Лабораторна робота №5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

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

Завдання на лабораторну роботу

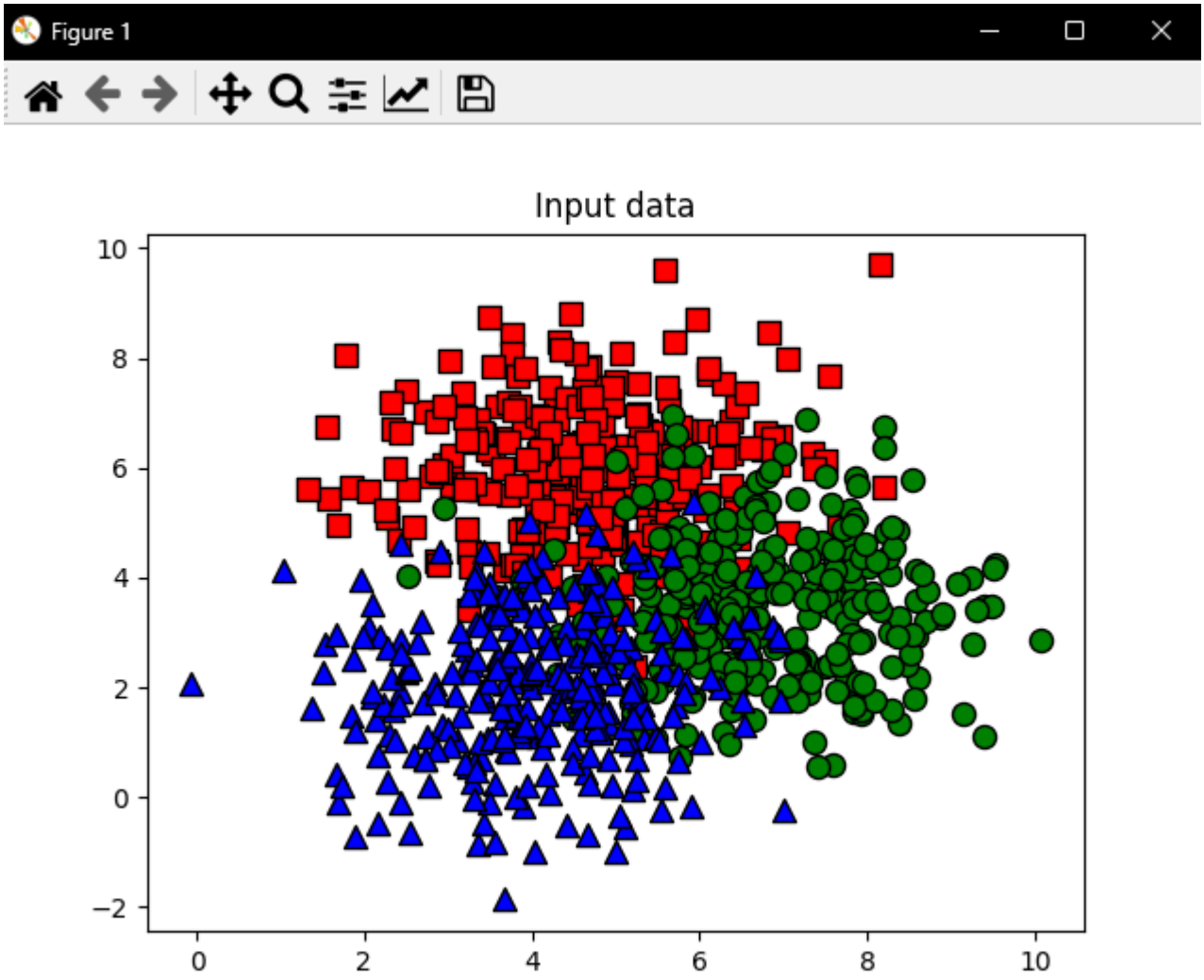
Завдання 2.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів

Лістинг програми

```
import argparse
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from utilities import visualize_classifier
from sklearn.model_selection import cross_val_score, train_test_split

def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify data using En-
semble Learning techniques')
    parser.add_argument("--classifier-type", dest="classifier_type", re-
quired=True, choices=['rf', 'erf'],
                        help="Type of classifier to use; can be either 'rf'
or 'erf'")
    return parser
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type
    input_file = 'data_random_forests.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, Y = data[:, :-1], data[:, -1]
    print(X)
    class_0 = np.array(X[Y == 0])
```

```
    class_1 = np.array(X[Y == 1])
    class_2 = np.array(X[Y == 2])
    plt.figure()
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='red',
edgecolors='black', linewidth=1, marker='s')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='green',
edgecolors='black', linewidth=1, marker='o')
    plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='blue',
edgecolors='black', linewidth=1, marker='^')
plt.title('Input data')
plt.show()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25,
random_state=5)
params = {'n_estimators': 100, 'max_depth': 4, '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, 'Training dataset')
class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
Y_train_pred = classifier.predict(X_train)
print(classification_report(Y_train, Y_train_pred, tar-
get_names=class_names))
print("#" * 40 + "\n")
print("#" * 40)
print("\nClassifier performance on test dataset\n")
Y_test_pred = classifier.predict(X_test)
print(classification_report(Y_test, Y_test_pred, target_names=class_names))
print("#" * 40 + "\n")
```

Результат виконання



Завдання 2.2. Обробка дисбалансу класів

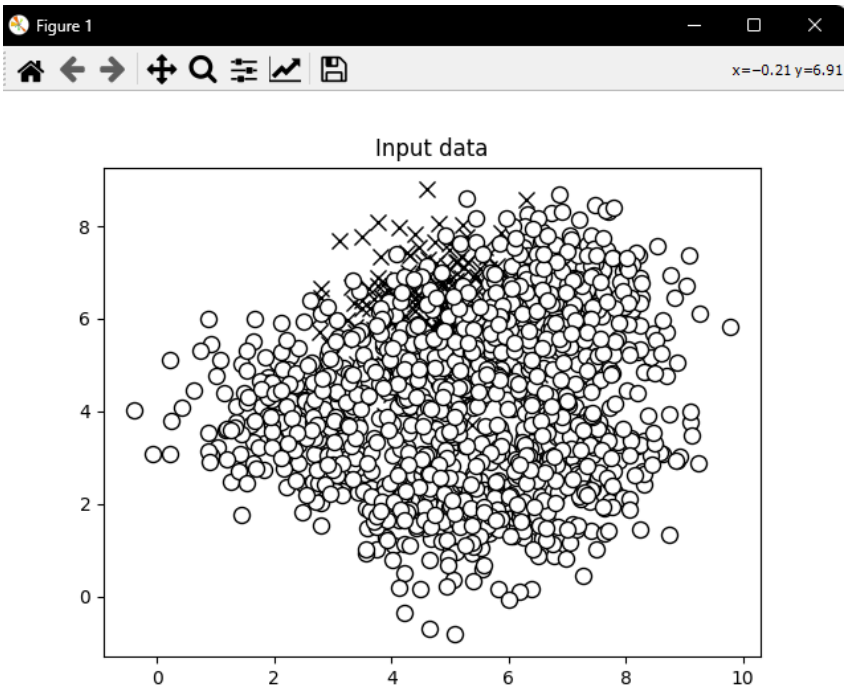
Лістинг програми

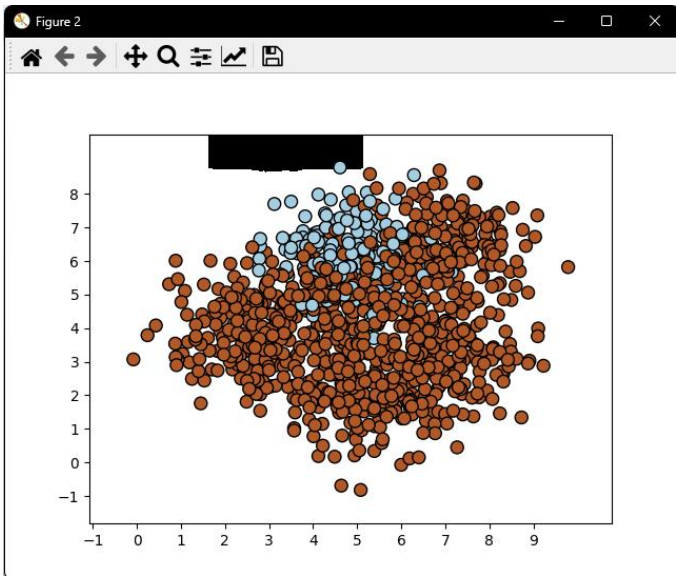
```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report
from utilities import visualize_classifier

if __name__ == '__main__':
    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
    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('Input data')
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.25, random_state=5)
    params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}

    if len(sys.argv) > 1:
        if sys.argv[1] == 'balance':
            params['class_weight'] = 'balanced'
        else:
            raise TypeError("Invalid input argument; should be 'balance' or
nothing")
    classifier = ExtraTreesClassifier(**params)
    classifier.fit(X_train, Y_train)
    visualize_classifier(classifier, X_train, Y_train)
    Y_test_pred = classifier.predict(X_test)
    class_names = ['Class-0', 'Class-1']
    print("\n" + "#"*40)
    print("Classifier performance on training dataset")
    print(classification_report(Y_test, Y_test_pred, tar-
get_names=class_names))
    print("#"*40)
    print("Classifier performance on test dataset")
    print(classification_report(Y_test, Y_test_pred, tar-
get_names=class_names))
    print("#"*40 + "\n")
    plt.show()
```

Результат виконання

```
              precision    recall  f1-score   support

     Class-0       0.00      0.00      0.00        69
     Class-1       0.82      1.00      0.90       306

    accuracy                           0.82       375
   macro avg       0.41      0.50      0.45       375
weighted avg       0.67      0.82      0.73       375

#############################################

PS E:\labs\WIKI\lab5>
```



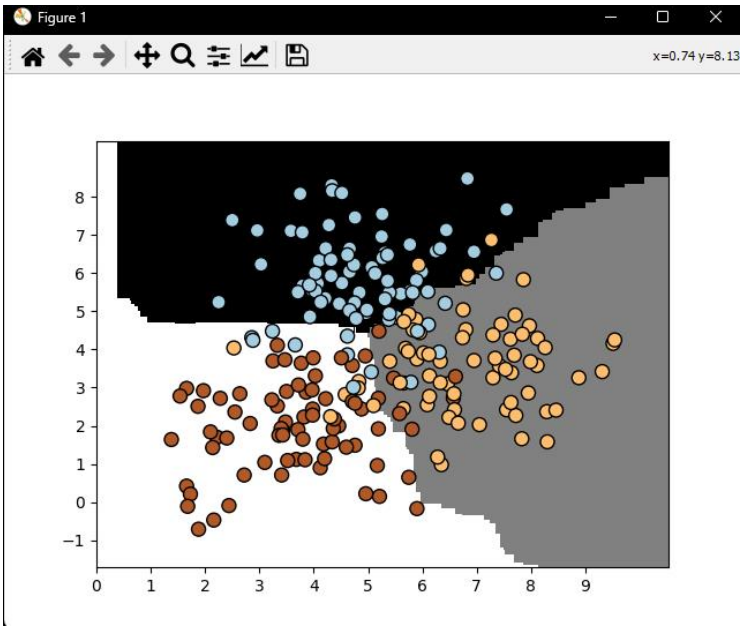


Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Лістинг програми

```
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report
from utilities import visualize_classifier

input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, Y = data[:, :-1], data[:, -1]
class_0 = np.array(X[Y == 0])
class_1 = np.array(X[Y == 1])
class_2 = np.array(X[Y == 2])
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25,
random_state=5)
parameter_grid = [{'n_estimators': [100], 'max_depth': [2, 4, 7, 12, 16]},
                  {'max_depth': [4], 'n_estimators': [25, 50, 100, 250]}]
metrics = ['precision_weighted', 'recall_weighted']
for metric in metrics:
    print("#### Searching optimal parameters for", metric)
    classifier = GridSearchCV(ExtraTreesClassifier(random_state=0), parame-
ter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, Y_train)
    print("\nScores across the parameter grid:")
    for params, avg_score in classifier.cv_results_.items():
        print(params, '-->', avg_score)
    print("\nHighest scoring parameter set:", classifier.best_params_)
    Y_test_pred = classifier.predict(X_test)
    class_names = ['Class-0', 'Class-1', 'Class-2']
    print("#"*40)
    print("Classifier performance on training dataset")
    print(classification_report(Y_test, Y_test_pred, tar-
get_names=class_names))
    print("#"*40 + "\n")
    visualize_classifier(classifier, X_test, Y_test)
```

Результат виконання




```
#### Searching optimal parameters for precision_weighted

Scores across the parameter grid:
mean_fit_time --> [0.1740413 0.17519169 0.17724514 0.18294024 0.18784242 0.0460608
 0.09085336 0.1753593  0.43098412]
std_fit_time --> [0.00151285 0.00094305 0.00107238 0.00071455 0.00251826 0.00055027
 0.00131814 0.00195356 0.00184127]
mean_score_time --> [0.01391373 0.01352015 0.01381817 0.01444988 0.01491914 0.00731082
 0.00921469 0.01351194 0.02624383]
std_score_time --> [5.82672699e-04 3.16949201e-04 5.04759275e-04 4.05305958e-04
 2.05942960e-04 4.09997158e-04 4.00342619e-04 6.80124257e-06
 4.10701432e-04]
param_max_depth --> [2 4 7 12 16 4 4 4 4]
param_n_estimators --> [100 100 100 100 100 25 50 100 250]
params --> [{'max_depth': 2, 'n_estimators': 100}, {'max_depth': 4, 'n_estimators': 100}, {'max_depth': 7, 'n_estimators': 100}, {'max_depth': 12, 'n_estimators': 100}, {'max_depth': 16, 'n_estimators': 100}, {'max_depth': 4, 'n_estimators': 25}, {'max_depth': 4, 'n_estimators': 50}, {'max_depth': 4, 'n_estimators': 100}, {'max_depth': 4, 'n_estimators': 250}]
split0_test_score --> [0.87460317 0.85323424 0.85381333 0.81554507 0.80837449 0.87558579
 0.84512618 0.85323424 0.86067019]
split1_test_score --> [0.87694315 0.86737731 0.87662655 0.87651671 0.85190389 0.85594956
 0.85035187 0.86737731 0.87887866]
split2_test_score --> [0.81956872 0.82834119 0.82611079 0.81030267 0.77569734 0.834651
 0.83704535 0.82834119 0.82834119]
split3_test_score --> [0.82078374 0.80260075 0.80192593 0.80351421 0.7942149  0.7931046
 0.80260075 0.80260075 0.80192593]
split4_test_score --> [0.8568842  0.85412387 0.86062409 0.85389639 0.86023157 0.86857693
 0.86332922 0.85412387 0.85412387]
mean_test_score --> [0.8497566  0.84113547 0.84382014 0.83195501 0.81648444 0.84557357
 0.83985067 0.84113547 0.84478797]
std_test_score --> [0.02513165 0.02303185 0.02656029 0.02833428 0.03342873 0.02969796
 0.02040062 0.02303185 0.0260872 ]
rank_test_score --> [1 5 4 8 9 2 7 5 3]

Highest scoring parameter set: {'max_depth': 2, 'n_estimators': 100}
#####################################
Classifier performance on training dataset
              precision    recall  f1-score   support

     Class-0       0.94      0.81      0.87        79
     Class-1       0.81      0.86      0.83        70
     Class-2       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

#####################################
```

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

Scores across the parameter grid:
mean_fit_time --> [0.1742377 0.17374768 0.17563477 0.18124676 0.18491387 0.04645476
 0.09118355 0.1744308  0.43034592]
std_fit_time --> [0.00050346 0.00064953 0.00053573 0.00063935 0.00053129 0.00097097
 0.0042179  0.00134347 0.00106242]
mean_score_time --> [0.0133471  0.01342573 0.01391935 0.01463642 0.0150826  0.00701056
 0.00911579 0.01402206 0.02599821]
std_score_time --> [0.00020679 0.00037287 0.00019823 0.00037524 0.00061698 0.00044763
 0.00048367 0.00063592 0.00049553]
param_max_depth --> [2 4 7 12 16 4 4 4 4]
param_n_estimators --> [100 100 100 100 100 25 50 100 250]
params --> [{'max_depth': 2, 'n_estimators': 100}, {'max_depth': 4, 'n_estimators': 100}, {'max_depth': 7, 'n_estimators': 100}, {'max_depth': 12, 'n_estimators': 100}, {'max_depth': 16, 'n_estimators': 100}, {'max_depth': 4, 'n_estimators': 25}, {'max_depth': 4, 'n_estimators': 50}, {'max_depth': 4, 'n_estimators': 100}, {'max_depth': 4, 'n_estimators': 250}]
split0_test_score --> [0.87407407 0.85185185 0.85185185 0.81481481 0.8        0.87407407
 0.84444444 0.85185185 0.85925926]
split1_test_score --> [0.86666667 0.85925926 0.87407407 0.87407407 0.85185185 0.85185185
 0.84444444 0.85925926 0.87407407]
split2_test_score --> [0.80740741 0.82222222 0.82222222 0.80740741 0.77037037 0.82962963
 0.82962963 0.82222222 0.82222222]
split3_test_score --> [0.81481481 0.8        0.8        0.8        0.79259259 0.79259259
 0.8        0.8        0.8        ]
split4_test_score --> [0.85185185 0.85185185 0.85925926 0.85185185 0.85925926 0.86666667
 0.85925926 0.85185185 0.85185185]
mean_test_score --> [0.84296296 0.83703704 0.84148148 0.82962963 0.81481481 0.84296296
std_test_score --> [0.02707586 0.02246778 0.02674884 0.02849687 0.03474382 0.02940657
 0.02089579 0.02246778 0.02674884]
rank_test_score --> [1 5 3 8 9 1 7 5 3]

Highest scoring parameter set: {'max_depth': 2, 'n_estimators': 100}
#####################################
Classifier performance on training dataset
              precision    recall  f1-score   support

     Class-0       0.94      0.81      0.87        79
     Class-1       0.81      0.86      0.83        70
     Class-2       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

#####################################
```

Завдання 2.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів

Лістинг програми

```
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor
from sklearn import preprocessing

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)
label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
    if item.isdigit():
        X_encoded[:, i] = data[:, i]
    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)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25,
random_state=5)
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, Y_train)
Y_pred = regressor.predict(X_test)
print("Mean absolute error =", round(mean_absolute_error(Y_test, Y_pred),
2))
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(test_datapoint[i])
    else:
        test_datapoint_encoded[i] = int(label_encoder[count].trans-
form([test_datapoint[i]]))
        count = count + 1
test_datapoint_encoded = np.array(test_datapoint_encoded)
print("Predicted traffic:", int(regressor.predict([test_datapoint_en-
coded])[0]))
```

Завдання 2.6. Створення навчального конвеєра (конвеєра машинного навчання)

Лістинг програми

```
from sklearn.datasets import _samples_generator
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier
X, Y = _samples_generator.make_classification(n_samples=150, n_features=25,
n_classes=3, n_informative=6, n_redundant=0, random_state=7)
k_best_selector = SelectKBest(f_regression, k=10)
classifier = ExtraTreesClassifier(n_estimators=60, max_depth=4)
processor_pipeline = Pipeline([('selector', k_best_selector), ('erf', clas-
sifier)])
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)
processor_pipeline.fit(X, Y)
print("Predicted output:", processor_pipeline.predict(X))
print("Score:", processor_pipeline.score(X, Y))
status = processor_pipeline.named_steps['selector'].get_support()
selected = [i for i, x in enumerate(status) if x]
print("Selected features:", selected)
```

Результат виконання

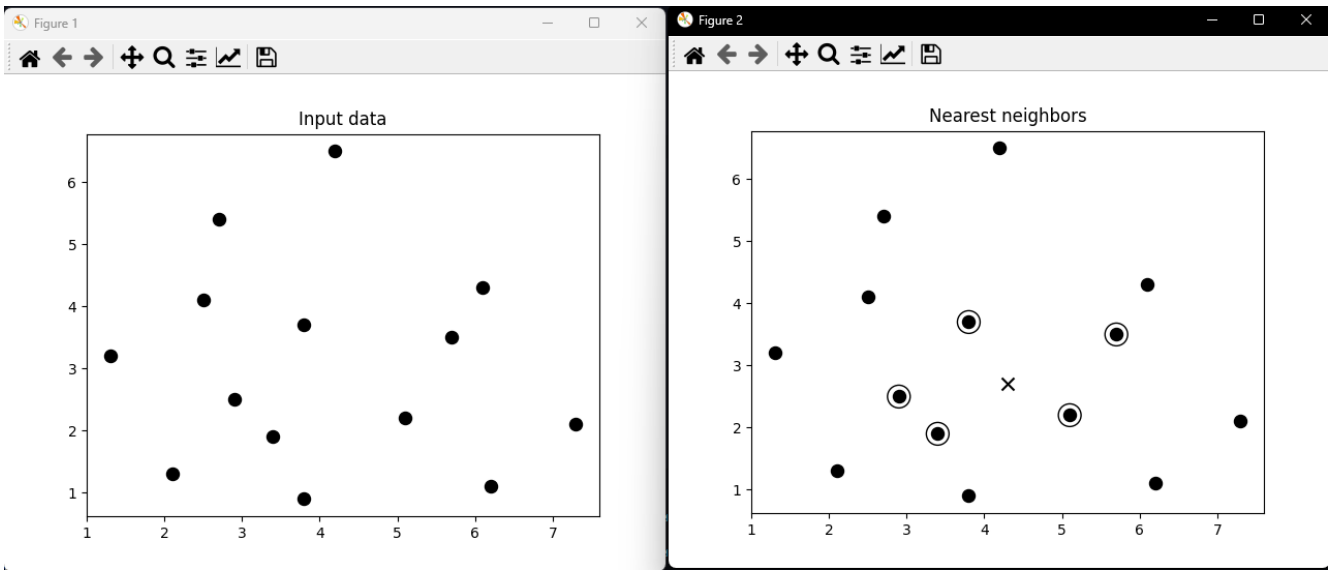
```
Predicted output: [1 2 2 0 2 0 2 1 0 1 1 2 1 0 2 2 1 0 0 1 0 2 1 1 2 2 0 0 1 2 1 0 1 0 2 2 1
 1 2 2 2 0 1 2 2 1 2 2 1 0 1 2 2 2 2 0 2 2 0 2 2 0 1 0 2 1 1 1 1 2 0 1 0 2
 0 0 1 2 2 0 0 1 2 2 2 0 0 0 2 2 2 1 2 0 2 1 2 2 0 0 1 1 1 1 2 2 2 2 0 1 1
 0 2 1 1 0 1 1 1 1 0 0 0 1 2 1 0 0 2 1 2 0 0 1 0 1 1 0 1 1 1 1 2 2 1 1 2 0
 2 2]
Score: 0.9
Selected features: [4, 7, 8, 12, 14, 17, 22]
PS E:\labs\WIKI\lab5>
```

Завдання 2.7. Пошук найближчих сусідів

Лістинг програми

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors
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]])
k = 5
test_datapoint = [4.3, 2.7]
plt.figure()
plt.title('Input data')
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')
knn_model = NearestNeighbors(n_neighbors=k, algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors([test_datapoint])
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])
plt.figure()
plt.title('Nearest neighbors')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1],
        marker='o', s=250, color='k', facecolors='none')
plt.scatter(test_datapoint[0], test_datapoint[1],
        marker='x', s=75, color='k')
plt.show()
```

Результат виконання

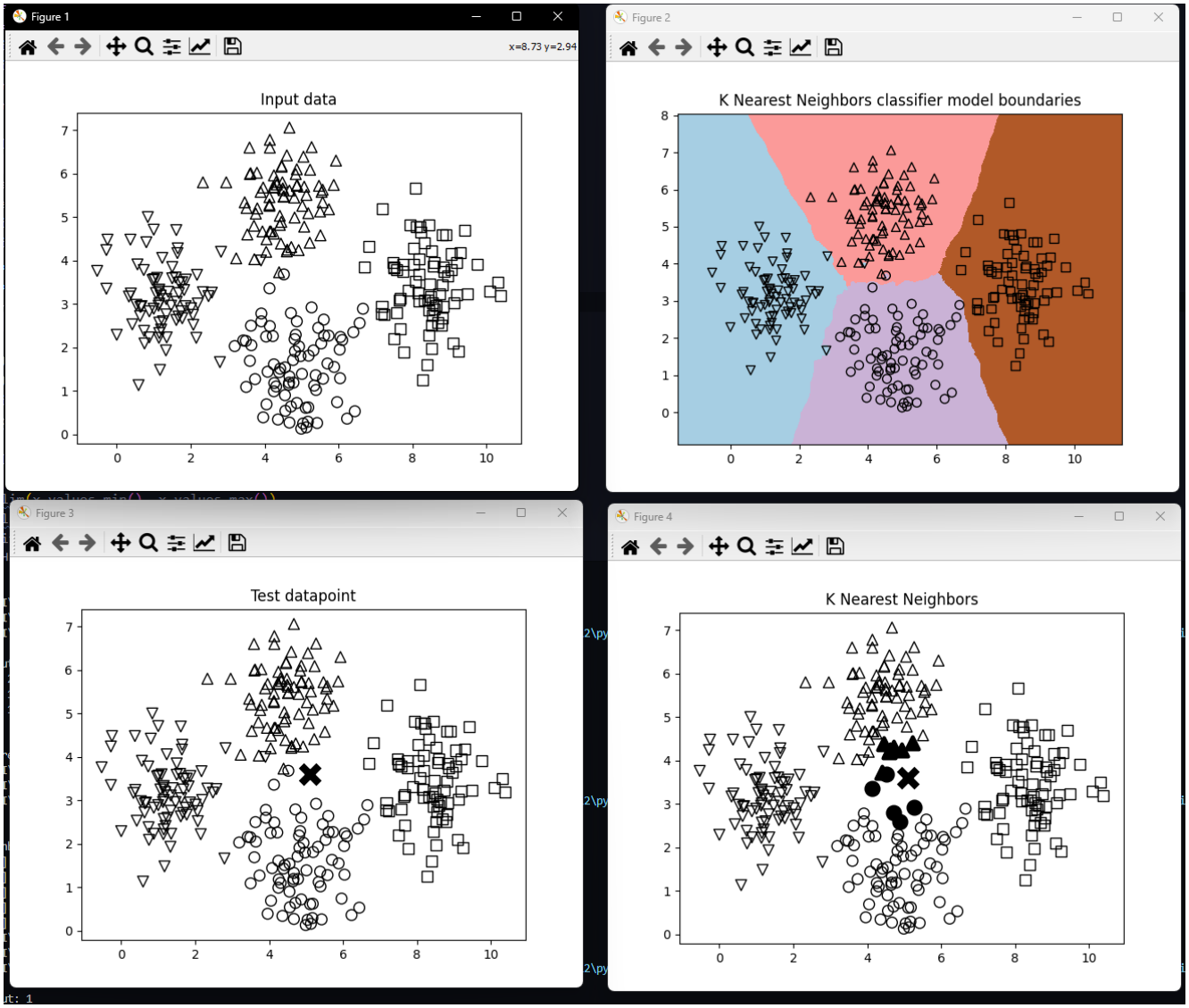


Завдання 2.8. Створити класифікатор методом k найближчих сусідів

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors

input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(int)
plt.figure()
plt.title('Input data')
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')
num_neighbors = 12
step_size = 0.01
classifier = neighbors.KNeighborsClassifier(num_neighbors, weights='dis-
tance')
classifier.fit(X, y)
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))

output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])
output = output.reshape(x_values.shape)
plt.figure()
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=50, edgecolors='black', facecolors='none')
plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())
plt.title('K Nearest Neighbors classifier model boundaries')
test_datapoint = [5.1, 3.6]
plt.figure()
plt.title('Test datapoint')
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.scatter(test_datapoint[0], test_datapoint[1], marker='x',
        linewidth=6, s=200, facecolors='black')
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(int)[0]
plt.figure()
plt.title('K Nearest Neighbors')
for i in indices:
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
            linewidth=3, s=100, facecolors='black')
plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
        linewidth=6, s=200, facecolors='black')
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')
print("Predicted output:", classifier.predict([test_datapoint])[0])
plt.show()
```

Результат виконання



Завдання 2.9. Обчислення оцінок подібності

Літсинг програми

```
import argparse
import json
import numpy as np

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 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_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1
    if len(common_movies) == 0:
        return 0
    squared_diff = []
    for item in dataset[user1]:
        if item in dataset[user2]:
            squared_diff.append(np.square(dataset[user1][item] - da-
taset[user2][item]))
    return 1 / (1 + np.sqrt(np.sum(squared_diff)))
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_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1
    num_ratings = len(common_movies)
    if num_ratings == 0:
        return 0
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    user1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in
common_movies])
```

```
    user2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in
common_movies])
    sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item]
for item in common_movies])
    Sxy = sum_of_products - (user1_sum * user2_sum / num_ratings)
    Sxx = user1_squared_sum - np.square(user1_sum) / num_ratings
    Syy = user2_squared_sum - np.square(user2_sum) / num_ratings
    if Sxx * Syy == 0:
        return 0
    return Sxy / np.sqrt(Sxx * Syy)
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    user1 = args.user1
    user2 = args.user2
    score_type = args.score_type
    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))
```

Результат виконання

```
PS E:\labs\WIKI\lab5> python lab5_task9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean

Euclidean score:
0.30383243470068705
PS E:\labs\WIKI\lab5> python lab5_task9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281
PS E:\labs\WIKI\lab5>
```

Посилання на GitHub - https://github.com/MischenchukMykola/lab5

Висновок: виконуючи цю лабораторну роботу я використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив методи ансамблів