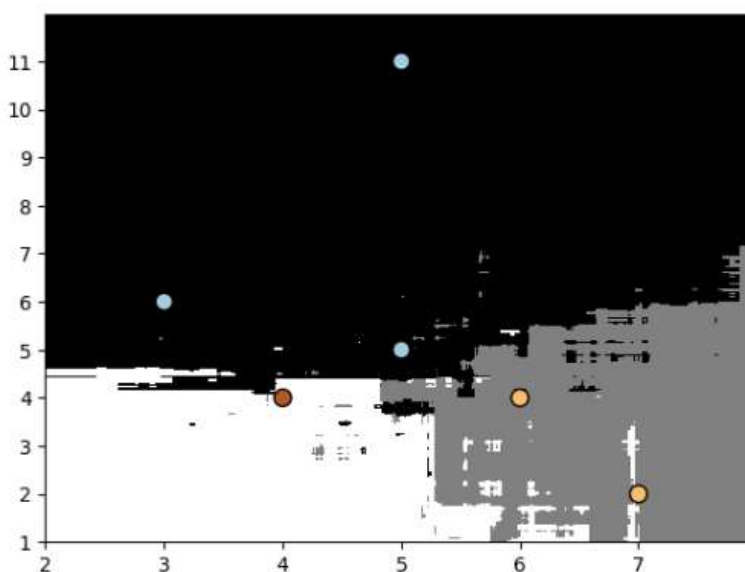


Рис.7. Візуалізація точок даних.

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		6

Прапорець erf:

```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2

Datapoint: [ 5 11]
Predicted class: Class-0
```

Рис.8. Класифікація точок та рівень довірливості.

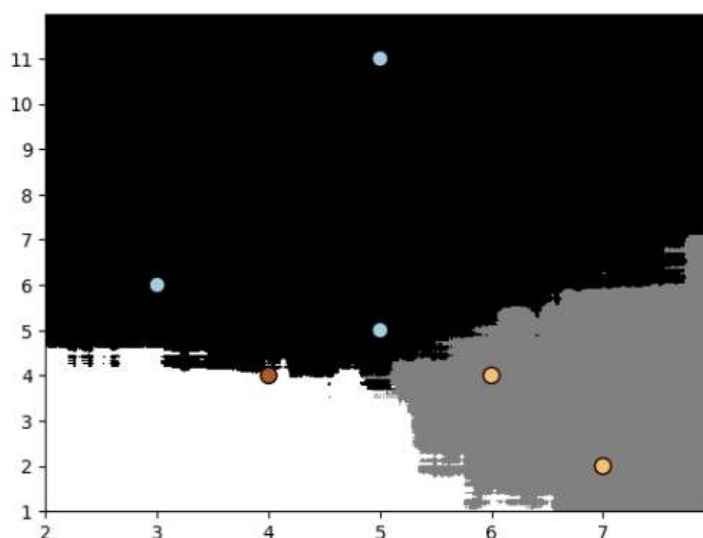


Рис.9. Візуалізація точок даних.

Висновки: під час виконання даного завдання було використано два класифікатори: **Random Forest (rf)** та **Extra Random Forest (erf)**.

Результати на навчальному наборі даних показали високу точність та повноту для обох класифікаторів. Це свідчить про їхню здатність ефективно розпізнавати класи на даних, на яких вони навчалися. Моделі також стабільні на нових даних, що показали результати на тестовому наборі. Візуально **графіки** з результатами класифікацій схожі. Проте **Random Forest** краще спрогнозував дані на тренувальному наборі, **Extra Random Forest** – на тестувальному.

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

Візуалізація тестових точок даних на підставі меж класифікатора також майже ідентична, через те що, класифікатори однаково передбачили класи для окремих точок обчислюючи рівні довірливості.

Завдання 2

Лістинг програми:

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import model_selection
from sklearn.metrics import classification_report
from utilities import visualize_classifier
# Завантаження вхідних даних
input_file = 'data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
# Поділ вхідних даних на два класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
# Візуалізація вхідних даних
import warnings

# Встановлення рівня повідомлень про попередження на "ignore"
warnings.filterwarnings("ignore")
plt.figure()

plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black', edgecolors='black', linewidth=1, marker='x')

plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='o')

plt.title('Вхідні дані')
# Розбиття даних на навчальний та тестовий набори
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)
import sys

# Встановлення значення sys.argv
sys.argv = ['', 'balance'] # 'balance' як аргумент

params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		


```

if len(sys.argv) > 1:
    if sys.argv[1] == 'balance':
        params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0,
                  'class_weight': 'balanced'}
    else:
        print("Invalid input argument; should be 'balance'")
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train)
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test)
# Обчислення показників ефективності класифікатора
class_names = ['Class-0', 'Class-1']

print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train, classifier.predict(X_train), tar-
get_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)

print("\nClassifier performance on test dataset\n")

print(classification_report(y_test, y_test_pred, target_names=class_names))

print("#" * 40 + "\n")

plt.show()

```

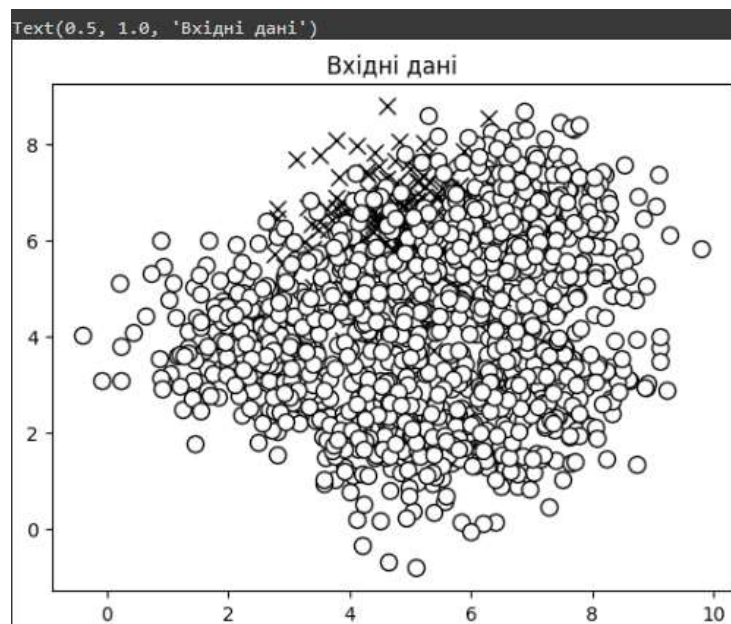


Рис.10. Вхідні дані.

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

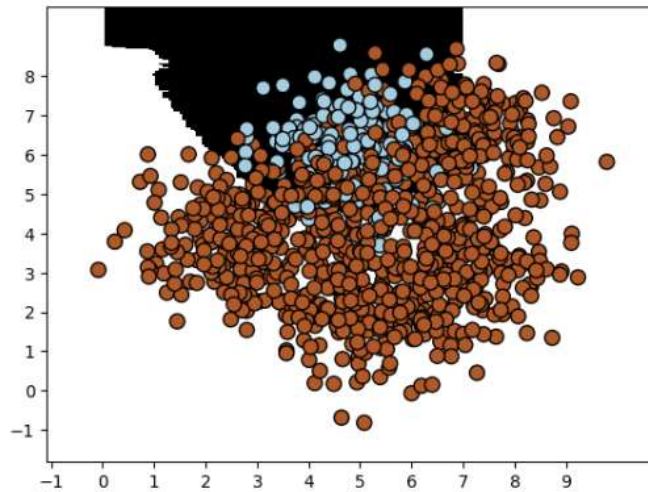


Рис.11. Візуалізація результату на тренувальному наборі.

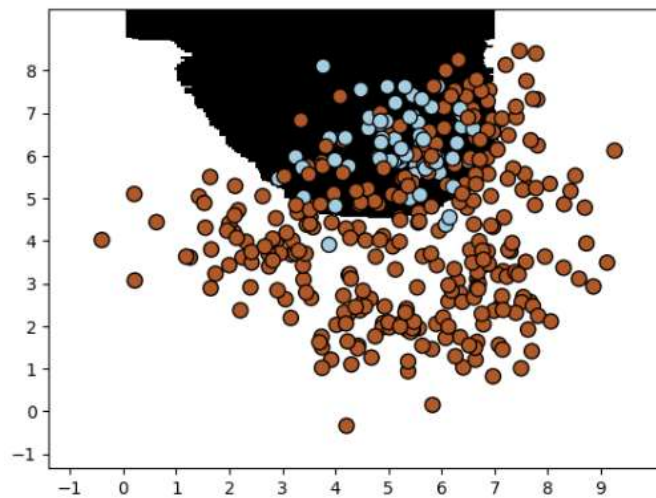


Рис.12. Візуалізація результату на тестовому наборі.

```
#####
Classifier performance on training dataset
      precision    recall  f1-score   support

Class-0       0.44       0.93       0.60       181
Class-1       0.98       0.77       0.86       944

 accuracy          0.80       1125
macro avg          0.71       0.85       0.73       1125
weighted avg       0.89       0.80       0.82       1125

#####
#####
Classifier performance on test dataset
      precision    recall  f1-score   support

Class-0       0.45       0.94       0.61        69
Class-1       0.98       0.74       0.84       306

 accuracy          0.78       375
macro avg          0.72       0.84       0.73       375
weighted avg       0.88       0.78       0.80       375

#####
```

Рис.13. Звіт із результатами класифікації.

Висновки: Модель класифікатора виявила труднощі в визначенні чіткої межі між двома класами. Це зрозуміло з графіку класифікації. Видно, що модель не може чітко розділити точки обох класів, і є чорна пляма в верхній частині малюнка, яка вказує на невизначеність. Точки обох класів переплітаються, що робить завдання класифікації складним.

- Модель має низьку точність та високу повноту для **Class-0** - вона визначає багато негативних прикладів як позитивні.
- Модель має високу точність та низьку повноту для **Class-1** - вона визначає багато позитивних прикладів як негативні.

На тестовому наборі даних результати подібні до результатів на навчальному. Загальна точність становить приблизно 78%.

Завдання 3

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import model_selection
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report
from utilities import visualize_classifier

# Завантаження вхідних даних
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Розбиття даних на три класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])

# Розбивка даних на навчальний та тестовий набори
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)

# Визначення сітки значень параметрів
parameter_grid = [
    {'n_estimators': [100], 'max_depth': [2, 4, 7, 12, 16]},
    {'max_depth': [4], 'n_estimators': [25, 50, 100, 250]}
]
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

```

# Метрики для вимірювання якості класифікації з урахуванням ваги класів
metrics = ['precision_weighted', 'recall_weighted']
# Створення окремих класифікаторів для precision і recall
precision_classifier = model_selection.GridSearchCV(
    ExtraTreesClassifier(random_state=0),
    parameter_grid, cv=5, scoring='precision_weighted')
recall_classifier = model_selection.GridSearchCV(
    ExtraTreesClassifier(random_state=0),
    parameter_grid, cv=5, scoring='recall_weighted')

for metric, classifier in [('precision_weighted', precision_classifier), ('re-
call_weighted', recall_classifier)]:
    print("\n#### Searching optimal parameters for", metric)
    classifier.fit(X_train, y_train)

    # Пошук оптимальних параметрів класифікатора для поточної метрики
    print("\nGrid scores for the parameter grid:")
    for params, avg_score, _ in zip(classifier.cv_results_['params'], classi-
fier.cv_results_['mean_test_score'], classifi-
er.cv_results_['std_test_score']):
        print(params, '-->', round(avg_score, 3))

    print("\nBest parameters for", metric, ":", classifier.best_params_)
# Звіт із результатами роботи класифікаторів та результати передбачень
precision_y_pred = precision_classifier.predict(X_test)
recall_y_pred = recall_classifier.predict(X_test)

print("\nPerformance report for precision:\n")
print(classification_report(y_test, precision_y_pred))

print("\nPerformance report for recall:\n")
print(classification_report(y_test, recall_y_pred))

```

```

#### Searching optimal parameters for precision_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.85
{'max_depth': 4, 'n_estimators': 100} --> 0.841
{'max_depth': 7, 'n_estimators': 100} --> 0.844
{'max_depth': 12, 'n_estimators': 100} --> 0.832
{'max_depth': 16, 'n_estimators': 100} --> 0.816
{'max_depth': 4, 'n_estimators': 25} --> 0.846
{'max_depth': 4, 'n_estimators': 50} --> 0.84
{'max_depth': 4, 'n_estimators': 100} --> 0.841
{'max_depth': 4, 'n_estimators': 250} --> 0.845

Best parameters for precision_weighted : {'max_depth': 2, 'n_estimators': 100}

```

Рис.14. Оцінка для кожної комбінації параметрів (показник precision).

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

```
##### Searching optimal parameters for recall_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.843
{'max_depth': 4, 'n_estimators': 100} --> 0.837
{'max_depth': 7, 'n_estimators': 100} --> 0.841
{'max_depth': 12, 'n_estimators': 100} --> 0.83
{'max_depth': 16, 'n_estimators': 100} --> 0.815
{'max_depth': 4, 'n_estimators': 25} --> 0.843
{'max_depth': 4, 'n_estimators': 50} --> 0.836
{'max_depth': 4, 'n_estimators': 100} --> 0.837
{'max_depth': 4, 'n_estimators': 250} --> 0.841

Best parameters for recall_weighted : {'max_depth': 2, 'n_estimators': 100}
```

Рис.15. Оцінка для кожної комбінації параметрів (показник recall).

Performance report for precision:					
	precision	recall	f1-score	support	
0.0	0.94	0.81	0.87	79	
1.0	0.81	0.86	0.83	70	
2.0	0.83	0.91	0.87	76	
accuracy			0.86	225	
macro avg	0.86	0.86	0.86	225	
weighted avg	0.86	0.86	0.86	225	

Performance report for recall:					
	precision	recall	f1-score	support	
0.0	0.94	0.81	0.87	79	
1.0	0.81	0.86	0.83	70	
2.0	0.83	0.91	0.87	76	
accuracy			0.86	225	
macro avg	0.86	0.86	0.86	225	
weighted avg	0.86	0.86	0.86	225	

Рис.16. Звіт із результатами роботи класифікаторів.

Висновки: Загалом, отримані результати показують, що найкращі параметри моделі для обох метрик є однаковими, хоча для recall усі значення трохи менше. Найкраща комбінація параметрів: {'max_depth': 2, 'n_estimators': 100} Також модель має високі показники точності та відновлення для всіх класів.

Завдання 4

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

```

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import datasets
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.utils import shuffle

# Завантаження даних із цінами на нерухомість
from sklearn.datasets import fetch_california_housing
housing_data = fetch_california_housing()
# Перемішування даних
X, y = shuffle(housing_data.data, housing_data.target, random_state=7)
# Розбиття даних на навчальний та тестовий набори
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
# Модель на основі регресора AdaBoost
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor

regressor = AdaBoostRegressor(
    DecisionTreeRegressor(max_depth=4),
    n_estimators=400, random_state=7
)

regressor.fit(X_train, y_train)

# Обчислення показників ефективності регресора AdaBoost
y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

evs = explained_variance_score(y_test, y_pred)

print("\nADABOOST REGRESSOR")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
# Вилучення важливості ознак
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names
# Нормалізація значень важливості ознак
feature_importances = 100.0 * (feature_importances/max(feature_importances))
# Сорткування та перестановка значень
index_sorted = np.flipud(np.argsort(feature_importances))
# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

```
# Создаем датафрейм с важностью признаков
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Строим столбчатую диаграмму
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df, orient='h')
plt.xlabel('Relative Importance')
plt.title('Оцінка важливості признаков з використанням регресора AdaBoost')
plt.show()
```

```

AdaBoostRegressor
└─ estimator: DecisionTreeRegressor
   └─ DecisionTreeRegressor

# Обчислення показників ефективності регресора AdaBoost
y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

evs = explained_variance_score(y_test, y_pred)

print("\nADABOOST REGRESSOR")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))

```

ADABOOST REGRESSOR
Mean squared error = 1.18
Explained variance score = 0.47

Рис.17. Результат створення моделі та оцінка ефективності регресора.

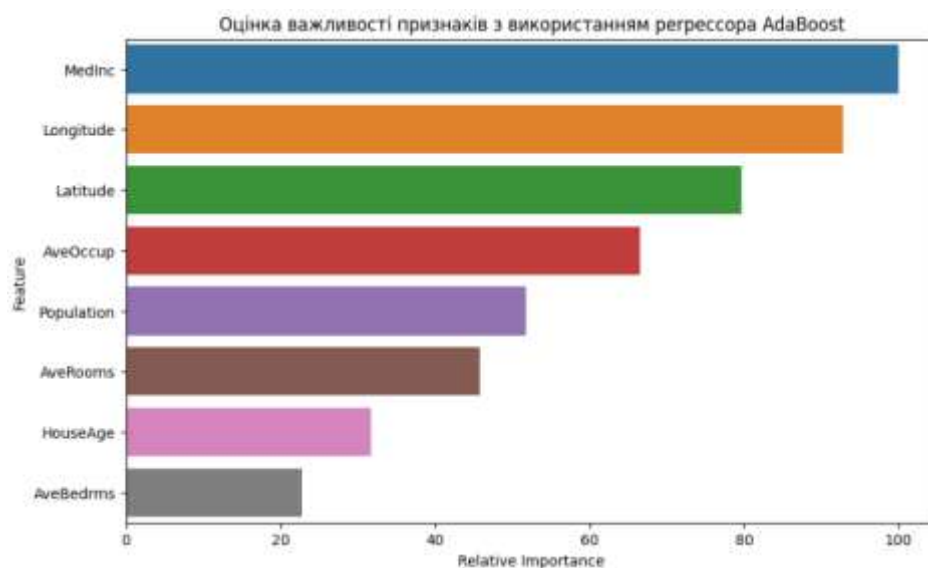


Рис.18. Столпчаста діаграма «Оцінка важливості признаков з використанням регресора AdaBoost».

Висновки: Відповідно до результатів діаграми **найважливішу** роль відіграє "MedInc" (середній дохід), а також значення "Longitude" (довгота), "Latitude" (широта) та "AveOccup" (середня кількість осіб) – усе це ознаки, що можуть бути важливими факторами для прийняття рішень.

"AveBedrms" (середня кількість спалень) має **найменший вплив** на модель, тому цією ознакою можна знехтувати.

Explained Variance Score дорівнює 0.47 - модель пояснює приблизно 47% варіації у цільовій змінній. Хоча це не ідеальний показник, він свідчить про те, що модель має певну спроможність пояснювати зміни в цінах на нерухомість.

MSE дорівнює 1.18, що залежно від задачі може вважатися завеликим показником, а в інших випадках свідчити про гарний прогноз моделі.

Завдання 5

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, mean_absolute_error
from sklearn import preprocessing
from sklearn.ensemble import ExtraTreesRegressor
#Завантажимо дані із файлу traffic_data.txt.
import numpy as np

input_file = 'traffic_data.txt'
data = []

with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line[:-1].split(',')
        data.append(items)

data = np.array(data)
# Перетворення рядкових даних на числові
from sklearn.preprocessing import LabelEncoder

label_encoder = []
X_encoded = np.empty(data.shape)

for i, item in enumerate(data[0]):
    if item.isdigit():
        X_encoded[:, i] = data[:, i]
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		


```

else:
    label_encoder.append(preprocessing.LabelEncoder())
    X_encoded[:, i] = label_encoder[-1].fit_transform(data[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)

# Розбиття даних на навчальний та тестовий набори
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)

# Регресор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)

# Обчислення характеристик ефективності регресора на тестових даних
y_pred = regressor.predict(X_test)
print("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))
from sklearn.preprocessing import LabelEncoder
import numpy as np

# Ваш список label_encoder уже созданий и заполнен

test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
test_datapoint_encoded = [-1] * len(test_datapoint)
count = 0

for i, item in enumerate(test_datapoint):
    if item.isdigit():
        test_datapoint_encoded[i] = int(item)
    else:
        test_datapoint_encoded[i] =
int(label_encoder[count].transform([item])[0])
        count = count + 1

test_datapoint_encoded = np.array(test_datapoint_encoded)

# Прогнозування результату для тестової точки даних
print("Predicted traffic:",
int(regressor.predict([test_datapoint_encoded])[0]))

```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

```
43] # Обчислення характеристик ефективності регресора на тестових даних
y_pred = regressor.predict(X_test)
print("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))

Mean absolute error: 7.42
```

Рис.19. Середня абсолютна похибка.

```
# Прогнозування результату для тестової точки даних
print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))

Predicted traffic: 26
```

Рис.20. Вихідний результат. (26 - вірно)

Завдання 6

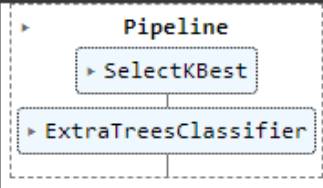
Лістинг програми:

```
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier
# Генерування даних
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=15, n_features=25, n_classes=3,
n_informative=6, n_redundant=0, random_state=7)
k_best_selector = SelectKBest(f_regression, k=9)
# Ініціалізація класифікатора на основі гранично випадкового лісу
classifier = ExtraTreesClassifier(n_estimators=60, max_depth=4)
# Створення конвеєра
processor_pipeline = Pipeline ([('selector',
k_best_selector), ('erf', classifier)])
# Встановлення параметрів
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)
# Навчання конвеєра
processor_pipeline.fit(X, y)
# Прогнозування результатів для вхідних даних
output = processor_pipeline.predict(X)
print("\nPredicted output\n", output)
# Виведення оцінки
print("\nScore:", processor_pipeline.score(X, y))
# Виведення ознак, відібраних селектором конвеєра
status = processor_pipeline.named_steps['selector'].get_support()
# Вилючення та виведення індексів обраних ознак
selected = [i for i, x in enumerate(status) if x]
print("\nIndices of selected features:", ', '.join(map(str, selected)))
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

```
[8] # Встановлення параметрів
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)
```



```
[9] # Навчання конвеєра
processor_pipeline.fit(X,y)
```

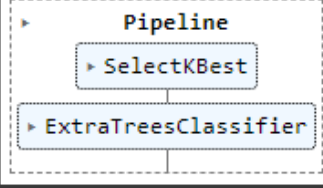


Рис.21. Результат навчання.

```
# Прогнозування результатів для вхідних даних
output = processor_pipeline.predict(X)
print("\nPredicted output\n", output)
```

Predicted output
[0 1 0 1 0 2 1 2 0 0 1 2 2 1 2]

Рис.22. Прогнозування.

```
[14] # Вилучення та виведення індексів обраних ознак
selected = [i for i, x in enumerate(status) if x]
print("\nIndices of selected features:", ', '.join(map(str, selected)))
```

Indices of selected features: 1, 2, 8, 10, 11, 14, 23

Рис.23. Обрані ознаки.

Висновки: Перший список містить прогнозовані класи для вхідних даних. Кожен елемент списку вказує на клас, до якого модель віднесла відповідний приклад. Значення **Score** вказує на точність моделі. У нашому випадку Score дорівнює 1.0, що означає - модель ідеально підходить для заданих даних.

В останньому рядку «Indices of selected features» містить індекси вибраних ознак селектором в пайплайні. Ці індекси вказують на те, які ознаки були визначені як найбільш важливі селектором SelectKBest.

Завдання 7

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors
# Вхідні дані
import numpy as np

X = np.array([
    [2.1, 1.3], [1.3, 3.2], [2.9, 2.5],
    [2.7, 5.4], [3.8, 0.9], [7.3, 2.1],
    [4.2, 6.5], [3.8, 3.7], [2.5, 4.1],
    [3.4, 1.9], [5.7, 3.5], [6.1, 4.3],
    [5.1, 2.2], [6.2, 1.1]
])

#кількість найближчих сусідів, котрі хочемо витягти.
k=5
# Тестова точка даних
test_datapoint = [4.3,2.7]
# Відображення вхідних даних на графіку
plt.figure()
plt.title('Вхідні дані')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color="black")
# Побудова моделі на основі методу k найближчих сусідів
from sklearn.neighbors import NearestNeighbors

knn_model = NearestNeighbors(n_neighbors=k, algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors(X)
# Виведемо 'k' найближчих сусідів
print("\nK Nearest Neighbors: ")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==> " + str(X[index]))
import matplotlib.pyplot as plt

plt.figure()
plt.title('Найближчі сусіди')

# Виведемо всі точки
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k', label='Навчальні точки')

# Виведемо найближчі сусіди для тестової точки
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		20

```
plt.scatter(X[indices][0][:, 0], X[indices][0][:, 1], marker='o', s=250, color='k', facecolors='none', label='Найближчі сусіди')

# Виведемо тестову точку
plt.scatter(test_datapoint, test_datapoint, marker='x', s=75, color='r', label='Тестова точка')

# Додамо легенду для пояснень
plt.legend()

plt.show()
```

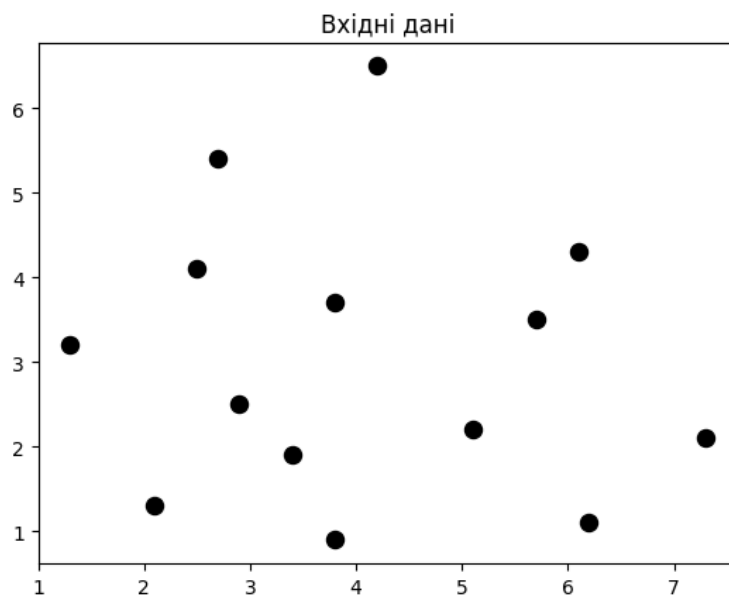


Рис.24. Вхідні дані.

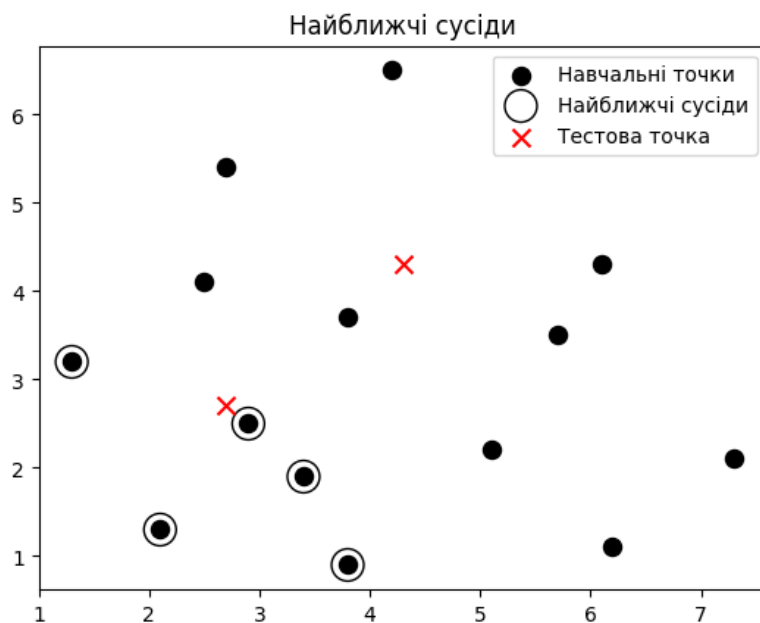


Рис.25. Результат візуалізації найближчих сусідів разом із тестовою точкою даних

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

```
K Nearest Neighbors:
1 ==> [2.1 1.3]
2 ==> [3.4 1.9]
3 ==> [2.9 2.5]
4 ==> [3.8 0.9]
5 ==> [1.3 3.2]
```

Рис.26. 'k' найближчих сусідів.

Висновки: на першому графіку ми можемо побачити вказані нами початкові дані у двовимірному просторі (на площині). На другому графіку видно результат роботи моделі на основі методу k найближчих сусідів. Показана тестова точка у виді (X) та її найближчі сусіди 5 штук. У вікні терміналу ми бачимо ці самі найближчі точки, але у числовому форматі, від найбільш схожої – до найменш схожої.

Завдання 8

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets
# Завантаження вхідних даних
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(int)
# Відображення вхідних даних на графіку
plt.figure()
plt.title('Вхідні дані')
marker_shapes = 'v^os'
mapper = [marker_shapes[i] for i in y]

for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i], s=75, edgecolors='black',
                facecolors='none')

plt.show()
# Кількість найближчих сусідів
num_neighbors = 12
# крок сітки
step_size = 0.01
# Створення класифікатора на основі методу k найближчих сусідів
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=num_neighbors,
                                weights='distance')
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

```

# Навчання моделі на основі методу k найближчих сусідів
classifier.fit(X,y)
# Створення сітки для відображення меж на графіку
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x_values, y_values = np.meshgrid(np.arange(x_min, x_max, step_size),
np.arange(y_min, y_max, step_size))
# Виконання класифікатора на всіх точках сітки
output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])
# Візуалізація передбачуваного результату
output = output.reshape (x_values.shape)
plt.figure()
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)

# Накладання навчальних точок на карту
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i], s=50, edgecolors='black',
facecolors='none')

#границні значення для осей X та Y та вкажемо заголовок
plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())

plt.title('Границі можелі класифікатора на основі найближчих к сусідів')
# Тестування вхідної точки даних
test_datapoint = [5.1, 3.6]

plt.figure()
plt.title('Тестова точка даних')

for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker='o', s=75, edgecolors='black', face-
colors="none")

plt.scatter(test_datapoint[0], test_datapoint[1], marker='x', linewidth=6,
s=200, color='black')

plt.show()
# Вилучення K найближчих сусідів
_, indices = classifier.kneighbors([test_datapoint] )
indices = indices.astype(int) [0]

```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

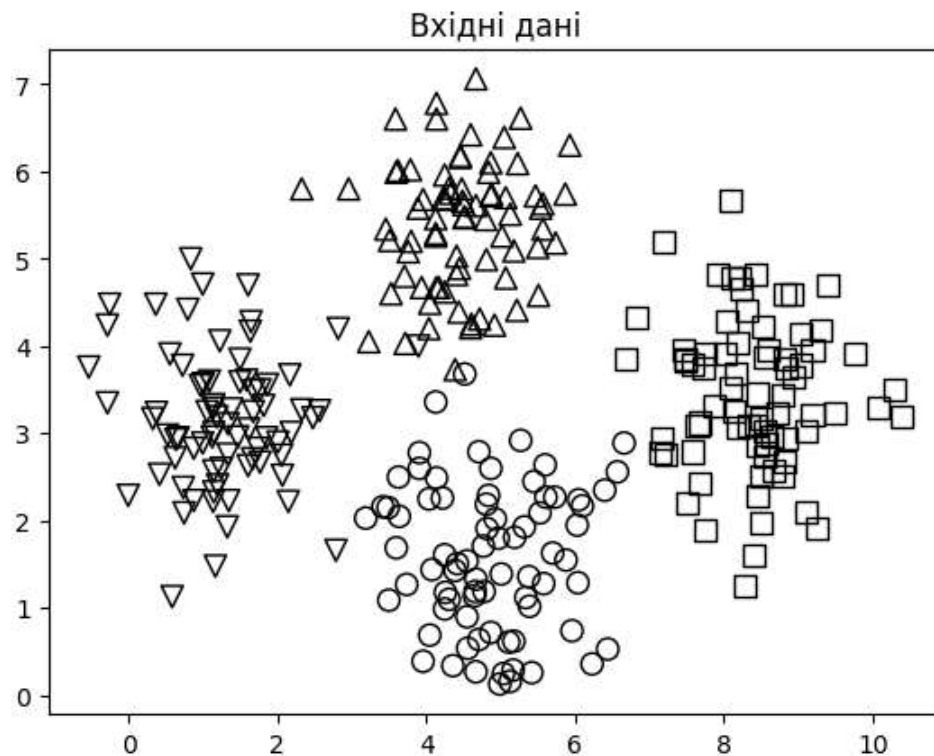


Рис.27. Вхідні дані.

Границі можелі класифікатора на основі найближчих к сусідів

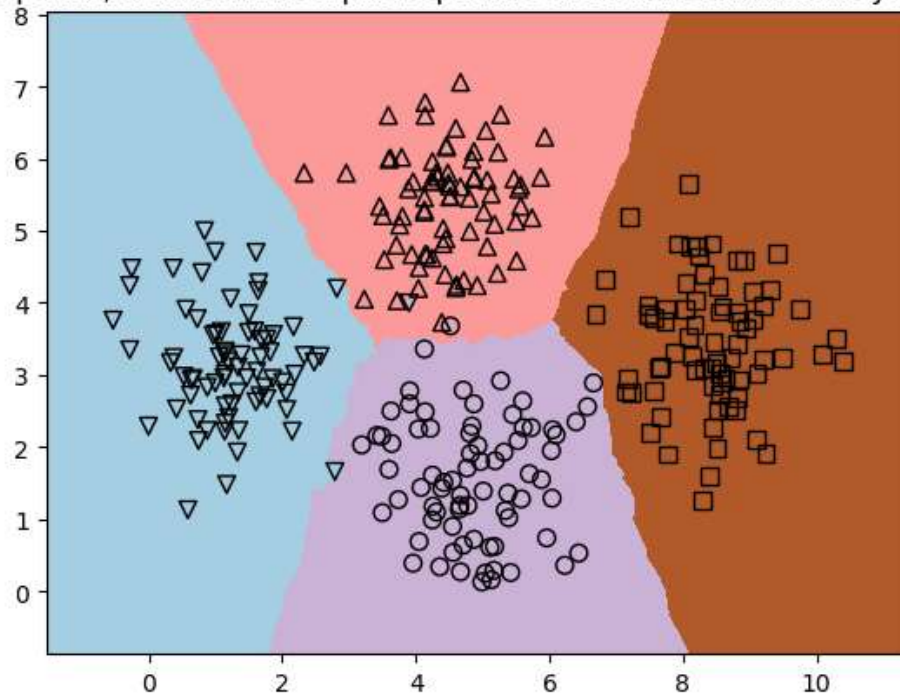


Рис.28. Візуалізація виконання класифікатора на всіх точках сітки.

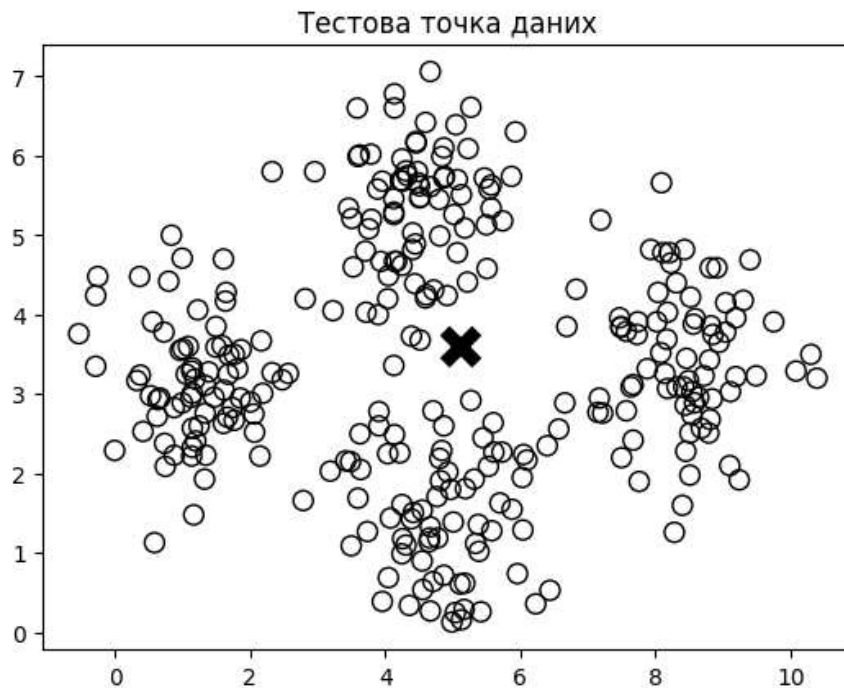


Рис.29. Тестування вхідної точки даних. Відображення навчальних точок даних разом із тестовою.

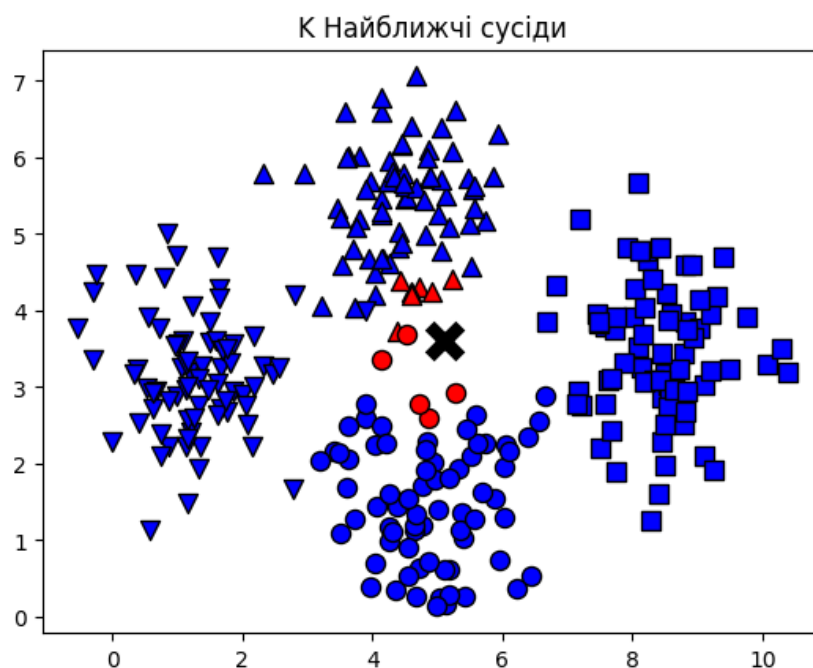


Рис.30. Відображення K найближчих сусідів тестової точки.

```
# Виведемо прогнозований результат
predicted_output = classifier.predict([test_datapoint])[0]
print("Predicted output:", predicted_output)
```

Predicted output: 1

Рис.31. Прогнозований результат.

Висновки: У ході дослідження було використано метод k-найближчих сусідів (KNN) для класифікації тестової точки даних. Результат класифікації показав, що вона була віднесена до класу з міткою "1". Також, чотири різні класи даних на графіках досить добре розподілені між собою, тому можливо відносно точно класифікувати тестові точки на основі їхніх найближчих сусідів.

Завдання 9

Лістинг програми:

```
import argparse
import json
import numpy as np
#парсер для обробки вхідних аргументів
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Compute similarity score')
    parser.add_argument('--user1', dest='user1', required=True, help='First user')
    parser.add_argument('--user2', dest='user2', required=True, help='Second user')
    parser.add_argument("--score-type", dest="score_type", required=True, choices=['Euclidean', 'Pearson'], help="Similarity metric to be used")
    return parser

def find_common_ratings(dataset, user1, user2):
    common_ratings = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_ratings[item] = 1
    return common_ratings

def euclidean_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError("Cannot find " + user1 + " in the dataset")
    if user2 not in dataset:
        raise TypeError("Cannot find " + user2 + " in the dataset")

    common_ratings = find_common_ratings(dataset, user1, user2)

    if len(common_ratings) == 0:
        return 0

    squared_diff = []
    for item in common_ratings:
        squared_diff.append(np.square(dataset[user1][item] - dataset[user2][item]))
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

```

similarity_score = 1 / (1 + np.sqrt(np.sum(squared_diff)))
return similarity_score

def pearson_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')
    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')

    common_ratings = find_common_ratings(dataset, user1, user2)

    if len(common_ratings) == 0:
        return 0

    sum1 = sum(dataset[user1][item] for item in common_ratings)
    sum2 = sum(dataset[user2][item] for item in common_ratings)

    sum1_squared = sum(pow(dataset[user1][item], 2) for item in com-
mon_ratings)
    sum2_squared = sum(pow(dataset[user2][item], 2) for item in com-
mon_ratings)

    product_sum = sum(dataset[user1][item] * dataset[user2][item] for item in
common_ratings)
    num = product_sum - (sum1 * sum2 / len(common_ratings))
    den = np.sqrt((sum1_squared - pow(sum1, 2) / len(common_ratings)) *
(sum2_squared - pow(sum2, 2) / len(common_ratings)))

    if den == 0:
        return 0

    similarity_score = num / den
    return similarity_score

if __name__ == '__main__':
    user1 = "David Smith"
    user2 = "Bill Duffy"
    score_type = "Euclidean"
    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    if score_type == 'Euclidean':
        print("\nEuclidean score:")
        print(euclidean_score(data, user1, user2))
    else:
        print("\nPearson score:")
        print(pearson_score(data, user1, user2))

```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

1) David Smith та Bill Duffy

Евклідова оцінка подібності:

Euclidean score:
0.585786437626905

Оцінка подібності Пірсона:

Pearson score:
0.9909924304103233

2) David Smith та Brenda Peterson

Евклідова оцінка подібності:

Euclidean score:
0.1424339656566283

Оцінка подібності Пірсона:

Pearson score:
-0.7236759610155113

3) David Smith та Samuel Miller

Евклідова оцінка подібності:

Euclidean score:
0.30383243470068705

Оцінка подібності Пірсона:

Pearson score:
0.7587869106393281

4) David Smith та Julie Hammel

Евклідова оцінка подібності:

Euclidean score:
0.2857142857142857

Оцінка подібності Пірсона:

Pearson score:
0

5) David Smith та Clarissa Jackson

Евклідова оцінка подібності:

Euclidean score:
0.28989794855663564

Оцінка подібності Пірсона:

Pearson score:
0.6944217062199275

6) David Smith та Adam Cohen

Евклідова оцінка подібності:

Euclidean score:
0.38742588672279304

Оцінка подібності Пірсона:

Pearson score:
0.9081082718950217

7) David Smith та Chris Duncan

Евклідова оцінка подібності:

Euclidean score:
0.38742588672279304

Оцінка подібності Пірсона:

Pearson score:
1.0

Висновки: У порівнянні обох метрик, оцінки подібності за метрикою "Пірсона" зазвичай вищі, ніж за метрикою "Евклідова". Це свідчить про те, що метрика "Пірсона" враховує кореляцію між оцінками, в той час як метрика "Евклідова" враховує відстані між оцінками без урахування кореляції. Найвища оцінка подібності за двома метриками між **David Smith та Bill Duffy** (0.58, 0.99), в той час коли найнижча між **David Smith та Brenda Peterson** (0.14, -0.7).

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдання 10

Лістинг програми:

```
import argparse
import json
import numpy as np

from scipy.stats import pearsonr

def pearson_score(user1_ratings, user2_ratings):
    # фільми, які оцінили обидва користувачі
    common_movies = [movie for movie in user1_ratings if movie in user2_ratings]

    if not common_movies:
        return 0 # Повертаємо 0, якщо немає спільних оцінок

    # середні оцінки кожного користувача для спільних фільмів
    user1_mean = np.mean([user1_ratings[movie] for movie in common_movies])
    user2_mean = np.mean([user2_ratings[movie] for movie in common_movies])

    # чисельник і знаменник для формули Пірсона
    numerator = sum((user1_ratings[movie] - user1_mean) * (user2_ratings[movie] - user2_mean) for movie in common_movies)
    denominator_user1 = np.sqrt(sum((user1_ratings[movie] - user1_mean)**2 for movie in common_movies))
    denominator_user2 = np.sqrt(sum((user2_ratings[movie] - user2_mean)**2 for movie in common_movies))

    if denominator_user1 == 0 or denominator_user2 == 0:
        return 0 # Повертаємо 0, якщо один з користувачів не має дисперсії

    # коефіцієнт кореляції Пірсона
    pearson = numerator / (denominator_user1 * denominator_user2)

    return pearson

# Визначимо функцію для парсингу вхідних аргументів.
def build_arg_parser():
    parser = argparse.ArgumentParser(description="Find users who are similar to the input user")
    parser.add_argument('--user', dest='user', required=True, help='Input user')
    return parser

# Визначимо функцію, яка знаходитиме в наборі даних користувачів, аналогічних зазначеному.
def find_similar_users(dataset, input_user, num_users):
    if input_user not in dataset:
        raise TypeError('Cannot find ' + input_user + ' in the dataset')

    # Оцінюємо подібність користувача до інших користувачів та зберігаємо результати
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    scores = np.array([[x, pearson_score(dataset[input_user], dataset[x])] for
x in dataset if x != input_user])

    # Сортуювання оцінок за спаданням
    scores_sorted = scores[scores[:, 1].argsort()[::-1]]

    # Вилучення оцінок перших 'num_users' користувачів
    top_users = scores_sorted[:num_users]

    return top_users
# Визначимо основну функцію
if __name__ == '__main__':
    user = "Bill Duffy"
    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    print('\nUsers similar to ' + user + ':\n')
    similar_users = find_similar_users(data, user, 3)
    print('User\t\t\tSimilarity score')
    print('-' * 41)
    for item in similar_users:
        print(item[0], '\t\t\t', round(float(item[1]), 2))

```

```

Users similar to Bill Duffy:

User                                Similarity score
-----
David Smith                        0.99
Samuel Miller                      0.88
Adam Cohen                        0.86

```

Рис.32. Результат пошуку користувачів зі схожими уподобаннями до Bill Duffy методом колаборативної фільтрації.

```

Users similar to Clarissa Jackson:

User                                Similarity score
-----
Chris Duncan                       1.0
Bill Duffy                         0.83
Samuel Miller                      0.73

```

Рис.33. Результат пошуку користувачів зі схожими уподобаннями до Clarissa Jackson методом колаборативної фільтрації.

Висновки: виконавши дане завдання ми написали програму, що може знайти користувачів зі схожими смаками, шляхом розрахунку оцінки за Пірсоном. Усі отримані оцінки не сягнули нижче 0.7, що є досить високим коефіцієнтом.

Завдання 11

Лістинг програми:

```
import argparse
import json
import numpy as np
# Визначимо функцію для парсингу вхідних аргументів.
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Find movie recommendations
for the given user')
    parser.add_argument('--user', dest='user', required=True, help='Input us-
er')
    return parser
# Отримати рекомендації щодо фільмів
# для вказаного користувача
def get_recommendations(dataset, input_user):
    if input_user not in dataset:
        raise TypeError('Cannot find ' + input_user + ' in the dataset')

# Визначимо змінні для відстеження оцінок.
    overall_scores = {}
    similarity_scores = {}
#Обчислимо оцінку подібності між вказаним користувачем та всіма
#іншими користувачами у наборі даних.
    for user in [x for x in dataset if x != input_user]:
        similarity_score = pearson_score(dataset[input_user], dataset[user])
        # Якщо оцінка подібності менша за 0, переходимо до наступного
        if similarity_score <= 0:
            continue
#Відслідкуємо зважену рейтингову оцінку для кожного елемента
#відфільтрованого списку, виходячи з оцінок подібності.
        for item in dataset[user]:
            if item not in dataset[input_user] or dataset[input_user][item] ==
0:
                overall_scores[item] = overall_scores.get(item, 0) + da-
taset[user][item] * similarity_score
                similarity_scores[item] = similarity_scores.get(item, 0) +
similarity_score
#У разі відсутності відповідних фільмів, ми не можемо надати жодних
рекомендацій.

    if len(overall_scores) == 0:
        return ['No recommendations possible']
# Генерація рейтингів фільмів за допомогою їх нормалізації
```

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		


```

    movie_scores = np.array([[score / similarity_scores[item], item] for item,
score in overall_scores.items()])
# Сортування за спаданням
    movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]
# Вилучення рекомендацій фільмів
    movie_recommendations = [movie for _, movie in movie_scores]
    return movie_recommendations

# Визначимо основну функцію
if __name__ == '__main__':
    user = "Julie Hammel"
    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    print("Data loaded successfully.")
    print(data.get(user, "User not found in the dataset"))

    movies = get_recommendations(data, user)

    if movies == ['No recommendations possible']:
        print("No movie recommendations available.")
    else:
        print("\nMovie recommendations for " + user + ":")
        for i, movie in enumerate(movies):
            print(str(i + 1) + '. ' + movie)

```

```

Data loaded successfully.
{'Scarface': 2.5, 'Roman Holiday': 4.5, 'Goodfellas': 3.0}

Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull

```

Рис.34. Результат роботи програми.

```

Data loaded successfully.
{'The Apartment': 1.5, 'Raging Bull': 4.5}

Movie recommendations for Chris Duncan:
1. Vertigo
2. Scarface
3. Goodfellas
4. Roman Holiday

```

Рис.35. Результат роботи програми.

Висновки: Програма успішно завантажує дані з файлу та визначає користувача для отримання рекомендацій.

		Дяченко В.В.			ДУ «Житомирська політехніка».22.122.4.000 – Лр4	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		