**Лекція 3. Класифікація**

1. **Поняття класифікації. Постановка задачі**
2. **Основні алгоритми та методи класифікації в машинному навчанні**
3. **Оцінка точності**

Класифікація є однією з основних задач машинного навчання, яка полягає у прогнозуванні мітки класу для нового зразка на основі навчальних даних. Класифікаційні моделі широко застосовуються у розпізнаванні зображень, медичній діагностиці, аналізі текстів та інших сферах.

## **Основи класифікації**

Класифікація передбачає наявність цільової змінної, яка приймає дискретні значення (класи). Основні етапи процесу класифікації:

* **Збір даних**: формування навчального набору даних із мітками класів.
* **Попередня обробка**: очищення даних, нормалізація, обробка пропущених значень.
* **Розділення даних**: навчальна, валідаційна та тестова вибірки.
* **Навчання моделі**: використання алгоритму класифікації для знаходження залежностей у даних.
* **Оцінка якості**: перевірка точності класифікації на тестовій вибірці.

##  Типи класифікаційних задач

* **Бінарна класифікація**: два класи (наприклад, "позитивний" і "негативний").
* **Багатокласова класифікація**: більше двох класів (наприклад, розпізнавання рукописних цифр від 0 до 9).
* **Багатоміткова класифікація**: кожен об'єкт може належати до кількох класів одночасно (наприклад, категоризація статей).

## **Основні алгоритми класифікації**

1. **Логістична регресія**: лінійний метод, який обчислює ймовірність належності до класу.
2. **Дерева рішень**: модель, яка розбиває простір ознак за допомогою умовних операторів.
3. **Підтримувальні векторні машини (SVM)**: визначення гіперплощини, яка розділяє класи з максимальним відступом.
4. **Наївний байєсівський класифікатор**: ймовірнісна модель, яка використовує припущення незалежності ознак.
5. **Нейронні мережі**: потужні моделі, здатні знаходити складні нелінійні залежності.

**Метод опорних векторів (SVM)** — це потужний алгоритм для класифікації, який шукає оптимальну межу (гіперплощину) для відокремлення зразків різних класів у багатовимірному просторі. Метою SVM є максимізація відстані між найближчими зразками різних класів та гіперплощиною — це відстань називається "запасом" (margin). Такий підхід дозволяє мінімізувати помилки класифікації, особливо на тестових даних.



Основна ідея SVM полягає в наступному: знайти гіперплощину, яка відокремлює дані так, щоб запас був максимальним.

### **Простий приклад**

1. **Припустимо, що в нас є учні зростом:**
	* 150 см, 155 см, 160 см (низькі)
	* 170 см, 175 см, 180 см (високі)
2. **Побудова гіперплощини**
	* SVM шукає найкращу межу (лінію), яка розділить ці дві групи.
	* У цьому випадку логічно провести пряму між 160 см та 170 см (наприклад, на 165 см).
3. **Опорні вектори**
	* Це ті точки (учні), які знаходяться найближче до межі.
	* Тут це 160 см (з низьких) і 170 см (з високих).
	* SVM вибирає таку межу, щоб відстань від неї до найближчих точок була максимальною.
4. **Як працює класифікація?**
	* Якщо новий учень має зріст **162 см**, модель скаже, що він **низький**.
	* Якщо 172 см – **високий**.

Ось приклад програми для реалізації методу опорних векторів (SVM) за допомогою бібліотеки scikit-learn, використовуючи вигадані дані про вік і бюджет, щоб передбачити, чи піде людина на фільм.

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Створення вигаданого набору даних

data = {

 'Age': [22, 25, 47, 35, 18, 23, 32, 54, 46, 29],

 'Budget': [15, 25, 50, 20, 10, 30, 40, 60, 70, 35],

 'Will\_Watch': ['No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes']

}

# Перетворення даних у DataFrame

df = pd.DataFrame(data)

# Визначення ознак (features) і цільової змінної (target)

X = df[['Age', 'Budget']]

y = df['Will\_Watch']

# Перетворення цільової змінної у числовий формат

y = np.where(y == 'Yes', 1, 0) # 'Yes' -> 1, 'No' -> 0

# Розділення даних на навчальну та тестову вибірки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Створення та навчання моделі SVM

model = SVC(kernel='linear') # Використовуємо лінійне ядро

model.fit(X\_train, y\_train)

# Прогнозування на тестових даних

y\_pred = model.predict(X\_test)

# Оцінка точності моделі

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Точність моделі SVM: {accuracy \* 100:.2f}%')

# Візуалізація результатів

plt.figure(figsize=(10, 6))

# Відображення точок

plt.scatter(X['Age'], X['Budget'], c=y, cmap='viridis', label='Дані')

plt.scatter(X\_test['Age'], X\_test['Budget'], c='red', label='Тестові дані', marker='x')

# Відображення лінії рішень

xlim = plt.xlim()

ylim = plt.ylim()

xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100),

 np.linspace(ylim[0], ylim[1], 100))

Z=model.decision\_function(pd.DataFrame(np.c\_[xx.ravel(), yy.ravel()], columns=X\_train.columns))

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z > 0, alpha=0.5, colors=['#FFAAAA', '#AAAAFF'])

plt.scatter(X\_train['Age'], X\_train['Budget'], c=y\_train, edgecolor='k', marker='o', label='Навчальні дані')

plt.title('Метод опорних векторів (SVM)')

plt.xlabel('Вік')

plt.ylabel('Бюджет')

plt.legend()

plt.show()

**Пояснення програми:**

1. **Імпорт бібліотек**: Імпортуємо необхідні бібліотеки для обробки даних, побудови моделі SVM та візуалізації.
2. **Створення набору даних**: Створюємо вигаданий набір даних з інформацією про вік, бюджет на квитки та відповідь, чи піде людина на фільм.
3. **Перетворення даних у DataFrame**: Перетворюємо дані у формат DataFrame для зручності.
4. **Визначення ознак і цільової змінної**: Визначаємо ознаки (X) та цільову змінну (y).
5. **Перетворення цільової змінної**: Конвертуємо цільову змінну в числовий формат: 'Yes' в 1, 'No' в 0.
6. **Розділення даних**: Розділяємо дані на навчальну та тестову вибірки.
7. **Створення та навчання моделі SVM**: Створюємо модель SVC з лінійним ядром і навчаємо її на навчальних даних.
8. **Прогнозування та оцінка точності**: Прогнозуємо результати на тестових даних і обчислюємо точність моделі.
9. **Візуалізація**: Візуалізуємо результати, показуючи дані, тестові дані та лінію рішень, яку визначає модель SVM.



**Метод дерев рішень**

Дерево рішень — це один із популярних алгоритмів машинного навчання, який використовується для класифікації та регресії. Це графічна модель, що показує, як приймаються рішення на основі значень вхідних ознак. Дерево рішень нагадує структуру дерева, де кожен вузол відповідає перевірці певної умови (ознаки), а гілки представляють можливі результати цієї перевірки.

Основні характеристики дерева рішень:

Вузли:

Кореневий вузол: Це верхній вузол дерева, з якого починається процес прийняття рішень.

Внутрішні вузли: Ці вузли представляють перевірки або умови на певних ознаках.

Листові вузли: Це кінцеві вузли дерева, які представляють результати (клас для класифікації або значення для регресії).

Гілки: Гілки дерева показують результати перевірок і ведуть до інших вузлів або до листових вузлів.

Розгалуження: Процес розділення даних на основі умов, які перевіряються у внутрішніх вузлах. Це може бути бінарне (двостороннє) або множинне розгалуження.

Як працює дерево рішень:

Початок з кореневого вузла: Дерево починає з кореневого вузла, де оцінюється умова на основі вхідних ознак.

Прийняття рішень: В залежності від результату перевірки, дані направляються по одній з гілок до наступного вузла.

Повторення процесу: Цей процес повторюється, поки не буде досягнуто листового вузла, де буде прийнято рішення (клас або значення).

Прогнозування: Для нових зразків даних дерево рішень пройде через кореневий вузол, потім через внутрішні вузли, поки не досягне листового вузла, що надасть відповідний клас або значення.

**Простий приклад для дерев рішень**

Уявімо, що ми хочемо передбачити, чи підуть людина на прогулянку в залежності від погоди. У нас є кілька простих ознак:

Температура: високий чи низький.

Вологість: висока чи низька.

І ось наш набір даних:



Будуємо дерево рішень

Перша умова: дерево починає з аналізу температури.

Якщо температура висока, ми перевіряємо вологість.

Якщо вологість низька, то людина підуть на прогулянку (відповідь Так).

Якщо вологість висока, то людина не піде на прогулянку (відповідь Ні).

Якщо температура низька, то людина не піде на прогулянку (відповідь Ні).

Остаточне дерево:

 Температура?

 / \

 Висока Низька

 / \ |

 Низька Висока Ні

 | |

 Так Ні

Як працює дерево рішень:

Якщо температура висока і вологість низька, людина піде на прогулянку.

Якщо температура висока і вологість висока, людина не піде на прогулянку.

Якщо температура низька, незалежно від вологості, людина не піде на прогулянку.

Це дуже простий приклад, але дерево рішень може бути набагато складнішим, коли ознак більше або якщо дані мають більше можливих варіантів. Дерева рішень працюють добре для задач класифікації, де можна легко визначити умови, які розділяють класи.

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

# Створення складнішого набору даних

data = {

 'Credit\_Score': [600, 700, 750, 500, 400, 650, 680, 720, 690, 580, 710, 730, 800, 450, 700, 670, 760, 620, 740, 590],

 'Debt': [3, 2, 0, 5, 7, 2, 1, 0, 4, 6, 1, 0, 0, 8, 2, 3, 0, 4, 1, 5],

 'Employment\_Years': [1, 3, 10, 5, 0, 2, 8, 15, 12, 4, 9, 14, 20, 1, 11, 7, 17, 6, 13, 3],

 'Will\_pay': ['No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',

 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No']

}

# Перетворення даних у DataFrame

df = pd.DataFrame(data)

# Визначення ознак (features) і цільової змінної (target)

X = df[[ 'Credit\_Score', 'Debt', 'Employment\_Years']]

y = df['Will\_pay']

# Нормалізація ознак для рівномірного розподілу впливу

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Розділення даних на навчальну та тестову вибірки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3, random\_state=42)

# Створення більш складної моделі дерева рішень

model = DecisionTreeClassifier(criterion='entropy', max\_depth=5, min\_samples\_split=3, random\_state=42)

model.fit(X\_train, y\_train)

# Оцінка точності моделі

accuracy = model.score(X\_test, y\_test)

print(f'Точність моделі: {accuracy \* 100:.2f}%')

text\_representation = tree.export\_text(model, feature\_names=list(X.columns))

print(text\_representation)

# Візуалізація дерева рішень

plt.figure(figsize=(14, 8))

tree.plot\_tree(model, feature\_names=X.columns, class\_names=model.classes\_, filled=True)

plt.title('Більш деталізоване дерево рішень')

plt.show()

****

**Застосуємо нейронну мережу до розв’язання цієї задачі**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score

import warnings

warnings.filterwarnings("ignore") # Ігноруємо попередження для чистого виводу

# Створення складнішого набору даних

data = {

 'Credit\_Score': [600, 700, 750, 500, 400, 650, 680, 720, 690, 580, 710, 730, 800, 450, 700, 670, 760, 620, 740, 590],

 'Debt': [3, 2, 0, 5, 7, 2, 1, 0, 4, 6, 1, 0, 0, 8, 2, 3, 0, 4, 1, 5],

 'Employment\_Years': [1, 3, 10, 5, 0, 2, 8, 15, 12, 4, 9, 14, 20, 1, 11, 7, 17, 6, 13, 3],

 'Will\_pay': ['No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',

 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No']

}

# Перетворення даних у DataFrame

df = pd.DataFrame(data)

# Визначення ознак (features) і цільової змінної (target)

X = df[['Credit\_Score', 'Debt', 'Employment\_Years']]

y = df['Will\_pay']

# Перетворення цільової змінної у числовий формат

encoder = LabelEncoder()

y\_encoded = encoder.fit\_transform(y) # "No" -> 0, "Yes" -> 1

# Нормалізація ознак

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Розділення даних на навчальну та тестову вибірки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_encoded, test\_size=0.3, random\_state=42)

# Створення та навчання нейронної мережі

mlp = MLPClassifier(

 hidden\_layer\_sizes=(20, 10), # Збільшена кількість нейронів

 activation='relu',

 solver='adam',

 alpha=0.01, # Регуляризація для уникнення переобчислення

 learning\_rate\_init=0.001, # Оптимальний коефіцієнт навчання

 max\_iter=5000, # Більше ітерацій для стабільного навчання

 random\_state=42

)

mlp.fit(X\_train, y\_train)

# Оцінка точності моделі

y\_pred = mlp.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Точність моделі (MLP): {accuracy \* 100:.2f}%')

MLPClassifier — це клас з бібліотеки scikit-learn, який реалізує багатошаровий персептрон (MLP) для задач класифікації. Багатошаровий персептрон — це тип нейронної мережі, що складається з кількох шарів нейронів (вхідний шар, один або більше прихованих шарів і вихідний шар). Цей алгоритм використовує метод зворотного поширення помилки для навчання моделі на даних.

Основні параметри MLPClassifier:

hidden\_layer\_sizes: визначає кількість та розміри прихованих шарів (наприклад, (100,) означає один прихований шар з 100 нейронами).

activation: функція активації, наприклад, 'relu', 'tanh', 'logistic'.

solver: алгоритм оптимізації, що використовується для тренування, наприклад, 'adam', 'sgd', 'lbfgs'.

max\_iter: максимальна кількість ітерацій для навчання.

alpha: параметр регуляризації.

**Методи оцінки точності класифікації**

Уявімо, що у вас є модель регресії для виявлення спаму в електронній пошті, яка прогнозує значення від 0 до 1, що представляє ймовірність того, що певний лист є спамом. Прогноз 0,50 означає 50% ймовірність того, що лист є спамом, прогноз 0,75 означає 75% ймовірність, що лист є спамом, і так далі.

Ви хочете розгорнути цю модель у додатку електронної пошти для фільтрації спаму в окрему папку. Однак для цього вам потрібно перетворити необроблений числовий результат моделі (наприклад, 0,75) у одну з двох категорій: «спам» або «не спам».

Для цього ви обираєте порогову ймовірність, яку називають порогом класифікації. Приклади з ймовірністю вище порогового значення присвоюються позитивному класу, тобто класу, на який ви тестуєте (у цьому випадку, спам). Приклади з меншою ймовірністю належать до негативного класу, альтернативного класу (у цьому випадку не спам).

Вам може бути цікаво: що станеться, якщо прогнозоване значення дорівнює порогу класифікації (наприклад, оцінка 0,5, де поріг класифікації також дорівнює 0,5)? Обробка цього випадку залежить від конкретної реалізації, обраної для моделі класифікації. Бібліотека Keras прогнозує негативний клас, якщо оцінка і поріг однакові, але інші інструменти або платформи можуть обробляти цей випадок по-різному.

Уявімо, що модель оцінює одне електронне повідомлення як 0,99, прогнозуючи, що ймовірність того, що воно є спамом, становить 99%, а інший лист отримує оцінку 0,51, припускаючи, що ймовірність того, що воно є спамом, становить 51%. Якщо ви встановите поріг класифікації на рівні 0,5, модель класифікує обидва листи як спам. Якщо ви встановите порогове значення на рівні 0,95, лише лист з оцінкою 0,99 буде класифіковано як спам.

Хоча 0,5 може здатися інтуїтивним порогом, це не завжди є хорошою ідеєю, якщо вартість одного типу неправильних класифікацій більша, ніж іншого, або якщо класи несбалансовані. Якщо лише 0,01% електронних листів є спамом, або якщо неправильно позначити законний лист як спам гірше, ніж пропустити спам у вхідні, позначати все, що модель вважає спамом з ймовірністю 50% і більше, може призвести до небажаних результатів.

**Матриця плутанини**

У прикладі з класифікатором спаму, якщо ви розмістите істину у вигляді стовпців, а прогноз моделі — у вигляді рядків, результатом буде таблиця, яку називають **матрицею плутанини**.

|  |  |  |
| --- | --- | --- |
|  | **Фактичний позитивний** | **Фактичний негативний** |
| **Прогнозний позитивний** | Істинно позитивний результат (TP): спам-повідомлення правильно класифікується як спам. Це спам-повідомлення, автоматично відправлені в папку «Спам». | Хибне спрацьовування (FP): електронний лист, який не є спамом, помилково класифікований як спам. Це легітимні електронні листи, що потрапляють до папки «Спам». |
| **Прогнозний негативний** | Хибно негативний результат (FN): спам-повідомлення, помилково класифіковане як не спам. Це спам-повідомлення, які не відслідковуються спам-фільтром і потрапляють у вхідні. | Істинно негативний результат (TN): електронний лист, який не є спамом, правильно класифікований як не спам. Це легітимні електронні листи, що відправляються безпосередньо у поштову скриньку. |

Зверніть увагу, що сума в кожному рядку дає всі прогнозовані позитивні значення (TP + FP) та всі прогнозовані негативні значення (FN + TN), незалежно від їх достовірності. Тим часом, сума в кожному стовпці дає всі реальні позитивні результати (TP + FN) та всі реальні негативні значення (FP + TN), незалежно від класифікації моделі.

Якщо загальна кількість фактичних позитивних результатів не близька до загальної кількості фактичних негативних значень, набір даних є несбалансованим. Прикладом несбалансованого набору даних може бути набір із тисяч фотографій хмар, де цікавий вам рідкісний тип хмар, скажімо, спіральні хмари, з'являється лише кілька разів.

Різні порогові значення зазвичай призводять до різної кількості істинно позитивних і хибнопозитивних результатів, а також істинно негативних і хибнонегативних результатів.

У міру збільшення порога модель, скоріш за все, буде прогнозувати менше позитивних результатів, як істинних, так і хибних. Класифікатор спаму з порогом 0,9999 позначить електронний лист як не спам лише в тому випадку, якщо він вважає, що класифікація має ймовірність не менше 99,99%. Це означає, що дуже ймовірно пропустити справжнє повідомлення.

Як істинні, так і хибно негативні результати зростають. У міру збільшення порога модель, скоріш за все, буде прогнозувати більше негативних результатів, як істинних, так і хибних. При дуже високому порозі майже всі електронні листи, як спам, так і не спам, будуть класифіковані як не спам.

**Основні показники моделі класифікації**

Істинні та хибні позитивні й негативні результати використовуються для розрахунку кількох корисних показників для оцінки моделей. Які метрики оцінки є найзначнішими, залежить від конкретної моделі та задачі, вартості різних неправильних класифікацій і того, чи є набір даних збалансованим або несбалансованим.

Точність — це частка всіх класифікацій, які були правильними, як позитивними, так і негативними. Математично це визначається як:



де:

* TP — істинно позитивні результати,
* TN — істинно негативні результати,
* FP — хибнопозитивні результати,
* FN — хибнонегативні результати.

Ідеальна модель повинна мати нуль хибнопозитивних і хибнонегативних результатів і, отже, точність 1,0 або 100%.

Оскільки вона включає всі чотири результати матриці плутанини (TP, FP, TN, FN), при наявності збалансованого набору даних з однаковою кількістю прикладів у обох класах точність може слугувати грубою мірою якості моделі. Через це вона часто є метрикою оцінки за замовчуванням, що використовується для загальних або невизначених моделей, які виконують загальні або невизначені завдання.

Однак, коли набір даних несбалансований або помилка одного типу (FN або FP) обходиться дорожче, ніж інша, що має місце в більшості реальних застосувань, краще оптимізувати один з інших показників.

**Істинно позитивний рівень (TPR)** або частка всіх фактичних позитивних результатів, які були правильно класифіковані як позитивні, також відомий як **відгук**.

Математично його можна визначити як:



Хибно негативні результати — це фактичні позитивні результати, які були помилково класифіковані як негативні, тому вони з'являються в знаменнику. **У прикладі з класифікацією спаму відгук вимірює частку спам-повідомлень, які були правильно класифіковані як спам**. Ось чому інше назва відгуку — **ймовірність виявлення**: воно відповідає на запитання: «Яка частка спам-повідомлень виявляється цією моделлю?»

**Ложнопозитивний показник (FPR)** — це частка всіх фактичних негативних результатів, які були помилково класифіковані як позитивні, також відома як **ймовірність хибної тривоги**.

Математично це визначається як:



Хибнопозитивні результати — це фактичні негативні результати, які були неправильно класифіковані, тому вони з'являються в знаменнику. **У прикладі з класифікацією спаму FPR вимірює частку легітимних електронних листів, які були помилково класифіковані як спам**, або рівень хибних спрацьовувань моделі.

**Точність** — це частка всіх позитивних класифікацій моделі, які насправді є позитивними.

Математично це визначається як:



**У прикладі з класифікацією спаму точність вимірює частку листів, класифікованих як спам, які насправді були спамом.**

**Загальна рекомендація з використання метрик:**

**Відгук (TPR)**: Використовуйте, коли хибно негативні результати дорожчі за хибнопозитивні.

**Хибнопозитивний показник (FPR)**: Використовуйте, коли хибнопозитивні результати обходяться дорожче, ніж хибно негативні.

**Точність**: Використовуйте, коли дуже важливо, щоб позитивні прогнози були точними.