ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

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

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

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

def build_arg_parser():  1 usage  new *
    parser = argparse.ArgumentParser(description='Classify data using Ensemble Learning techniques')
    parser.add_argument(
        *name_or_flags: '--classifier-type',
        dest='classifier_type',
        choices=['rf', 'erf'],
        default='rf',
        help="Type of classifier to use; can be either 'rf' or 'erf'")
    return parser

if __name__=='__main__':
    # Parse the input arguments
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type
    # Load input data
    input_file = 'C:/Users/Администратор/PycharmProjects/lab5/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])

    # Visualize input data
    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('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)
```

Рис 2.1 – лістинг програми

```python
# 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)

# Ensemble Learning classifier
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)
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test)

# Evaluate classifier performance
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), 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")
test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])
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)
    print('Probability:', np.max(probabilities))
# Visualize the datapoints
visualize_classifier(classifier, test_datapoints, [0] * len(test_datapoints))
plt.show()
```

Рис 2.2 – лістинг програми

```
              precision    recall  f1-score   support

     Class-0       0.91      0.86      0.88       221
     Class-1       0.84      0.87      0.86       230
     Class-2       0.86      0.87      0.86       224

    accuracy                           0.87       675
   macro avg       0.87      0.87      0.87       675
weighted avg       0.87      0.87      0.87       675

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


Classifier performance on test dataset for RF

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
     Class-1       0.86      0.84      0.85        70
     Class-2       0.84      0.92      0.88        76

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

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


Confidence measure for RF:

Datapoint: [5 5]
Predicted class: Class-0
Probability: 0.814275318675482

Datapoint: [3 6]
Predicted class: Class-0
Probability: 0.9357445782050678

Datapoint: [6 4]
Predicted class: Class-1
Probability: 0.7451078021772837

Datapoint: [7 2]
Predicted class: Class-1
Probability: 0.7066022643054627

Datapoint: [4 4]
Predicted class: Class-2
Probability: 0.6388176473387874

Datapoint: [5 2]
Predicted class: Class-2
Probability: 0.8528526682816628
```

Рис 2.3 – результат виконання

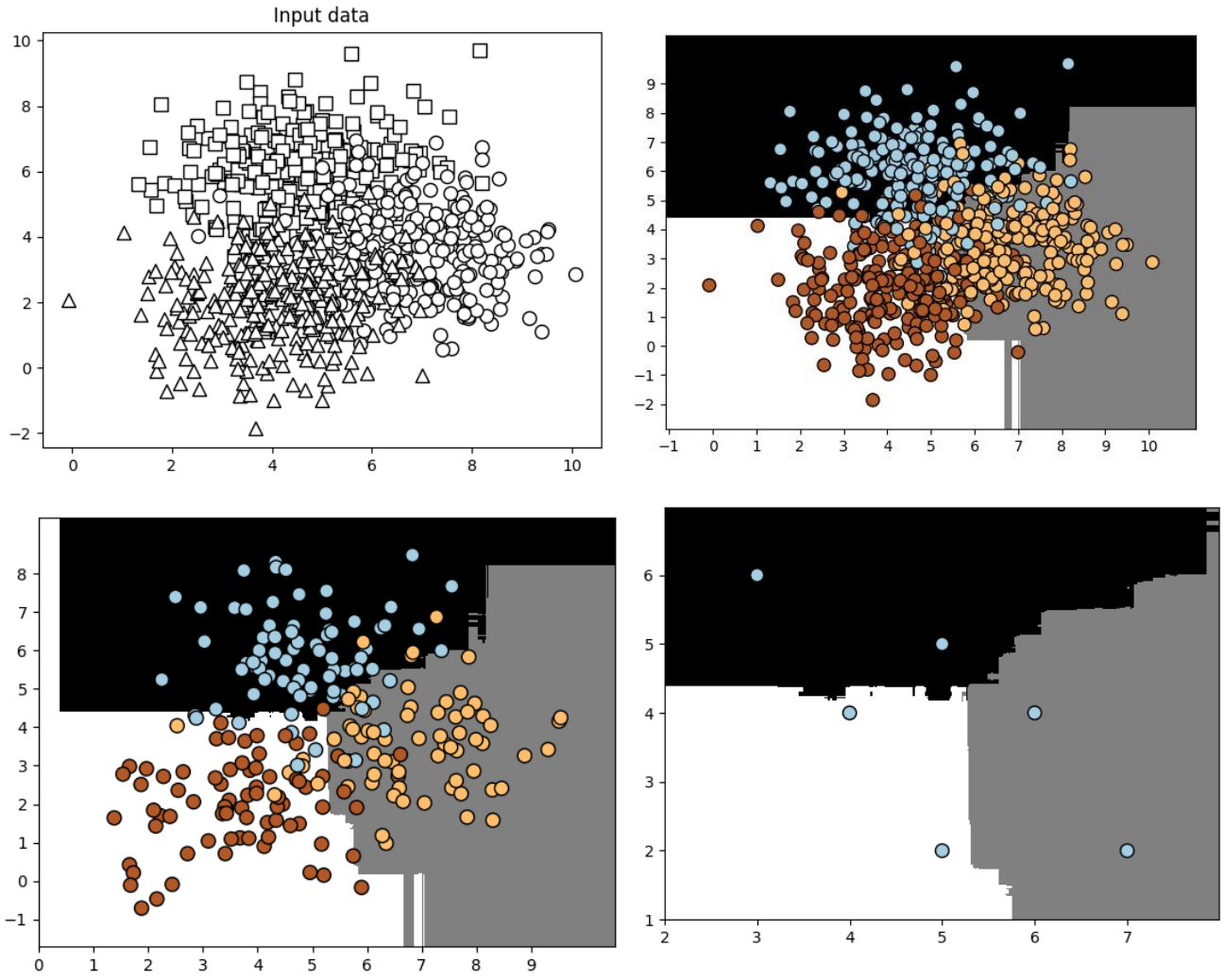


Рис 2.4 – результат виконання

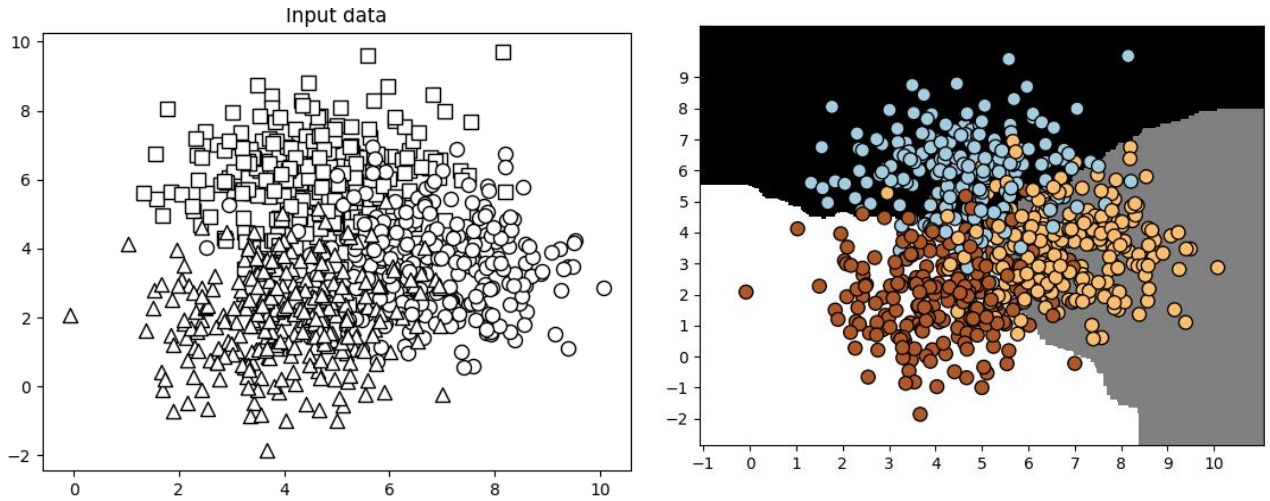


Рис 2.5 – результат виконання

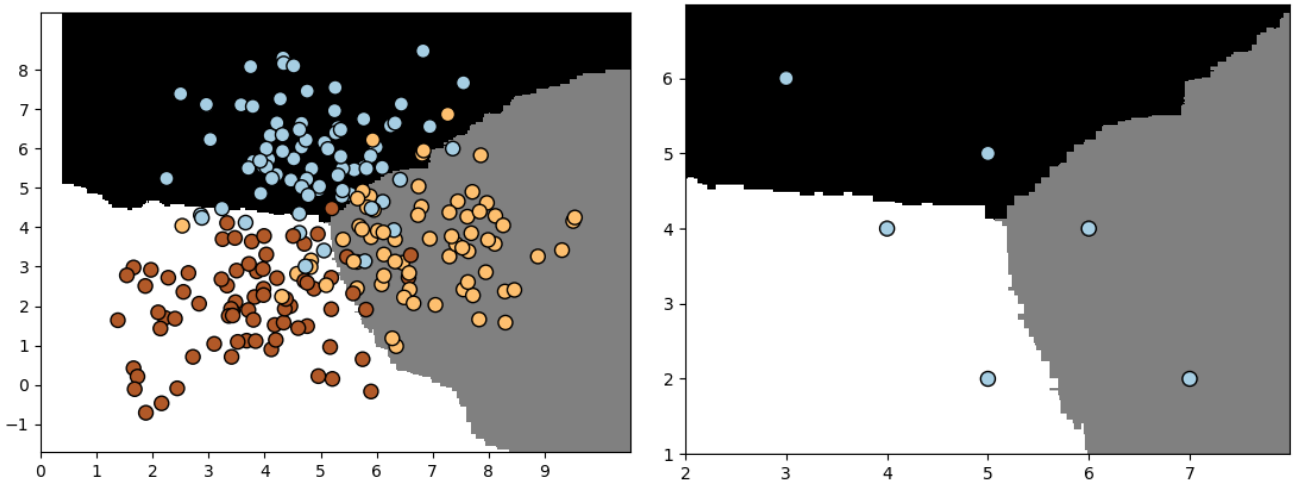


Рис 2.6 – результат виконання

```
Classifier performance on training dataset for ERF

              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 for ERF

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
     Class-1       0.84      0.84      0.84        70
     Class-2       0.85      0.92      0.89        76

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

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

Рис 2.7 – результат виконання

```
Confidence measure for ERF:

Datapoint: [5 5]
Predicted class: Class-0
Probability: 0.4890441872546998

Datapoint: [3 6]
Predicted class: Class-0
Probability: 0.6670738316255893

Datapoint: [6 4]
Predicted class: Class-1
Probability: 0.49535143822679545

Datapoint: [7 2]
Predicted class: Class-1
Probability: 0.6246676978884234

Datapoint: [4 4]
Predicted class: Class-2
Probability: 0.4512103944624855
```

Рис 2.8 – результат виконання

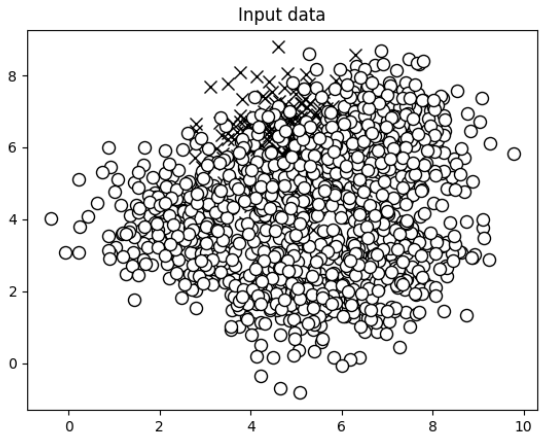
RandomForestClassifier використовує метод бутстрапування для створення підвибірок даних, на основі яких будується окреме дерево рішень для кожної підвибірки. При прогнозуванні класу алгоритм визначає підсумковий результат шляхом голосування дерев, вибираючи клас із найбільшою кількістю голосів (мода класів). Завдяки цьому RandomForestClassifier демонструє стійкість до шуму в даних і ризику перенавчання, забезпечуючи високу точність класифікації.

ExtraTreesClassifier працює схоже на RandomForestClassifier, але має кілька ключових відмінностей. Під час розщеплення вузлів він використовує всі доступні ознаки (p), на відміну від випадкової підвибірки (m), як у RandomForest. Крім того, поріг розщеплення для кожної ознаки обирається випадковим чином, а не визначається як найкращий. Завдяки цим особливостям ExtraTreesClassifier працює швидше та ефективніше, особливо на наборах даних із великою кількістю ознак.

Завдання 2.2. Обробка дисбалансу класів

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

input_file = 'data_imbalance.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])

plt.figure()
# Remove 'edgecolors' since 'x' marker doesn't support edge coloring
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='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 = {'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)

y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test)
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()
```

Рис 2.9 – лістинг програми



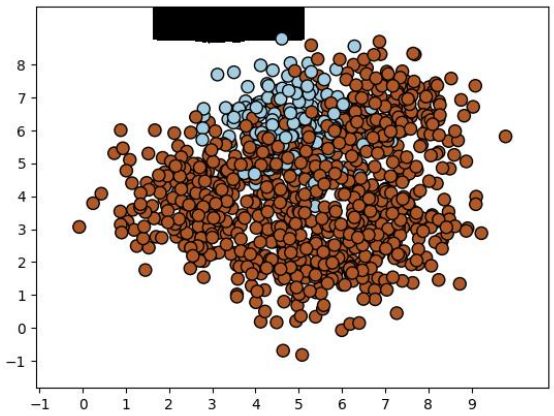


Рис 2.10 – результат програми

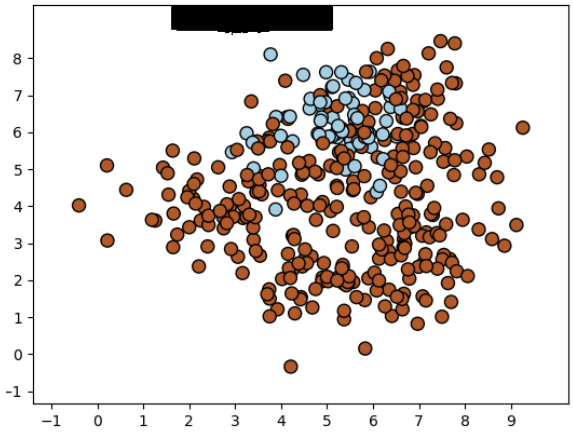


Рис 2.11 – результат виконання

```
Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       1.00      0.01      0.01       181
     Class-1       0.84      1.00      0.91       944

    accuracy                           0.84      1125
   macro avg       0.92      0.50      0.46      1125
weighted avg       0.87      0.84      0.77      1125

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

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

Classifier performance on test dataset

              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
```

Рис 2.12 – результат виконання

До збалансування модель значно недооцінювала менший клас (Class-0), демонструючи високі показники для більшого класу (Class-1), але низьку чутливість і F1-міру для Class-0. Після використання параметра class_weight='balanced' точність і чутливість для Class-0 суттєво покращилися, хоча загальна точність трохи знизилася. Збалансування є ефективним підходом для забезпечення точнішої класифікації обох класів, особливо у випадках із сильним дисбалансом даних.

Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report
from sklearn.ensemble import ExtraTreesClassifier

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("\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 i in classifier.cv_results_['params']:
        print(i, '-->', round(classifier.cv_results_['mean_test_score'][classifier.cv_results_['params'].index(i)], 3))
    print("\nBest parameters:", classifier.best_params_)
    y_pred = classifier.predict(X_test)
    print("\nPerformance report:\n")
    print(classification_report(y_test, y_pred))
```

Рис 2.13 – лістинг програми

```
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: {'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 2.14 – результат виконання

```
#### 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: {'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 2.15 – результат виконання

Код використовує GridSearchCV для автоматизованого підбору оптимальних параметрів моделі ExtraTreesClassifier за допомогою крос-валідації. Під час цього перевіряються різні значення параметрів n_estimators та max_depth, а найкращі параметри визначаються на основі метрик precision_weighted і recall_weighted. Це дає змогу знайти параметри, які забезпечують максимальну точність і повноту класифікації на тестових даних.

Завдання 2.4. Обчислення відносної важливості ознак

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')
import pandas as pd
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.utils import shuffle

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
housing_data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

label_encoder = preprocessing.LabelEncoder()
y = label_encoder.fit_transform(target)

X, y = shuffle(housing_data, y, random_state=7)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)

regressor = AdaBoostClassifier(DecisionTreeClassifier(max_depth=4), n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)

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))
```

Рис 2.16 – лістинг програми

```
feature_importances = regressor.feature_importances_
feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']

feature_importances = 100.0 * (feature_importances / max(feature_importances))

index_sorted = np.flipud(np.argsort(feature_importances))

pos = np.arange(index_sorted.shape[0]) + 0.5

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

Рис 2.17 – лістинг програми

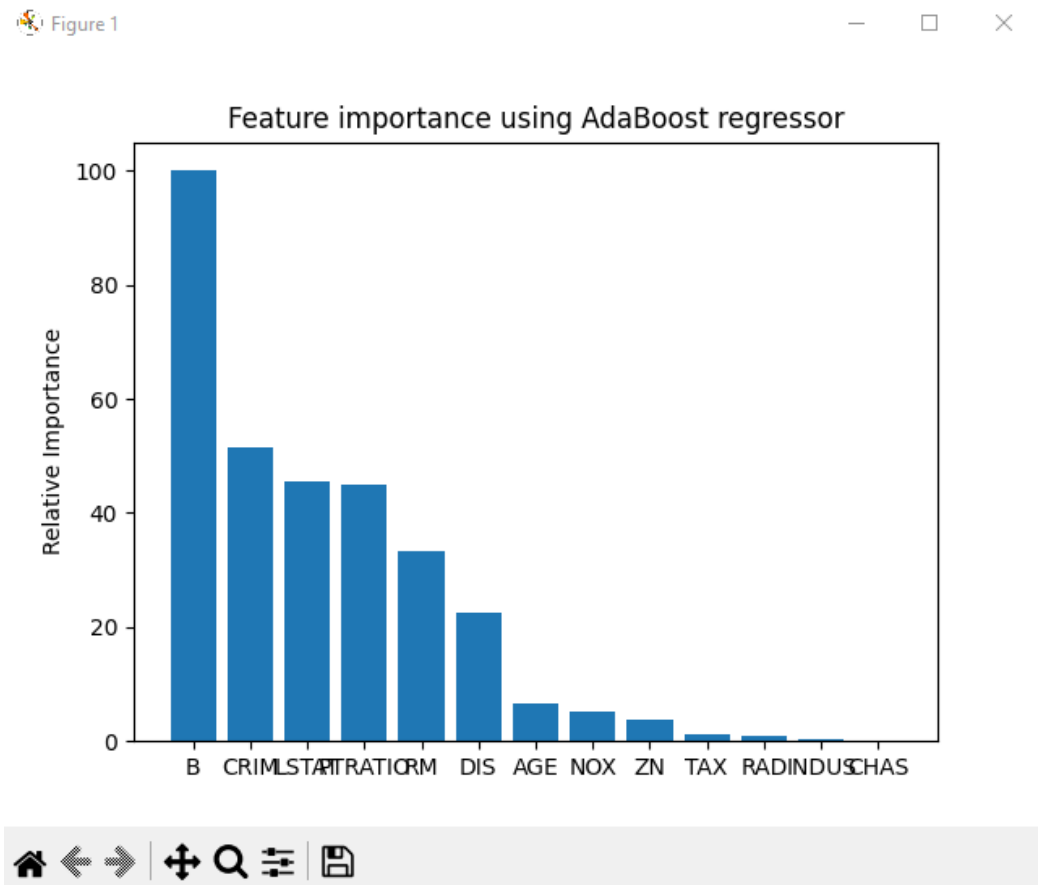


Рис 2.18 – результат виконання

```
ADABOOST REGRESSOR
Mean squared error = 1659.99
Explained variance score = 0.6
```

Рис 2.19 – результат виконання

Ознаки на діаграмі представляють різні соціально-економічні та екологічні характеристики житлових районів:

- **B** — пропорція афроамериканського населення в районі, може впливати на соціальні аспекти ціноутворення.
- **CRIM** — рівень злочинності, зазвичай негативно корелює з цінами на житло.
- **LSTAT** — відсоток населення з низьким соціально-економічним статусом, часто найбільш вагома ознака.
- **PTRATIO** — співвідношення учнів до вчителів у школах, впливає на привабливість району для сімей.
- **RM** — середня кількість кімнат у будинках, важливий фактор вартості житла.
- **DIS** — відстань до центрів зайнятості, показник доступності робочих місць.
- **AGE** — частка старих будівель, може свідчити про стан інфраструктури.
- **NOX** — рівень забруднення повітря оксидами азоту, впливає на екологічну привабливість району.
- **ZN** — частка земель під забудову великими житловими будинками, пов'язана з щільністю населення.
- **TAX** — рівень податкового навантаження, може відображати рівень місцевих послуг.
- **RAD** — доступ до великих магістралей, визначає транспортну зручність.
- **INDUS** — частка промислових площ, впливає на екологію та зайнятість.
- **CHAS** — близькість до річки Чарльз, підвищує екологічну та рекреаційну привабливість.

В аналізі важливості ознак для моделі AdaBoost ознака **B** виявилася найбільш значущою з максимальною відносною важливістю. За нею йдуть CRIM, LSTAT та PTRATION мають найбільшу роль, тоді як TAX, RAD INDUS та CHAS можна знехтувати, адже їх показники замалі.

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

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, mean_absolute_error
from sklearn.ensemble import ExtraTreesRegressor
# 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)

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].transform([test_datapoint[i]])[0])
        count += 1

test_datapoint_encoded = np.array(test_datapoint_encoded)

print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

Рис 2.20 – лістинг програми

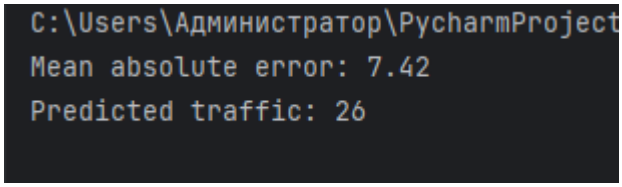


Рис 2.21 – результат виконання

Посилання на репозиторій:

https://github.com/ipz211/shi_prokopchuk_oleksandra_ipz-21-1_