Лабораторна робота №5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

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

Хід роботи

Посилання на GitHub: [ross3005/ai_labs_hordeiev_pi60 (github.com)]

Завдання №1:

Напишемо код для кластеризації даних за допомогою методу k-середніх:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import metrics

# Load input data
X = np.loadtxt('data_clustering.txt', delimiter=',')
num_clusters = 5

# Plot input data
plt.figure()
plt.scatter(X[:,0], X[:,1], marker='o', facecolors='none',
        edgecolors='black', s=80)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
plt.title('Input data')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

# Create KMeans object
kmeans = KMeans(init='k-means++', n_clusters=num_clusters, n_init=10)

# Train the KMeans clustering model
kmeans.fit(X)

# Step size of the mesh
step_size = 0.01

# Define the grid of points to plot the boundaries
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_vals, y_vals = np.meshgrid(np.arange(x_min, x_max, step_size),
        np.arange(y_min, y_max, step_size))

# Predict output labels for all the points on the grid
output = kmeans.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
```

```
# Plot different regions and color them
output = output.reshape(x_vals.shape)
plt.figure()
plt.clf()
plt.imshow(output, interpolation='nearest',
           extent=(x_vals.min(), x_vals.max(),
               y_vals.min(), y_vals.max()),
           cmap=plt.cm.Paired,
           aspect='auto',
           origin='lower')

# Overlay input points
plt.scatter(X[:,0], X[:,1], marker='o', facecolors='none',
        edgecolors='black', s=80)

# Plot the centers of clusters
cluster_centers = kmeans.cluster_centers_
plt.scatter(cluster_centers[:,0], cluster_centers[:,1],
        marker='o', s=210, linewidths=4, color='black',
        zorder=12, facecolors='black')

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
plt.title('Boundaries of clusters')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

Результати:

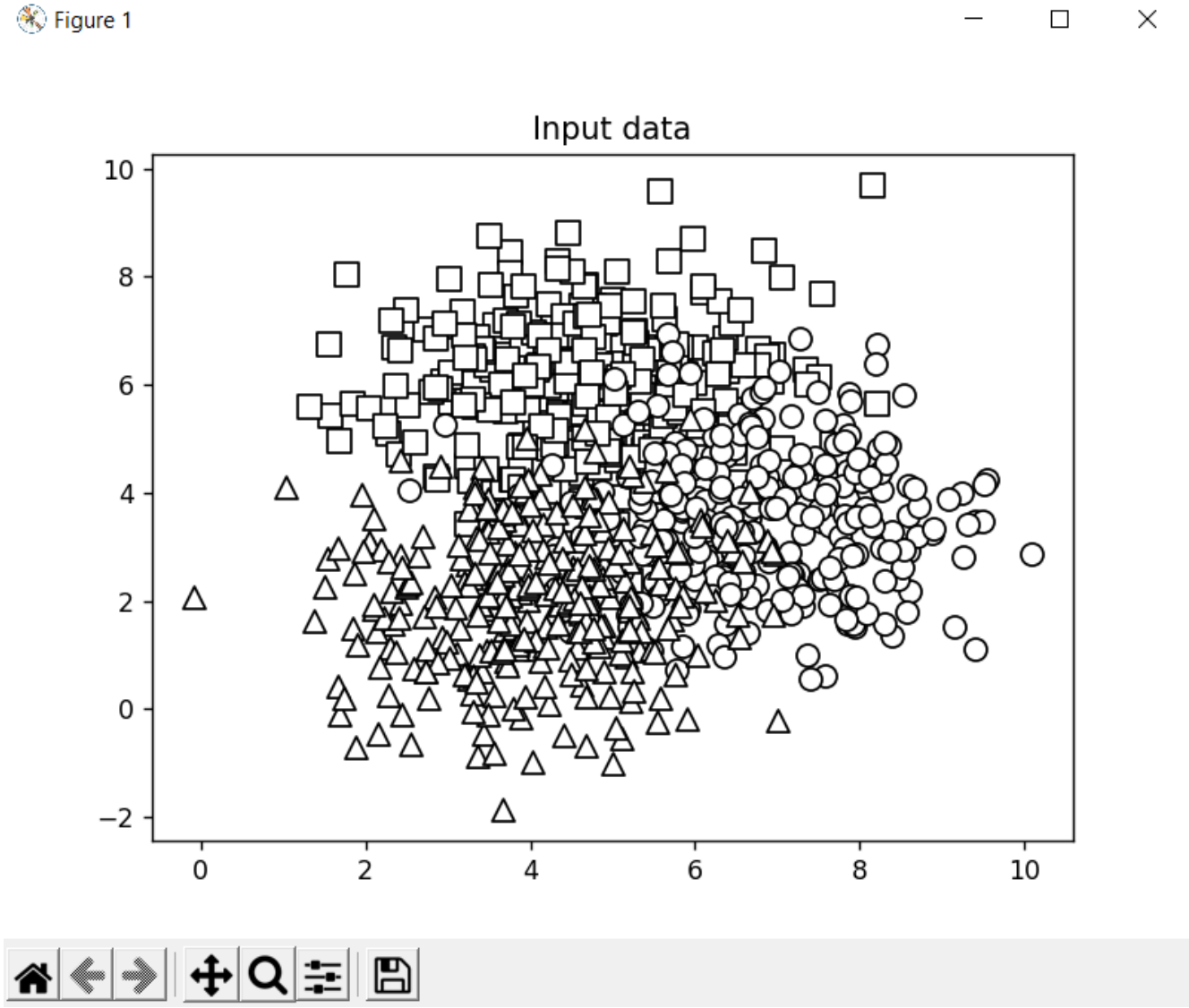

Рис. 1. Графік з вхідними даними

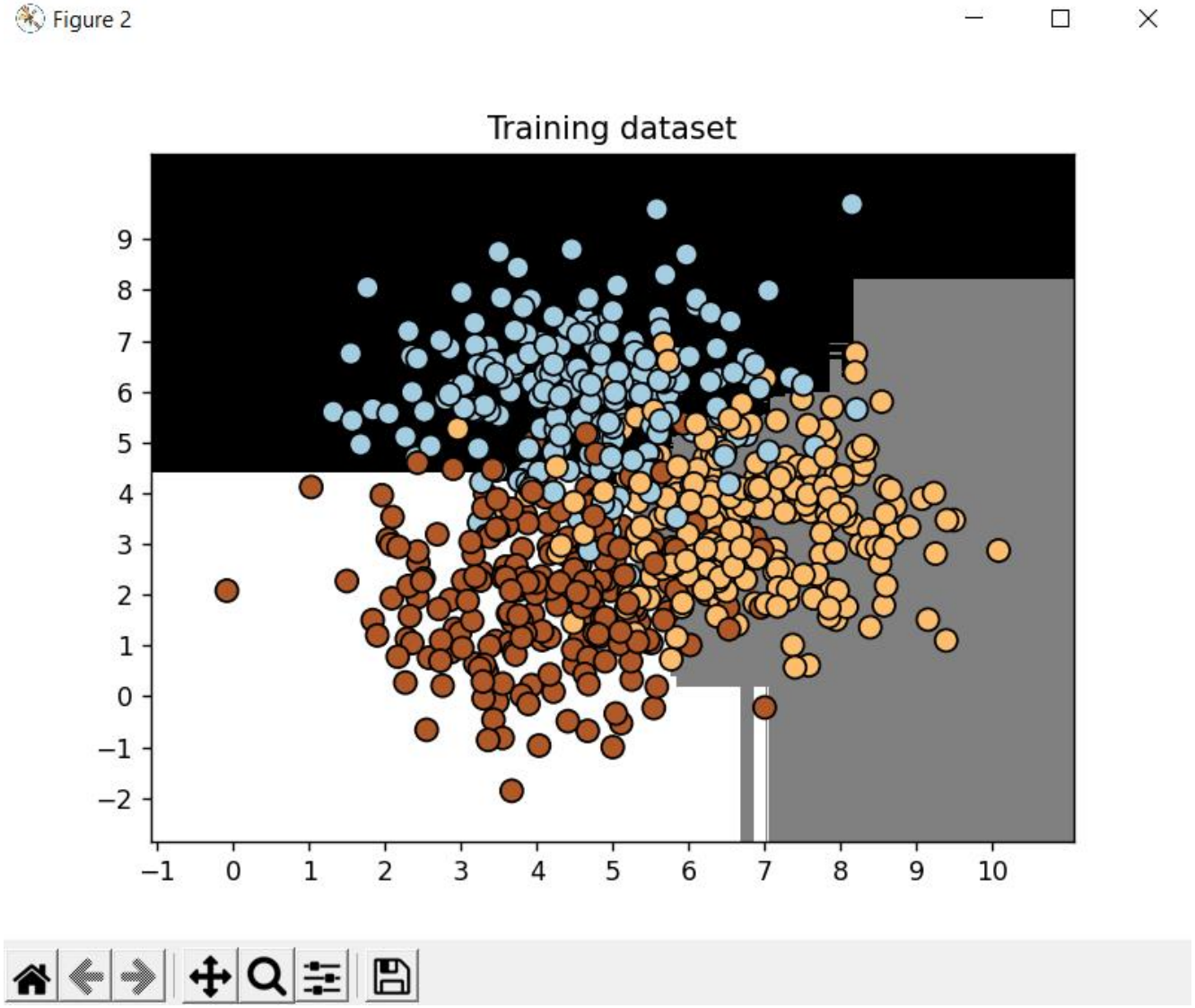

Рис. 2. Графік з кордонами класифікатора з параметром rf

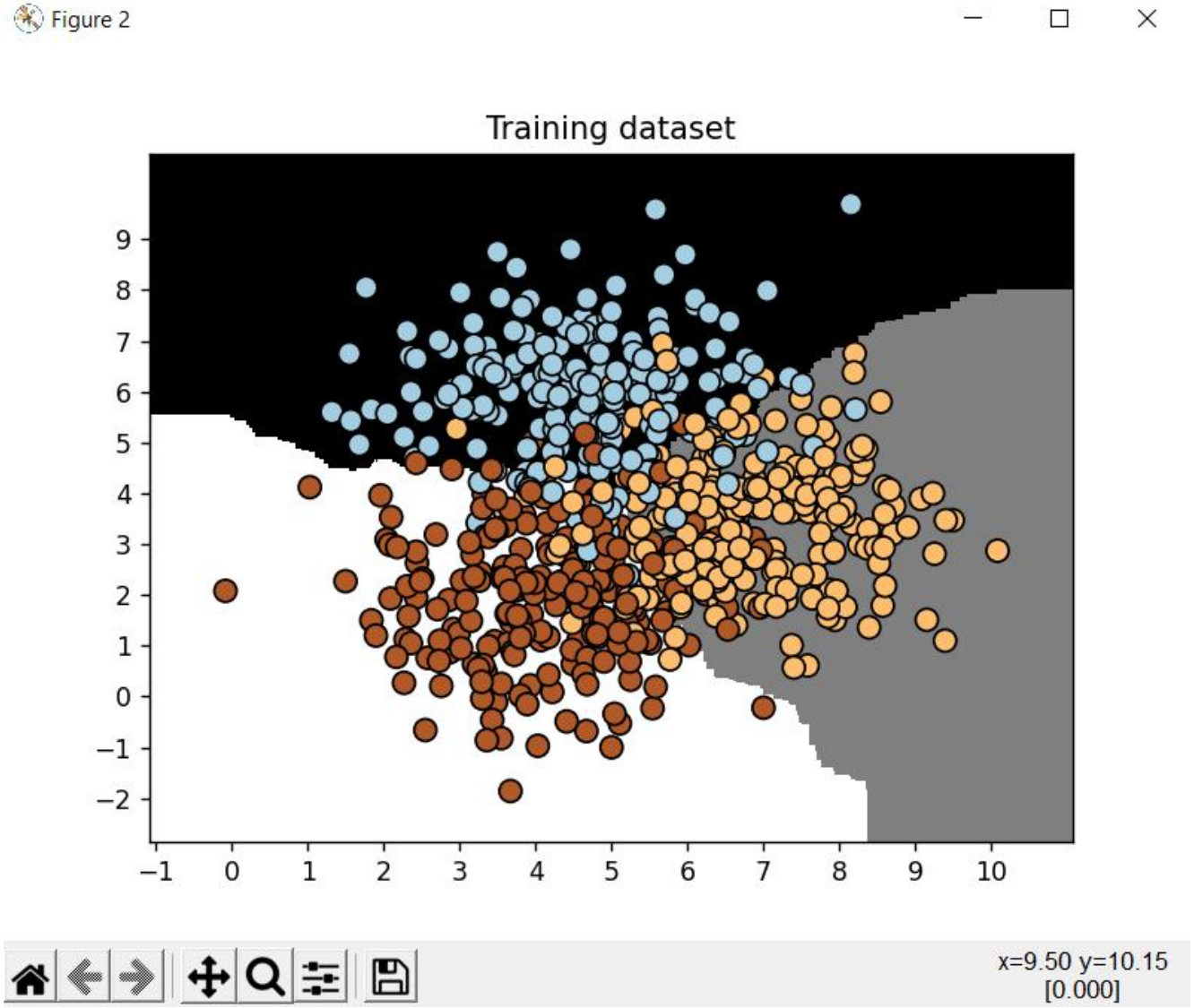

Рис. 3. Графік з кордонами класифікатора з параметром erf

Отже, судячи з графіків, варіант з параметром erf дає більш лагідні піки.

```
              precision    recall  f1-score   support

     Class-0       0.89      0.83      0.86       221
     Class-1       0.82      0.84      0.83       230
     Class-2       0.83      0.86      0.85       224

    accuracy                           0.85       675
   macro avg       0.85      0.85      0.85       675
weighted avg       0.85      0.85      0.85       675

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

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

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
Predicted class: Class-1

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

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

Datapoint: [5 2]
Predicted class: Class-2
```

Рис. 4. Результат оцінки міри достовірності прогнозів з параметром rf (перша частина)

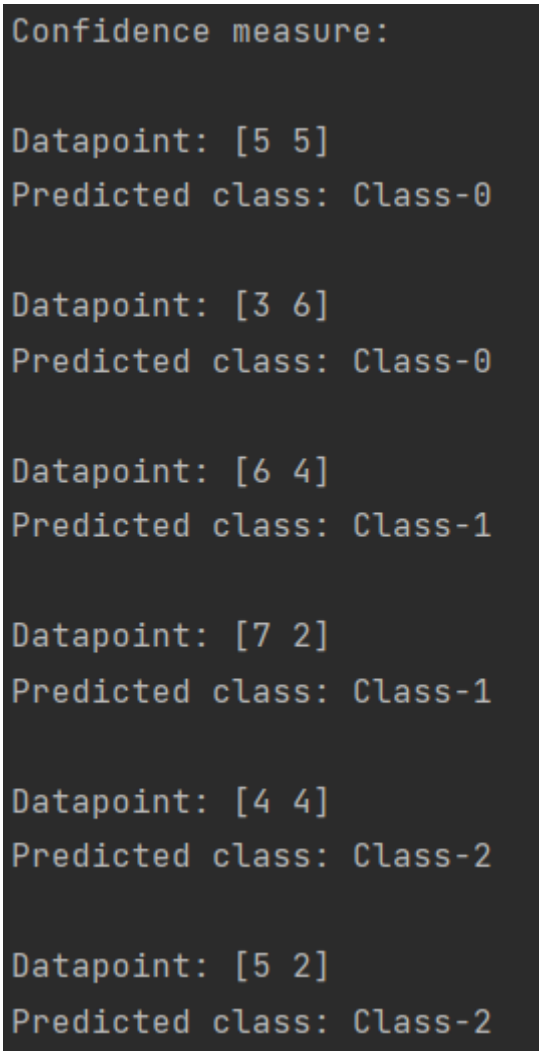

Рис. 5. Результат оцінки міри достовірності прогнозів з параметром rf (друга частина)

```
              precision    recall  f1-score   support

     Class-0       0.89      0.83      0.86       221
     Class-1       0.82      0.84      0.83       230
     Class-2       0.83      0.86      0.85       224

    accuracy                           0.85       675
   macro avg       0.85      0.85      0.85       675
weighted avg       0.85      0.85      0.85       675

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

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

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
Predicted class: Class-1

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

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

Datapoint: [5 2]
Predicted class: Class-2
```

Рис. 6. Результат оцінки міри достовірності прогнозів з параметром erf

Завдання №2:

Напишемо код для обробки з урахування дисбалансу класів:

```
import sys

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

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from utils import visualize_classifier

# Load input data
input_file = 'data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Separate input data into two classes based on labels
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])

# Visualize input data
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')

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5)

# Extremely Random Forests classifier
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
if len(sys.argv) > 1:
    if sys.argv[1] == 'balance':
        params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0,
'class_weight': 'balanced'}
    else:
        raise TypeError("Invalid input argument; should be 'balance'")

classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')

y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Test dataset')

# Evaluate classifier performance
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),
target_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()
```

Результати:

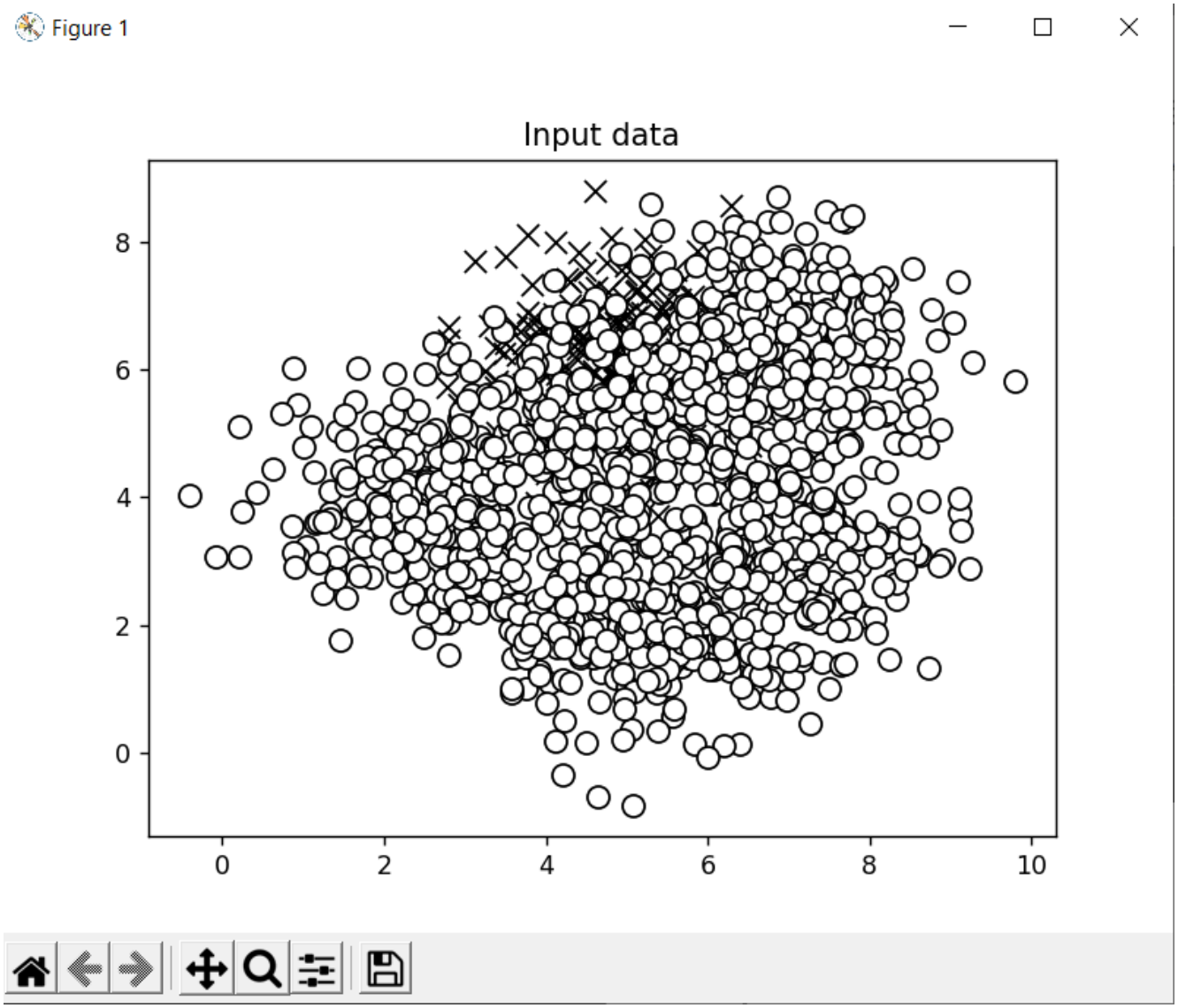

Рис. 7. Графік вхідних даних

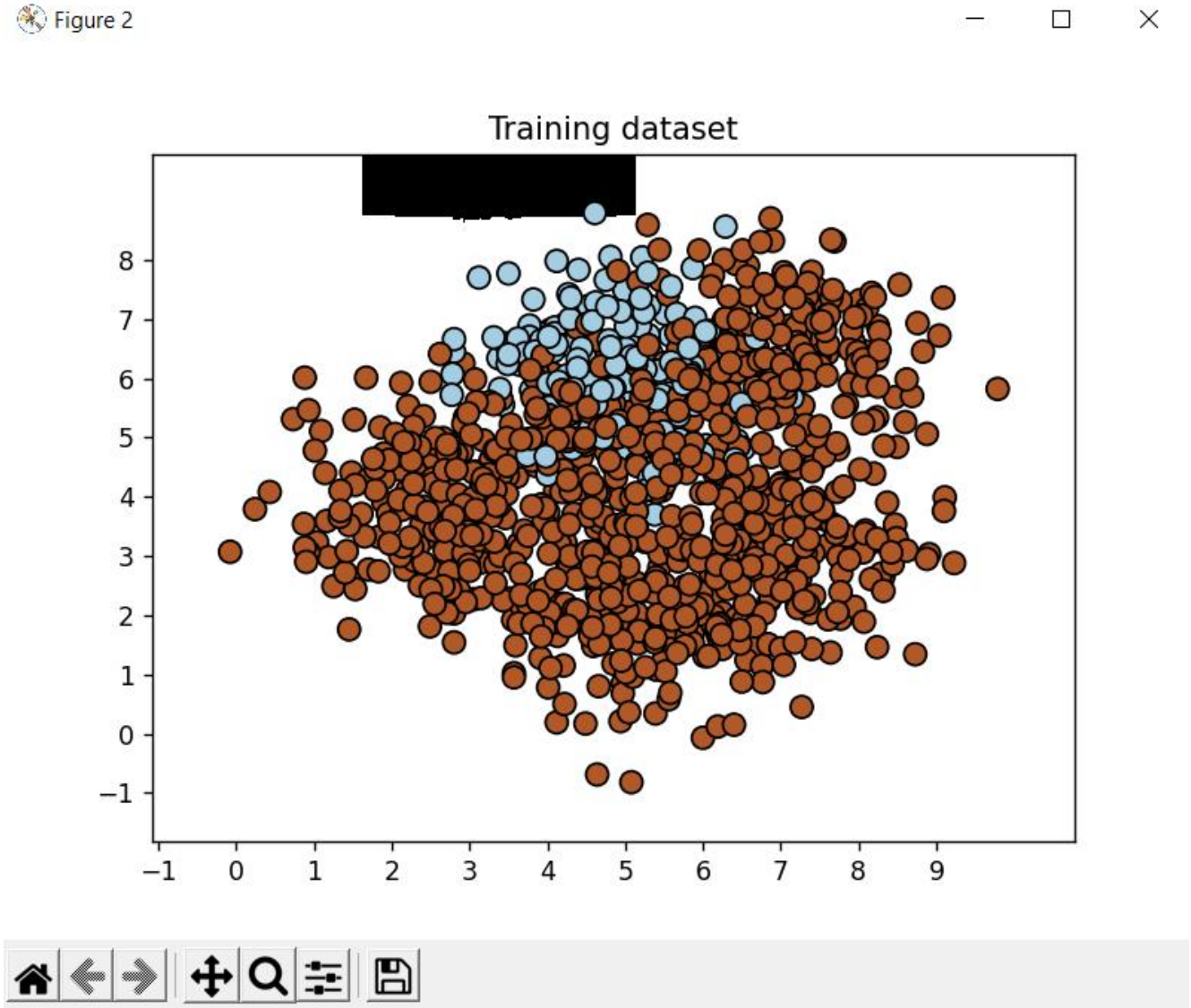

Рис. 8. Перший графік даних класифікатора для тестового набору

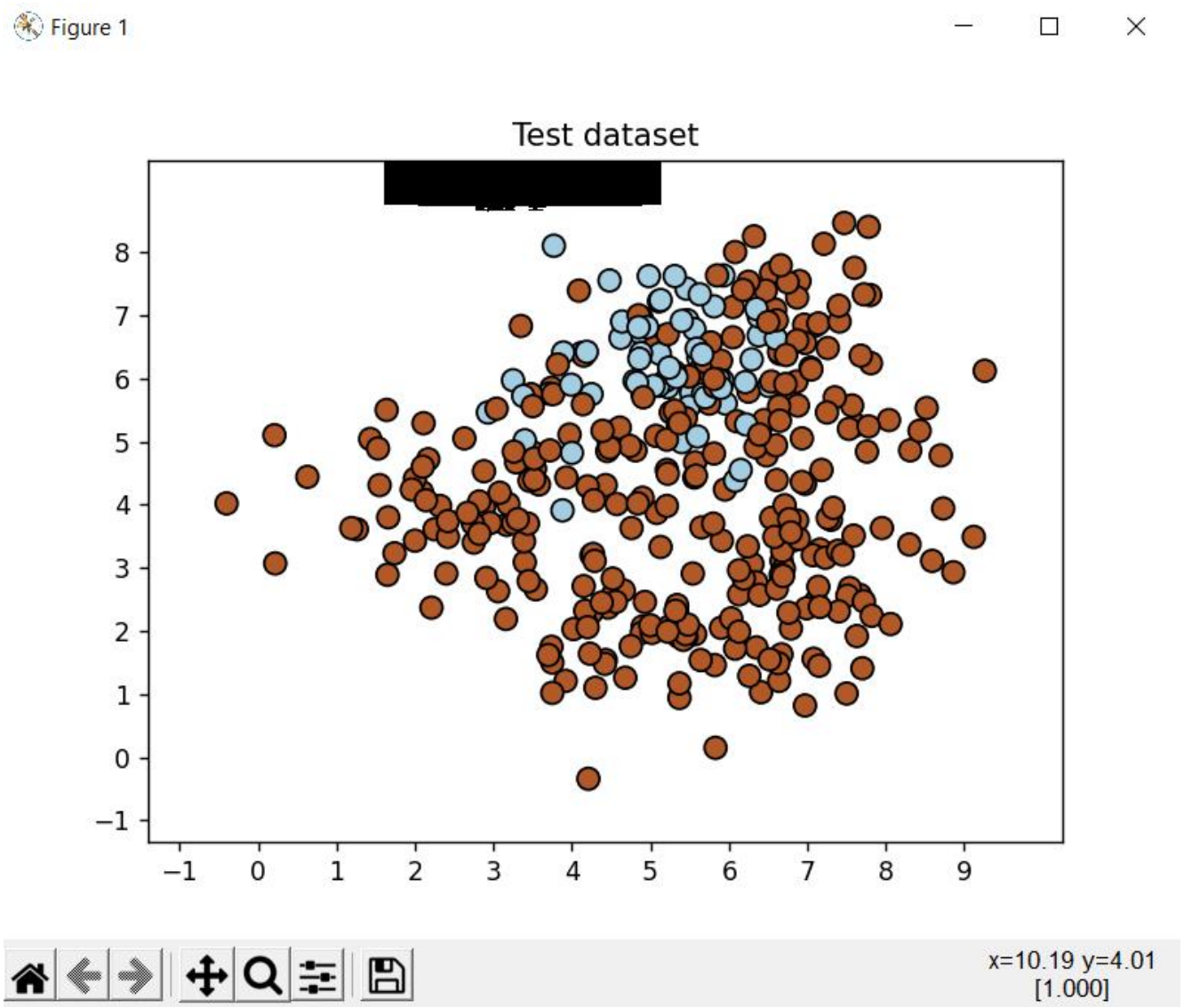

Рис. 9. Графік даних класифікатора для тестового набору

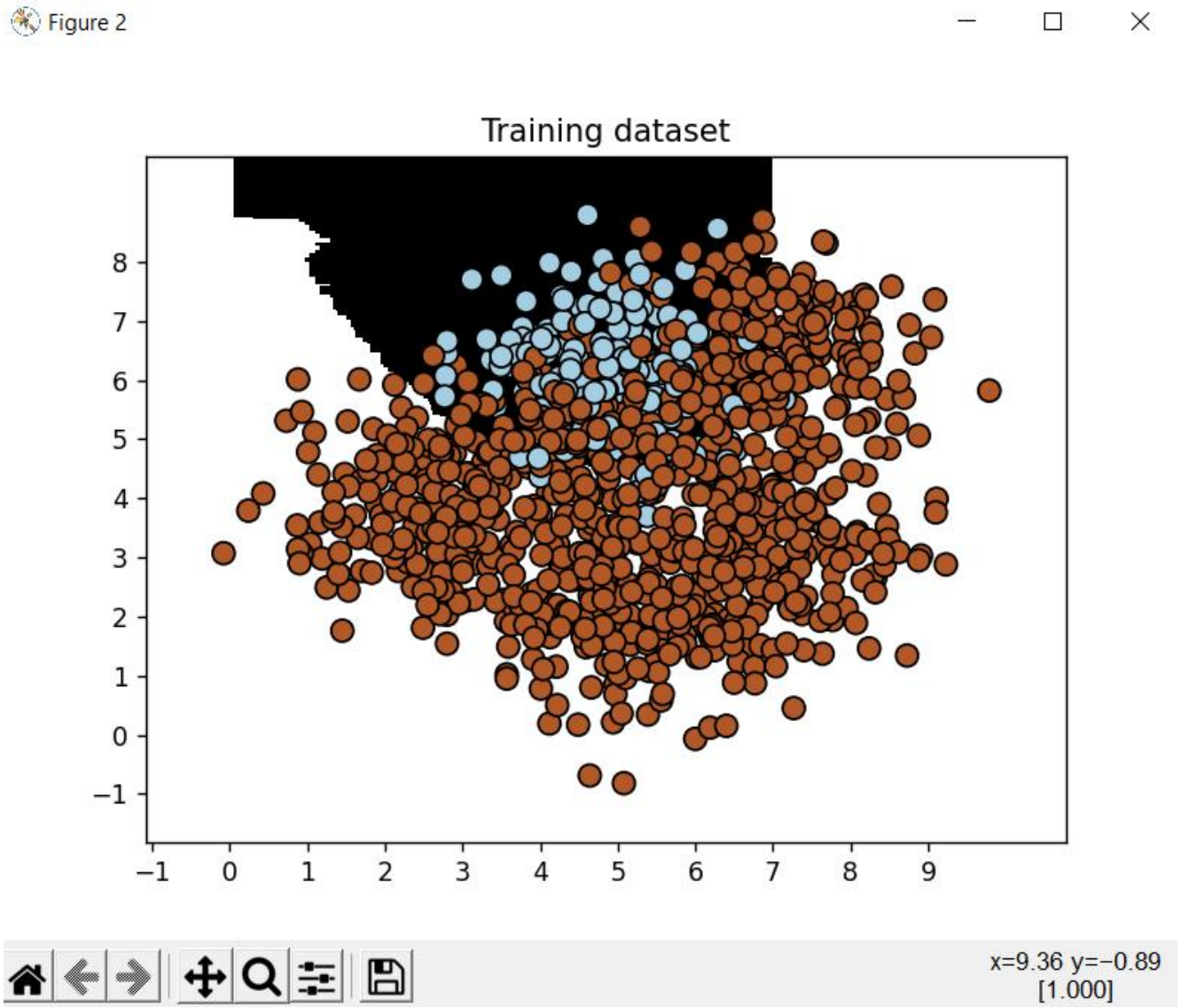

Рис. 10. Графік даних класифікатора для тренованого набору з урахування ділення на нуль

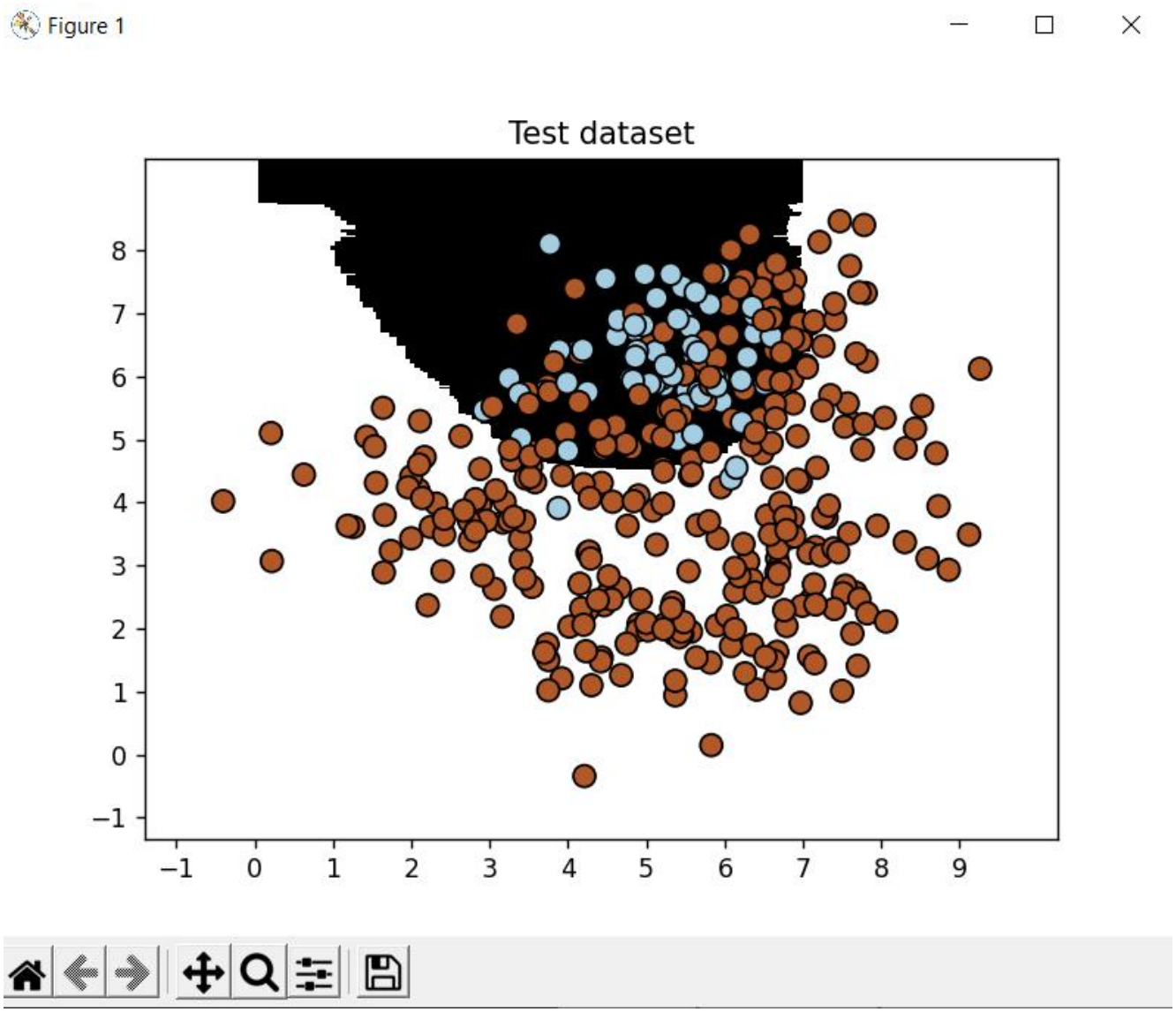

Рис. 11. Графік даних класифікатора для тестового набору з урахування ділення на нуль

```
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. 12. Оцінки якості наборів даних

Отже, з урахуванням дисбалансу класів, нам вдалося класифікувати точки даних для класу 0 з ненульовим значенням параметра точності.

Завдання №3:

Код для знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report

from utils import visualize_classifier

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

# Separate input data into three classes based on labels
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
class_2 = np.array(X[y==2])

# Split the data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5)

# Define the parameter grid
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("\n##### Searching optimal parameters for", metric)

    classifier = GridSearchCV(
            ExtraTreesClassifier(random_state=0),
            parameter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, y_train)

    print("\nGrid scores for the parameter grid:")
    for x, y in zip(classifier.cv_results_['params'],
classifier.cv_results_['mean_test_score']):
        print("{'n_estimators': " + str(x['n_estimators']) + ", 'max_depth': " +
str(x['max_depth']) + "} --> " + str(y))

    print("\nBest parameters:", classifier.best_params_)

    y_pred = classifier.predict(X_test)
    print("\nPerformance report:\n")
    print(classification_report(y_test, y_pred))
```

Результати:

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

Grid scores for the parameter grid:
{'n_estimators': 100, 'max_depth': 2} --> 0.8497565961090396
{'n_estimators': 100, 'max_depth': 4} --> 0.8411354716183237
{'n_estimators': 100, 'max_depth': 7} --> 0.8438201372795092
{'n_estimators': 100, 'max_depth': 12} --> 0.8319550120178777
{'n_estimators': 100, 'max_depth': 16} --> 0.8164844388128003
{'n_estimators': 25, 'max_depth': 4} --> 0.8455735749452762
{'n_estimators': 50, 'max_depth': 4} --> 0.8398506718022904
{'n_estimators': 100, 'max_depth': 4} --> 0.8411354716183237
{'n_estimators': 250, 'max_depth': 4} --> 0.8447879692444363

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

Performance report:

              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. 13. Найбільш оптимальні комбінації для показника precision

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

Grid scores for the parameter grid:
{'n_estimators': 100, 'max_depth': 2} --> 0.8497565961090396
{'n_estimators': 100, 'max_depth': 4} --> 0.8411354716183237
{'n_estimators': 100, 'max_depth': 7} --> 0.8438201372795092
{'n_estimators': 100, 'max_depth': 12} --> 0.8319550120178777
{'n_estimators': 100, 'max_depth': 16} --> 0.8164844388128003
{'n_estimators': 25, 'max_depth': 4} --> 0.8455735749452762
{'n_estimators': 50, 'max_depth': 4} --> 0.8398506718022904
{'n_estimators': 100, 'max_depth': 4} --> 0.8411354716183237
{'n_estimators': 250, 'max_depth': 4} --> 0.8447879692444363

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

Performance report:

              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. 14. Найбільш оптимальні комбінації для показника recall

Отже, для обох показників найкращим варіантом виявився: *{'max_depth': 2, 'n_estimators': 100}*.

Завдання №4:

Напишемо код для обчислення відносної важливості ознак:

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

```
from sklearn.utils import shuffle

from utils import visualize_classifier

# Load housing data
housing_data = datasets.load_boston()

# Shuffle the data
X, y = shuffle(housing_data.data, housing_data.target, random_state=7)

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=7)

# AdaBoost Regressor model
regressor = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
        n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)

# Evaluate performance of AdaBoost regressor
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))

# Extract feature importances
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names

# Normalize the importance values
feature_importances = 100.0 * (feature_importances / max(feature_importances))

# Sort the values and flip them
index_sorted = np.flipud(np.argsort(feature_importances))

# Arrange the X ticks
pos = np.arange(index_sorted.shape[0]) + 0.5

# Plot the bar graph
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Feature importance using AdaBoost regressor')
plt.show()
```

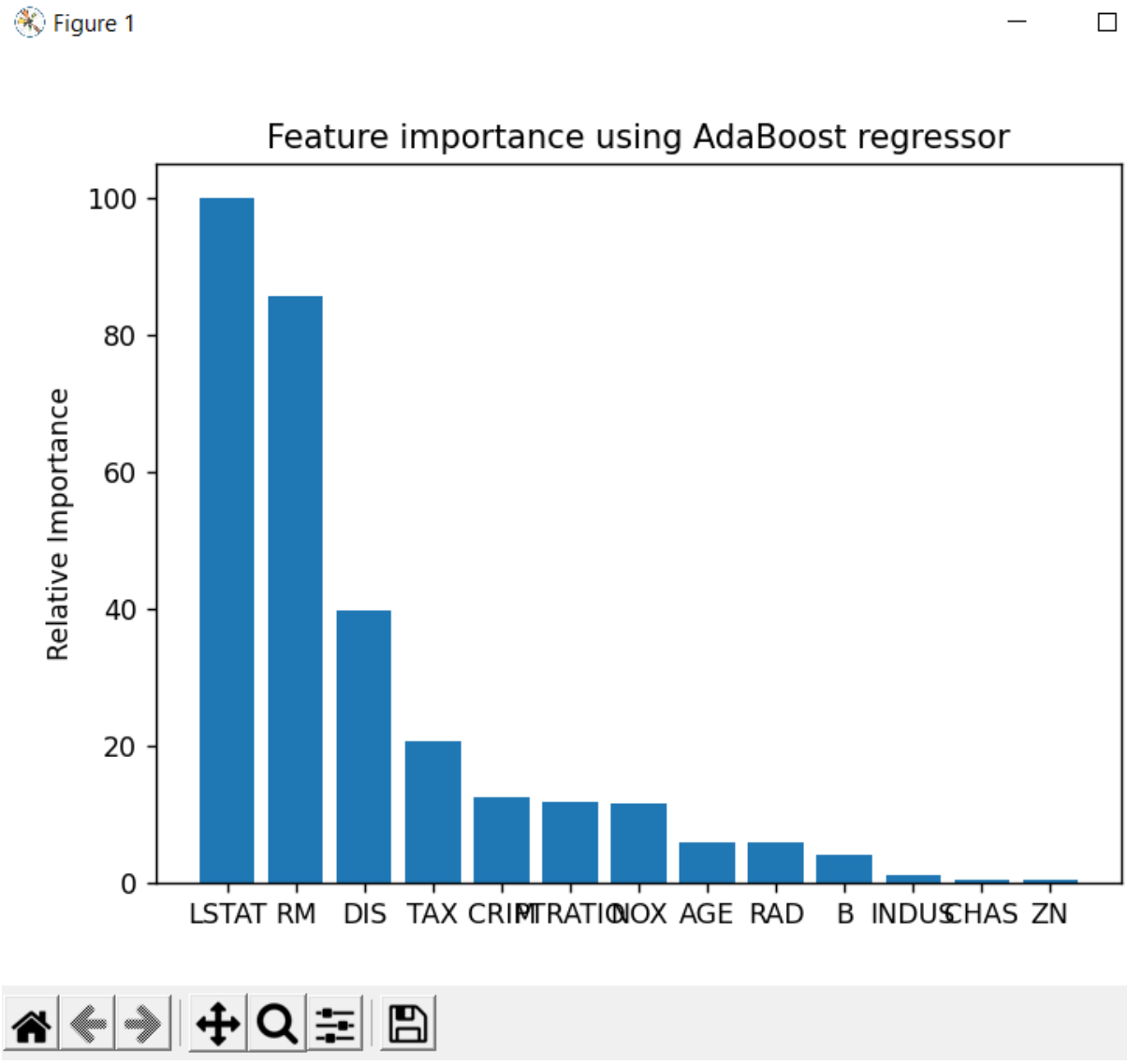

Рис. 15. Діаграма важливості ознак

Отже, судячи з графіку, найважливішим параметром є LSTAT.

Завдання №5:

Напишемо код для прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів:

```
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
from sklearn.model_selection import train_test_split

# Load input data
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)
```

```
# Convert string data to numerical 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)

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5)

# Extremely Random Forests regressor
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)

# Compute the regressor performance on test data
y_pred = regressor.predict(X_test)
print("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))

# Testing encoding on single data instance
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].transform(test_datapoint[i]))
        count = count + 1

test_datapoint_encoded = np.array(test_datapoint_encoded)

# Predict the output for the test datapoint
print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

Результат:

```
Mean absolute error: 7.42
Predicted traffic: 26
```

Рис. 16. Результат прогнозування інтенсивності дорожнього руху

Висновок: на цій лабораторній роботі я, використовуючи спеціалізовані бібліотеки та мову програмування Python, дослідив методи ансамблів у машинному навчанні.