МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Машинне навчання

27 серпня 2023 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Кочура Ю. П., Гордієнко Ю. Г.

Машинне навчання

Занурення в машинне навчання

Навчальний посібник

Рекомендовано для здобувачів ступеня “магістр”, які навчаються за освітніми програмами «Комп’ютерні системи та мережі» спеціальність 123 «Комп’ютерна інженерія» та «Інженерія програмного забезпечення комп’ютерних систем» спеціальність 121 «Інженерія програмного забезпечення»

Електронне видання

Навчальний посібник перебуває на етапі написання. Останню версію цього документа можна знайти тут:

Ми були б дуже вдячні за надану допомогу та пропозиції щодо наповнення та покращення цього документа. Повідомте про помилки, неточності

Чернетка. Версія 0.1.0

Зміст

Розділ 1

Загальна теорія

“Я не маю ніякого таланту, тільки допитливість.
Відповідно, відпадає питання про спадковість.”

– Альберт Ейнштейн

1.1 Мотивація та інтуїція

Для більшості задач машинного навчання вибір хороших ознак має першочергове значення. Навіть найкращий алгоритм не зможе продемонструвати хороших результатів, якщо для його навчання було використано погані ознаки. З іншого боку, вибір ознак часто є вкрай нетривіальним завданням. Наприклад, розглянемо кольорове зображення обличчя людини. Нехай у цьому випадку зображення обличчя людини буде представлено інтенсивністю пікселя для трьох кольорів: червоного, зеленого та синього (якщо використовується RGB-кодування). Тому, 1M піксельне кольорове зображення буде мати 3M ознак за якими ми можемо навчати нашу модель. З іншого боку, вираз обличчя людини, ймовірно, можна охарактеризувати ≤ 56 ознаками (на обличчі людини є ~ 56 м’язів). Отже, для ідентифікації конкретних виразів обличчя, дані великої розмірності можуть бути представлені відповідними ознаками меншої розмірності.

Таким чином, ідеальний екстрактор (видобувач) ознак буде приймати на вхід зображення обличчя, а на виході видавати видобуті ознаки, що характеризують вираз обличчя людини. Однак, на теперішній час, не існує ідеальних способів як це зробити. У традиційних методах розпізнавання образів, які були розроблені починаючи з 50-х років, ці екстрактори ознак були жорстко закодовані на основі суб’єктивної інтуїції дослідників. Основна ідея, яка прийшла до нас разом з нейронними мережами, полягає в тому, що хороші ознаки можуть бути вивчені мережею безпосередньо з даних, таким чином більше непотрібно досліднику видобувати їх вручну.

З іншого боку, базова модель слабо розвинулася з часу її створення починаючи з 1950-х років. Перша машина («Марк-1», Френк Розенблат в 1957 р.), яка реалізовувала алгоритм перцептрона (нейрон Маккалока-Пітса) була лінійним класифікатором, побудованим поверх простого жорстко прописаного екстрактора ознак. До сьогоднішнього дня на практиці для вирішення прикладних задач машинного навчання використовують видобування ознак.

1.1.1 Інжиніринг ознак

Інжиніринг (конструювання) ознак є дуже важливим етапом для створення моделі. Він передбачає видобування та вибір ознак. Під час видобування ознак витягуються з даних усі ознаки, які характеризують поставлену задачу. Під час вибору – визначаються усі найбільш важливі ознаки з метою покращення продуктивність моделі.

На рисунку 1.1 розглянуто ще один приклад класифікації зображень. Ручне вилучення ознак з даних вимагає глибоких знань як задачі, яка вирішується, так і предметної галузі. Крім того, цей спосіб є трудомістким. Ми можемо автоматизувати процес конструювання ознак за допомогою глибинного навчання!

Звичайно, якби нейронна мережа була представлена лише лінійними шарами (нейронами), сукупний ефект також був би лінійним і ми могли б згорнути всю архітектуру мережі лише в один шар (нейрон). Це пояснюється тим, що результат комбінації лінійних перетворень залишається лінійним перетворенням. З іншого боку, введення нелінійності відкриває можливість для побудови мережі з кількох шарів, таким чином кожен окремий нейрон та шар буде вивчати різні ознаки.

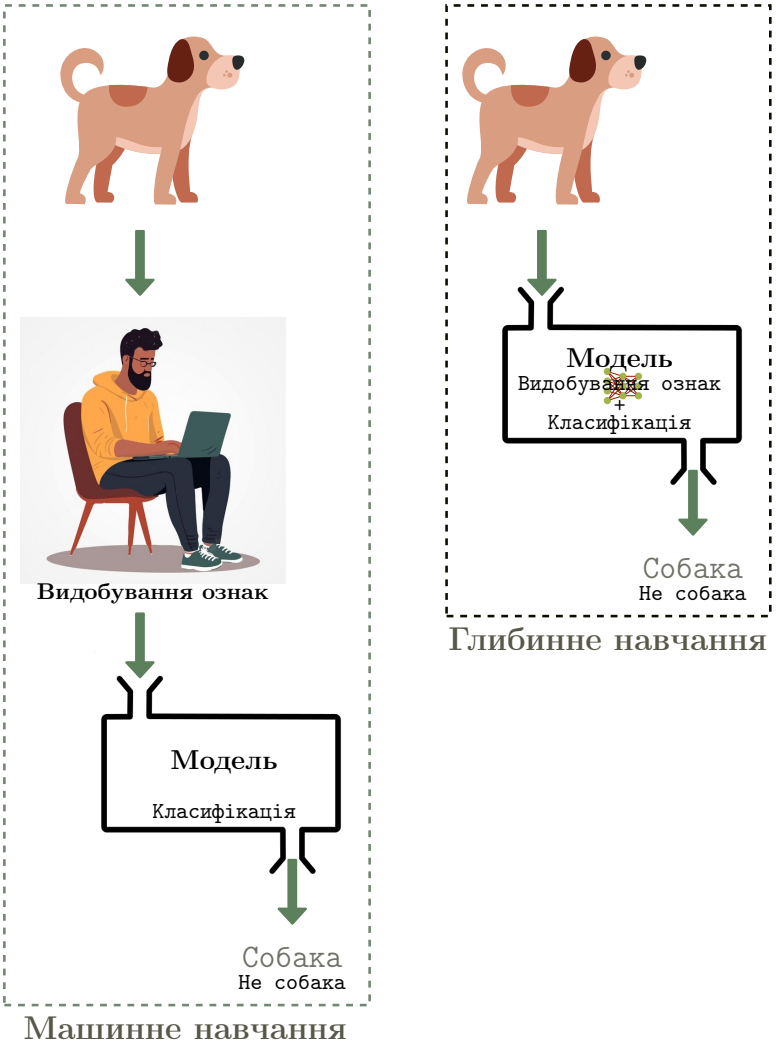

Рис. 1.1: Інжиніринг ознак в машинному навчанні та глибинному навчанні

—- Виконуючи це завдання, Ви познайомитесь з математичним апаратом, який лежить в основі лінійної регресії для моделювання взаємозв'язку між скалярною залежною змінною $\hat{y}$ та однією або кількома незалежними змінними $\boldsymbol{X}$. Отриманий досвід буде корисним для пошуку тренду в даних, коли залежна змінна, яку намагаємось змоделювати приймає дійсні значення ($\hat{y} \in \mathbb{R}$).

1.2 Завдання з програмування

Відкрийте завдання:
https://nbviewer.org/github/YKochura/ai-lab/blob/main/linear-regression/linear_regression.ipynb

На рисунку 1.2 показана модель лінійної регресії.

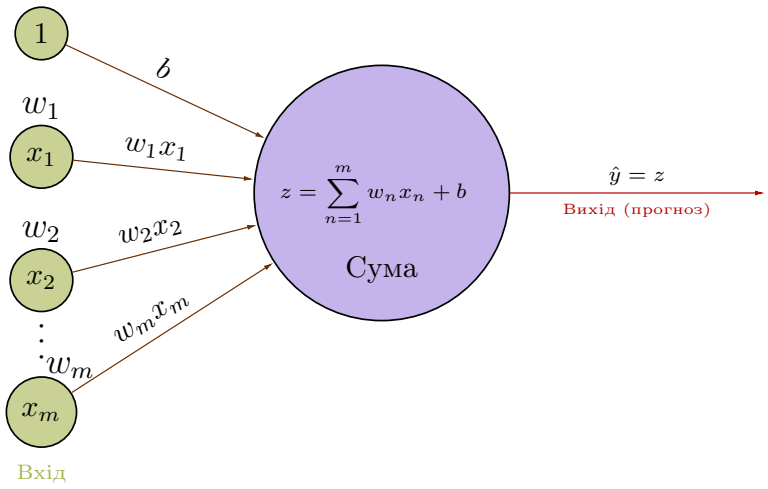

Рис. 1.2: Модель лінійної регресії

Метод лінійної регресії полягає у знаходженні лінійної комбінації вхідних ознак (зважена сума) з урахуванням зміщення:

$$\hat{y} = z = \sum_{i=1}^{m} w_i x_i + b \tag{1.1}$$

Вам потрібно імплементувати декілька функцій для прямого та зворотного поширення лінійної регресії. Функції, які потрібно імплементувати позначено у завданні так:

```
# TODO
```

Розміщуйте свою реалізацію функцій між

```
# BEGIN_YOUR_CODE

# END_YOUR_CODE
```

Крок 0: Ініціалізувати ваги та зсув

```
# TODO
def parameters_inititalization(m):
    """
    Ця функція ініціалізує вектор-рядок випадкових дійсних значень ваг форми (1, m),
    отриманих з нормального розподілу та зсув (довільне дійсне значення)

    Параметри:
    m -- кількість вхідних ознак для кожного навчального прикладу

    Повертає:
    W -- вектор-рядок ваг форми (1, m)
    b -- зсув (скаляр)
    """

    # BEGIN_YOUR_CODE
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

Крок 1: Обчислити лінійну комбінацію вхідних ознак та ваг, включаючи зсув

```
# TODO
def forwardPropagate(X, W, b):
    """
    Ця функція обчислює лінійну комбінацію вхідних ознак та ваг, включаючи зсув

    Параметри:
    X -- вхідний вектор ознак форми (1, 375)
    W -- вектор-рядок ваг форми (1, m)
    b -- зсув моделі (скаляр)

    Повертає:
    z -- загальна зважена сума вхідних ознак, включаючи зсув
    y_hat -- прогноз моделі
    """

    # BEGIN_YOUR_CODE
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

Крок 2: Обчислити усереднену втрату на всьому навчальному наборі даних. Цільова функція

```
# TODO
def cost(n, y_hat, y_true):
    """
    Ця функція обчислює середнє квадратичне відхилення на всьому навчальному наборі даних

    Параметри:
    n -- загальна кількість навчальних прикладів
    y_hat -- вихідне значення лінійної регресії
    y_true -- істинне значення залежної змінної

    Повертає:
    J --  середнє квадратичне відхилення на всьому навчальному наборі даних
    """

    # BEGIN_YOUR_CODE
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

Крок 3: Розрахувати градієнти цільвої функції відносно ваг та зсуву

```
# TODO
def backwardPropagate(n, X, y_hat, y_true):
    """
    Ця функція обчислює градієнти цільвої функції відносно ваг та зсуву

    Параметри:
    n -- загальна кількість навчальних прикладів
    X -- вхідний вектор ознак форми (1, 375)
    y_hat --  вихідне значення лінійної регресії
    y_true -- істинне значення залежної змінної

```

```
    Повертає:
    dW --  градієнт цільової функції відносно ваг моделі
    db -- градієнт цільової функції відносно зсуву моделі
    """

    # BEGIN_YOUR_CODE
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

Крок 4: Оновити ваги та зсув

```
# TODO
def update(alpha, dW, db, W, b):
    """
    Ця функція оновлює навчальні параметри моделі (ваги та зсув ) у напрямку мінімізації цільової функції

    Параметри:
    alpha -- швидкість  навчання (крок навчання)
    dW --  градієнт цільової функції відносно ваг моделі
    db -- градієнт цільової функції відносно зсуву моделі
    W -- вектор-рядок ваг моделі форми (1, m)
    b -- зсув моделі (скаляр)

    Повертає:
    W -- оновлений вектор-рядок ваг моделі форми (1, m)
    b -- оновлений зсув моделі (скаляр)
    """


    # BEGIN_YOUR_CODE
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

Коли усі функції будуть реалізовані, дослідіть збіжність оптимізаційного алгоритму залежно від кількості зроблених ітерацій та швидкості навчання. Подайте власні спостереження, щодо вливу цих двох гіперпараметрів на навчання моделі.

Отримані результати порівняйте між собою та подайте власні спостереження, щодо вливу *alpha* на навчання моделі.

1.3 Оцінювання

- 60% – завдання з програмування
- 40% – підготовлено звіт у якому досліджено збіжність оптимізаційного алгоритму залежно від швидкості навчання та кількості ітерацій навчання. Очікується формальний звіт, написаний в LaTeX.

Розділ 2

Лабораторна 2: Логістична регресія

“Грам власного досвіду коштує дорожче тонни чужих повчань.”

– Магатма Ґанді

2.1 Вступ

Виконуючи це завдання, Ви познайомитесь з математичним апаратом, який лежить в основі навчання найпростішої нейронної мережі, що складається з одного нелінійного нейрона. Ця модель носить назву логістична регресія. Отриманий досвід буде корисним для подальшого розуміння принципу роботи глибинних нейронних мереж.

2.2 Класичне програмування та машинне навчання

`Комп'ютери` та `обчислення` допомагають нам досягати більш складних цілей і кращих результатів у вирішенні проблем, ніж ми могли б досягти самі. Однак, багато сучасних завдань вийшли за рамки обчислень через один основний обмежуючий фактор: **традиційно, комп'ютери можуть дотримуватися лише конкретних вказівок/інструкцій, які їм дають.**

Вирішення проблем з `програмування` вимагає написання конкретних покрокових інструкцій, які має виконувати комп'ютер. Ми називаємо ці кроки алгоритмами. У цьому випадку, комп'ютери можуть допомогти нам там, де ми:

1. Розуміємо як вирішити проблему.
2. Можемо описати проблему за допомогою чітких покрокових інструкцій, які комп'ютер може зрозуміти.

Методи машинного навчання дозволяють комп'ютерам “учитися” на прикладах. Вирішення проблем із застосуванням машинного навчання вимагає виявлення деякого шаблону[1], а потім, коли такий шаблон готовий, дозволяють, наприклад, нейронній мережі вивчити карту переходів між вхідними та вихідними даними. Ця особливість відкриває нові типи проблем, де комп'ютери можуть допомогти нам у їх розв'язанні, за умови, коли ми:

1. Визначили шаблон проблеми.
2. Маємо достатньо даних, що ілюструють шаблон.

На рисунку 2.1 графічно показана відмінність класичного програмування від машинного навчання.

[1]Пошук прикладів, що висвітлюють обидві сторони шаблону: вхід і вихід.

Рис. 2.1: Відмінність класичного програмування від машинного навчання.

2.3 Письмове завдання

Покажіть, що похідна сигмоїди дорівнює цьому виразу:

$$\frac{d\hat{y}}{dz} = \frac{d\sigma(z)}{dz} = \sigma(z)\big(1 - \sigma(z)\big), \tag{2.1}$$

де $\hat{y} = \sigma(z) = \frac{1}{1+\exp(-z)}$

2.4 Завдання з програмування

Відкрийте завдання:
https://nbviewer.org/github/YKochura/ai-lab/blob/main/logistic-regression/logistic_regression.ipynb

Вам потрібно імплементувати декілька функцій для прямого та зворотного поширення одного навчального прикладу логістичної регресії. Функції, які потрібно імплементувати позначено у завданні так:

```
# TODO
```

Розміщуйте свою реалізацію функцій між

```
# BEGIN_YOUR_CODE

# END_YOUR_CODE
```

Коли усі функції будуть реалізовані, дослідіть два пункти, які подані у кінці завдання:

1. Повторіть кроки 1-5 та з'ясуйте як зміниться значення цільової функції, ваг та зсуву моделі. Швидкість навчання використовуйте $alpha = 0.0001$
2. Повторіть кроки 0-5 для більшої швидкості навчання $alpha = 0.003$. Порівняйте отримані результати для $alpha = 0.0001$

2.5 Оцінювання

Ваша оцінка за виконання завдання буде залежати від:

- 10% – письмове завдання
- 60% – завдання з програмування
- 30% – підготовлено звіт у якому подано розв'язок письмового завдання та досліджено зміну цільової функції, ваг та зсуву моделі залежно від швидкості навчання та кількості ітерацій навчання. Очікується формальний звіт, написаний в LaTeX.