МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
„ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

С.Ю. Гавриленко, В.В. Челак, О.А. Горносталь, В.Д. Зозуля

МАШИННЕ НАВЧАННЯ

Лабораторний практикум для студентів комп’ютерних спеціальностей вищих навчальних закладів

Затверджено
редакційно-видавничою
радою університету,
протокол № 1 від 29 квітня 2022 р.

Харків
НТУ «ХПІ»
2022

УДК 004.85
ББК 16.63
Г12

Рецензенти:

В.Г. Іванов, канд. техн. наук, проф. Харківського Національного технічного університету радіоелектроніки;

П.О. Качанов, д-р техн. наук, проф. Національного технічного університету «Харківський політехнічний інститут»

Гавриленко С.Ю., Челак В.В., Горносталь О.А., Зозуля В.Д.
Г 12 Машинне навчання: Лабораторний практикум. – Х.: НТУ «ХПІ», 2022.– 86 с.

Лабораторний практикум містить необхідний матеріал для виконання лабораторних робіт: варіанти завдань, стислий теоретичний матеріал, методичні вказівки та зразок виконання лабораторних робіт, приклади текстів програм та екранних форм, які демонструють результати роботи методів машинного навчання для класифікації, кластеризації та попередньої обробки даних.

Призначено для студентів комп'ютерних спеціальностей вищих навчальних закладів

Іл. 50. Табл. 8. Бібліогр: 32 назв.

УДК 004.85
ББК 16.63

ВСТУП

Сучасна проблема швидкої та якісної обробки даних об'єднує в єдине ціле три напрямки: Data Science, Data Mining та класичне програмування.

Основна мета фахівців інформаційних технологій полягає в тому, щоб зменшити час обробки даних та отримати неявну інформацію та висновки з великих, а інколи незначних обсягів даних. Для вирішення таких задач стали використовувати системи штучного інтелекту, які призначенні для часткової або повної імітації процесів, що проходять у мозку тварин, птахів та людей. Якщо додати до цієї наукової галузі велику кількість даних та обмеження системи на конкретних задачах – отримуємо розділ цієї галузі, а саме машинне навчання, яке спроможне давати висновки значне краще ніж звичайний експерт, гірше ніж аналітичний розв’язок задачі (якщо такий існує), та потребує значно менший час та обсяги обчислювальних ресурсів для експлуатації моделі, при цьому витрачаючи ресурсі на навчальному етапі. Машинне навчання вирішує ряд задач. Це задача класифікації, коли відомі найменування класів, є зразки та потрібно побудувати модель, яка буде спроможна класифікувати нові дані за новими ознаками. Задача кластеризації, коли даних багато, є приблизне знання про кількість класів, але самі дані не мають інформації, який зразок до якого класу відноситься. Також машинне навчання займається задачами регресії, асоціації, побудови систем, де пам’ять адресується за змістом.

Метою лабораторного практикуму є вивчення та практичне використання студентами методів машинного навчання для аналізу великих обсягів даних, способів попередньої їх обробки з метою оптимізації, зменшення навантаження системи при навчанні та збільшення швидкості обробки зразків.

Лабораторний практикум включає в себе 8 лабораторних робіт, серед яких перша знайомить студентів з процедурами попередньої даних. Друга робота розглядає систему метрик та показників оцінки якості методів машинного навчання. Третя лабораторна робота присвячена методу рішення задач кластеризації DBSCAN, який набуває популярності у широкому сегменті задач та перевірений часом. З четвертої по сьому лабораторні роботи розглядаються різні групи методів для рішення задачі класифікації. В четвертій роботі розглянуто один із методів опорних векторів для виявлення аномалій та шумів в даних. В п’ятій роботі досліджено методи, побудови дерев рішень. В шостій і сьомій роботі

студенти досліджують два кластери ансамблевих об’єднань: boosting та bagging. В восьмій лабораторній роботі розглядається багато-задачний метод, а саме метод глибинного машинного навчання (багатошарові нейронні мережі).

Лабораторна робота № 1
“Попередня обробка даних”

1 Мета роботи

Дослідження методів попередньої обробки даних. Розробка програмних засобів найпростіших програм фільтрів та програм попередньої обробки даних.

2 Теми для попереднього опрацювання

Очищення даних.
Шуми та аномалії в даних.
Масштабування даних.
Перетворення вданих
Скорочення даних.
Секціонування даних.

3 Короткі теоретичні відомості

В цілому, попередня обробка даних складається з п'яти основних завдань: очищення, масштабування, перетворення, скорочення, та секціонування даних.

Очищення (Data cleaning) даних спрямовано на підвищення якості даних шляхом заповнення відсутніх значень і видалення шуму або викидів в даних.

Масштабування або нормалізація (Data transformation) даних спрямовано на перетворення вихідних даних в відповідні діапазони для прогнозного моделювання.

Перетворення даних (Scaling to a specific range) – це впорядкування вихідних даних у відповідні формати для різних алгоритмів інтелектуального аналізу даних (наприклад, перетворення числових даних в категоріальні дані і навпаки).

Скорочення або ущільнення даних (Data reduction) застосовується шляхом вибору вибірок (об’єктів) і атрибутів з метою спрощення обробки даних та зниження обчислювальних витрат.

Секціонування даних (Data partitioning) спрямовано на поділ всього набору даних на різні групи з метою підвищення чутливості та надійності подальшого аналізу.

В рамках лабораторної роботи досліджено наступні завдання попередньої обробки даних:

1. Масштабування даних.
2. Шуми та аномалії в даних.
3. Пропуски значень в даних.
4. Оптимізація критеріїв.

4 Порядок виконання індивідуального завдання

Залежно від номера прізвища за списком вибрати індивідуальне завдання (Табл.1.1). Розробити два програмні модулі: модуль генерації псевдовипадкових даних та модуль попередньої обробки згенерованих даних.

Програмний модуль генерації псевдовипадкових даних повинен генерувати таблицю з вихідними даними, які належать діапазону [0-100] розміру MxN, де N – кількість атрибутів (ознак) об'єкту, M – кількість вибірок або об'єктів, мати вихідний формат даних – ***.csv** та містити наступні налаштування (табл.1.1):

1. Налаштування діапазону значень ознаки.
2. Налаштування частоти появи суперечливої інформації даних.
3. Налаштування частоти появи пропусків в даних.
4. Налаштування частоти появи аномалії.
5. Налаштування частоти появи помилок введення даних.

Таблиця 1.1 – Індивідуальне завдання

№ % 32	Частота появи суперечливої інформації	Частота появи пропусків	Частота появи аномалій	Частота появи помилок введення	M	N	L	k	X
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	23%	25%	27%	36%	53000	40000	8	1.0	530
1	6%	21%	26%	31%	26000	29000	3	1.5	360
2	8%	21%	31%	37%	33000	43000	7	2.0	960
3	10%	15%	27%	31%	33000	27000	9	2.5	340
4	5%	7%	11%	22%	46000	44000	3	3.0	960
5	0%	2%	9%	35%	31000	52000	6	1.0	350
6	0%	36%	38%	39%	32000	23000	1	1.5	70
7	13%	15%	27%	28%	36000	51000	2	2.0	560
8	21%	25%	30%	31%	21000	24000	4	2.5	860
9	13%	17%	27%	33%	48000	48000	7	3.0	140
10	4%	23%	24%	26%	42000	48000	8	1.0	10
11	8%	27%	28%	29%	19000	33000	1	1.5	120
12	9%	10%	16%	35%	49000	28000	10	2.0	590
13	15%	22%	23%	33%	35000	26000	6	2.5	620
14	10%	17%	27%	38%	24000	46000	6	3.0	650
15	4%	21%	24%	36%	48000	45000	4	1.0	780
16	19%	31%	34%	40%	23000	33000	2	1.5	330
17	7%	27%	35%	37%	34000	44000	7	2.0	220
18	6%	8%	22%	25%	22000	45000	9	2.5	480
19	14%	24%	25%	27%	29000	37000	8	3.0	850
20	0%	4%	23%	34%	41000	40000	3	1.0	510
21	2%	4%	30%	38%	47000	41000	7	1.5	460
22	4%	25%	26%	29%	30000	36000	9	2.0	80
23	22%	26%	31%	39%	34000	29000	3	2.5	770
24	5%	12%	33%	40%	26000	37000	6	3.0	780
25	2%	7%	30%	40%	32000	46000	1	1.0	40
26	2%	8%	19%	32%	47000	47000	2	1.5	390

1	2	3	4	5	6	7	8	9	10
27	3%	14%	19%	20%	32000	53000	4	2.0	30
28	17%	27%	33%	40%	43000	44000	7	2.5	700
29	5%	20%	23%	31%	35000	44000	8	3.0	430
30	7%	22%	27%	29%	50000	32000	1	1.0	300
31	2%	13%	14%	39%	26000	34000	10	1.5	320

де № – номер студента за списком у журналі групи,; % – операція знаходження залишку від цілочислового ділення, L – порядковий номер кожної аномальної вибірки, k –ваговий коефіцієнт, X – поріг прийняття рішення щодо аномальності даних.

Модуль попередньої обробки даних повинен виконувати наступні функції:

- Обробку пропусків в даних.
- Виявлення суперечливої інформації.
- Обробку аномальних значень.
- Оптимізацію критеріїв.
- Масштабування даних.
- Формування звіту щодо результатів попередньої обробки даних, який містить таку інформацію: начальний та кінцевий розмір даних, кількість відсотків та абсолютне значенням видалених даних.

4.1 Обробка пропусків в даних

Обробка пропусків в даних виконується таким чином:

– Якщо ваше прізвище та ім’я починаються на приголосну, то замінити пропущені значення **середньо- арифметичним значенням**.

– Якщо ваше прізвище та ім’я починаються на голосну то замінити пропущені значення **середньо- гармонічним значенням**.

– Якщо ваше прізвище починається на приголосну а ім’я на голосну, то замінити пропущені значення **середньо-геометричним значенням**.

– Якщо ваше прізвище починається на голосну а ім’я на приголосну, то замінити пропущені значення **середньо- квадратичним значенням**.

4.2 Обробка суперечної інформації.

Суперечливою є інформація, яка містить розбіжності. Наприклад, за одних и тих же вихідних даних цільова функція приймає різне значення.

Необхідно знайти усі **суперечливі дані** та видалити їх або замінити значення цільової функції на найбільш вірогідну.

4.3 Обробка аномалій

Аномалія – це неправильність, відхилення від норми або від загальної закономірності. Основними методами виявлення аномалій в даних є:метод стандартного відхилення (Standard Deviation Method), метод

міжквартильного діапазону (Interquartile Range Method), метод локального рівня викидів (Local Outlier Factor), метод Ізолюючого лісу (Isolation Forest).

В рамках лабораторної роботи розглянуто метод стандартного відхилення.

Основою методу стандартного відхилення є розрахунок середніх значень ряду

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

та середньоквадратичного відхилення

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2},$$

де N– кількість елементів, x_i – і-й елемент вибірки.

Поріг визначення аномалій, зазвичай, визначається як

$$anomaly\_score = x_i \pm 2\sigma.$$

В рамках даної лабораторної роботи поріг визначення аномалій визначається як

$$anomaly\_score = x_i \pm k\sigma.$$

де *k* ваговий коефіцієнт (див. 9 стовбець табл. 1.1).

Аномальною будемо вважати вибірку або об'єкт (рядок згенерованої таблиці), *L* -показників якої (див. восьмий стовбець табл. 1.1) виходять за межі *anomaly_score*. Аномальні вибірки необхідно видали із даних.

4.4 Оптимізація критеріїв

Дані, які мають близько-нульову дисперсія, не мають вагомого впливу на цільову функцію, тобто не є інформативними. Необхідно визначити дані у яких дисперсія менш ніж **X (див. 10 стовбець табл. 1.1).**)та видалити їх.

4.5 Масштабування даних

Виконати нормалізацію даних за методом міні-макса: лінійне перетворення даних в діапазоні, наприклад, від 0 до 1, де мінімальне і максимальне масштабовані значення відповідають 0 і 1 відповідно:

$$x_{norm} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}.$$

5 Зміст звіту з лабораторної роботи

- Титульний лист.
- Тема роботи.

- Мета роботи.
- Завдання.
- Алгоритм програми (текстовий та/або графічний вигляд).
- Текст програми.
- Результати виконання роботи програми.
- Висновки.

6. Програмна реалізація модуля попередньої обробки даних

6.1. Бібліотеки та методи

Бібліотеки, що використовуються:

Бібліотеки, що використовуються:

- pandas==1.2.4
- numpy==1.22.2
- sklearn==1.0.2
- scipy==1.7.3
- matplotlib==3.4.0

Методи pandas рекомендовані до розгляду:

- pandas.DataFrame
- pandas.DataFrame.shape
- pandas.DataFrame.drop
- pandas.DataFrame.dropna
- pandas.DataFrame.fillna
- pandas.DataFrame.apply
- pandas.DataFrame.value_counts
- pandas.DataFrame.isna
- pandas.DataFrame.sort_values
- pandas.DataFrame.duplicated
- pandas.DataFrame.drop_duplicates
- pandas.DataFrame.reset_index

Розробимо модуль генерування датасету та допоміжні функції:

```
from sklearn.datasets import make_classification
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import pandas as pd
import random
random.seed(42)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer,
RobustScaler
from scipy.stats import hmean, gmean
from scipy import mean as arithmetic_mean
import math

def generate_dataset(M,
                     N,
                     non_informative_feature,
                     n_classes,
                     feature_range,
                     nan_percentage,
                     outlier_percentage,
                     outlier_range=(999,9999)):

    if (nan_percentage + outlier_percentage) > 1.0:
        print(f'[nan_percentage + outlier_percentage] should be not greater
than 1.0')
        return

    X, y = make_classification(
        n_samples=M,
        n_features=N,
        n_classes=n_classes,
        n_redundant=non_informative_feature,
        n_informative=(N - non_informative_feature),
        random_state=42
    )

    scaler = MinMaxScaler(feature_range=feature_range)
    X = scaler.fit_transform(X)

    for idx, x in np.ndenumerate(X):
        i = idx[0]
        j = idx[1]
        X[i][j] = np.int16(x)

    for r in range(0, int(M*nan_percentage) ):
        while True:
            i = random.randint(0,M-1)

            if not any(np.isnan(x) for x in X[i].flatten()):
                j = random.randint(0,N-1)
                X[i][j] = np.nan
                break


    for r in range(0, int(M*outlier_percentage) ):
        while True:
            i = random.randint(0,M-1)

            if not any(np.isnan(x) for x in X[i].flatten()) and not any(x
for x in X[i].flatten() if outlier_range[0]<x<outlier_range[1]):
                j = random.randint(0,N-1)
                X[i][j] = random.randint(outlier_range[0],outlier_range[1])
                break

    df = pd.concat([
        pd.DataFrame(y, columns=['label']),
```

```
        pd.DataFrame(X, columns=[f'column_{c}' for c in range(0,N)])
    ], axis=1)

    return df


def get_feature_importance(df, method):
    X = df.drop(columns=['label'])
    y = df['label']

    if method == 'randomforest':
        forest = RandomForestClassifier(random_state=42)
        forest.fit(X, y)
        importances = forest.feature_importances_
        forest_importances = pd.Series(importances, index=df.columns[1:])
        return forest_importances
    elif method == 'decisiontree':
        model = DecisionTreeClassifier()
        model.fit(X, y)
        importances = model.feature_importances_
        dectree_importances = pd.Series(importances, index=df.columns[1:])
        return dectree_importances
    elif method == 'logreg':
        model = LinearRegression()
        model.fit(X, y)
        importances = model.coef_
        logreg_importances = abs(pd.Series(importances,
index=df.columns[1:]))
        return logreg_importances
    else:
        print(f'Incorrect method.')
        return np.nan


def process_nan(x, method):
    x_with_no_nan = x.dropna()
    if method == 'mean':
        return arithmetic_mean(x_with_no_nan)
    elif method == 'hmean':
        return hmean(x_with_no_nan)
    elif method == 'gmean':
        return gmean(x_with_no_nan)
    elif method == 'rms':
        n = len(x_with_no_nan)
        square = (x_with_no_nan ** 2).sum()
        mean = (square / (float)(n))
        root = math.sqrt(mean)
        return root
    else:
        print(f'Incorrect method.')
        return np.nan

def scale_data(X, method):
    if method == 'minmax':
        return MinMaxScaler().fit_transform(X)
    elif method == 'standard':
        return StandardScaler().fit_transform(X)
    elif method == 'norm':
        return Normalizer().fit_transform(X)
    elif method == 'robust':
        return RobustScaler().fit_transform(X)
    else:
        print(f'Incorrect method.')
```

```
            return np.nan
```

6.2. Опис алгоритму генерації та попередньої обробка даних

Використовуючи розроблені функції формуємо датасет (рис. 1.1) та виводимо перші п’ять рядків (df.head(20) – буде виведено 20 рядків, NaN – означає, що значення відсутнє):

```
df = generate_dataset(M=20000,
                      N=10,
                      non_informative_feature=2,
                      n_classes=2,
                      feature_range=(1,20),
                      nan_percentage=0.4,
                      outlier_percentage=0.3)
df.head()
```

	label	column_0	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9
0	0	10.0	10.0	11.0	8.0	10.0	9.0	10.0	13.0	12.0	11.0
1	1	12.0	14.0	NaN	13.0	12.0	9.0	6.0	5.0	7.0	4.0
2	1	10.0	9.0	8.0	NaN	13.0	14.0	10.0	8.0	10.0	7.0
3	0	9.0	5.0	6.0	14.0	9.0	13.0	11.0	10.0	NaN	13.0
4	1	8.0	9.0	9.0	11.0	11.0	13.0	11.0	11.0	11.0	12.0

Рисунок 1.1 – Перші 5 зразків отриманого датасету

Виконаємо розрахунок кількості аномалій в отриманих даних:

```
df.apply(lambda x: any(xx for xx in x if 999<xx<9999) ,
axis=1).value_counts()
```

```
False   14001
True     5999
dtype: int64
```

Знайдемо дані, які містять пропуски:

```
df.isna().any(axis=1).value_counts()
```

```
False   12000
True     8000
dtype: int64
```

Виконаємо обробку пропущених даних (*NaN*) за допомогою функції *process_nan(x, method)*. Підрахуємо середнє значення даних колонки *x* методом *method* та замінимо пропущенні значення на знайдені.

Значення, що приймаються:

- Параметр x: *pandas.Series* – колонка таблиці df.
- Параметр method : *str* – метод, який використовується для заповнення перепусток. Можливі значення параметра method:
  - mean (Arithmetic Mean)
  - hmean (Harmonic Mean)

- gmean (Geometric Mean)
- rms (Root Mean Square)

Значення, що повертається - *float* (середнє значення в залежності від методу method).

```
df = df.fillna(df.apply(lambda x: process_nan(x,'hmean'),axis=0))
df.head()
```

	label	column_0	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9
0	0	10.0	10.0	11.000000	8.000000	10.0	9.0	10.0	13.0	12.000000	11.0
1	1	12.0	14.0	8.504062	13.000000	12.0	9.0	6.0	5.0	7.000000	4.0
2	1	10.0	9.0	8.000000	8.755474	13.0	14.0	10.0	8.0	10.000000	7.0
3	0	9.0	5.0	6.000000	14.000000	9.0	13.0	11.0	10.0	10.868354	13.0
4	1	8.0	9.0	9.000000	11.000000	11.0	13.0	11.0	11.0	11.000000	12.0

Рисунок 1.2 – Перші 5 зразків після обробки пропусків в даних

Видалимо дублікати в даних використовуючи вбудований метод pandas.DataFrame. drop_duplicates (рис. 1.3)

```
columns_without_label = df.columns.to_list()[1:]
df[df.duplicated(subset=columns_without_label,
keep=False)].sort_values(by=columns_without_label)
```

	label	column_0	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9
8604	0	8.0	6.000000	10.0	12.0	10.0	10.000000	11.0	12.0	8.000000	10.0
17726	0	8.0	6.000000	10.0	12.0	10.0	10.000000	11.0	12.0	8.000000	10.0
5532	0	9.0	6.000000	9.0	10.0	11.0	11.000000	11.0	12.0	10.000000	11.0
15536	0	9.0	6.000000	9.0	10.0	11.0	11.000000	11.0	12.0	10.000000	11.0
4829	0	9.0	7.000000	10.0	10.0	12.0	10.188979	11.0	12.0	11.000000	10.0
5403	0	9.0	7.000000	10.0	10.0	12.0	10.188979	11.0	12.0	11.000000	10.0
8321	0	9.0	8.000000	11.0	10.0	12.0	9.000000	11.0	12.0	11.000000	10.0
15709	0	9.0	8.000000	11.0	10.0	12.0	9.000000	11.0	12.0	11.000000	10.0
14151	0	9.0	8.721197	13.0	8.0	11.0	10.000000	11.0	14.0	13.000000	11.0
14899	0	9.0	8.721197	13.0	8.0	11.0	10.000000	11.0	14.0	13.000000	11.0
7956	1	9.0	9.000000	9.0	9.0	12.0	10.000000	12.0	11.0	11.000000	9.0
13233	1	9.0	9.000000	9.0	9.0	12.0	10.000000	12.0	11.0	11.000000	9.0
701	0	10.0	8.000000	12.0	10.0	12.0	8.000000	10.0	13.0	10.000000	9.0
9852	0	10.0	8.000000	12.0	10.0	12.0	8.000000	10.0	13.0	10.000000	9.0
368	0	10.0	9.000000	11.0	7.0	15.0	11.000000	11.0	12.0	13.000000	8.0
1609	0	10.0	9.000000	11.0	7.0	15.0	11.000000	11.0	12.0	13.000000	8.0
169	1	11.0	10.000000	7.0	8.0	10.0	10.000000	12.0	11.0	11.000000	8.0
18704	1	11.0	10.000000	7.0	8.0	10.0	10.000000	12.0	11.0	11.000000	8.0
3272	0	12.0	10.000000	6.0	10.0	10.0	12.000000	11.0	10.0	10.868354	13.0
17439	0	12.0	10.000000	6.0	10.0	10.0	12.000000	11.0	10.0	10.868354	13.0

Рисунок 1.3 – Список дублікатів

```
df = df.drop_duplicates(ignore_index=True)
```

```
df.shape
(19990, 11)
```

Визначимо аномальні дані використовуючи метод IsolationForest та видалимо їх:

```
from sklearn.ensemble import IsolationForest

X = df.drop(columns=['label'])
X = IsolationForest(random_state=42).fit_predict(X)

mask = X == -1

df = df.drop(df[mask].index).reset_index(drop=True)
df.shape
(19862, 11)
```

Виконаємо оптимізацію даниих за рахунок пошуку та видалення слабко-інформативних ознак. Для оцінки слабко-інформативних ознак використовується функція get_feature_importance(df, метод). У якості параметру функції method використовуються методи машинного навчання.

Значення, що приймаються:

- Параметр df: pandas.DataFrame – усі наші дані.
- Параметр method: str – метод, який використовується. Можливі значення параметра method: randomforest (RandomForestClassifier), decisiontree (DecisionTreeClassifier), logreg (LinearRegression).

Значення, що повертається – pandas.Series.

Результати оцінки інформативності ознак наведено на рис. 1.4.

```
import matplotlib.pyplot as plt

importances = get_feature_importance(df, 'randomforest')
fig, ax = plt.subplots()
importances.plot.bar(ax=ax)
fig.tight_layout()

# df = df.drop(columns=['column_0','column_3','column_n'])
```

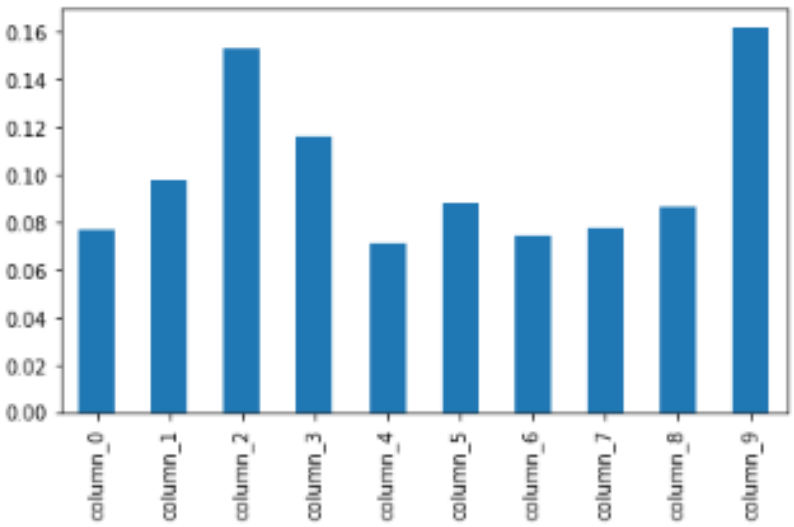

Рисунок 1.4 – Критерії та їх коефіцієнти важливості

Масштабування даних виконується за допомогою функції scale_data. Функція scale_data(X, method) масштабує дані X методом method.

Значення, що приймаються:

- Параметр X: *pandas.DataFrame* – вихідні дані, за винятком залежної змінної label.
- Параметр method: *str* – метод масштабування даних. Можливі значення параметра method:
  - minmax (MinMaxScaler)
  - standard (StandardScaler)
  - norm (Normalizer)
  - robust (RobustScaler)

Значення, що повертається – *ndarray* (масштабовані дані X). Датасет до та після масштабування представлено на рис.1.5 та рис. 1.6 відповідно.

```
df
X = df.drop(columns=['label'])
y = df['label']

X = scale_data(X, 'robust')

df = pd.concat([
    pd.DataFrame(y, columns=['label']),
    pd.DataFrame(X, columns=[f'column_{c}' for c in range(0,X.shape[1])])
], axis=1)

df.shape
```

[10]:

	label	column_0	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9
0	0	10.0	10.0	11.000000	8.000000	10.0	9.0	10.000000	13.0	12.000000	11.0
1	1	12.0	14.0	8.504062	13.000000	12.0	9.0	6.000000	5.0	7.000000	4.0
2	1	10.0	9.0	8.000000	8.755474	13.0	14.0	10.000000	8.0	10.000000	7.0
3	0	9.0	5.0	6.000000	14.000000	9.0	13.0	11.000000	10.0	10.868354	13.0
4	1	8.0	9.0	9.000000	11.000000	11.0	13.0	11.000000	11.0	11.000000	12.0
...	...	...	...	...	...	...	...	...	...	...	...
19857	1	7.0	7.0	10.000000	9.000000	9.0	10.0	13.000000	13.0	11.000000	11.0
19858	0	5.0	9.0	2515.000000	11.000000	10.0	10.0	7.000000	9.0	7.000000	16.0
19859	0	13.0	12.0	5.000000	9.000000	12.0	16.0	10.139866	9.0	15.000000	11.0
19860	0	5.0	6.0	8.504062	16.000000	9.0	11.0	9.000000	9.0	5.000000	14.0
19861	0	10.0	11.0	10.000000	9.000000	9.0	7.0	10.139866	13.0	12.000000	12.0

Рисунок 1.5 – Оброблений датасет до процедури масштабування (перші та останні 5 зразків)

[12]:

	label	column_0	column_1	column_2	column_3	column_4	column_5	column_6	column_7	column_8	column_9
0	0	0.268502	0.333333	0.666667	-0.333333	-0.333333	-0.333333	-0.046622	0.622071	0.500000	0.333333
1	1	0.935169	1.666667	-0.165313	1.333333	0.333333	-0.333333	-1.379955	-2.044595	-2.000000	-2.000000
2	1	0.268502	0.000000	-0.333333	-0.081509	0.666667	1.333333	-0.046622	-1.044595	-0.500000	-1.000000
3	0	-0.064831	-1.333333	-1.000000	1.666667	-0.666667	1.000000	0.286711	-0.377929	-0.065823	1.000000
4	1	-0.398165	0.000000	0.000000	0.666667	0.000000	1.000000	0.286711	-0.044595	0.000000	0.666667
...	...	...	...	...	...	...	...	...	...	...	...
19857	1	-0.731498	-0.666667	0.333333	0.000000	-0.666667	0.000000	0.953378	0.622071	0.000000	0.333333
19858	0	-1.398165	0.000000	835.333333	0.666667	-0.333333	0.000000	-1.046622	-0.711262	-2.000000	2.000000
19859	0	1.268502	1.000000	-1.333333	0.000000	0.333333	2.000000	0.000000	-0.711262	2.000000	0.333333
19860	0	-1.398165	-1.000000	-0.165313	2.333333	-0.666667	0.333333	-0.379955	-0.711262	-3.000000	1.333333
19861	0	0.268502	0.666667	0.333333	0.000000	-0.666667	-1.000000	0.000000	0.622071	0.500000	0.666667

Рисунок 1.6 – Оброблений датасет після процедури масштабування (перші та останні 5 зразків)

Посилання

1. https://ru.education-wiki.com/1053656-data-preprocessing-in-machine-learning
2. https://neurohive.io/ru/tutorial/primer-reshenija-realnoj-zadachi-po-mashinnomu-obucheniju-na-python/
3. https://habr.com/ru/post/511132/
4. https://www.bigdataschool.ru/blog/data-preparation-operations.html
5. https://colab.research.google.com/github/KrishnaswamyLab/SingleCellWorkshop/blob/master/exercises/Preprocessing/notebooks/01_Loading_and_preprocessing_your_own_data.ipynb#scrollTo=cZZOws5li73K

Лабораторна робота № 2
“Метрики якості. Матриця невідповідності. ROC-аналіз”

1 Мета роботи

Дослідження методів оцінки якості класифікації даних. Розробка програмних засобів для оцінки якості побудованих моделей машинного навчання.

2 Теми для попереднього опрацювання

Confusion Matrix.
ROC крива.
Оцінка точності методів класифікації.
Помилки першого та другого роду.

3 Короткі теоретичні відомості

Матриця невідповідності (*confusion matrix*), або матриця помилок (*error matrix*) – це таблиця особливого компонування, що дає можливість унаочнювати продуктивність алгоритму, зазвичай керованого машинним навчання.

Кожен з рядків цієї матриці представляє зразки **прогнозованого** класу, тоді як кожен зі стовпців представляє зразки **фактичного** класу (рис.2.1).

		Прогнозований стан	
Фактичний стан	Загальна сукупність (Total Population) **T=P+N**	Позитивний прогнозований стан (Predicted Condition Positive, **PCP**);	Негативний прогнозований стан (Predicted Condition Negative, **PCN**);
	Позитивна популяція (Positive Population) **P**	істинно позитивний (True positive, **TP**)	хибно негативний, (False negative, **FN)** помилка II роду
	Негативна популяція (Negative Population) **N**	хибно позитивний, (False positive, **FP**), помилка I роду	істинно негативний (True negative, **TN**)

Рисунок 2.1 – Матриця невідповідності

де TP (істинно-позитивний) – кількість вірно класифікованих позитивних подій (*модель сказала так і вгадала*); FP (хибно-позитивний, **помилка I**

роду, false alarm, помилкова тривога) – кількість невірно класифікованих позитивних подій (модель сказала так і помилилася); TN (істинно-позитивний) – кількість вірно класифікованих негативних подій (модель сказала ні і вгадала); FN (хибно-негативний, **помилка II роду**, Miss Target, пропуск цілі) – кількість невірно класифікованих негативних подій *(модель сказала ні і помилилася*) .

Для оцінки якості, використовують такі показники ефективності (key performance indicators, **KPI):**

1. Частка правильних відповідей (Classification Accuracy):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}.$$

2. Точність (чутливість), частка позитивних класів які класифіковані правильно:

$$\text{Precision} = \frac{TP}{TP + FP} = S_e.$$

Мета *Precision* (*Sensitivity,* S_e) – класифікувати всі позитивні екземпляри як позитивні, не допускаючи помилкових визначень негативних об'єктів, як позитивних.

3. Повнота (Recall) або частка позитивних класів, які правильно були прогнозовані:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

4. F1score – це середнє гармонійне значення точності та запам'ятовування. Він враховує як хибно-позитивні, так і хибно-негативні події. Тому він добре працює на незбалансованому наборі даних.

$$F1score = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \cdot (\text{Precision} \cdot \text{Recall})}{(\text{Precision} + \text{Recall})}$$

Точність і повнота характеризують різні сторони якості класифікатора. Чим вище точність, тим менше помилкових спрацьовувань (помилка I роду). Чим вище повнота, тим менше помилкових пропусків (помилка II роду).

Рішення про те, що слід використовувати точність або повноту, залежить від типу проблем.

Приклад 1. Є зображення і необхідно максимально визначити всі автомобілі всередині нього. Оскільки мета полягає в тому, щоб виявити всі автомобілі, необхідно використовувати класифікатор з максимальним значенням **Recall** (повноти), тобто зменшити помилку II роду (пропуск цілі).

Такий підхід може помилково класифікувати деякі об’єкти як автомобілі, але в кінцевому підсумку працює для прогнозування всіх автомобілів.

Приклад 2. Є знімок з результатами мамографії, і необхідно визначити наявність раку. Оскільки мета полягає в тому, щоб не тільки виявити захворювання, але і бути впевненими у відсутності невірної ідентифікації зображень як злоякісного (помилкової тривоги), тому що лікування є надто токсичним для організму людини. Таким чином, бажаним показником у даному випадку є максимальне значення **Precission** (точності).

Приклад 3 . Із 100 пацієнтів 9 пацієнтів мають захворювання. За результати тестування маємо таку матрицю невідповідності (рис.2.2).

		Прогнозований стан	
Фактичний стан	Загальна сукупність =100	Позитивний прогнозований стан	Негативний прогнозований стан
	Позитивний стан=**9**	Істинно-позитивний (True positive, **TP=1**)	Хибно-негативний, (False negative, **FN=8)** помилка II роду
	Негативний стан=**91**	Хибно-позитивний, (False positive, **FP=1**), помилка I роду	Істинно-негативний, (True negative, **TN=90**)

Рисунок 2.2 – Матриця невідповідності для прикладу 1

Тоді: $Precision = \frac{TP}{TP+FP} = \frac{1}{1+1} = 50\%,\ Recall = \frac{TP}{TP+FN} = \frac{1}{1+8} = 11\%,$

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} = \frac{90+1}{1+1+8+90} = 91\%.$$

ROC-крива показує залежність (рис.3.3) *кількості вірно класифікованих позитивних прикладів (TPR)* від *кількості невірно класифікованих позитивних прикладів (FPR).*

Кількість вірно класифікованих позитивних прикладів (TPR) співпадає з Precision або Sensitivity.

$$TPR = \text{Precision} = \frac{TP}{TP+FP}.$$

Кількість невірно класифікованих позитивних прикладів(*FPR*) може бути визначена як

$$FPR=1-Recall.$$

Також кількість невірно класифікованих позитивних прикладів може бути визначена як

$$FPR=1- S_p$$

де Sp – Специфічність (Specificity) або частка істинно негативних випадків, які були правильно ідентифіковані моделлю:

$$S_p = \frac{TN}{TN + FP}$$

У термінології ROC-аналізу передбачається, що у класифікатора є деякий параметр, міняючи який, ми будемо отримувати різне розбиття на два класи. Цей параметр часто називають порогом, або точкою відсікання (англ. **cut-off** value, treshold). Залежно від нього будуть виходити різні величини помилок I і II роду. При збільшенні обсягу вибірок ROC-криві, побудовані за вибірками, будуть сходитися до теоретичної кривої (побудованої для розподілів).

ROC-крива будується наступним чином:

Для кожного значення порога відсікання, яке змінюється від 0 до 1 з кроком d_x (наприклад, 0.01) розраховуються значення кількості вірно класифікованих позитивних прикладів (TPR) та кількості невірно класифікованих позитивних прикладів специфічності (FPR). Як альтернатива порогом може бути кожне наступне значення прикладу у вибірці. Будується графік залежності: по осі Y відкладається чутливість TPR, по осі X – FPR.

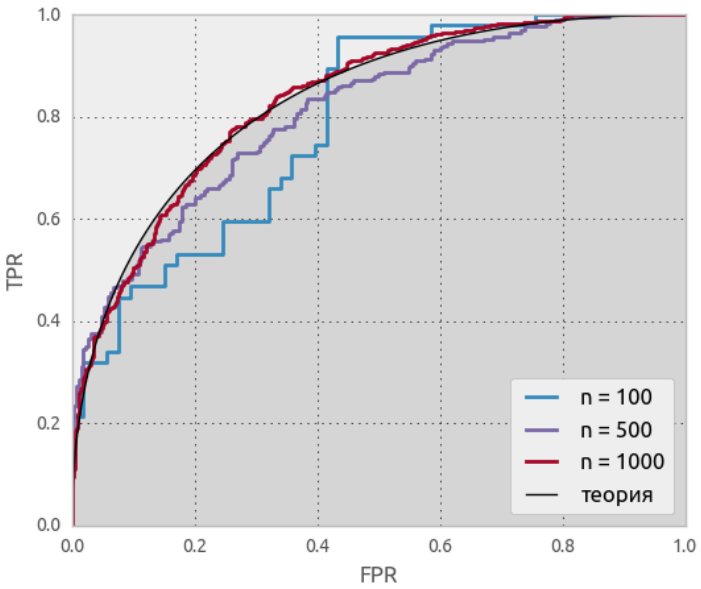

Рисунок 2.3 – Приклад ROC-кривої

Якість класифікації визначається площею під ROC-кривою (Табл.2.1).

Таблиця 2.1 – Оцінка якості класифікації

Інтервал	AUC
0,9–1,0	Відмінна
0,8–0,9	Дуже добра
0,7–0,8	Добра
0,6–0,7	Середня
0,5–0,6	Незадовільна

Для ідеального класифікатора графік ROC-кривої проходить через верхній лівий кут, де частка істинно позитивних випадків становить 100 %, або 1,0 (ідеальна чутливість), а частка хибно-позитивних прикладів дорівнює нулю. Тому чим ближче крива до верхнього лівого кута, тим вища попереджувальна здатність моделі. Навпаки, чим менше вигин кривої і чим ближче вона розташована до діагональної прямої, тим менш ефективна модель. Діагональна лінія відповідає «марному» класифікатору, тобто повній нерозрізненості двох класів.

4 Порядок виконання індивідуального завдання

Залежно від номера прізвища за списком вибрати індивідуальне завдання (Табл.2). Розробити два програмні модулі:

- Програму генерації псевдовипадкових даних.
- Програму оцінки якості класифікатору.

Програмний модуль генерації псевдовипадкових даних повинен генерувати таблицю розміру Nx2, де N – кількість вихідних даних(об’єктів або вибірок). Така таблиця буде імітувати результат роботи класифікатору. Перший стовпець таблиці – це результат класифікації на основі побудованої моделі, другий стовпець – це очікуваний результат. Перший стовпець повинен мати псевдовипадкові значення в діапазоні [a, b], які вказані в індивідуальному завданні (**табл.1**). Другий стовпець повинен мати псевдовипадкові значення 0 або 1 с заданою частотою появи «1» (**табл.1**). Вихідний формат даних – це файл з типу ***.csv**.

Програмний модуль оцінки якості побудованої моделі машинного навчання має обов’язковий параметр – порогове значення (θ), завдяки якому виконується правило, яке приймає рішення. За формулою:

$$Y(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

На основі отриманих вихідних даних побудувати матрицю невідповідності .

На основі матриці невідповідності знайти значення наступних характеристик:

- True Positive (TP);
- False Positive (FP);
- False Negative (FN);
- True Negative (TN);
- Condition Positive (CP);
- Condition Negative (CN);
- Predicted Condition Positive (PCP);Predicted Condition Negative (PCN);
- Total Population (T);
- Accuracy (ACC);
- Precision;
- Recall;
- F_1 score.

Знайти порогове значення **cut-off** (*значення, при якому точка на ROC-графіку буде найближче до точки { FPR= 0; TPR=1}*). Побудувати графік ROC кривої.

Знайти площу під кривою ROC – Area Under Curve (AUC). Метод обчислення площі задано в табл. 2.2. Зробити висновки, щодо якості моделі.

Таблиця 2.2 – Індивідуальне завдання

№ % 32	Частота появи «1»	N	a	b	K	Метод обчислення площі під ROC кривою
1	2	3	4	5	6	7
0	54	4946	0	100	100	Метод правих прямокутників
1	47	3629	100	200	1000	Метод лівих прямокутників
2	53	3575	200	300	10	Метод середніх прямокутників
3	51	3473	300	400	100	Метод трапецій
4	46	4797	400	500	1000	Метод парабол (Метод Сімпсона)
5	44	4481	500	600	10	Метод правих прямокутників
6	57	3936	600	700	100	Метод лівих прямокутників
7	40	3721	700	800	1000	Метод середніх прямокутників
8	52	3201	800	900	10	Метод трапецій
9	58	3127	900	1000	100	Метод парабол (Метод Сімпсона)
10	60	4850	1000	1100	1000	Метод правих прямокутників
11	55	3461	1100	1200	10	Метод лівих прямокутників
12	44	3815	1200	1300	100	Метод середніх прямокутників
13	44	3728	1300	1400	1000	Метод трапецій
14	45	4429	1400	1500	10	Метод парабол (Метод Сімпсона)
15	57	4386	1500	1600	100	Метод правих прямокутників

1	2	3	4	5	6	7
16	44	4601	1600	1700	1000	Метод лівих прямокутників
17	41	3599	1700	1800	10	Метод середніх прямокутників
18	57	4167	1800	1900	100	Метод трапецій
19	40	3238	1900	2000	1000	Метод парабол (Метод Сімпсона)
20	59	3332	2000	2100	10	Метод правих прямокутників
21	53	3483	2100	2200	100	Метод лівих прямокутників
22	43	4508	2200	2300	1000	Метод середніх прямокутників
23	50	4130	2300	2400	10	Метод трапецій
24	51	4464	2400	2500	100	Метод парабол (Метод Сімпсона)
25	43	4640	2500	2600	1000	Метод правих прямокутників
26	55	4336	2600	2700	10	Метод лівих прямокутників
27	45	3269	2700	2800	100	Метод середніх прямокутників
28	57	4701	2800	2900	1000	Метод трапецій
29	40	3830	2900	3000	10	Метод парабол (Метод Сімпсона)
30	58	3911	3000	3100	100	Метод правих прямокутників
31	54	3515	3100	3200	1000	Метод лівих прямокутників

де № – номер студента за списком у журналі групи, % – операція знаходження залишку від цілочислового ділення.

5 Зміст звіту

– Титульний лист.
– Тема роботи.
– Мета роботи.
– Завдання.
– Алгоритм програми (текстовий та/або графічний вигляд).
– Текст програми.
– Результати виконання роботи програми.
– Висновки.

6. Програмна реалізація оцінки якості класифікації

6.1. Бібліотеки та методи

Бібліотеки, що використовуються:

- pandas==1.2.4
- numpy==1.22.2
- sklearn==1.0.2
- matplotlib==3.4.0

Методи sklearn рекомендовані до розгляду:

- sklearn.metrics.auc
- sklearn.metrics.roc_curve

- sklearn.metrics.confusion_matrix
- sklearn.metrics.accuracy_score
- sklearn.metrics.classification_report
- sklearn.model_selection.train_test_split

Розробимо необхідні функції для розрахунків метрик якості навчання методів машинного навчання:

```
random_seed = 42

import random
random.seed(random_seed)
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import ConfusionMatrixDisplay, auc,
classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler



def generate_dataset(M, one_class_freq, feature_range):

    X, y = make_classification(
        n_samples=M,
        n_features=1,
        n_classes=2,
        n_informative=1,
        n_redundant=0,
        n_clusters_per_class=1,
        weights=[1 - one_class_freq],
        random_state=random_seed,
    )

    scaler = MinMaxScaler(feature_range=feature_range)
    X = scaler.fit_transform(X)

    for idx, x in np.ndenumerate(X):
        i = idx[0]
        j = idx[1]
        X[i][j] = np.int16(x)

    return X, y

def roc_curve_from_scratch(y_true, y_prob):

    fpr = []
    tpr = []
    thresholds = np.unique(np.sort(np.append(y_prob, 2)))

    for threshold in thresholds:

        y_pred = np.where(y_prob >= threshold, 1, 0)

        fp = np.sum((y_pred == 1) & (y_true == 0))
        tp = np.sum((y_pred == 1) & (y_true == 1))
```

```
            fn = np.sum((y_pred == 0) & (y_true == 1))
            tn = np.sum((y_pred == 0) & (y_true == 0))

            fpr.append(fp / (fp + tn))
            tpr.append(tp / (tp + fn))

    return np.array(fpr), np.array(tpr), np.array(thresholds)


def draw_roc_curve(fpr, tpr, imax, model_name):
    plt.scatter(fpr[imax], tpr[imax], marker="o", color="black",
label="Best")
    plt.plot([0, 1], [0, 1], linestyle="--", label="No Skill")
    plt.plot(fpr, tpr, marker=".", label=model_name)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()


def calc(y_test, y_pred):
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    print(f"True Negative = {tn}")
    print(f"False Positive = {fp}")
    print(f"False Negative = {fn}")
    print(f"True Positive = {tp}")
    print("=" * 30)

    accuracy = (tp + tn) / (tp + tn + fp + fn)
    print(f"Accuracy = {accuracy}")

    precision = tp / (tp + fp)
    print(f"Precision = {precision}")

    recall = tp / (tp + fn)
    print(f"Recall = {recall}")

    f1_score = 2 * ((recall * precision) / (recall + precision))
    print(f"F1_score = {f1_score}")
    print("=" * 30)

    condition_positive = tp + fn
    print(f"Condition Positive = {condition_positive}")

    condition_negative = fp + tn
    print(f"Condition Negative = {condition_negative}")

    predicted_condition_positive = tp + fp
    print(f"Predicted Condition Positive = {predicted_condition_positive}")

    predicted_condition_negative = fn + tn
    print(f"Predicted Condition Negative = {predicted_condition_negative}")
    print("=" * 30)

    total = tn + fp + fn + tp
    print(f"Total = {total}")
```

6.2 Генерація даних, навчання на базовому методі та оцінка якості.

Функція generate_dataset(M, one_class_freq, feature_range). Генерує датасет із двома колонками. Одна колонка – незалежна змінна X, інша – залежна змінна y. Датасет, що генерується, є датасетом бінарної класифікації.

Значення, що приймаються:

- Параметр M: int – Кількість рядків (примірників).
- Параметр one_class_freq: float – Частота появи класу 1. Набуває значення від 0 до 1.
- Параметр feature_range : tuple – діапазон значень змінної X.
- Значення, що повертається - * tuple * (X, y).

```
X, y = generate_dataset(M=1000, one_class_freq=0.9, feature_range=(1, 100))

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=random_seed, stratify=y
)
```

Навчаємо лінійну модель класифікації на основі методу лінійної регресії (Logistic regression).

```
model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)
y_prob = y_prob[:, 1]
```

Виконуємо побудову ROC-кривої використовуючи функцію roc_curve_from_scratch(y_test, y_prob). На основі порогів ймовірностей thresholds підраховує FPR та TPR.

Значення, що приймаються:

- Параметр y_test: *list* | *ndarray* – тестові передбачення.
- Параметр y_prob: *list* | *ndarray* - імовірнісні прогнози моделі для класу 1.

Значення, що повертаються – * tuple * (fpr, tpr, thresholds).

Результат побудови представлено на рис. 2.4.

```
fpr, tpr, thresholds = roc_curve_from_scratch(y_test, y_prob)
gmeans = np.sqrt(tpr * (1 - fpr))
imax = np.argmax(gmeans)
print(f"Best Threshold = {thresholds[imax]}")

auc_score = auc(fpr, tpr)
print(f"AUC = {auc_score}")
```

```
Best Threshold = 0.8289311538746801
AUC = 0.9765617433414044
```

```
draw_roc_curve(fpr, tpr, imax, "Log.reg.")
```

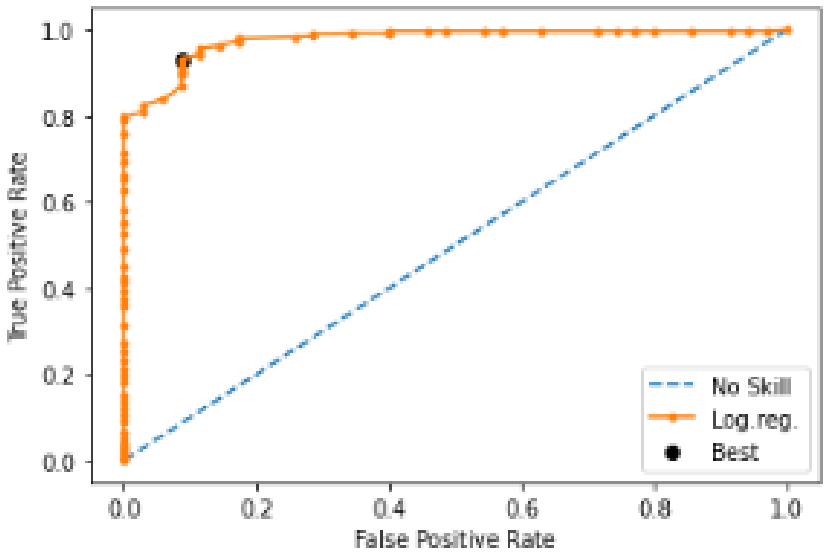

Рисунок 2.4 – ROC-крива для логістичної регресії

```
# робимо підрахунки за завданням, використовуючи тільки TN, FP, FN, TP
calc(y_test, y_pred)
True Negative = 26
False Positive = 9
False Negative = 5
True Positive = 290
==============================
Accuracy = 0.9575757575757575
Precision = 0.9698996655518395
Recall = 0.9830508474576272
F1_score = 0.9764309764309764
==============================
Condition Positive = 295
Condition Negative = 35
Predicted Condition Positive = 299
Predicted Condition Negative = 31
==============================
Total = 330
```

```
# Можна скористатися утилітою sklearn та порівняти отримані результати
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.84	0.74	0.79	35
1	0.97	0.98	0.98	295
accuracy			0.96	330
macro avg	0.90	0.86	0.88	330
weighted avg	0.96	0.96	0.96	330

Для отрмиання презентативного зображення можна використовувати такі утиліти (рис. 2.5)

```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot();
```

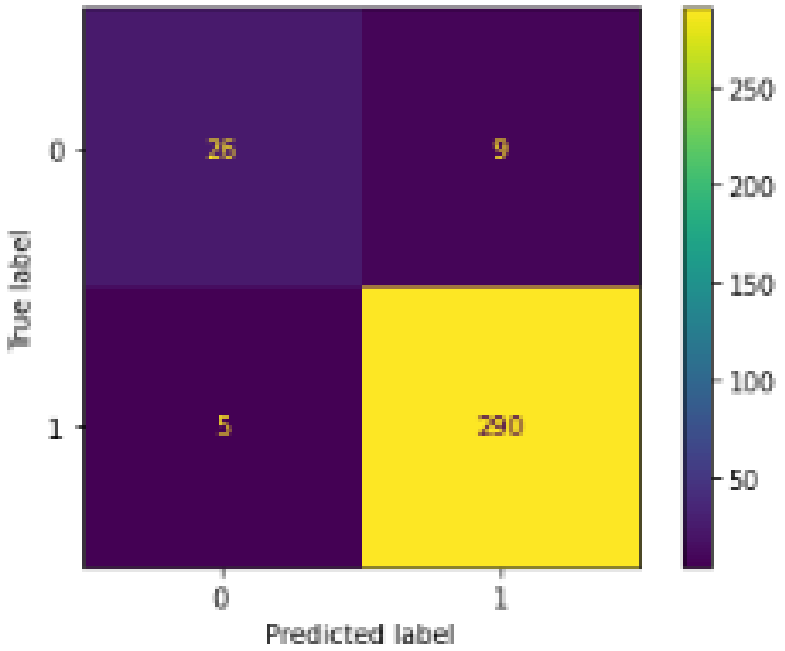

Рисунок 2.5 – Матриця невідповідності начальної моделі

Посилання

1. https://www.bigdataschool.ru/blog/machine-learning-confusion-matrix.html https://www.youtube.com/watch?v=CYy0TZ6OIDw

2. https://dyakonov.org/2017/07/28/auc-roc-%D0%BF%D0%BB%D0%BE%D1%89%D0%B0%D0%B4%D1%8C-%D0%BF%D0%BE%D0%B4-%D0%BA%D1%80%D0%B8%D0%B2%D0%BE%D0%B9-%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA/

3. https://loginom.ru/blog/logistic-regression-roc-auc

Лабораторна робота № 3
«Метод просторової кластеризації для даних з шумами DBSCAN»

1 Мета роботи

Дослідження методу просторової кластеризації для даних з шумами (*Density-based spatial clustering of applications with noise)*.

2 Теми для попереднього опрацювання

DBSCAN.

Щільність та ймовірність.

Формули відстані.

3 Короткі теоретичні відомості

Метод просторової кластеризації для даних з шумами (*Density-based spatial clustering of applications with noise,* DBSCAN) – це алгоритм кластеризації даних, який групує разом точки, які щільно розташовані, позначаючи як викиди точки, які знаходяться самотньо в областях з малою щільністю. DBSCAN потребує двох гіперпараметрів: *minPts* – мінімальне число об'єктів в кластері та ε – радіус гіперсфери або кластерна відстань.

Ідеальне значення *minPts* залежить від мети аналізу. Воно повинно відповідати мінімальному розміру, який буде вважатися істотним кластером. Збільшення мінімального числа об'єктів в кластері може привести до злиття деяких невеликих кластерів.

Кластерна відстань ε для кожної точки – це максимальна відстань між сусідніми точками кластеру.

Існує безліч досліджень підбору оптимальних гіперпараметрів алгоритму DBSCAN (ε та MinPts). Апріорі значення ε та MinPts, пов’язують з пороговою щільністю найменш щільного кластеру. Для автоматичного визначення ε запропоновано підхід на основі розрахунку всіх попарних відстаней між об'єктами. Застосовується також алгоритм диференціальної еволюції.

3.1 Алгоритм методу DBSCAN

Алгоритм методу DBSCAN є наступним:

1) Задається ε – радіус гіперсфери або кластерна відстань та *minPts* – мінімальне число об'єктів в кластері.

2) Береться довільний ще не оброблений об'єкт. Для нього перевіряється умова, що в ε-околі точки є деякий мінімум *j*-об'єктів де $j \geq minPts$, відстань до яких: $d(x_i, y_i) \leq \varepsilon$. Якщо це не так, то ця точка є аномальною або шумом і ті самі дії повторюються для наступної точки.

3) Якщо умова 2 виконується, то точка помічається як та, що належить кластеру. Це так звана коренева точка або центр кластеру. Точки, що знаходяться навколо неї, заносяться в окрему категорію.

4) Кожна не оброблена точка з цієї категорії спочатку помічається як та, що належить кластеру та перевіряється умова, що в ε-околі точки є деякий мінімум j-об'єктів де $j \geq$ minPts, відстань до яких: $d(x_i, y_i) < \varepsilon$. Якщо це так, то ця точка заносяться до цього ж кластеру. У разі якщо у нас кількість точок менше ніж *minPts*, але більше 1 (і при цьому один з сусідів належить до кластеру) точка є крайовою, інакше точка не належить кластеру і є необробленою. Пункт 4 повторюємо до тих пір, поки не проаналізуємо усі необроблені точки.

5) Якщо кількість кластерів не змінюється, то закінчуємо процес кластеризації, інакше повертаємося до кроку 2.

6) Точки, які не увійшли до жодного кластеру є шумами або аномаліями.

4 Порядок виконання індивідуального завдання

Залежно від номера прізвища за списком вибрати індивідуальне завдання (Табл.1). Розробити два програмні модулі:

- Програмний модуль генерації псевдовипадкових даних.
- Програмний модуль просторової кластеризації даних з шумами на основі методу DBSCAN.

Програмний модуль генерації псевдовипадкових даних повинен генерувати таблицю розміру Nx2, де N – кількість вихідних даних(об’єктів або вибірок), [a,b] –діапазон даних (табл.3.1.). Вихідний формат даних – ***.csv**.

Програмний модуль DBSCAN реалізує метод просторової кластеризації даних з шумами з урахуванням гіперпараметрів *minPts* –та ε, наведених в Табл.1. Функція знаходження відстані від центру кластеру до об’єкту *distfunc* реалізована за різними алгоритмами (Табл.1).

В межах цієї лабораторної необхідно продемонструвати:

Результат кластеризації даних методом DBSCAN з урахуванням заданих параметрів ε, *minPts* та *distfunc*. Визначити кількість аномальних даних.

Результат кластеризації даних методом DBSCAN за умови налаштування параметрів ε, *minPts* таким чином, щоб кількість аномалій даних була меншою або дорівнювала заданому значенню W, яке наведено в табл.1. Наприклад, якщо маємо 100 об’єктів та відповідно до завдання N=5%, то потрібно знайти такі параметри налаштування ε та minPts, щоб за результатами роботи модель DBSCAN ідентифікувала 5 об’єктів як аномальні.

Таблиця 3.1 – Індивідуальне завдання

№ % 32	M	N	a	b	ε	minPts	DistFunc	W – відсоток шуму в даних
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	4	4946	0	100	1	3	Euclidean distance	1%
1	7	3629	100	200	5	4	Taxicab geometry	3%
2	3	3575	200	300	10	5	Euclidean distance	5%
3	1	3473	300	400	1	6	Taxicab geometry	10%
4	6	4797	400	500	5	7	Euclidean distance	15%
5	4	4481	500	600	10	3	Taxicab geometry	20%
6	7	3936	600	700	1	4	Euclidean distance	25%
7	10	3721	700	800	5	5	Taxicab geometry	30%
8	2	3201	800	900	10	6	Euclidean distance	1%
9	8	3127	900	1000	1	7	Taxicab geometry	3%
10	10	4850	1000	1100	5	3	Euclidean distance	5%
11	5	3461	1100	1200	10	4	Taxicab geometry	10%
12	4	3815	1200	1300	1	5	Euclidean distance	15%
13	4	3728	1300	1400	5	6	Taxicab geometry	20%
14	5	4429	1400	1500	10	7	Euclidean distance	25%
15	7	4386	1500	1600	1	3	Taxicab geometry	30%
16	4	4601	1600	1700	5	4	Euclidean distance	1%
17	1	3599	1700	1800	10	5	Taxicab geometry	3%
18	7	4167	1800	1900	1	6	Euclidean distance	5%
19	10	3238	1900	2000	5	7	Taxicab geometry	10%
20	9	3332	2000	2100	10	3	Euclidean distance	15%
21	3	3483	2100	2200	1	4	Taxicab geometry	20%
22	3	4508	2200	2300	5	5	Euclidean distance	25%
23	10	4130	2300	2400	10	6	Taxicab geometry	30%
24	1	4464	2400	2500	1	7	Euclidean distance	1%
25	3	4640	2500	2600	5	3	Taxicab geometry	3%
26	5	4336	2600	2700	10	4	Euclidean distance	5%
27	5	3269	2700	2800	1	5	Taxicab geometry	10%
28	7	4701	2800	2900	5	6	Euclidean distance	15%
29	0	3830	2900	3000	10	7	Taxicab geometry	20%
30	8	3911	3000	3100	1	3	Euclidean distance	25%
31	4	3515	3100	3200	5	4	Taxicab geometry	30%

де № – номер студента за списком у журналі групи, % – операція знаходження залишку від цілочислового ділення.

5 Зміст звіту

- Титульний лист.
- Тема роботи.
- Мета роботи.
- Завдання.
- Алгоритм програми (текстовий та/або графічний вигляд).
- Текст програми.
- Результати виконання роботи програми.
- Висновки.

6 Програмна реалізація алгоритму DBSCAN

6.1 Бібліотеки та методи

Бібліотеки, що використовуються:

- pandas==1.2.4
- numpy==1.22.2
- sklearn==1.0.2
- scipy==1.7.3
- matplotlib==3.4.0

Методи, що рекомендуються до розгляду:

- matplotlib.axes.Axes.scatter
- matplotlib.axes.Axes.fill
- matplotlib.pyplot.subplots
- sklearn.cluster.DBSCAN

Розробимо необхідні функції для лабораторної роботи.

```
random_seed = 42

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from scipy import interpolate
from scipy.spatial import ConvexHull


def plot(df, y_column, ax, colors):
    for i in df[y_column].unique():
        if i == -1:
            dff = df[df[y_column]==i]
            ax.scatter(dff['x1'], dff['x2'], c='r', alpha = 0.9, s=30)
            continue

        # get the convex hull
        points = df[df[y_column] == i][['x1', 'x2']].values
        hull = ConvexHull(points)
```

```
        x_hull = np.append(points[hull.vertices,0],
                           points[hull.vertices,0][0])
        y_hull = np.append(points[hull.vertices,1],
                           points[hull.vertices,1][0])

        # interpolate
        dist = np.sqrt((x_hull[:-1] - x_hull[1:])**2 + (y_hull[:-1] -
y_hull[1:])**2)
        dist_along = np.concatenate(([0], dist.cumsum()))
        spline, u = interpolate.splprep([x_hull, y_hull],
                                        u=dist_along, s=0)
        interp_d = np.linspace(dist_along[0], dist_along[-1], 50)
        interp_x, interp_y = interpolate.splev(interp_d, spline)
        # plot shape
        try:
            ax.fill(interp_x, interp_y, '--', c=colors[i], alpha=0.25)
            dff = df[df[y_column]==i]
            ax.scatter(dff['x1'], dff['x2'], c=colors[i], alpha = 0.7,
s=60,edgecolors='k')
        except IndexError:
            raise IndexError('Add more colors to colors list.')
```

6.2. Тестування розробленого модуля

Генеруємо датасет для кластеризації за допомогою утиліти make_bolbs, яка є спрощеною версією make_classification. Основне завдання методу – генерація заданої кількості кластерів з нормальним розподілом.

```
X, y_true = make_blobs(
    n_samples=400,
    n_features=2,
    centers=4,
    center_box=(0, 9),
    random_state=random_seed
)
```

Для зручності створимо таблицю типу pandas.DataFrame.

```
df = pd.DataFrame(
    data={
        'x1':X[:,0],
        'x2':X[:,1],
        'y_true':y_true
    }
)
```

Запускаємо кластеризацію DBSCAN. Найголовніші параметри алгоритму:

- eps – Максимальна відстань між двома точками, при якій одна вважається сусідньою з іншою.
- min_samples – Мінімальна кількість сусідніх точок на околиці точки, при якому точка не вважається шумом.

- metric – Спосіб за яким вважається розточення між точками.
- Бібліотека sklearn надає 6 методів:
  - cityblock
  - cosine
  - euclidean
  - l1
  - l2
  - manhattan

```
df['y_pred'] = DBSCAN(eps=0.5, min_samples=5,
metric='euclidean').fit_predict(df[['x1','x2']])
```

У наведеному датафреймі df змінні y_true та y_pred означають відповідно очікуване та передбачене значення залежної змінної (рис. 3.1.).

```
df.head()
```

	x1	x2	y_true	y_pred
0	-0.757677	9.550379	3	-1
1	0.751116	9.378331	0	0
2	0.277009	7.522862	3	0
3	7.563065	5.240869	1	1
4	7.158836	6.523492	1	1

Рисунок 3.1 – Перші 5 точок датасету після кластеризації

Зробимо візуалізацію отриманих результатів (див. розділ 7):

```
fig, (ax1, ax2) = plt.subplots(1,2,figsize=(18, 7))
colors =
['blue','green','cyan','magenta','yellow','orange','lime','grey','indigo','
gold']


ax1.set_title(f'Expected clusters [{len(df["y_true"].unique())} clusters]',
fontweight="bold", fontsize=20)
plot(df, 'y_true', ax1, colors)

ax2.set_title(f'Predicted clusters [{len(df["y_pred"].unique()) - 1}
clusters]', fontweight="bold", fontsize=20)
      plot(df, 'y_pred', ax2, colors)
```

7. Результати виконання лабораторної роботи

На рис. 3.1. представлено кластери, які було задано до кластеризації. На рис. 3.3 представлено кластери, що було побудовано за допомогою алгоритму DBSCAN.

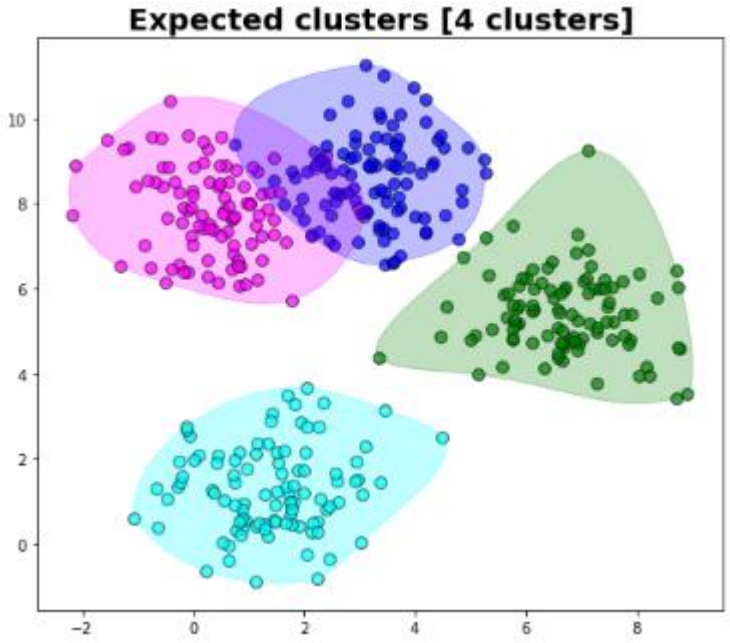

Рисунок 3.2 – Очікувані кластери

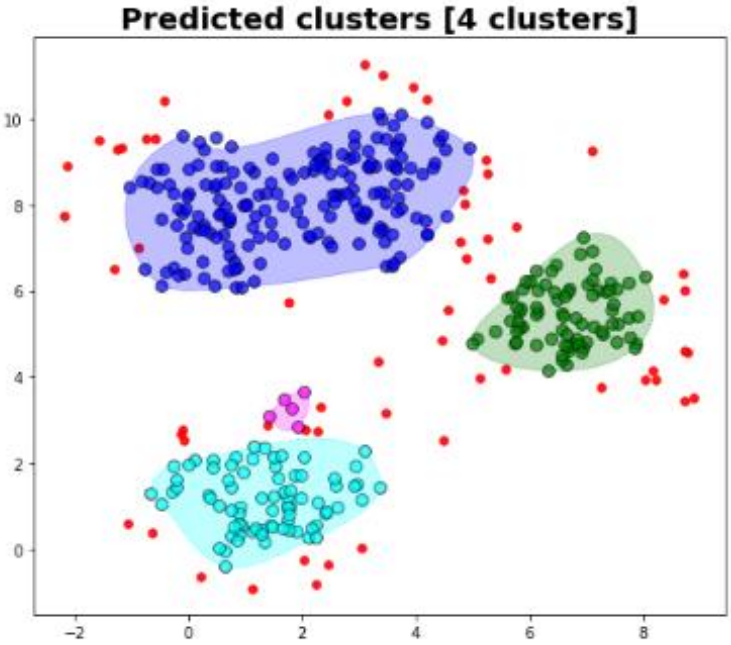

Рисунок 3.3 – Кластерізація DBSCAN (Eps=0.5, minPts=5, distFunc=’Euclidian’)

Посилання

1. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

2. https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html

Лабораторна робота № 4
“Розробка класифікатору на основі алгоритму One Class SVM ”

1 Мета роботи

Дослідження методів класифікації даних. Отримання навичок аналізу даних на прикладі класифікатору на основі алгоритму One Class SVM.

2 Теми для попереднього опрацювання

Maximal Margin Classifiers.
Soft Margin Classifiers.
Support Vector Machines.
Kernel Trick.
The Polynomial Kernel Function.
One Class SVM.

3 Короткі теоретичні відомості

Класифікація – це набір методів які розділяють об'єкти за заздалегідь відомою ознакою на основі аналізу значень атрибутів (ознак). Класифікувати об'єкт означає, вказати номер (або найменування класу), до якого належить даний об'єкт. Для класифікації завжди потрібен вчитель – розмічені дані з ознаками і категоріями, на яких буде проводитися навчання.

Для вирішення задачі класифікації використовують метричні, лінійні та нелінійні методи, лінійну регресія, ймовірнісні та логічні методи.

Метричні методи – намагаються знайти в даних точки, в деякому сенсі ізольовані від інших, використовують різні метрики близькості об'єктів в просторі ознак. До таких методів відносяться методи: найближчого сусіда (Nearest Neighbor, NN); k–найближчого сусіда (k-Nearest Neighbor, kNN); *k*-зваженого найближчого сусіди (k-weighted nearest neighbors , KWNN).

Лінійні методи шукають рішення у вигляді лінійної комбінації ознак.

До таких методів відносяться: лінійна регресія, лінійний SVM (Support Vector Machine) та ін.

До *нелінійних методів* відноситься: логістична регресія, нелінійний SVM, нейронні мережі.

Ймовірнісні методи класифікації використовують методи теорії статистичних рішень. Найбільш популярними є класифікатор Байеса.

Логічні методи використовують логічні операції І, АБО, над предикатами. До логічних методів відносяться дерева рішень та їх ансамблі. Найбільш популярними є класифікатори на основі мета-алгоритмів бустингу, беггінку.

В рамках лабораторної роботи досліджено класифікатор на основі, методу опорних векторів (SVM) який є одним з найбільш потужних методів класифікації.

Метод опорних векторів (SVM, support vector machines) – група алгоритмів класифікації, заснованих на навчанні з учителем, які використовують лінійний поділ об'єктів у просторі ознак за допомогою гіперплощини. Метод застосовується для вирішення завдання бінарної класифікації. Основною проблемою методу є вибір оптимальної гіперплощини, що дозволяє розділити класи з максимальною точністю. Для цього розділяюча гіперплощина повинна бути обрана таким чином, щоб відстань між найближчими точками, розташованими по різні боки від неї, була б максимальною. Ця відстань називається зазором (margin), а самі точки – опорними векторами (support vectors). Тоді розділяюча гіперплощина повинна бути обрана таким чином, щоб максимізувати зазор, що забезпечить більш впевнений поділ класів.

Модель SVM є наступною. Для заданих навчаючих векторів $x_i \in \Re^d$ i=1,..., *n*, і вектора $y \in \{1, -1\}^n$, кожен компонент якого вказує клас {1 або -1} до якого належить точка x_i, розділяючі гіперплощини можуть бути описані рівнянням:

$$w \cdot x - b = 0$$

де *w* – вектор нормалі до цієї гіперплощини, параметр $\frac{b}{\|w\|}$ визначає зсув гіперплощини від початку координат вздовж вектора нормалі *w*. (рис. 4.1)

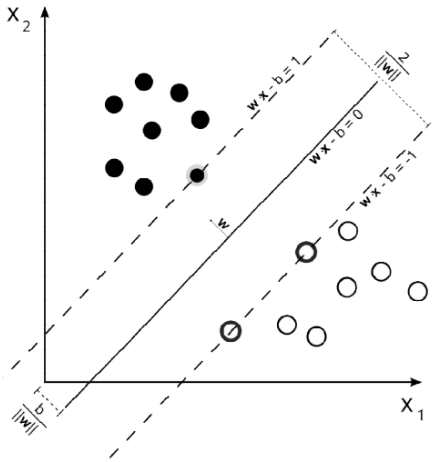

Рисунок 4.1 – модель SVM

Якщо дані вибірки є лінійно роздільні, то паралельні гіперплощини які розділяють дані таким чином, що відстань між ними є якомога більшою описуються рівняннями:

$$w \cdot x - b = 1,$$
$$w \cdot x - b = -1$$

З геометричної точки зору відстань між цими двома гіперплощинами є $\frac{2}{\|w\|}$, де w – перпендикуляр до роздільної гіперплощини. Отже, для максимізації відстані між ними, потрібно мінімізувати $\|w\|$.

Додавши наступні нерівності ми обмежуємо попадання точок даних до розділення:

$$w \cdot x - b \leq -1, \text{ якщо } y_i = 1$$
$$w \cdot x - b \geq -1, \text{ якщо } y_i = 1.$$

Ці обмеженні стверджують, що кожна точка повинна лежати з «правильного» боку розділення. Останні вирази можна переписати так:

$$y(w \cdot x - b) \geq -1, \text{ для всіх } \; 1 \leq i \leq n$$

і в загальному вигляді оптимізаційна задача матиме вигляд:

Мінімізувати $\|w\|$ за умови $y(w \cdot x - b) \geq -1$, для всіх $\; 1 \leq i \leq n$.

Таким чином, w та b визначають завдання класифікації так: $x \to \operatorname{sgn}(w \cdot x - b)$

Важливим наслідком наведеної геометричної інтерпретації є те, що максимально розділова гіперплощина повністю визначається тими векторами x, які лежать найближче до неї. Ці вектори і називають **опорними векторами**.

В рамках лабораторної роботи досліджено класифікатор на основі алгоритму One Class SVM. Для навчання алгоритму достатньо мати дані та один максимально далекий зразок за відстанню, що використовує алгоритм. Алгоритм будує модель, яка буде видаляти заданий відсоток даних, що знаходяться близько до заданого зразку, використовуючи опорний вектор.

4 Порядок виконання індивідуального завдання

Залежно від номера прізвища за списком вибрати індивідуальне завдання (Табл.1). Розробити два програмні модулі:

– Програмний модуль генерації псевдовипадкових даних.

– Програмний модуль класифікатора на основі методу One Class SVM для виявлення аномальних об’єктів.

– Виконати оцінку точності класифікації за допомогою **ROC**-аналізу. Для оцінки якості класифікації використати програмне забезпечення, розроблене в лабораторній роботі №2.

Програмний модуль генерації псевдовипадкових даних повинен генерувати таблицю розміру Nx2, де N – кількість вхідних даних, [a, b] – діапазон даних (табл.4.1.). Вихідний формат даних – ***.csv**.

Програмний модуль класифікатора реалізує метод класифікації One Class SVM з урахуванням параметру *X*, який задає процентне співвідношення аномалій або шуму в даних та координати Origin Point, наведених в Табл.1. Координати Origin point– це координати нового об'єкту, відносно до якого виконується ідентифікація на приналежність до класу нормальних чи аномальних об'єктів. Координати визначаються наступним чином:

- Up-Left. Знайти мінімальне значення за першим стовбцем та максимальне значення за другим. Відняти від мінімального 1000 та додати до максимального 1000. Результуюча пара точок – координати Origin point.
- Up-Right. Знайти максимальне значення за першим стовбцем та максимальне значення за другим. Додати до максимальних значень 1000. Результуюча пара точок – координати Origin point.
- Down-Left. Знайти мінімальне значення за першим стовбцем та мінімальне значення за другим. Відняти від мінімальних значень 1000. Результуюча пара точок – координати Origin point.
- Down-Right. Знайти максимальне значення за першим стовбцем та мінімальне значення за другим. Відняти від мінімального 1000 та додати до максимального 1000. Результуюча пара точок – координати Origin point.

Створена програмна модель повинна зберігатись у файлі та ідентифікувати новий об'єкт, координати якого задаються двома числами Origin Point.

Таблиця 4.1 – Індивідуальне завдання

№ % 32	N	a	b	Origin point	Кількість шуму.
1	**2**	**3**	**4**	**5**	**6**
0	4946	0	100	Up-Left	1%
1	3629	100	200	Up-Right	2%
2	3575	200	300	Down-Left	3%
3	3473	300	400	Down-Right	4%
4	4797	400	500	Up-Left	5%
5	4481	500	600	Up-Right	6%
6	3936	600	700	Down-Left	7%
7	3721	700	800	Down-Right	8%
8	3201	800	900	Up-Left	9%
9	3127	900	1000	Up-Right	10%
10	4850	1000	1100	Down-Left	1%
11	3461	1100	1200	Down-Right	2%
12	3815	1200	1300	Up-Left	3%
13	3728	1300	1400	Up-Right	4%
14	4429	1400	1500	Down-Left	5%
15	4386	1500	1600	Down-Right	6%
16	4601	1600	1700	Up-Left	7%

1	2	3	4	5	6
17	3599	1700	1800	Up-Right	8%
18	4167	1800	1900	Down-Left	9%
19	3238	1900	2000	Down-Right	10%
20	3332	2000	2100	Up-Left	1%
21	3483	2100	2200	Up-Right	2%
22	4508	2200	2300	Down-Left	3%
23	4130	2300	2400	Down-Right	4%
24	4464	2400	2500	Up-Left	5%
25	4640	2500	2600	Up-Right	6%
26	4336	2600	2700	Down-Left	7%
27	3269	2700	2800	Down-Right	8%
28	4701	2800	2900	Up-Left	9%
29	3830	2900	3000	Up-Right	10%
30	3911	3000	3100	Up-Left	1%
31	3515	3100	3200	Up-Right	2%

де № – номер студента за списком у журналі групи, % – операція знаходження залишку від цілочислового ділення.

5 Зміст звіту до лабораторної роботи

- Титульний лист.
- Тема роботи.
- Мета роботи.
- Завдання.
- Алгоритм програми (текстовий та/або графічний вигляд).
- Текст програми.
- **Оцінка результатів побудованої моделі (ROC-аналіз)**
- Результати виконання роботи програми.
- Висновки.

6 Програмна реалізація методуOne-class SVM

6.1 Бібліотеки та методи

Бібліотеки, що використовуються:

- pandas==1.2.4
- numpy==1.22.2
- sklearn==1.0.2
- matplotlib==3.4.0

Опишемо функції для роботи з методами опорних векторів:

```
random_seed = 42
import random
random.seed(random_seed)
import pandas as pd
import numpy as np
import itertools
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.svm import OneClassSVM


def one_class_svm_research(df, kernels, nus, outlier_index=0):
    params = [{'kernel':p[0],'nu':p[1]} for p in
list(itertools.product(kernels, nus))]

    fig, all_ax = plt.subplots(len(kernels),len(nus),figsize=(len(nus)*6,
len(kernels)*5))

    param_info_list = []
    outlier_percentage_list =  []
    point_detected_list = []
    for p, ax in zip(params, all_ax.flatten()):
        df['y_pred'] = OneClassSVM(kernel=p['kernel'],
nu=p['nu']).fit_predict(df[['x1','x2']])

        param_info = f"({p['kernel']}, {p['nu']})"

        try:
            outlier_percentage =
df['y_pred'].value_counts(normalize=True).iloc[-1]
        except:
            outlier_percentage = 0

        point_detected = df.iloc[outlier_index]['y_pred'] == -1

        df.plot(x='x1',
y='x2',c='y_pred',kind='scatter',colormap='PiYG',s=60, alpha=0.7, ax=ax,
title=param_info)

        param_info_list.append(param_info)
        outlier_percentage_list .append(outlier_percentage)
        point_detected_list.append(point_detected)

    result = pd.DataFrame(data={
        'params':param_info_list,
        '%_outliers':outlier_percentage_list,
        'point_detected':point_detected_list
    })

          return result
```

6.2 Тестування розробленого модуля

Генеруємо датасет для кластеризації за допомогою утиліти make_bolbs, яка є спрощеною версією make_classification. Основне завдання методу – генерація заданої кількості кластерів з нормальним розподілом.

```
X, y_true = make_blobs(
    n_samples=500,
    n_features=2,
```

```
    cluster_std=1,
    centers=1,
    center_box=(0, 20),
    random_state=random_seed
)
```

Штучно додаємо "точку-шум". Нехай це буде точка із індексом 0.

```
outlier_index = 0

max_x1 = X[:,0].max() + 10
max_x2 = X[:,1].max() + 10

X[outlier_index][0] = max_x1
X[outlier_index][1] = max_x2
y_true[outlier_index] = -1
```

Для зручності створимо таблицю типу pandas.DataFrame. Потім візуалізуємо таблицю. (рис. 4.3)

```
df = pd.DataFrame(
    data={
        'x1':X[:,0],
        'x2':X[:,1],
        'y_true':y_true
    }
)

df.plot(x='x1',
y='x2',c='y_true',kind='scatter',colormap='PiYG',s=50,alpha=0.7,
figsize=(8,6));
```

Функція one_class_svm_research(df, kernels, nus, outlier_index=0). Перебирає всілякі параметри kernel та nu методу OneClassSVM. Підраховує та візуалізує результати дослідження.

Значення, що приймаються:

- Параметр df: *pandas.DataFrame* – вихідні дані.
- Параметр kernels : *list* –досліджувані значення параметра *kernel* моделі OneClassSVM.
- Параметр nus : *list* – досліджувані значення параметра *nus* моделі OneClassSVM.
- Параметр outlier_index : *int, default=0* – індекс штучно створеної "точки-шуму".

Значення, що повертається – *pandas.DataFrame*.

Створимо таблицю типу pandas.DataFrame візуалізуємо її (рис. 4.4).

```
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
nus = [0.9, 0.5, 0.1, 0.01, 0.005, 0.0001]

result = one_class_svm_research(df, kernels, nus)
```

Позначення колонок:

- params - параметри, серед яких потрібно вибрати потрібний
- %_outliers - відсоток аномалій/шумів від загальної кількості даних
- point_detected - чи змогла модель виявити нашу штучно створену "точку-шум"

```
result
```

[8]:

	params	%_outliers	point_detected
0	(linear, 0.9)	0.100	False
1	(linear, 0.5)	0.500	False
2	(linear, 0.1)	0.100	False
3	(linear, 0.01)	0.008	False
4	(linear, 0.005)	0.006	False
5	(linear, 0.0001)	1.000	False
6	(poly, 0.9)	0.102	False
7	(poly, 0.5)	0.500	False
8	(poly, 0.1)	0.100	False
9	(poly, 0.01)	0.010	False
10	(poly, 0.005)	0.006	False
11	(poly, 0.0001)	0.002	False
12	(rbf, 0.9)	0.100	True
13	(rbf, 0.5)	0.500	True
14	(rbf, 0.1)	0.100	True
15	(rbf, 0.01)	0.010	True
16	(rbf, 0.005)	0.004	True
17	(rbf, 0.0001)	0.006	False
18	(sigmoid, 0.9)	0.126	False
19	(sigmoid, 0.5)	0.478	False
20	(sigmoid, 0.1)	0.086	False

Рисунок 4.2 – Результати побудованої моделі OneClass SVM з різними функціями ядр та їх параметрами

7. Результати виконання лабораторної роботи.

На рис. 4.3 представлено графічну візуалізацію моделі OneClass SVM з точкою Origin Point, яка розташовано у верхньому-правому куту.

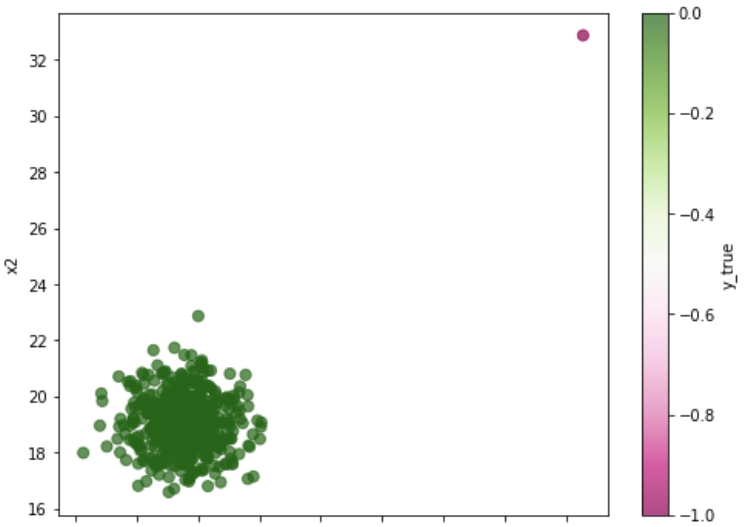

Рисунок 4.3 – Візуалізація таблиці з псевдовипадковими даними.

На рис. 4.4 зображено результати побудови OneClass SVM моделей з різними показниками шумів. (Функція ядра – лінійна)

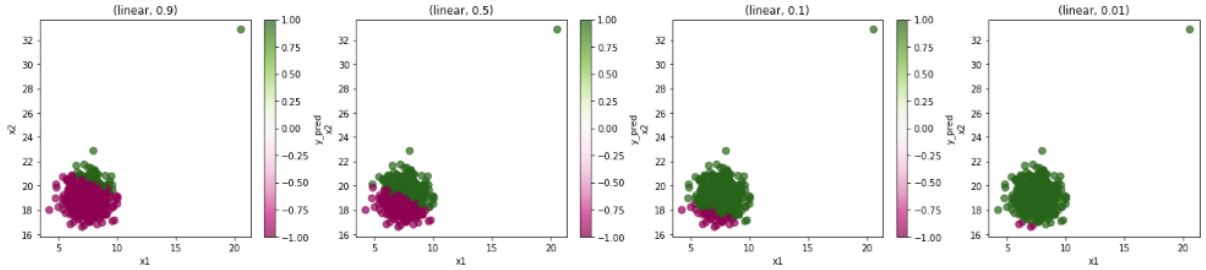

Рисунок 4.4 – Моделі з 10%, 50%, 90% та 99% шумів відповідно.

Посилання

1. https://habr.com/ru/post/428503/
2. https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

Лабораторна робота № 5
“Розробка класифікатору даних на основі дерева прийняття рішення”

1 Мета роботи

Дослідження методів класифікації даних. Отримання навичок аналізу даних на прикладі класифікатору на основі Дерев прийняття рішень (Decision tree).

2 Теми для попереднього опрацювання

Попередня обробка даних.
Аномалії та новизна у даних.
Регресія (лінійна та бінарна).
Оцінка точності методів класифікації та виявлення аномалій.

3 Короткі теоретичні відомості

Дерева прийняття рішень – це спосіб представлення правил в ієрархічній структурі, де кожному об'єкту відповідає єдиний вузол, що дає результуюче рішення (рис.5.1). Під правилом розуміється логічна конструкція, представлена у вигляді «якщо ... то ...» . Дерево має:

– Кореневий вузол (самий верхній вузол дерева).

– Лист, листовий або термінальний вузол(вузол, який не має дочірніх елементів).

– Внутрішній вузол (будь-який вузол дерева, що має нащадків, і таким чином, не є листовим вузлом).

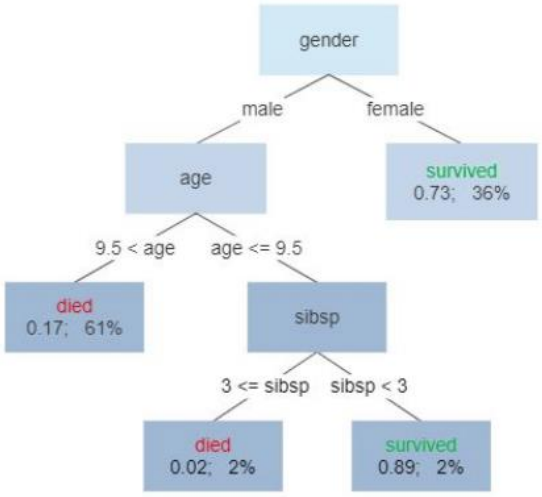

Рисунок 5.1– Дерево прийняття рішення

Алгоритми побудови дерев рішень розрізняються наступними характеристиками:

• Вид розгалуження - бінарний (binary), множинний (multi-way) майже не використовується.

• Критерії розгалуження або критерій прийняття рішення – ентропія (теоретично-інформаційний критерій), Gini (статистичний), ін.

Інформаційна ентропія визначається як

$$H = -\sum_{i=1}^{p} \frac{N_i}{N} \log\left(\frac{N_i}{N}\right),$$

де n – число класів у вихідному підмножині, Ni – число прикладів і–го ступеня, N – загальне число прикладів в підмножині.

Процедура побудови дерева рішень наведена на рис. 5.2.

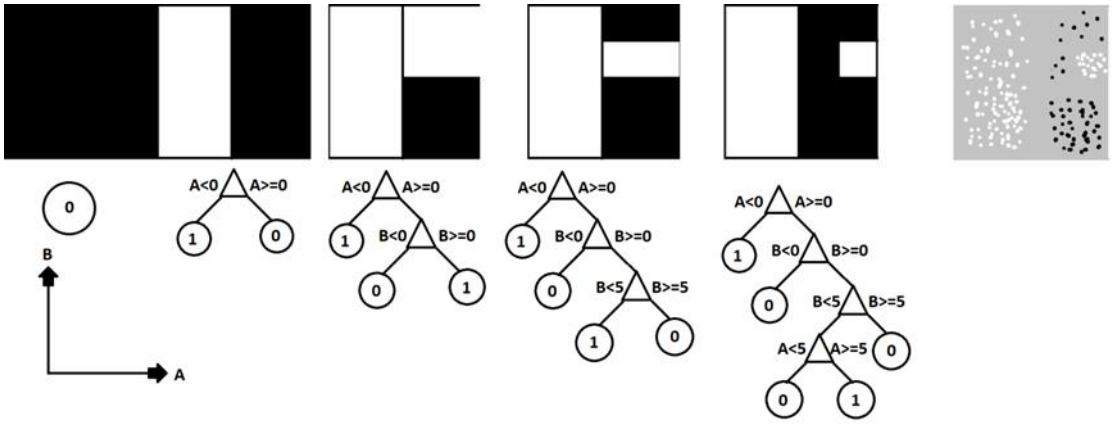

Рисунок 5.2 – Процедура побудови дерева рішень

Побудову дерева рішень розглянемо на наступному прикладі. Будемо передбачати колір кульки за її координатою. У якості критерію прийняття рішення використаємо значення ентропії .

Маємо 9 синіх кульок і 11 жовтих (рис.5.3). Якщо ми навмання витягли кульку, то вона з ймовірністю $p_1 = 9/20$ буде синьою і з ймовірністю $p_2 = 11/20$ – жовтою. Значить, ентропія стану $S_0 = -9/20 \log_2 (9/20) - 11/20 \log_2(11/20) \approx 1$. Саме це значення поки ні про що нам не говорить. Тепер подивимося, як зміниться ентропія, якщо розбити кульки на дві групи - з координатою $X \leq 12$ або $X > 12$.

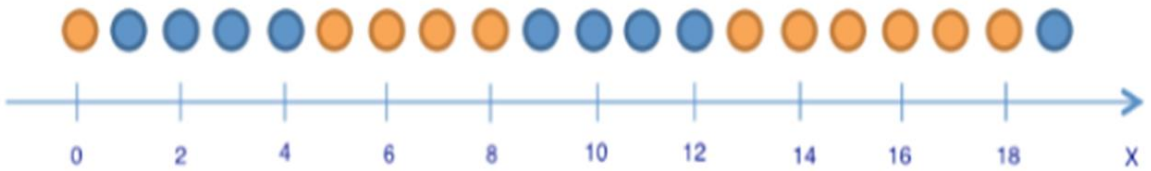

Рисунок 5.3– Вихідні дані прикладу

У лівій групі виявилося 13 куль, з яких 8 синіх і 5 жовтих (рис.5.4). Ентропія цієї групи дорівнює S1=-5/13 $\log_2$ (5/13) - 8/13 $\log_2$ (8/13) ≈0.96. У правій групі виявилося 7 куль, з яких 1 синя і 6 жовтих. Ентропія правої групи дорівнює S2=-1/7$\log_2$ (1/7)- 6/7 $\log_2$ (6/7) ≈ 0.6. Як бачимо, ентропія зменшилася в обох групах у порівнянні з початковим станом, хоч в лівій групі і не сильно.

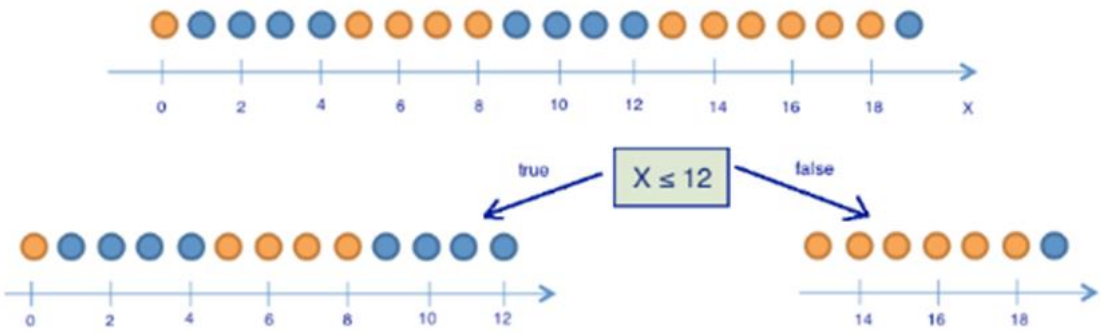

Рисунок 5.4– Приклад побудови дерева рішень

Оскільки ентропія – це ступінь хаосу (або невизначеності) в системі, зменшення ентропії є приростом інформації. Формально приріст інформації (*information gain, IG*) при розбитті вибірки за ознакою Q (в нашому прикладі це ознака x≤12) визначається як:

$$IG(Q) = S_0 - \sum_{i=1}^{q} \frac{N_i}{N} S_i,$$

де q – число груп після розбиття, N_i – число елементів вибірки, у яких ознака Q має і–те значення. У нашому випадку після поділу вийшло дві групи (q = 2) – одна з 13 елементів (N_1=13), друга – з 7 (N_2=7). **Приріст інформації** дорівнює:

$$IG(x \le 12) = S_0 - \frac{12}{20} S_1 - \frac{7}{20} S_2 = 1 - \frac{13}{20} \cdot 0,98 - \frac{7}{20} \cdot 0,6 \approx 0,16$$

Виходить, розділивши кульки на дві групи за ознакою “координата менше або дорівнює 12”, ми вже отримали більш впорядковану систему, ніж на початку. Продовжимо розподіл кульок на групи до тих пір, поки в кожній групі кульки не будуть одного кольору.

Для правої групи було потрібно всього одне додаткове розбиття за ознакою “координата менше або дорівнює 18”, для лівої – ще три. Очевидно, ентропія групи з кульками одного кольору дорівнює 0 $\log_2$1 = 0, що відповідає уявленню, що група кульок одного кольору – впорядкована.У підсумку ми побудували дерево рішень, яке пророкує колір кульки за його

координатою. Відзначимо, що таке дерево рішень може погано працювати для нових об'єктів (визначення кольору нових кульок), оскільки воно ідеально налаштоване під навчальну вибірку (початкові 20 кульок). Для класифікації нових кульок краще буде дерево з меншим числом “питань”, або вузлів, нехай навіть воно і не ідеально розбиває за кольорами навчальну вибірку. Це є проблемою перенавчання.

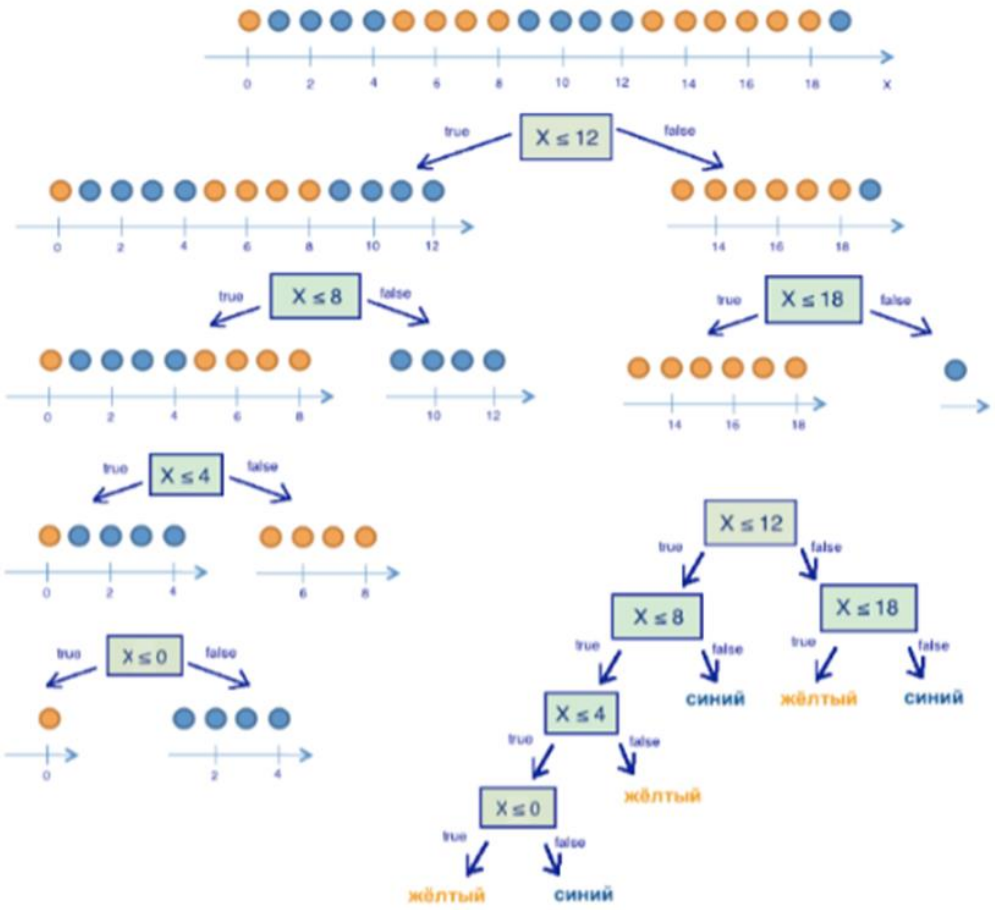

Рисунок 5.5 – Побудовано дерево рішень

4 Порядок виконання індивідуального завдання

Залежно від номера прізвища за списком вибрати індивідуальне завдання (Табл.5.1). Розробити два програмні модулі:

– Програмний модуль генерації псевдовипадкових даних.

– Програмний модуль класифікатора на основі дерева прийняття рішень.

– Виконати оцінку точності класифікації за допомогою **ROC**-аналізу. Для оцінки якості класифікації використати програмне забезпечення, розроблене в лабораторній роботі №2.

Програмний модуль генерації псевдовипадкових даних повинен генерувати таблицю розміру NxL, де N – кількість вихідних даних, [a, b] – діапазон даних), L=5 та відображає кількість ознак або атрибутів об'єкту(табл.1). Також необхідно згенерувати стовбець Y, $Y \in [0,C]$ який розмічає або класифікує вихідні дані відповідно до кількості класів C (табл.1). Вихідний формат даних – ***.csv**.

Програмний модуль класифікатора реалізує метод класифікації на основі дерева прийняття рішень з урахуванням параметру *D*, який задає правило зупинки побудови дерева прийняття рішень. Побудована програмна модель класифікатора на основі дерева прийняття рішень повинна зберігатись у файлі та ідентифікувати новий об'єкт, який задається випадковими ознаками.

Таблиця 5.1 – Індивідуальне завдання

№ % 32	N	A	b	C – Кількість класів	D– Правило зупинки
1	**2**	**3**	**4**	**5**	**6**
0	3456	3100	3200	3	Задана точність
1	7231	3000	3100	4	Максимальна кількість розгалужень
2	2231	2900	3000	5	Максимальна глибина (кількість вузлів за вертикаллю)
3	9503	2800	2900	6	Задана точність
4	3151	2700	2800	3	Максимальна кількість розгалужень
5	3533	2600	2700	4	Максимальна глибина (кількість вузлів за вертикаллю)
6	9938	2500	2600	5	Задана точність
7	3653	2400	2500	6	Максимальна кількість розгалужень
8	6435	2300	2400	3	Максимальна глибина (кількість вузлів за вертикаллю)
9	2345	2200	2300	4	Задана точність
10	9634	2100	2200	5	Максимальна кількість розгалужень
11	8543	2000	2100	6	Максимальна глибина (кількість вузлів за вертикаллю)
12	7634	1900	2000	3	Задана точність
13	3456	1800	1900	4	Максимальна кількість розгалужень
14	8345	1700	1800	5	Максимальна глибина (кількість вузлів за вертикаллю)
15	2423	1600	1700	6	Задана точність
16	7754	1500	1600	3	Максимальна кількість розгалужень

1	2	3	4	5	6
17	9923	1400	1500	4	Максимальна глибина (кількість вузлів за вертикаллю)
18	5435	1300	1400	5	Задана точність
19	4495	1200	1300	6	Максимальна кількість розгалужень
20	7685	1100	1200	3	Максимальна глибина (кількість вузлів за вертикаллю)
21	7758	1000	1100	4	Задана точність
22	8743	900	1000	5	Максимальна кількість розгалужень
23	8675	800	900	6	Максимальна глибина (кількість вузлів за вертикаллю)
24	9536	700	800	3	Задана точність
25	4798	600	700	4	Максимальна кількість розгалужень
26	9576	500	600	5	Максимальна глибина (кількість вузлів за вертикаллю)
27	8675	400	500	6	Задана точність
28	8344	300	400	3	Максимальна кількість розгалужень
29	7655	200	300	4	Максимальна глибина (кількість вузлів за вертикаллю)
30	8799	100	200	5	Задана точність
31	7588	0	100	6	Максимальна кількість розгалужень

де № - номер студента за списком у журналі групи .

5 Зміст звіту

- Титульний лист.
- Тема роботи.
- Мета роботи.
- Завдання.
- Алгоритм програми (текстовий та/або графічний вигляд).
- Текст програми.
- Оцінка результатів побудованої моделі (ROC-аналіз)
- Результати виконання роботи програми.
- Висновки.

6 Програмна реалізація методу побудови дерева рішень

6.1. Бібліотеки таметоди

Виконувані бібліотеки:

- pandas==1.2.4

- numpy==1.22.2
- sklearn==1.0.2
- matplotlib==3.4.0

Методи sklearn рекомендовані до розгляду:

- sklearn.tree.DecisionTreeClassifier

Реалізуаємо необхідні функції для генераці даних та побудови дерева рішень.

```
random_seed = 42

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from sklearn import tree
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report


def plot(X, X_test, y_pred, model, title, figsize=(8,6), cmap_name =
'Set1'):
    n_classes = len(np.unique(y_pred))

    colors = matplotlib.cm.get_cmap(cmap_name).colors
    if n_classes > len(colors):
        print(f'Max classes for visualizing = {len(colors)}.')
        return

    colors = colors[:n_classes]
    cmap = matplotlib.colors.ListedColormap(colors)
    fig, ax = plt.subplots(1,figsize=figsize)

    x1_min, x1_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    x2_min, x2_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

    x1, x2 = np.meshgrid(np.arange(x1_min, x1_max, 0.01), np.arange(x2_min,
x2_max, 0.01))

    Z = model.predict(np.c_[x1.ravel(), x2.ravel()])
    Z = Z.reshape(x1.shape)
    ax.contourf(x1, x2, Z, alpha=0.4, cmap=cmap)

    for i in range(n_classes):
        idx_points = np.where(y_pred == i)
        ax.scatter(X_test[idx_points, 0], X_test[idx_points, 1], s=40,
color=colors[i], edgecolors='k')

    ax.set_title(title, fontweight="bold", fontsize=20)
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
```

```
    legends = [Line2D([0],[0], marker='o', markeredgewidth=1,
markeredgecolor='k', linestyle='', color=c, label=i) for c, i in
zip(colors, range(n_classes))]
          ax.legend(handles=legends)
```

6.2 Генерація даних та перевірка працездатності розробленого метода

Створюємо датасет:

```
X, y = make_blobs(
    n_samples=1300,
    n_features=2,
    centers=3,
    center_box=(0, 9),
    random_state=random_seed
)
```

Навчаємо дерево рішень на розпізнавання 8 класів. Отримаємо розділений на сегменти простір. Така мапа може виступати заміною дерева рішень для прогназування результатів дерева рішень.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=random_seed, stratify=y
)
model = DecisionTreeClassifier(random_state=random_seed).fit(X_train,
y_train)
y_pred = model.predict(X_test)
plot(X, X_test, y_pred, model, 'Decision Tree')
```

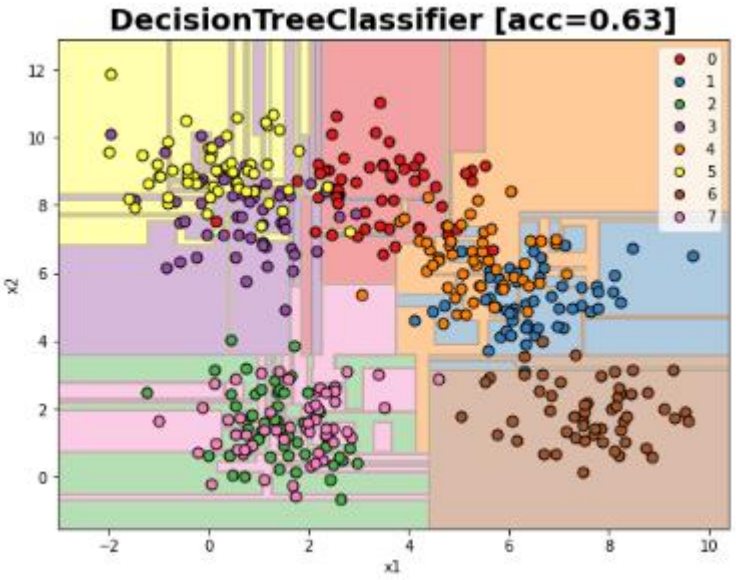

Рисунок 5.6 – Простір з сегментами рішень

Візуалізуємо дерево рішень (рис. 5.7)

```
fig = plt.figure(figsize=(15,10))
tree.plot_tree(
    model,
    feature_names=['x1','x2'],
    class_names=[str(y) for y in np.unique(y_test)],
    filled=True
);
```

```
# fig.savefig("decistion_tree.png", facecolor='white')
```

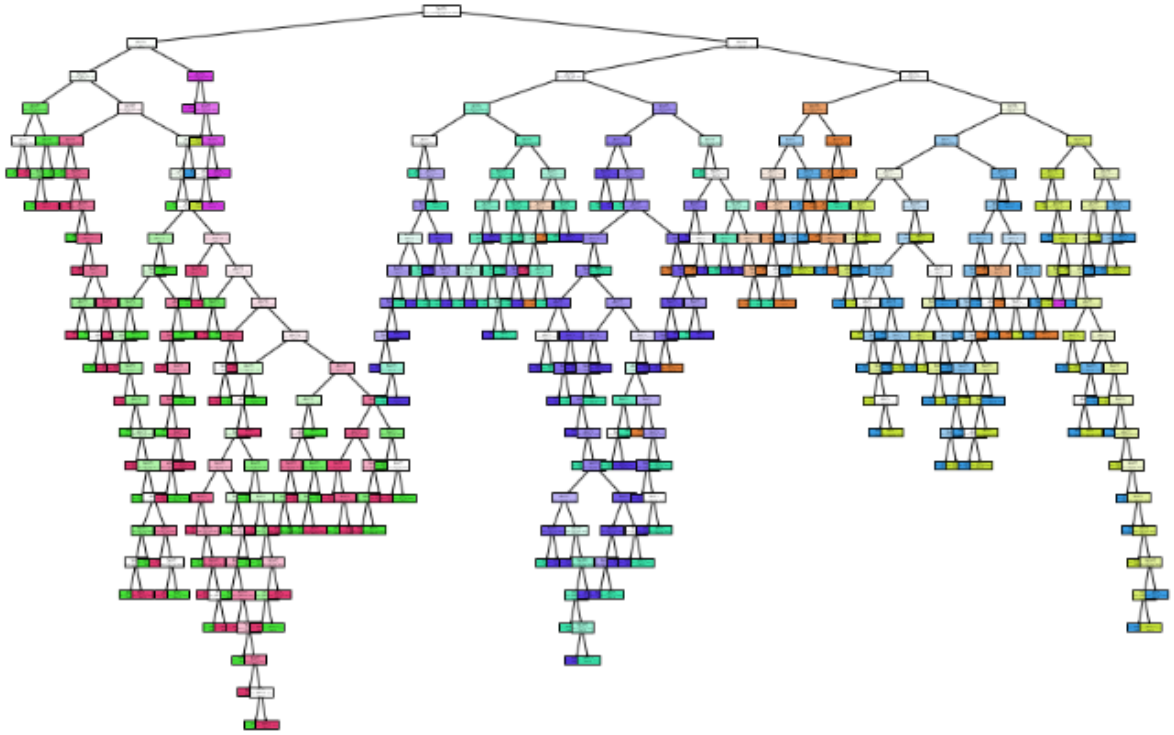

Рисунок 5.7 – Дерево рішень

Порівняємо результати побудованого дерева рішень з очікуваними результатами.

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.76	0.69	0.72	54
1	0.65	0.69	0.67	54
2	0.51	0.44	0.48	54
3	0.53	0.54	0.53	54
4	0.55	0.58	0.57	53
5	0.64	0.64	0.64	53
6	0.96	0.94	0.95	54
7	0.49	0.55	0.52	53
accuracy			0.63	429
macro avg	0.64	0.63	0.63	429
weighted avg	0.64	0.63	0.63	429

Рисунок 5.8 – Багатовимірна матриця невідповідності

Посилання

1. https://ranalytics.github.io/data-mining/043-Decision-Trees.html
2. https://elib.bsu.by/bitstream/123456789/7693/4/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8

%D0%B8%20%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%B8.pdf
3. https://loginom.ru/blog/decision-tree-p1
4. https://ami.nstu.ru/~vms/lecture/data_mining/trees.htm
5. http://math.nsc.ru/AP/datamine/decisiontree.htm

Лабораторна робота № 6
“Розробка класифікатору даних на основі ансамблевих мета-алгоритмів”

1 Мета роботи

Дослідження методів класифікації даних. Отримання навичок аналізу даних на прикладі побудови класифікатору на основі ансамблевих мета-алгоритмів.

2 Теми для попереднього опрацювання

Методи класифікації
Дерева прийняття рішення
Ансамблеві методи класифікації: *Bagging*, *Boosting*, *Stacking*
Оцінка точності методів класифікації та виявлення аномалій

3 Короткі теоретичні відомості

Ансамблеві методи – це парадигма машинного навчання, де кілька моделей навчаються для вирішення однієї і тієї ж проблеми і об'єднуються для отримання кращих результатів.

У загальному випадку ансамбль записується як:

$$a(x) = b(b_1(x),...,b_n(x))$$

де b - деякий алгоритм (тобто функція), яку прийнято називати мета-алгоритмом (*meta-estimator*).

Типи мета-алгоритмів, які спрямовані на об'єднання слабких учнів:

- Беггінг або бутстреп.
- Бустінг.
- Стекінг.

3.1 Алгоритм беггінгу

Беггінг (*bagging or Bootstrap aggregating*) – це метаалгоритм класифікації, що використовує композиції класифікаторів, кожен з яких навчається на різних підвибірках даних незалежно і паралельно використовуючи один і той же алгоритм навчання. Найбільш популярними є мета-алгоритми: *Pasting Ensemble*, *Bootstrap Ensemble*, *Random Subspace Ensemble* та *Random Patches Ensemble* та *Random Forest*. Кожний із цих алгоритмів відрізняється формуванням вибірок для побудови класифікаторів.

3.1.1 Алгоритм Склеювання (*Pasting*)

Алгоритм Склеювання (*Pasting*) формує вибірки таким чином, що вони є унікальними і не повторюються. Нехай маємо набір даних з 10 записами. Спочатку ми розділяємо набір даних на навчальні та тестові: 6 записів (*N*) для навчання та 4 записи для тестування (рис.6.1).

	Income (X1)	Age (X2)	Outstanding Debt (X3)	Credit Card Limit (X4)	Loan Accepted (Yes/No) (Y Variable)
Training Dataset	245	26	416	5	0
	505	39	226	50	0
	453	67	357	15	0
	189	47	251	45	0
	564	26	131	55	1
	803	41	188	20	0
Testing Dataset	149	50	305	65	1
	965	35	207	50	1
	725	69	344	60	1
	836	51	210	40	1

Рисунок 6.1 – Приклад поділу даних на навчальну та тестову вибірки

Надалі із навчальних даних випадковим чином формуються вибірки для навчання класифікатора. Кількість вибірок складає близько 60% від розміру навчальних даних (рис.6.2).

Bag 1				Bag 2				Bag 3				Bag 4				Bag 5			
X1	X2	X3	X4	X1	X2	X3	X4	X1	X2	X3	X4	X1	X2	X3	X4	X1	X2	X3	X4
245	26	416	5	564	26	131	55	453	67	357	15	505	39	226	50	189	47	251	45
453	67	357	15	505	39	226	50	803	41	188	20	453	67	357	15	505	39	226	50
189	47	251	45	803	41	188	20	245	26	416	5	245	26	416	5	564	26	131	55

Рисунок 6.2 – Приклад формування вибірок за алгоритмом склеювання (*Pasting*)

3.1.2 Алгоритм *Bootstrap aggregating*

Алгоритм *Bootstrap aggregating* формує вибірки також випадковим чином, але вони не є унікальними, тобто можуть повторюватись (рис.6.3).

Bag 1				Bag 2				Bag 3				Bag 4				Bag 5			
X1	X2	X3	X4	X1	X2	X3	X4	X1	X2	X3	X4	X1	X2	X3	X4	X1	X2	X3	X4
245	26	416	5	564	26	131	55	453	67	357	15	453	67	357	15	189	47	251	45
453	67	357	15	505	39	226	50	803	41	188	20	453	67	357	15	505	39	226	50
245	26	416	5	564	26	131	55	803	41	188	20	245	26	416	5	189	47	251	45

Рисунок 6.3 – Приклад формування вибірок за алгоритмом *Bootstrap aggregating*

3.1.3 Алгоритм *Random Subspace*

Алгоритм *Random Subspace* формує вибірки шляхом випадкового вибору ознак, а не випадкового вибору спостережень. Спостереження (вибірки) вибираються випадковим чином, не є унікальними і можуть повторюватися. (рис.6.4).

Bag 1			Bag 2			Bag 3			Bag 4		
X1	X2	X4	X2	X3	X4	X1	X2	X3	X1	X3	X4
245	26	5	26	416	5	245	26	416	245	416	5
505	39	50	39	226	50	505	39	226	505	226	50
453	67	15	67	357	15	453	67	357	453	357	15
189	47	45	47	251	45	189	47	251	189	251	45
564	26	55	26	131	55	564	26	131	564	131	55
803	41	20	41	188	20	803	41	188	803	188	20

Рисунок 6.4 – Приклад формування вибірок за алгоритмом *Random Subspace*

3.1.4 Алгоритм *Random Patches*

Алгоритм *Random Patches* формує вибірки шляхом випадкового вибору як ознак, так і спостережень (вибірок) які не повторюються (рис.6.5).

Bag 1			Bag 2			Bag 3			Bag 4		
X1	X2	X4	X2	X3	X4	X1	X2	X3	X1	X3	X4
245	26	5	47	251	45	505	39	226	803	188	20
505	39	50	26	131	55	803	41	188	245	416	5
453	67	15	41	188	20	453	67	357	26	131	55

Рисунок 6.5 – Приклад формування вибірок за алгоритмом *Random Patches*

Алгоритм *Random Forest* буде розглянуто в лаботаторній роботі №7.

3.1.5 Алгоритм навчання класифікатора

Алгоритм беггінгу може бути сформовано наступним чином:

1.Із множини вихідних даних сформувати вибірки для навчання класифікатора, використовуючи різні мета-алгоритми.

2. На основі кожної вибірки побудувати класифікатор (наприклад, на основі дерева прийняття рішень), який навчається за одним і тим же алгоритмом.

3. Виконати налаштування класифікатора. Наприклад, для дерева прийняття рішень налаштування класифікатора відбувається шляхом підбору максимальної кількості ознак, що використовуються при побудові дерева; мінімальної кількості розгалужень при побудові дерева; мінімальної кількості листків та максимальної глибини.

4. Визначити оптимальну кількість дерев ансамблю.

5. Виконати класифікацію. Класифікація об'єктів проводиться шляхом голосування: кожен класифікатор комітету відносить об'єкт, який класифікується до одного з класів, а клас, який отримує більшість голосів, є відповіддю моделі ансамблю (це називається мажоритарних голосуванням):

$$S_L(.) = \arg_k \max[card(l \mid w_l(.) = k)]$$

–

де $S_L(.)$ – проста більшість голосів завдання класифікації.

3.2 Алгоритм бустінгу

Бустинг (підсилювання) – це мета-алгоритм класифікації, суть якого полягає у створенні сильного класифікатора на основі кількох слабких. Для цього спочатку створюється одна модель, а потім інша модель, яка намагається виправити помилки в першій. Моделі додаються до тих пір, поки тренувальні дані не будуть ідеально передбачатися або поки не буде перевищено необхідну кількість моделей.

Основні алгоритми бустингу: *AdaBoost, LogitBoost, BrownBoost, LPBoost, RankBoost, Gradient*

Одним із найпростіших алгоритмів є алгоритм *AdaBoost.*

Алгоритм *AdaBoost* будує «сильний» класифікатор виду:

$$F_T(x) = sign(f_T(x)) = sign(\sum_{t=1}^{T} \beta_t h(x, a_t))$$

де $w_t \in \mathbb{R}, h(x, a_t): X \times A \to \{-1, 1\}$ – "слабкий" класифікатор, що належить деякому сімейству класифікаторів *H*, а *A* – простір параметрів цього сімейства.

Для знаходження класифікатора $F_T(x)$ алгоритм *AdaBoost* послідовно шукає оптимальні параметри $\beta_t t$ і a_t $(1 \le t \le T)$ для побудови класифікатора $F_t(x)$, використовуючи знайдений раніше класифікатор F_{t-1}:

$$F_t(x) = sign(f_{t-1}(x) + \beta_t h(x, a_t)), 1 \le t \le T,$$

при умові, що $f_0(x) = 0.$

Крок 1: Ініціалізуємо ваги об'єктів:

$$w_i = \frac{1}{n}, i = 1, ..., n$$

Крок 2: Послідовно виконуємо побудову класифікаторів $F_t(x), 1 \le t \le T$ для $t = 1, ..., T$:

a) навчаємо слабкий класифікатор $h(x, a_t) \in H$ використовуючи в якості функції втрат :

$$L(a) = \sum_{i=1}^{n} w_i I[\![h(x_i, a) \neq y_i]\!]$$

де $I[\![h(x_i, a) \neq y_i]\!]$ – індикатор помилки.

b) Виконуємо підрахунок β_t:

$$\beta_t = \frac{1}{2} \ln \frac{1 - L(a_t)}{L(a_t)}$$

c) Оновлюємо вагу об’єктів:

$$w_i = \frac{w_i e^{-\beta_t y_i h(x_i, a_t)}}{Z_t}, i = 1, ..., n$$

де Z_t – множник нормування.

Крок 3: Будуємо підсумковий класифікатор :

$$F_T(x) = sign(\sum_{t=1}^{T} \beta_t h(x, a_t))$$

Проста ілюстрація цього підходу приведена на рис. 6.6

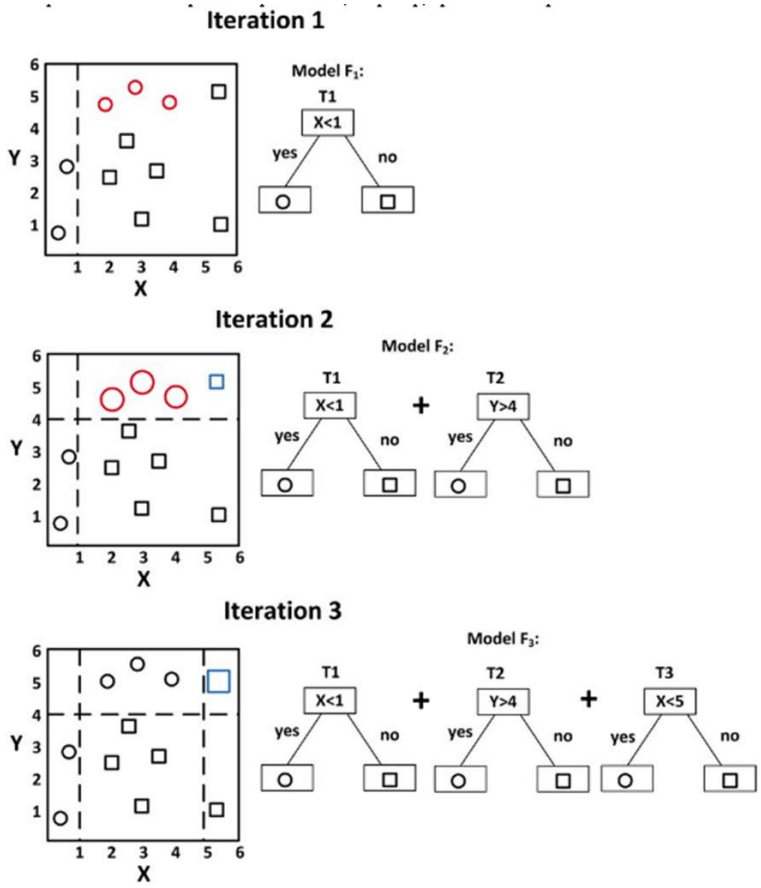

Рисунок 6.6 – Ілюстрація алгоритму *AdaBoost*.

3.3 Алгоритм стекінгу

Стекінг – це метаалгоритм класифікації, що використовує композиції класифікаторів, кожен з яких навчається на однакових підвибірках даних незалежно і паралельно використовуючи різні алгоритми навчання. Навчаємо кілька різних алгоритмів і передаємо їх результати на вхід до останнього класифікатора, який приймає остаточне рішення. Результуючий класифікатор може працювати за різним алгоритмом, в тому числі усереднювати.

Алгоритм стекінгу є наступним:

Крок 1. Будуємо групу різних моделей на вихідному наборі ознак (наприклад, лінійну регресію, регресію за *KNN*, дерево прийняття рішень, алгоритм *Random Forest*, нейромереж та ін.

Крок 2. Кожна з моделей на кроці 1 дала якісь передбачення, додаємо ці передбачення до початкового набору ознак. Отримуємо так звані "мета-ознаки".

Крок 3. Навчаємо модель прийняття остаточного рішення (наприклад, знову лінійну регресію) на наборі мета-ознак. Така модель називається

"мета-модель". Результат мета-моделі будемо вважати остаточним прогнозом.

4 Порядок виконання індивідуального завдання

Залежно від номера прізвища за списком вибрати індивідуальне завдання (Табл.1). Розробити два програмні модулі:

– Програмний модуль генерації псевдовипадкових даних.

– Програмний модуль класифікатора на основі ансамблевого мета-алгоритму, відповідно до індивідуального завдання.

– Виконати оцінку точності класифікації за допомогою ***ROC***-аналізу. Для оцінки якості класифікації використати програмне забезпечення, розроблене в лабораторній роботі №2.

Програмний модуль генерації псевдовипадкових даних повинен генерувати таблицю розміру NxL, де N – кількість вихідних даних, $[a, b]$ – діапазон даних), L=5 та відображає кількість ознак або атрибутів об’єкту(табл.6.1). Також необхідно згенерувати стовбець Y, $Y \in [0,C]$ який розмічає або класифікує вихідні дані відповідно до кількості класів C (табл.6.1). Вихідний формат даних – ****.csv***.

Побудована програмна модель класифікатора на основі мета-алгоритму повинна зберігатись у файлі та ідентифікувати новий об’єкт, який задається випадковими ознаками.

Таблиця 6.1 – Індивідуальне завдання

№ % 32	N	a	b	C-Кількість класів
1	**2**	**3**	**4**	**5**
0	3456	3100	3200	3
1	7231	3000	3100	4
2	2231	2900	3000	5
3	9503	2800	2900	6
4	3151	2700	2800	3
5	3533	2600	2700	4
6	9938	2500	2600	5
7	3653	2400	2500	6
8	6435	2300	2400	3
9	2345	2200	2300	4
10	9634	2100	2200	5
11	8543	2000	2100	6
12	7634	1900	2000	3
13	3456	1800	1900	4

1	2	3	4	5
14	8345	1700	1800	5
15	2423	1600	1700	6
16	7754	1500	1600	3
17	9923	1400	1500	4
18	5435	1300	1400	5
19	4495	1200	1300	6
20	7685	1100	1200	3
21	7758	1000	1100	4
22	8743	900	1000	5
23	8675	800	900	6
24	9536	700	800	3
25	4798	600	700	4
26	9576	500	600	5
27	8675	400	500	6
28	8344	300	400	3
29	7655	200	300	4
30	8799	100	200	5
31	7588	0	100	6

де № – номер студента за списком у журналі групи .

5 Зміст звіту

- Титульний лист.
- Тема роботи.
- Мета роботи.
- Завдання.
- Алгоритм програми (текстовий та/або графічний вигляд).
- Текст програми.
- **Оцінка результатів побудованої моделі (*ROC*-аналіз)**
- Результати виконання роботи програми.
- Висновки.

6 Програмна реалізація побудови ансамблю дерев рішень методом AdaBoost

6.1. Бібліотеки та методи

Виконувані бібліотеки:

- pandas==1.2.4
- numpy==1.22.2
- sklearn==1.0.2

- matplotlib==3.4.0

Методи sklearn рекомендовані до розгляду:

- sklearn.preprocessing.LabelBinarizer
- sklearn.preprocessing.OneHotEncoder
- sklearn.ensemble.AdaBoostClassifier
- pandas.get_dummies

Розробимо необхідні програмні модулі:

```
random_seed = 42

from sklearn.preprocessing import OneHotEncoder
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import roc_curve, auc, accuracy_score,
classification_report, plot_roc_curve
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

def plot(X,
         X_test,
         y_test,
         model,
         figsize=(8,6),
         save_png=True):


    plot_indent = 0.5
    plot_step =0.01
    edge_color = 'silver'
    cmap_name='Set1'
    marker = 'o'
    marker_size = 50

    # налаштування кольорів
    labels = np.unique(y_test)
    colors = matplotlib.cm.get_cmap(cmap_name).colors
    if len(labels) > len(colors):
        print(f'Max classes for visualizing = {len(colors)}.')
        return
    colors = [c for i,c in enumerate(colors) if i in labels]
#colors[:n_classes]
    cmap = matplotlib.colors.ListedColormap(colors)

    # створення картинки
    fig, ax = plt.subplots(figsize=figsize)

    # зафарбовування фону
    x1_min, x1_max = X[:, 0].min() - plot_indent, X[:, 0].max() +
plot_indent
```

```
    x2_min, x2_max = X[:, 1].min() - plot_indent, X[:, 1].max() +
plot_indent
    x1, x2 = np.meshgrid(np.arange(x1_min, x1_max, plot_step),
np.arange(x2_min, x2_max, plot_step))
    Z = model.predict(np.c_[x1.ravel(), x2.ravel()])
    Z = Z.reshape(x1.shape)
    cnt = ax.contourf(x1, x2, Z, alpha=0.4, levels=len(labels), cmap=cmap)
    for c in cnt.collections:
        c.set_edgecolor(edge_color)

    # розрахунок точності
    y_pred = model.predict(X_test)
    acc = round(accuracy_score(y_test, y_pred), 2)

    # відображення точок для кожного класу своїм кольором
    for c, l in zip(colors, labels):
        idx_points = np.where(y_test == l)
        ax.scatter(X_test[idx_points, 0], X_test[idx_points, 1],
marker=marker, s=marker_size, color=c, edgecolors='k')

    # відображення додаткової інформації
    ax.set_title(f'{type(model).__name__} [acc={acc}]', fontweight="bold",
fontsize=20)
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    legends = [Line2D([0],[0], marker=marker, markeredgewidth=1,
markeredgecolor='k', linestyle='', color=c, label=l) for c, l in
zip(colors, labels)]
    ax.legend(handles=legends)

    # зберігання зображення
    if save_png:
        fig.savefig(f'{type(model).__name__}.png',facecolor='white')

          return ax
```

6.2 Тестування розробленого модуля на відкритих та псевдовипадково сформованих даних

В цій лабораторній роботі в якості прикладу возьмемо реальні тестові дані замість генерації своїх.

Для цього необхідно рахувати дані в pandas.DataFrame. Pandas має велику кількість різних методів зчитування даних із файлу. Найпопулярніший метод - pandas.read_csv.

Два варіанти використання:

1. CSV файл розташований локально на комп'ютері. Для цього на вхід подаємо шлях файлу: pd.read_csv('filename.csv')
2. CSV-файл розташований віддалено, наприклад, у репозиторії github. Для цього необхідно в гітхабі відкрити файл та знайти кнопку Raw. Після цього скопіювати шлях та вставити у функцію: pd.read_csv('https://gist.githubusercontent.com/abcd/1234567/raw/1234567890/filename.csv'):

- names - імена стовпців
- index_col - стовпець для індексу
- header - рядок для імен стовпців (за замовчуванням береться нульовий рядок)
- dtype - тип даних (можна ставити як усієї таблиці, так і кожному стовпцю окремо)
- nrows - кількість рядків для зчитування (для великих таблиць)
- parse_dates - стовпці, які необхідно перетворити на дату
- encoding - корисно при роботі з різними мовами

Завантажимо датасет та відобразимо перші 5 зразків (рис. 6.7).

```
df =
pd.read_csv('https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21
398ea43cba4f7d537619d0e07d5ae3/iris.csv')
df.head()
```

[2]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

Рисунок 6.7 – Перші 5 зразків навчального датасету

```
X = df.drop(columns=['variety'])
y = df['variety']
n_classes = len(y.unique())

X.shape, y.shape, n_classes
```

Побудуємо ROC-криву. Нагадаю, що ROC-аналіз робиться тільки для бінарної класифікації. А що робити, якщо у нас класифікатор небінарний (3 і більше класів)? Тут необхідно застосувати прийом один-проти-всіх (one-VS-all). Сенс його у цьому, що оцінюємо роботу класифікатора з погляду точності класифікації кожного класу окремо. Спробуємо це реалізувати. Але для початку перетворюємо y.

Швидке кодування (One-Hot Encoding) – процес, з допомогою якого категоріальні змінні перетворюються на відповідну алгоритмам Машинного навчання (ML) форму.

Трохи змінимо *y* (рис. 6.8 - 6.9):

col1	col2	label
1	2	class_1
2	3	class_3
3	1	class_2
4	1	class_3
1	3	class_2
3	3	class_2

Рисунок 6.8 – Класи до виконання процедури швидкого кодування

col1	col2	class_1	class_2	class_3
1	2	1	0	0
2	3	0	0	1
3	1	0	1	0
4	1	0	0	1
1	3	0	1	0
3	3	0	1	0

Рисунок 6.9 – Назви классів після виконання процедури швидкого кодування

Це можна зробити трьома способами. Можна вибрати будь-який:

```
# 1 method
# y = y.values.reshape(-1, 1)
# y = OneHotEncoder().fit_transform(y).toarray()

# 2 method
# y = pd.get_dummies(y).values

# 3 method
y = LabelBinarizer().fit_transform(y)
```

Ділимо датасет на тренувальний та тестовий. А далі ініціалізуємо класифікатор AdaboostClassifier і додаємо його в класифікатор-обгортку OneVsRestClassifier. Знаходимо ймовірності для ROC-аналізу (так само як і в лаб.роботі №2).

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=random_seed, stratify=y
)
```

```
classifier =
OneVsRestClassifier(AdaBoostClassifier(random_state=random_seed))
y_prob = classifier.fit(X_train, y_train).predict_proba(X_test)
```

Виводимо результат, почергово відображаючи криву для кожного класу (рис. 6.10).

```
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])


plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'ROC curve of class {i} (auc =
{round(roc_auc[i],3)})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Analysis')
plt.legend(loc="lower right")
plt.show()
```

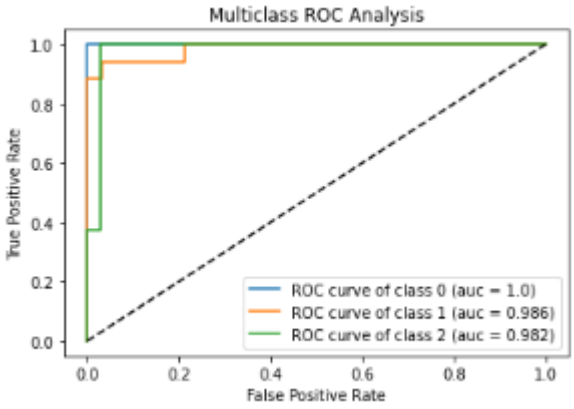

Рисунок 6.10 – ROC-аналіз для навчального датасету

Повторюємо те саме для згенерованих даних. (рис. 6.11)

```
n_classes = 5
X, y = make_blobs(
    n_samples=600,
    n_features=2,
    centers=n_classes,
    center_box=(0, 15),
    random_state=random_seed
)

y_bin = LabelBinarizer().fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(
    X, y_bin, test_size=0.33, random_state=random_seed, stratify=y_bin
)
```

```
classifier =
OneVsRestClassifier(AdaBoostClassifier(random_state=random_seed))
y_prob = classifier.fit(X_train, y_train).predict_proba(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])


plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'ROC curve of class {i} (auc =
{round(roc_auc[i],3)})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Analysis')
plt.legend(loc="lower right")
plt.show()
```

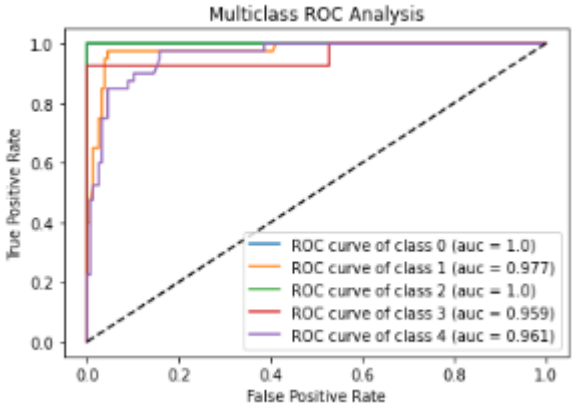

Рисунок 6.11 – ROC-аналіз для згенерованих даних

Будуємо ансамбль на базі дерев рішень з використанням бустинг мета-алгоритму (рис. 6.12).

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=random_seed, stratify=y
)

model = AdaBoostClassifier(random_state=random_seed)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot(X, X_test, y_test, model)
```

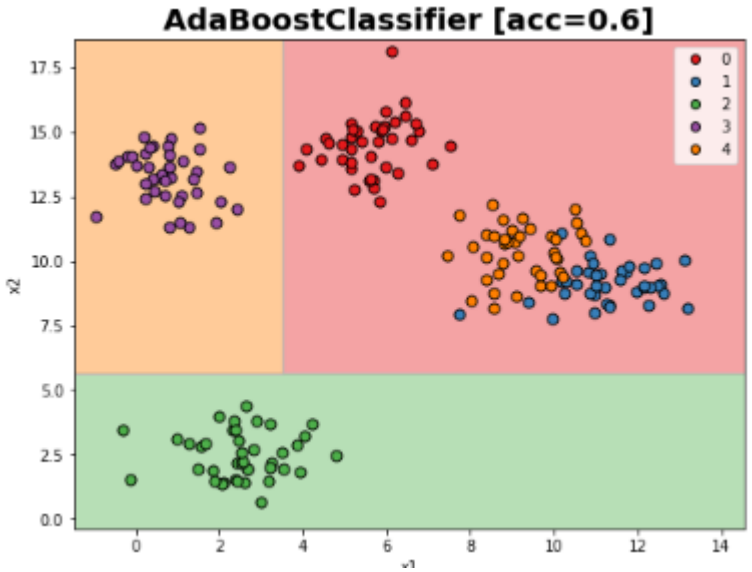

Рисунок 6.12 – ROC-аналіз

Отримаємо матрицю невідповідності для побудованого ансабля дерев рішень. (рис. 6.13)

```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.03	0.06	33
1	0.35	1.00	0.52	33
2	1.00	1.00	1.00	33
3	0.00	0.00	0.00	33
4	0.00	0.00	0.00	33
5	0.46	1.00	0.63	33
accuracy			0.51	198
macro avg	0.47	0.51	0.37	198
weighted avg	0.47	0.51	0.37	198

Рисунок 6.13 – Матріца невідповідностей для 6 класів

Посилання

1. https://habr.com/ru/company/ods/blog/324402/
2. https://www.datavedas.com/bagging/
3. https://habr.com/ru/company/ods/blog/324402/
4. https://ranalytics.github.io/data-mining/044-Ensembles.html
5. https://habr.com/ru/company/ods/blog/324402/
6. https://habr.com/ru/post/116385/
7. https://dyakonov.org/2017/03/10/c%D1%82%D0%B5%D0%BA%D0%B8%D0%BD%D0%B3-stacking-%D0%B8-%D0%B1%D0%BB%D0%B5%D0%BD%D0%B4%D0%B8%D0%BD%D0%B3-blending/

Лабораторна робота № 7
“Розробка класифікатору даних на основі мета-алгоритму *Random Forest*”

1 Мета роботи

Дослідження методів класифікації даних. Отримання навичок аналізу даних на прикладі побудови класифікатору на основі алгоритму *Random Forest*.

2 Теми для попереднього опрацювання

Методи класифікації.
Дерева прийняття рішення.
Ансамблеві мета-алгоритм *Random Forest*.
Оцінка точності методів класифікації.

3 Короткі теоретичні відомості

Випадковий ліс (*Random Forest_* – це різновид ансамблевого мета-алгоритму бегінгу, який поєднує дерева прийняття рішення.

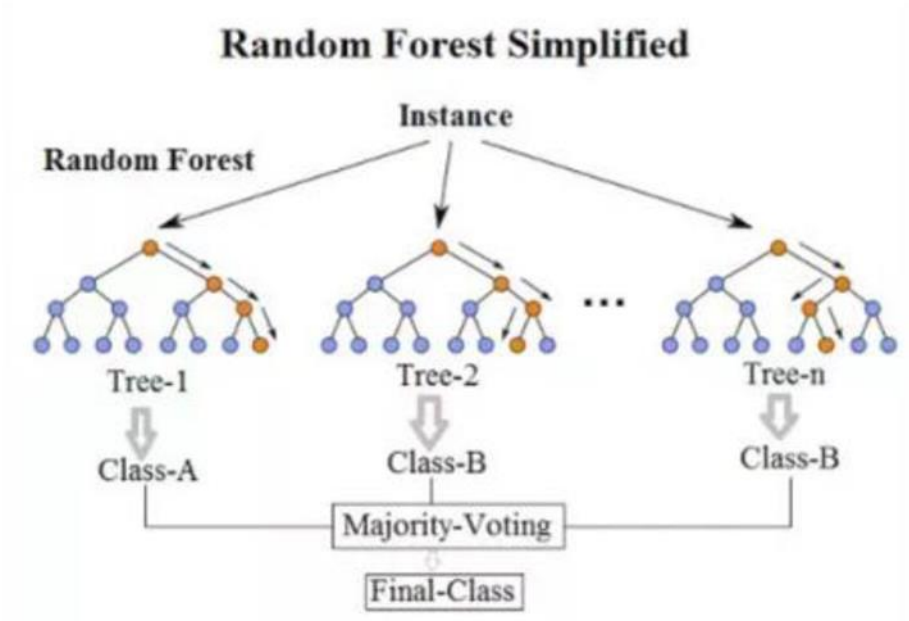

Рисунок 7.1 – Випадковий ліс

Для побудови дерев рішень із вихідних даних формуються випадкові вибірки із заміною, а також вибираючи випадкові ознаки які можуть повторюватися. Таким чином, кожна підмножина може мати дублікати вибірок (спостережень), проте є випадково обрані ознаки, які є унікальними. Приклад формування вибірок для дерев рішень наведено на рис.7.2.

Bag 1			Bag 2			Bag 3			Bag 4		
X1	X2	X4	X2	X3	X4	X1	X2	X3	X1	X3	X4
245	26	5	47	251	45	803	41	188	453	67	357
505	39	50	26	131	55	803	41	188	245	416	5
245	26	5	26	131	55	453	67	357	453	67	357

Рисунок 7.2 –Формування вибірок для алгоритму *Random Forest*

У звичайному дереві рішень, коли потрібно розділити вузол, розглядається кожна можлива ознака і вибирається та, яка сильніше ділить значення в вузлах. У випадковому лісі ознака розподілу може вибирати тільки з випадкової підмножини ознак. Це призводить до ще більшої варіації між деревами в моделі і в кінцевому підсумку до більш слабкої кореляції між деревами і більшої різноманітності.

3.1 Алгоритм навчання класифікатора RF

Алгоритм навчання класифікатора *RF* є наступним:

Нехай навчальна вибірка складається з N прикладів, розмірність простору ознак дорівнює M, і задано параметр m (в задачах класифікації зазвичай $m \approx \sqrt{M}$ або 40-60% від числа ознак), який визначає кількість випадково вибраних ознак.

Усі дерева комітету будуються незалежно один від одного за такою процедурою:

– Згенеруємо випадкову підвибірку з повторенням розміром n з навчальної вибірки. (Таким чином, деякі приклади потраплять в неї кілька разів, а приблизно $N/3$ прикладів не ввійдуть у неї взагалі)

– Побудуємо дерево рішень, яке класифікує приклади даної підвибірки, причому в ході створення чергового вузла дерева будемо вибирати ознаку, на основі якої проводиться розбиття, не з усіх M ознак, а лише з m випадково вибраних. Вибір найкращої з цих m ознак може здійснюватися різними способами, наприклад критерій Джині або приросту інформації).

– Дерево будується до повного вичерпання підвибірки і не піддається процедурі відсікання.

– Класифікація об'єктів проводиться шляхом голосування: кожне дерево комітету відносить об'єкт, який класифікується до одного з класів, а клас, який отримує більшість голосів, є відповіддю моделі ансамблю (це називається мажоритарних голосуванням):

$$- \quad S_L(.) = \arg_k \max[card(l \mid w_l(.) = k)]$$

де $S_L(.)$ – проста більшість голосів завдання класифікації.

4 Порядок виконання індивідуального завдання

Залежно від номера прізвища за списком вибрати індивідуальне завдання (Табл.1). Розробити два програмні модулі:

– Програмний модуль генерації псевдовипадкових даних.

– Програмний модуль класифікатора на основі ансамблевого мета-алгоритму *Random Forest* .

– Виконати оцінку точності класифікації за допомогою ***ROC***-аналізу. Для оцінки якості класифікації використати програмне забезпечення, розроблене в лабораторній роботі №2.

Програмний модуль генерації псевдовипадкових даних повинен генерувати таблицю розміру *NxL*, де *N* – кількість вихідних даних, [*a*, *b*] – діапазон даних), *L*– кількість ознак або атрибутів об'єкту(табл.7.1). Також необхідно згенерувати стовбець *Y*, $Y \in [0,1]$ який розмічає або класифікує вихідні дані відповідно до двох класів. Вихідний формат даних – ****.csv***.

Програмний модуль класифікатора реалізує метод класифікації на основі ансамблевого мета-алгоритму з урахуванням параметру *C*, який задає кількість дерев рішень. Побудована програмна модель класифікатора на основі мета-алгоритму повинна зберігатись у файлі та ідентифікувати новий об'єкт, який задається випадковими ознаками.

Таблиця 7.1 – Індивідуальне завдання

№ % 32	*N*	*a*	*b*	*L* –кількість ознак	C– Кількість дерев рішень
1	**2**	**3**	**4**	**5**	**6**
0	3456	3100	3200	7	30
1	7231	3000	3100	8	31
2	2231	2900	3000	9	32
3	9503	2800	2900	10	33
4	3151	2700	2800	11	34
5	3533	2600	2700	12	35
6	9938	2500	2600	7	36
7	3653	2400	2500	8	37
8	6435	2300	2400	9	38
9	2345	2200	2300	10	39
10	9634	2100	2200	11	40
11	8543	2000	2100	12	20
12	7634	1900	2000	7	21
13	3456	1800	1900	8	22
14	8345	1700	1800	9	23
15	2423	1600	1700	10	24
16	7754	1500	1600	11	25

1	2	3	4	5	6
17	9923	1400	1500	12	26
18	5435	1300	1400	7	27
19	4495	1200	1300	8	28
20	7685	1100	1200	9	29
21	7758	1000	1100	10	30
22	8743	900	1000	11	31
23	8675	800	900	12	32
24	9536	700	800	7	33
25	4798	600	700	8	34
26	9576	500	600	9	35
27	8675	400	500	10	36
28	8344	300	400	11	37
29	7655	200	300	12	38
30	8799	100	200	7	39
31	7588	0	100	8	40

де № – номер студента за списком у журналі групи .

5 Зміст звіту

- Титульний лист.
- Тема роботи.
- Мета роботи.
- Завдання.
- Алгоритм програми (текстовий та/або графічний вигляд).
- Текст програми.
- **Оцінка результатів побудованої моделі (*ROC*-аналіз)**
- Результати виконання роботи програми.
- Висновки.

6 Програмна реалізація побудови ансамблю дерев рішень методом RandomForest

6.1 Бібліотеки та методи

Бібліотеки, що використовуються:

- pandas==1.2.4
- numpy==1.22.2
- sklearn==1.0.2
- matplotlib==3.4.0

Методи sklearn рекомендовані до розгляду:

- sklearn.metrics.auc
- sklearn.metrics.roc_curve
- sklearn.metrics.roc_auc_score
- sklearn.metrics.confusion_matrix
- sklearn.metrics.accuracy_score
- sklearn.metrics.classification_report
- sklearn.model_selection.train_test_split

Розробимо необхідні функції для алгоритму RandomForest:

```
# змінити на None для отримання випадкових результатів
random_seed = 42

import random
random.seed(random_seed)
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import ConfusionMatrixDisplay, auc,
classification_report, confusion_matrix, roc_curve, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier

def generate_dataset(N, L, one_class_freq, feature_range):

    X, y = make_classification(
        n_samples=N,
        n_features=L,
        n_classes=2,
        n_informative=1,
        n_redundant=0,
        n_clusters_per_class=1,
        weights=[1 - one_class_freq],
        random_state=random_seed,
    )

    scaler = MinMaxScaler(feature_range=feature_range)
    X = scaler.fit_transform(X)

    for idx, x in np.ndenumerate(X):
        i = idx[0]
        j = idx[1]
        X[i][j] = np.int16(x)


          return X, y
```

6.2 Тестування розробленого модуля

Генеруємо датасет і ділимо його на дві частини. Функція generate_dataset(M, L, one_class_freq, feature_range). Генерує

датасет із двома колонками. Одна колонка – незалежна змінна X, інша – залежна змінна y. Датасет, що генерується, є датасетом бінарної класифікації.

Значення, що приймаються:

- Параметр N : *int* - Кількість рядків (примірників).
- Параметр L: *int* - Кількість стовпців (ознак).
- Параметр one_class_freq: *float* - Частота появи класу 1. Набуває значення від 0 до 1.
- Параметр feature_range : *tuple* – діапазон значень змінної X.

Значення, що повертається - *tuple* (X, y).

```
X, y = generate_dataset(N=3000, L=31, one_class_freq=0.9, feature_range=(1,
100))

# ділимо датасет на тренувальний та тестовий у пропорції 2:1 (задається
параметром test_size).
# параметр stratify відповідає за те, щоб класовий розподіл у двох
отриманих датасетах не відрізнявся від початкового.
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=random_seed, stratify=y
    )
```

Створюємо, навчаємо та тестуємо класифікатор

```
clf = OneVsRestClassifier(RandomForestClassifier(random_state=random_seed))
#clf = RandomForestClassifier()
      y_prob = clf.fit(X_train, y_train).predict_proba(X_test)
```

Будуємо ROC-криву та оцінюємо площу під нею. Для отримання значень fpr (false positive rate) та tpr (true positive rate) використовуємо функцію roc_curve(y_test, y_prob[:, 1]).

Для отримання оцінки площі під ROC-кривою використовуємо функцію auc(fpr, tpr). Крім того, можна використати функцію roc_auc_score(y_test, y_prob[:, 1]), якщо раніше не були отримані значення fpr і tpr.

```
fpr, tpr, _ = roc_curve(y_test, y_prob[:, 1])
roc_auc = auc(fpr, tpr)
# малюємо roc криву
plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve of class 1 (auc =
{round(roc_auc,3)})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Analysis')
plt.legend(loc="lower right")
plt.show()

      print(f"ROC AUC score = {auc_score}")
```

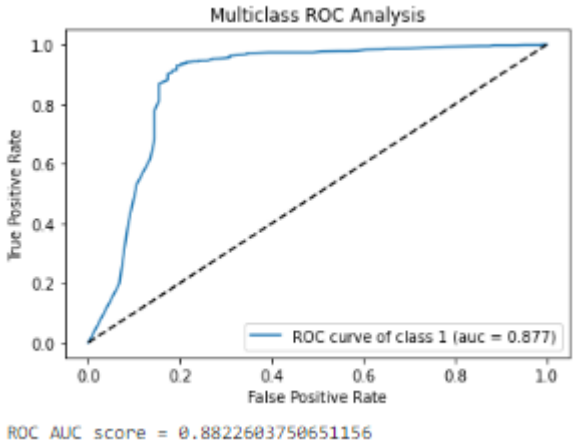

Рисунок 7.3 – ROC-оцінка ансамблевої моделі на RandomForest

Посилання

1. https://habr.com/ru/company/ods/blog/324402/
2. https://www.datavedas.com/bagging/
3. https://habr.com/ru/company/ods/blog/324402/
4. https://ranalytics.github.io/data-mining/044-Ensembles.html
5. https://habr.com/ru/company/ods/blog/324402/
6. https://habr.com/ru/post/116385/
7. https://dyakonov.org/2017/03/10/c%D1%82%D0%B5%D0%BA%D0%B8%D0%BD%D0%B3-stacking-%D0%B8-%D0%B1%D0%BB%D0%B5%D0%BD%D0%B4%D0%B8%D0%BD%D0%B3-blending/

Лабораторна робота № 8
“Згорткові нейронні мережі”

1 Мета роботи

Отримати практичні навички розробки, налаштування та тестування згорткових нейронних мереж

2 Теми для попереднього проороблення

Інтерполяція даних
Логістична регресія
Теорія обробки сигналів та зображень (згортки)
Нейронні мережі

3 Короткі теоретичні відомості

Згорткові нейронні мережі (*Convolutional Neural Network* (*ConvNet*/*CNN*) – це клас глибинних штучних нейронних мереж прямого поширення, який застосовується до аналізу зображень. *CNN* складається з шарів входу та виходу, а також із декількох прихованих шарів. Основною складовою прихованих шарів є шар згортки, що складається з декількох (зазвичай, від одиниць до кількох десятків) фільтрів, кожен з яких налаштовується у процесі навчання для виявлення характеристичних ознак на зображенні. Ці фільтри в режимі «ковзного вікна» проходять по всьому зображенню, записуючи результат згортки з елементами зображення у відповідний осередок вихідного зображення. Вихідне зображення у разі називається «***картою ознак***», оскільки відповідає виділеним за допомогою даного фільтра ознакам. Сама операція згортки, що реалізується з кожним фільтром у згортковому шарі, виявляється у попарному перемноженні елементів фільтра з елементами вхідного зображення, причому розмір вікна точно дорівнює розміру фільтру. Таким чином, роль *CNN* полягає в тому, щоб зменшити зображення у форму, яку легко обробляти, не втрачаючи функцій, які мають вирішальне значення для отримання гарного прогнозу.

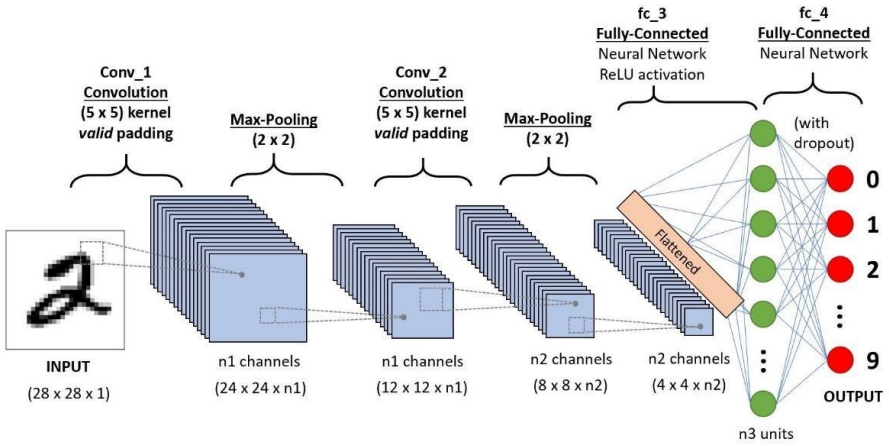

Рисунок 8.1– Приклад функціонування згорткової нейронної мережі

Результат операції згортки визначається як:

$$(f \cdot g)[m.\ n] = \sum_{k,l} f[m-k, n-k] \cdot g[k,l],$$

f – вихідна матриця зображення розміру n x m;
g –фільтр згортки розміру k x l .

Фільтри в перших шарах мережі будуть активуватися базовими ознаками: лінії, кути, блоки, при цьому при русі до більш глибоких шарів згорткової мережі фільтри в них виділятимуть більш складні ознаки (рис. 8.3). Розмір ядра зазвичай беруть у межах від 3x3 до 7x7. Якщо розмір ядра маленький, воно не зможе виділити будь-які ознаки, якщо занадто велике, то збільшується кількість зв'язку між нейронами. Кожен фільтр є матрицею ваг, що налаштовуються в процесі навчання моделі: це одна з головних особливостей згорткової нейронної мережі, що полягає в принципі «*shared weights*» (загальних ваг) та дозволяє скоротити число зв'язків, знаходячи одну і ту ж ознаку на всій області зображення.

При цьому залежно від методу обробки країв вихідної матриці результат згортки може бути меншим за вихідне зображення («*valid*») або такого ж розміру («*same*»). На практиці, щоб не втрачати інформацію з пікселів розмежування зображення, частіше використовують згортку із збереженням розміру вхідного зображення, для цього вихідну матрицю доповнюють одним або декількома шарами пікселів. Ця операція називається паддинг (*padding*) і описується параметром *p*, який відповідає за «товщину» доданого шару (рис.8.2). Найчастіше додаткові пікселі заповнюються нульовими значеннями.

Розмір карти ознак визначається як:

$$n_{out} = \left| \frac{n_{in} \_ 2p - k}{s} \right| + 1,$$

де: n_{in} – розмір вхідного зображення; n_{out} – розмір вихідного зображення (мапи ознак); k – розмір фільтру; p – розмір паддінгу; s – страйд

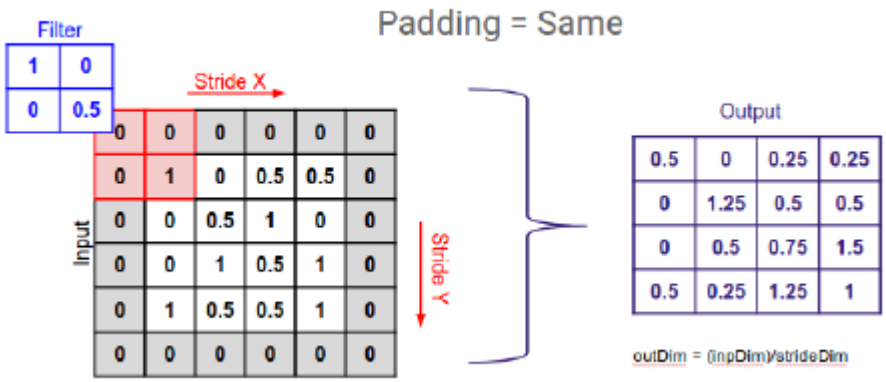

Рисунок 8.2 –Операція «*padding*»

За шаром згортки після застосування функції активації до карт майже завжди слідує шар субдискретизації або пулінгу (рис.8.3), який призначений для зменшення розмірності карт ознак попереднього шару. Зниження числа параметрів моделі дозволяє уникнути перенавчання. Також при зменшенні розмірності карт ознак ми здійснюємо перехід до іншого масштабу ознак, що в кінцевому підсумку дозволяє переходити від точок, ліній та плям на перших шарах до високорівневих ознак, що містить частини реальних об'єктів на останніх шарах мережі (рис.8.4). Розмір вікна пулінгу, зазвичай 2x2, при цьому застосовують два варіанти вибору значення у вікні: вибір максимального елемента – "*MaxPooling*") або середнього елемента – ("*AveragePooling*"), яке потім записують у відповідну комірку вихідної матриці.

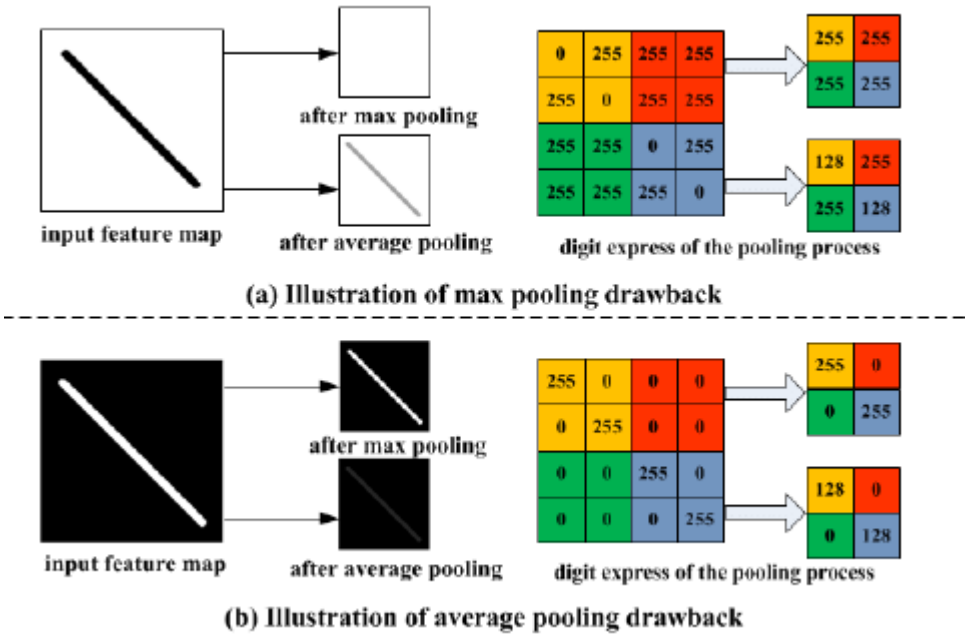

Рисунок 8.3 – Ілюстрація операцій субдискретизації

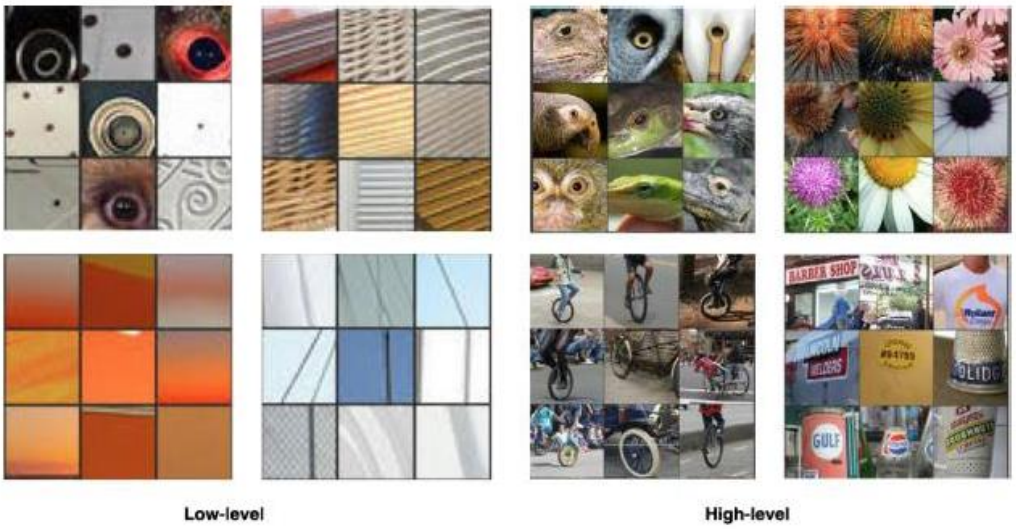

Рисунок 8.4 – Ознаки низького рівня (а), що максимізують активацію нейронів на перших шарах згорткової мережі, та ознаки високого рівня (б) (максимізують активацію нейронів на останніх шарах згорткової мережі).

4 Підготовка до виконання лабораторної роботи

Рекомендована мова для виконання - *Python* 3.6+

Бібліотека для реалізації нейронних мереж - *Keras https*://*keras*.*io*/.

Рекомендоване середовище для навчання мережі - *Google Colab*. Інтерфейс аналогічний *Jupyter Lab*, але є можливість прискорити процес навчання мережі шляхом підключення *GPU*, для цього необхідно в налаштуваннях середовища змінити *Runtime Type* на *GPU* (рис.8.5).

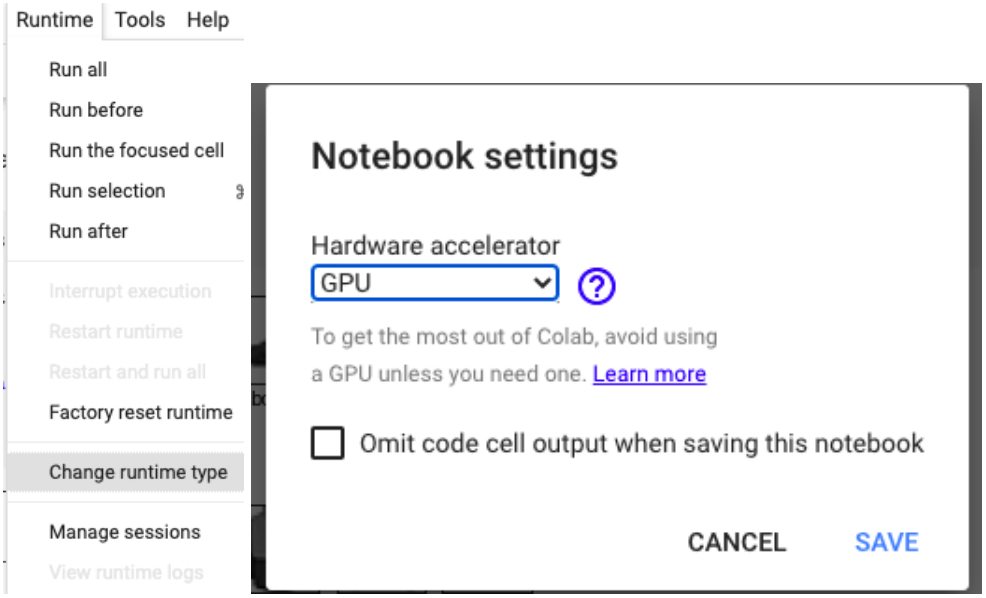

Рисунок 8.5– Налаштуваннях середовища

5. Порядок виконання індивідуального завдання

1. Виконати завантаження набору даних *CIFAR*-10 за допомогою бібліотеки *Keras* (*https*://*keras*.*io*/*api*/*datasets*/*cifar*10/).

2. Виконати візуалізацію декількох зображення для кожного класу.

3. Сформувати тензор для подачі в нейронну мережу, виконати нормалізацію значень.

4. Виконати підготовку *labels* (обробка категоріальної ознаки).

5. Виконати поділ набору даних на тренувальну, валідаційну і тестову вибірки.

6. Запропонувати 2 архітектури згорткової мережі для заданих зображень (при виборі кількості шарів і параметрів згортки / пулінгу використовувати знання, отримані при виконанні ПЗ №8).

7. Використовувати одну з методик запобігання перенавчання, відповідно до варіанту ($V = N \bmod 3 + 1$):

- Варіант 1 - *Dropout* *https*://*keras*.*io*/*api*/*layers*/*regularization_layers*/*dropout*/
- Варіант 2 - *BatchNormalization* *https*://*keras*.*io*/*api*/*layers*/*normalization_layers*/*batch_normalization*/
- Варіант 3 - *Early Stopping Callback https*://*keras*.*io*/*api*/*callbacks*/

8. Виконати навчання нейронної мережі (гіперпараметри задати самостійно).

9. Побудувати графіки, на яких буде візуалізовано значення точності класифікації та помилки під час навчання мережі.

5 Зміст звіту

- Титульний лист.
- Тема роботи.
- Мета роботи.
- Завдання.
- Алгоритм програми (текстовий та/або графічний вигляд).
- Текст програми.
- Результати виконання роботи програми.
- Висновки.

6 Навчання та перевірка згорткові нейронні мережі на наборі даних MNIST

6.1 Бібліотеки та методи

Бібліотеки, що використовуються:

- tensorflow
- PIL
- numpy
- matplotlib

Методи tensorflow рекомендовані до розгляду:

- tensorflow.keras.datasets
- tensorflow.keras.models
- tensorflow.keras.layers
- tensorflow.keras.preprocessing

```
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras import utils
from tensorflow.keras.preprocessing import image
from google.colab import files
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from tensorflow.keras.callbacks import EarlyStopping
%matplotlib inline
```

Виконуємо завантаження даних Fashion MNIST з бібліотеки Keras (рис. 8.6).

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

Рисунок 8.6– Завантаження датасету

Створюємо масив з назвами класів для більш наочного сприйняття

```
classes = ['t-shirt', 'trousers', 'sweetshot', 'dress', 'coat', 'shoes',
'shirt', 'sneakers', 'bag', 'boots']
```

Приклади досліджуваних зображень представлено на рис. 8.7.

```
plt.figure(figsize=(10,10))
for i in range(100,150):
    plt.subplot(5,10,i-100+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(classes[y_train[i]])
```

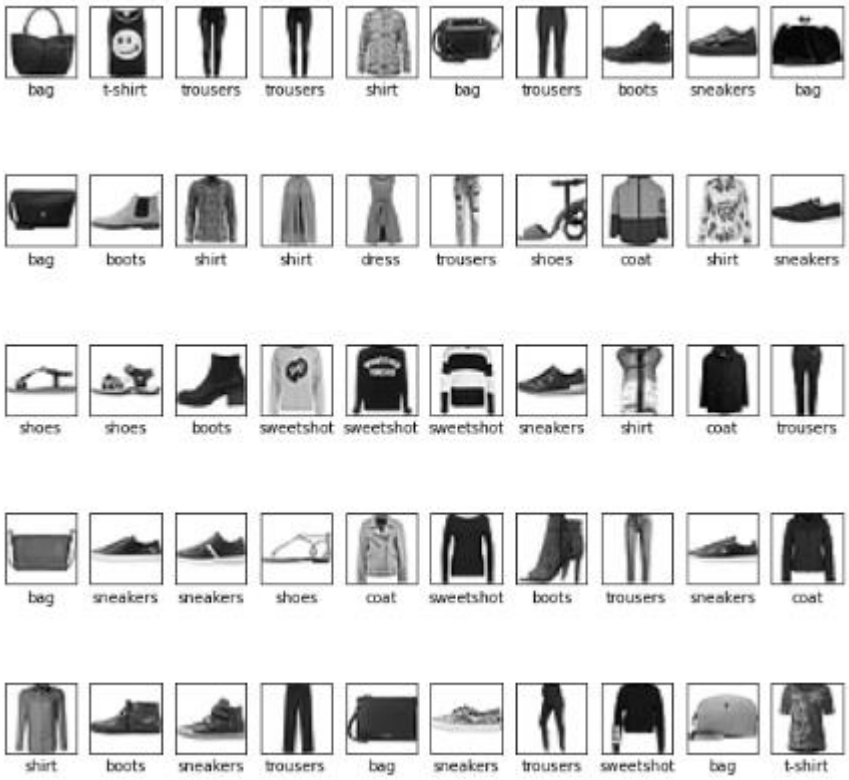

Рисунок 8.7– Приклади вхідних зображень

Виконуємо попередню обробку даних

1. Перетворюємо назву класу у категоріальну характеристику
2. Змінюмо розмірність вхідного зображеня, додавши. ще один вимір, для інтерпретації зображення у відтінках сірого
3. Нормалізуємо дані

```
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)
x_test = np.expand_dims(x_test, axis=3)
x_train = np.expand_dims(x_train, axis=3)
x_train = x_train / 255
x_test = x_test / 255
```

Створюємо модель з двома парами шарів згортки та підвибірки та одним прихованим повнозв'язним шаром. Для кожного шару згортки задається 2 параметри - кількість карток ознак та розмірність ядра згортки. Для шару підвибірки задається розмір ядра для знаходження максимуму. Вихідний шар складаєтсья з 10 нейронів, оскільки дані з набору можуть належати до одного з 10 класів. Додатково використано шар Dropoout для запобігання перенавчанню. (рис. 8.8)

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(28,28, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('sigmoid'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
activation_4 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
activation_5 (Activation)    (None, 11, 11, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               204928
_________________________________________________________________
activation_6 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_7 (Activation)    (None, 10)                0
=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 8.8– Налаштування нейронної мережі

Створимо callback для протидиї перенавчанню. Якщо протягом 10 епох навчання значення помилки на на валідаційній вибірці не зменшується, до навчання мережі зупиняється.

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
```

Виконуємо навчання моделі протягом 100 епох (якщо не спрацює callback), на валідацію виділяємо 10% з тестової вибірки

```
history = model.fit(x_train, y_train, epochs=100, validation_split=0.1,
callbacks = [es])
```

```
Epoch 1/100
1688/1688 [==============================] - 39s 4ms/step - loss: 0.7908 - accuracy: 0.7110 - val_loss: 0.3578 - val_accuracy: 0.8660
Epoch 2/100
1688/1688 [==============================] - 5s 3ms/step - loss: 0.3987 - accuracy: 0.8593 - val_loss: 0.3174 - val_accuracy: 0.8817
Epoch 3/100
1688/1688 [==============================] - 5s 3ms/step - loss: 0.3366 - accuracy: 0.8793 - val_loss: 0.2756 - val_accuracy: 0.8980
Epoch 4/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.2997 - accuracy: 0.8912 - val_loss: 0.2679 - val_accuracy: 0.9020
Epoch 5/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.2727 - accuracy: 0.9005 - val_loss: 0.2646 - val_accuracy: 0.8995
Epoch 6/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.2521 - accuracy: 0.9077 - val_loss: 0.2448 - val_accuracy: 0.9057
```

```
Epoch 7/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.2367 - accuracy: 0.9136 - val_loss: 0.2487 - val_accuracy: 0.9057
Epoch 8/100
1688/1688 [==============================] - 5s 3ms/step - loss: 0.2261 - accuracy: 0.9158 - val_loss: 0.2499 - val_accuracy: 0.9125
Epoch 9/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.2096 - accuracy: 0.9210 - val_loss: 0.2501 - val_accuracy: 0.9100
Epoch 10/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1959 - accuracy: 0.9272 - val_loss: 0.2668 - val_accuracy: 0.9068
Epoch 11/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1879 - accuracy: 0.9298 - val_loss: 0.2658 - val_accuracy: 0.9045
Epoch 12/100
1688/1688 [==============================] - 5s 3ms/step - loss: 0.1811 - accuracy: 0.9315 - val_loss: 0.2529 - val_accuracy: 0.9152
Epoch 13/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1751 - accuracy: 0.9329 - val_loss: 0.2398 - val_accuracy: 0.9198
Epoch 14/100
1688/1688 [==============================] - 5s 3ms/step - loss: 0.1634 - accuracy: 0.9377 - val_loss: 0.2655 - val_accuracy: 0.9175
Epoch 15/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1611 - accuracy: 0.9386 - val_loss: 0.2889 - val_accuracy: 0.9132
Epoch 16/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1525 - accuracy: 0.9436 - val_loss: 0.2664 - val_accuracy: 0.9187
Epoch 17/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1495 - accuracy: 0.9429 - val_loss: 0.2576 - val_accuracy: 0.9190
Epoch 18/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1411 - accuracy: 0.9452 - val_loss: 0.2797 - val_accuracy: 0.9185
Epoch 19/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1395 - accuracy: 0.9446 - val_loss: 0.2724 - val_accuracy: 0.9162
Epoch 20/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1349 - accuracy: 0.9474 - val_loss: 0.2716 - val_accuracy: 0.9153
Epoch 21/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1304 - accuracy: 0.9506 - val_loss: 0.2880 - val_accuracy: 0.9122
Epoch 22/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1234 - accuracy: 0.9526 - val_loss: 0.3140 - val_accuracy: 0.9168
Epoch 23/100
1688/1688 [==============================] - 6s 3ms/step - loss: 0.1222 - accuracy: 0.9528 - val_loss: 0.2917 - val_accuracy: 0.9140
Epoch 00023: early stopping
```

Перевіряємо точність роботи мережі на тестовій вибірці

```
scores = model.evaluate(x_test, y_test, verbose=1)

print("Accuracy on test data: ", round(scores[1] * 100, 4))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2996 - accuracy: 0.9109
Accuracy on test data:  91.09
```

Посилання

1. *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53*

ГАВРИЛЕНКО Світлана Юріївна
ЧЕЛАК Віктор Володимирович
ГОРНОСТАЛЬ Олексій Андрійович
ЗОЗУЛЯ Владислав Денисович

МАШИННЕ НАВЧАННЯ

Лабораторний практикум
для студентів комп'ютерних спеціальностей

Роботу до видання рекомендував М.Й. Заполовський

План 2022 р., поз. 89

Підп. до друку 29.01.2021. Формат 60 х 84 1/16. Папір друк. №2.
Riso-друк. Гарнітура Таймс. Ум. друк. арк. 5,0. Наклад. 100 прим.
Зам №_____. Ціна договірна.

Видавничий центр НТУ „ХПІ".
Свідоцтво про державну реєстрацію ДК № 116 від 10.07.2004 р.
61002, Харків, вул. Фрунзе, 21

Друкарня НТУ "ХПІ". 61002, Харків, вул. Фрунзе, 21