Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Інститут комп’ютерних наук та інформаційних технологій
Кафедра систем штучного інтелекту

Звіт до розрахунково-графічної роботи

з дисципліни

«Математичні методи дослідження операцій»

Виконала:

студентка групи ШІ – 23

Брездень Софія

Викладач:

Шиманський В. М.

Львів 2024 р.

Зміст

Вступ 3
Змістовна постановка задачі операційного дослідження 5
Формальна постановка задачі 7
Аналіз методів та алгоритмів розв'язування задачі 10
Описання ключових моментів програмної реалізації 12
Аналіз та інтерпретація отриманих результатів 17
Список літератури 23
Додаток 24

Вступ

Лінійна регресія є однією з найпоширеніших та найбільш простих технік у статистиці та машинному навчанні для аналізу залежностей між змінними. У своїй основі вона передбачає побудову лінійної моделі, яка описує взаємозв'язок між незалежними змінними (відомими також як пояснюючі фактори) та залежною змінною.

Цей метод завдяки своїй простоті та ефективності знайшов широке застосування у багатьох галузях, включаючи економіку, фінанси, медицину, соціологію та інші. Він може бути особливо корисним в ситуаціях, коли потрібно швидко отримати базове уявлення про зв'язок між змінними або коли складність моделі не є критичним фактором.

Одним з ключових переваг лінійної регресії є її інтерпретованість. Модель може бути легко пояснена навіть людям без фахової підготовки в області аналізу даних. Лінійна регресія не є обчислювально важкою і, отже, добре підходить у випадках, коли масштабування є важливим. Наприклад, модель може добре масштабуватися при збільшенні обсягу даних (великі дані).

Найшвидший градієнтний спуск - це алгоритм оптимізації, який використовує градієнт (вектор похідних) функції втрат для знаходження мінімуму цієї функції. Основна ідея полягає в тому, щоб крок за кроком оновлювати параметри моделі в напрямку, протилежному до напрямку градієнта, досягаючи таким чином локального або глобального мінімуму функції втрат.

У контексті лінійної регресії, ми шукаємо такі значення параметрів моделі, які мінімізують різницю між фактичними та прогнозованими значеннями цільової змінної. Задача полягає в тому, щоб мінімізувати функцію втрат, яка відображає різницю між спостережуваними значеннями та тими, які передбачає модель.

Алгоритм працює шляхом ітеративного оновлення параметрів моделі в напрямку, протилежному до градієнта функції втрат. Градієнт показує напрям, в якому функція швидко змінюється, тому оновлення параметрів в цьому напрямку допомагає зменшити значення функції втрат.

Цей процес повторюється до досягнення критерію зупинки, такого як задана кількість ітерацій або досягнення заданої точності.

Градієнтний спуск може бути ефективним методом для оптимізації параметрів у лінійній регресії через кілька важливих причин. По-перше, функція втрат у лінійній регресії, яка визначає різницю між фактичними та прогнозованими значеннями, є квадратичною, що робить її гладкою та легкою для оптимізації. Це сприяє швидкому збіганню алгоритму градієнтного спуску до мінімуму. Крім того, функція втрат у лінійній регресії лінійна відносно параметрів моделі, що дозволяє обчислити градієнт аналітично, що робить градієнтний спуск

ефективним. Таким чином, градієнтний спуск є ефективним методом для знаходження оптимальних параметрів у лінійній регресії завдяки комбінації простоти функції втрат та лінійності моделі.

Регуляризація у лінійній регресії є методом контролю складності моделі шляхом додавання додаткових обмежень до функції втрат. Головна мета полягає в тому, щоб уникнути перенавчання та покращити узагальнюючі здатності моделі, зокрема в умовах обмежених даних або коли кількість ознак велика.

У лінійній регресії застосовуються два основних типи регуляризації: L1 (лассо) і L2 (Ridge). Обидва методи додають штраф до функції втрат, але з різними способами врахування параметрів моделі.

В основі регуляризації L1, також відомої як Лассо (оператор найменшого абсолютного стиснення і відбору), лежить проста, але потужна модифікація функції втрат, що використовується в моделі машинного навчання. Стандартна функція втрат, яка вимірює різницю між прогнозованими і фактичними значеннями, розширюється шляхом додавання штрафу.

L2 регуляризація, широко відома як регресія Ridge, вводить інший тип штрафу у функцію втрат моделі машинного навчання порівняно з L1 регуляризацією. У регуляризації L2 штраф є сумою квадратів коефіцієнтів моделі.

Обидва методи регуляризації можуть бути ефективними інструментами для покращення прогностичних здатностей лінійної регресії та зменшення ризику перенавчання, що є важливими у контексті роботи з реальними даними.

Змістовна постановка задачі операційного дослідження

Задача передбачає створення моделі для прогнозування цін на мобільні телефони на основі різних технічних характеристик. Цей прогноз допоможе виробникам мобільних телефонів визначити оптимальні ціни на нові моделі, враховуючи їхні технічні параметри. Окрім того, споживачі зможуть оцінити, чи відповідає запропонована ціна специфікаціям телефону, що підвищує прозорість ринку та покращує процес прийняття рішень під час купівлі.

Вхідні дані:

Для розв'язання задачі використовуватиметься датасет, який містить більше 200 прикладів мобільних телефонів. Кожен приклад описаний наступними характеристиками:

- Розмір екрану (дюйми): Діагональний розмір екрана мобільного телефону в дюймах.
- ОЗП (ГБ): Об'єм оперативної пам'яті (RAM) в гігабайтах (ГБ).
- Обсяг сховища (ГБ): Ємність внутрішнього сховища мобільного телефону в гігабайтах (ГБ).
- Ємність батареї (мАг): Ємність батареї в міліампер-годинах (мАг).
- Якість камери (МП): Якість камери в мегапікселях (МП).
- Ціна ($): Ціна мобільного телефону в доларах ($).

У задачі прогнозування цін на мобільні телефони цільовою змінною є "Ціна ($)". Це змінна, яку ми прагнемо передбачити на основі інших характеристик мобільних телефонів, таких як розмір екрану, обсяг оперативної пам'яті, ємність внутрішнього сховища, ємність батареї та якість камери.

Для побудови прогнозної моделі буде використовуватися лінійна регресія, а оптимізація вибору параметрів моделі здійснюватиметься за допомогою методу найшвидшого градієнтного спуску. Цей метод дозволяє ефективно мінімізувати функцію втрат, а саме середньоквадратичну похибку, яка визначає різницю між фактичними та прогнозованими цінами.

У цій задачі ми хочемо знайти оптимальні значення вагових коефіцієнтів, які мінімізують середньоквадратичну помилку (MSE) між прогнозованими та фактичними цінами мобільних телефонів. Найшвидший градієнтний спуск є методом оптимізації, який використовується для знаходження мінімуму (або максимуму) функції шляхом послідовного переходу у напрямку найшвидшого зменшення (або збільшення) значення функції. У контексті лінійної регресії, градієнтний спуск використовується для оновлення параметрів моделі (вагових коефіцієнтів) з кожною ітерацією так, щоб мінімізувати функцію втрат. Цей процес триває до досягнення зазначеного критерію зупинки, такого як досягнення певного рівня точності або певної кількості ітерацій.

Метою є побудова регресійної моделі, яка зможе точно прогнозувати ціну мобільного телефону на основі зазначених характеристик. Успіх буде вимірюватися за допомогою метрик точності, такої як середньоквадратична помилка (MSE) між прогнозованими та фактичними цінами.

Розв'язання задачі

Для розв'язання задачі прогнозування цін мобільних телефонів використовуватиметься метод лінійної регресії з оптимізацією параметрів за допомогою найшвидшого градієнтного спуску. Першим етапом є підготовка даних. Цей процес включає збір та огляд даних для виявлення пропущених значень та аномалій, які можуть вплинути на результати моделі. Після цього виконується попередня обробка даних, яка включає нормалізацію або стандартизацію числових характеристик для забезпечення рівномірного внеску кожної змінної у модель. Після підготовки даних вони розподіляються на навчальну та тестову вибірки. Навчальна вибірка використовується для навчання моделі, тоді як тестова - для оцінки її продуктивності. Наступним етапом є побудова моделі лінійної регресії. Це включає визначення цільової змінної (ціна) та характеристики телефонів, формулювання моделі і вирішення задачі оптимізації параметрів за допомогою найшвидшого градієнтного спуску. Після побудови моделі вона оцінюється на тестовій вибірці, вимірюються метрики точності, такі як середньоквадратична помилка , і проводиться аналіз її відповідності даним. Завершальним етапом є тестування моделі на нових даних і її впровадження в практичне використання після успішного тестування.

Для покращення результатів моделі про прогнозування цін на мобільні телефони, я вирішила порівняти вплив регуляризації L1 та L2. Регуляризація впливає на процес навчання моделі, допомагаючи уникнути перенавчання шляхом додавання до функції втрат штрафного елемента, який контролює складність моделі. Регуляризація L1 (Lasso) сприяє створенню розріджених моделей, зводячи до нуля коефіцієнти деяких змінних, тоді як регуляризація L2 (Ridge) зменшує коефіцієнти змінних, не зводячи їх до нуля.

Після того, як моделі були оптимізовані за допомогою обох видів регуляризації, я оцінила їх точність. Для цього використовуються відповідні метрики, такі як середньоквадратична похибка, яка дозволяє кількісно оцінити похибки моделей та їх здатність точно передбачати ціни на мобільні телефони. Порівнюючи результати моделей з регуляризацією L1 та L2, я можу визначити, яка з них краще підходить для мого набору даних і забезпечує найкращі прогнози.

Формальна постановка задачі

Раніше ми припускали, що незалежними ознаками є характеристики телефона, тобто X, а відповідна ціна телефона y - залежна змінна. Якщо припустити, що існує лінійний зв'язок між X та y, тоді ціна телефона може бути передбачена за допомогою:

$$Y\_pred = W * X + b$$

де Y_pred - це передбачене значення (прогнозована величина)

W - це ваговий коефіцієнт (вектор параметрів), який помножений на вхідну змінну X

b - це зсув

Це базове рівняння лінійної регресії, де модель намагається знайти оптимальні значення вагових коефіцієнтів W та зсуву b, щоб найкраще відобразити залежність між вхідними даними X та вихідними даними Y_pred. Це досягається шляхом мінімізації функції втрат.

Ми визначаємо функцію втрат, щоб оцінити, наскільки добре наша модель працює. Для лінійної регресії часто використовується середньоквадратична помилка (Mean Squared Error, MSE), яка обчислюється як середнє значення квадратів відхилень між фактичними значеннями Y і передбаченими значеннями Y_pred:

$$Loss\ function = \frac{1}{n}\sum_{i=1}^{n}({Y_{pred}}^{(i)} - y^{(i)})^2$$

де n - кількість спостережень, а i - індекс кожного спостереження.

Ця функція втрат визначає, наскільки добре наша модель працює на навчальних даних, порівняно з фактичними значеннями. Буде використовуватись найшвидший градієнтний спуск для пошуку мінімуму цієї функції втрат (Loss function). Градієнтний спуск - це метод оптимізації, який дозволяє знайти мінімум функції втрат, шляхом ітеративного крокового руху в напрямку найшвидшого зменшення функції. Після обчислення градієнтів функції втрат відносно параметрів W та b, ми оновлюємо їх значення згідно з такими правилами:

$$W = W - beta * \frac{dLoss\ function}{dW}$$
$$b = b - beta * \frac{dLoss\ function}{db}$$

де beta - крок спуску (learning rate), який визначає швидкість збіжності градієнтного спуску

Швидкість спуску beta підбирається таким чином, щоб мінімізувати функцію втрат:

$$Loss\ function(W - beta * \frac{dLoss\ function}{dW}, b - beta * \frac{dLoss\ function}{db})$$

Це може бути виконано, наприклад, за допомогою методу золотого перетину або інших методів одновимірної оптимізації. Мною буде використано метод Фібоначчі для мінімізації вказаної функції. Після цього обновлення значень параметрів W та b здійснюється для того, щоб мінімізувати функцію втрат.

Оновлені значення параметрів W та b допомагають покращити точність моделі і підвищити її здатність до передбачення цільової змінної y на основі вхідних ознак X. Ми продовжуємо оновлювати значення параметрів W та b за допомогою градієнтного спуску, поки не досягнемо зупинки за певною умовою (наприклад, задана кількість ітерацій або величина зміни функції втрат менше заданого порогу).
Таким чином ми оптимально обрали значення W, b і можемо обрахувати яку середньоквадратичну помилку вони дають таким чином підставивши знайдені числові значення W, b:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - (W * X + b))^2$$

Таким чином ми отримали підібране значення W, b та оцінку яку похибку дає побудована нами модель.

Для покращення результатів моделі про прогнозування цін на мобільні телефони, було використано функцію втрат з L1 регуляризацією (також відома як Mean Absolute Error з регуляризацією), яка визначається як:

$$Loss\ function\ L1 = \frac{1}{n}\sum_{i=1}^{n}({Y_{pred}}^{(i)} - y^{(i)})^2 + \alpha * \sum_{j=1}^{m} | W_j |$$

де α - параметр регуляризації, який контролює силу штрафу

Мета полягає в тому, щоб знайти значення ваг W та зсуву b, які мінімізують функцію втрат Loss function L1 , включаючи в розрахунок штраф за величину ваг. Це досягається шляхом оптимізації параметрів W та b за допомогою методів градієнтного спуску. Як було згадано раніше, за допомогою найшвидшого градієнтного спуску ми оновлюємо W,b поки не виконається умова виходу. Також було обраховано значення середньоквадратичної похибки для оцінки доцільності застосування L1 регуляризації для мого датасету.

Також для L2 регуляризації функція втрат з L2 регуляризацією виглядає наступним чином:

$$Loss\ function\ L2 = \frac{1}{n}\sum_{i=1}^{n}({Y_{pred}}^{(i)} - y^{(i)})^2 + \alpha * \sum_{j=1}^{m} W_j^2$$

де α - параметр регуляризації, який контролює силу штрафу

Завдання полягає в тому, щоб визначити значення ваг W і зсуву b, які зведуть до мінімуму функцію втрат Loss function L2, враховуючи штраф за величину ваг. Аналогічно це досягається шляхом обрання оптимальних параметрів W і b за допомогою методів градієнтного спуску. Для визначення необхідності застосування L2 регуляризації було також обраховано MSE для порівняння з L1 регуляризацією.

:

Аналіз методів та алгоритмів розв'язування задачі

Лінійна регресія є базовою технікою в машинному навчанні та статистиці для моделювання відносин між змінними. Окрім найшвидшого градієнтного спуску,який я обрала, існує кілька інших методів для розв'язання задачі лінійної регресії. Кожен з цих методів має свої переваги та недоліки, і їх вибір залежить від конкретних характеристик задачі, обсягу даних, наявних обчислювальних ресурсів тощо. Давайте розглянемо найбільш поширені:

1. Метод найменших квадратів (Ordinary Least Squares, OLS)

Метод найменших квадратів є одним із найпоширеніших методів розв'язання задачі лінійної регресії. Мета методу полягає в мінімізації суми квадратів різниць між спостережуваними значеннями та передбаченими значеннями.

Формально, розв'язок для коефіцієнтів W та b у векторизованій формі можна знайти як:

$$W = (X^T * X)^{-1} X^T y$$

$$b = \bar{y} * W^T \bar{X}$$

де $\bar{y}$- це середнє значення цільової змінної y, $\bar{X}$ - середнє значення матриці ознак X

Перевагою даного методу є точність, а саме те, що він забезпечує точний розв'язок для лінійних задач. Крім того, для невеликих розмірів даних цей метод є швидким і ефективним. Однак, цей метод має і недоліки. Одним з них є обчислювальна складність, оскільки для великих розмірів даних обчислення інверсії матриці може бути дуже затратним. Також метод є чутливим до мультиколінеарності: коли ознаки сильно корелюють між собою, інверсія матриці може бути нестабільною.

2. Стохастичний градієнтний спуск (Stochastic Gradient Descent, SGD)

Стохастичний градієнтний спуск (SGD) є варіантом методу градієнтного спуску, який використовується для мінімізації функції втрат у машинному навчанні та статистиці. Основна ідея полягає в тому, щоб оновлювати параметри моделі поступово, використовуючи одну або декілька вибірок даних замість всього набору даних. Перевагою цього методу є його швидкість обчислень. Оскільки SGD оновлює параметри після кожного зразка або невеликої групи зразків, він часто може збігатися швидше, ніж звичайний градієнтний спуск, який обробляє весь набір даних на кожній ітерації. Ще однією перевагою є можливість обробки великих наборів даних. SGD може працювати з дуже великими наборами даних, оскільки він не вимагає завантаження всього набору даних у пам'ять. Недоліком є нестабільність алгоритму. Через стохастичність, SGD може бути менш стабільним у порівнянні з методами, які використовують весь набір

даних для оновлень. Це може вимагати більшої кількості ітерацій для досягнення збіжності та робити процес навчання довшим. Важливо також правильно вибрати гіперпараметри, зокрема розмір кроку (learning rate). Вибір відповідного розміру кроку є критичним для ефективності SGD. Неправильний вибір цього параметру може призвести до повільної збіжності або навіть до збіжності до неправильного мінімуму. Занадто великий розмір кроку може призвести до нестабільного навчання, тоді як занадто малий — до повільного прогресу.

3. Метод імпульсів (Momentum Method)

Метод імпульсів є вдосконаленням класичного градієнтного спуску, що допомагає прискорити збіжність до оптимального рішення та зменшити коливання під час навчання. Основна ідея в тому, що метод імпульсів використовує попередні градієнти для того, щоб зберігати напрямок руху і не "гальмувати" на кожній ітерації. Це досягається шляхом додавання до поточного градієнту частки попереднього градієнту.

При порівнянні методів, можна зазначити, що GD найточніший, але може бути повільним на великих наборах даних. SGD та метод імпульсів швидші та ефективні для великих даних, але вимагають налаштування гіперпараметрів. Адаптивні методи, такі як Adagrad, RMSprop та Adam, автоматично налаштовуються для кожного параметра, що робить їх популярними для практичного застосування.

Описання ключових моментів програмної реалізації

Мій датасет збережений у форматі csv і для отримання коректних даних мені потрібно правильно зчитати мої дані та записати ознаки в X а цільову змінну “Price” в y. Було створено функцію read, яка відкриває файл, читає його вміст рядок за рядком та перетворює кожен рядок у словник, використовуючи csv.DictReader. Потім ці словники зберігаються в список data, який повертається функцією. Функція receive(data) обробляє отримані дані, визначаючи список ознак feature_names, які будуть використовуватися у моделі. Ознаки обираються за їхніми назвами, що передаються у списку. Після цього функція витягує значення ознак та ціну з кожного рядка даних та створює два окремі списки: features - список списків, де кожен вкладений список містить значення ознак для кожного рядка, та price - список цін. Нарешті, функція викликає preprocessing.normalize для нормалізації ознак та цін, після чого повертає нормалізовані ознаки та ціни у формі масивів NumPy.

```
import csv
from sklearn import preprocessing

data_url = "Mobile Price Prediction.csv"
def read(data_url):
    with open(data_url, 'r') as f:
        reader = csv.DictReader(f)
        data = [row for row in reader]
    return data


def receive(data):
    feature_names = [key for key in data[0].keys() if key  in [
        'Screen Size (inches)', 'RAM (GB)', 'Storage (GB)',
        'Battery Capacity (mAh)', 'Camera Quality (MP)']]
    print("here",feature_names)
    features = [[float(row[key]) for key in feature_names] for row
     in data]
    price = [float(row['Price ($)']) for row in data]
    return preprocessing.normalize(np.array(features)),
     preprocessing.normalize([np.array(price)])[0]
```

Наступним кроком розділяю датасет на навчальний набір даних та тренувальний у відношенні 80 на 20 відповідно.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Після цього, мені потрібно задати функцію втрат та диференціали по W та по b. Ця функція обчислює помилку моделі на основі передбачених значень та фактичних значень цільової змінної. У випадку лінійної регресії часто використовується середньоквадратична помилка (MSE), яка обчислюється як

середнє значення квадратів різниць між фактичними та передбаченими значеннями.

```
def Loss(X, y, W, b):
    return np.sum((y - (W @ X.T + b)) ** 2) / X.shape[0]
```

Градієнти функції втрат по вагам і зсуву визначають, як швидко функція втрат змінюється при зміні ваг(W) і зсуву(b). Ці градієнти потрібні для оновлення параметрів моделі під час навчання за допомогою градієнтного спуску. Їх було обчислено як часткові похідні по W та по b:

```
def dldw(X, y, W, b):
    return -2 * X.T @ (y - (W @ X.T + b)) / X.shape[0]


def dldb(X, y, W, b):
    return -2 * np.sum(y - (W @ X.T + b)) / X.shape[0]
```

Оновлення параметрів моделі в градієнтному спуску полягає в корекції значень ваг і зсуву моделі з метою мінімізації функції втрат. Після обчислення градієнтів функції втрат по вагам і зсуву ми знаємо напрямок, в якому функція втрат швидко зменшується. Цей напрямок вказує, якім чином потрібно змінити значення параметрів моделі для зменшення помилки.

Оновлення параметрів моделі відбувається згідно з такими правилами:

Оновлення ваг: Ваги оновлюються в напрямку, протилежному до напрямку градієнта. Це означає, що кожна вага зменшується на деяку величину, пропорційну градієнту функції втрат відносно цієї ваги.

Формула для оновлення ваг за градієнтним спуском:

$$W = W - beta * \frac{dLoss\ function}{dW}$$
$$b = b - beta * \frac{dLoss\ function}{db}$$

де beta підбирається за допомогою функції get_beta, яка використовується метод одновимірної оптимізації для підбору оптимального beta - кроку навчання (learning rate) за допомогою методу Фібоначчі. Основна мета цієї функції полягає в тому, щоб знайти значення кроку, яке мінімізує функцію втрат при оновленні параметрів моделі (ваг і зсуву). Найкраще знайдене значення кроку повертається як результат роботи функції get_beta.

```
def get_beta(X, y, W, b):
    epsilon = 1e-5

    def approximation(beta):
        return Loss(X, y, W - beta * dldw(X, y, W, b), b - beta *
         dldb(X, y, W, b))

    beta, *other = fib_method(approximation, 0, 10, epsilon)
    return beta
```

Функція gradient_descent реалізує алгоритм градієнтного спуску для навчання моделі лінійної регресії. Алгоритм починається з ініціалізації лічильника ітерацій, який використовується для обмеження максимальної кількості ітерацій у циклі. У циклі while перевіряється, чи не було досягнуто максимальної кількості ітерацій. Далі обчислюється значення функції втрат для поточних параметрів моделі. Потім використовується функція get_beta для отримання значення кроку градієнтного спуску. За допомогою цього кроку і градієнтів функції втрат обчислюються нові значення параметрів моделі. Після цього обчислюється нове значення функції втрат для нових параметрів моделі. Якщо зміна втрат менше, ніж допустиме значення tol, то алгоритм завершується. В іншому випадку параметри моделі оновлюються до нових значень, і ітерації продовжуються. Під час кожної ітерації виводиться інформація про поточну ітерацію. Цей процес повторюється до тих пір, поки не буде досягнуто максимальної кількості ітерацій або досягнута достатня зміна у значенні функції втрат. Після завершення алгоритм повертає оновлені значення параметрів моделі.

```
def gradient_descent(X, y, W, b, max_iterations=10000, tol=1e-9):
    itr = 0
    while itr < max_iterations:
        loss = Loss(X, y, W, b)
        beta = get_beta(X, y, W, b)
        W_new = W - beta * dldw(X, y, W, b)
        b_new = b - beta * dldb(X, y, W, b)
        new_loss = Loss(X, y, W_new, b_new)
        if np.abs(loss - new_loss) < tol:
            break
        W, b = W_new, b_new
        print(f"Iteration {itr} | W = {W}| b = {b}")
        itr += 1
    return W, b
```

Після завершення алгоритму ми отримали оптимальні значення параметрів W та b, які найкраще апроксимують залежність між ознаками X та цільовою змінною y. Ці значення можна використовувати для передбачення цільових значень для нових вхідних даних.

Для реалізації L1 регуляризації у лінійній регресії додамо до функції втрат штрафний член, який буде залежати від суми абсолютних значень вагових коефіцієнтів W. Основні моменти програмної реалізації є наступними:

- Оновлення функції втрат: До функції втрат Loss_L1 додам штрафний член alpha , який враховує суму абсолютних значень вагових коефіцієнтів W.

```
4 usages
def Loss_L1(X, y, W, b, alpha):
    return np.sum((y - (W @ X.T + b)) ** 2) / X.shape[0] + alpha * np.sum(np.abs(W))
```

- Додаю в окремі функції частинні похідні за ваговими коефіцієнтами W і зсувом b з урахуванням штрафного члена L1 регуляризації.

```
def dldw_L1(X, y, W, b, alpha):
    return -2 * X.T @ (y - (W @ X.T + b)) / X.shape[0] + alpha * np.sign(W)


2 usages
def dldb_L1(X, y, W, b, alpha):
    return -2 * np.sum(y - (W @ X.T + b)) / X.shape[0]
```

- Як було згадано раніше параметри моделі W та b оновлюються згідно ітераційного процесу градієнтного спуску з урахуванням оновлених градієнтів. Алгоритм завершує роботу, коли досягнута необхідна точність або після заданої кількості ітерацій.
- Таким чином маємо підібрані коефіцієнти W та b які мінімізують функцію втрат для L1 регуляризації.

Для L2 регуляризації в лінійній регресії додамо до функції втрат штрафний член, який буде залежати від квадратичної суми вагових коефіцієнтів W. Відповідно цільова функція для L2 регуляризації є такою:

```
def Loss_L2(X, y, W, b, alpha):
    return np.sum((y - (W @ X.T + b)) ** 2) / X.shape[0] + alpha * np.sum(W ** 2)
```

Відповідно часткові похідні по W та по b набудуть такого вигляду:

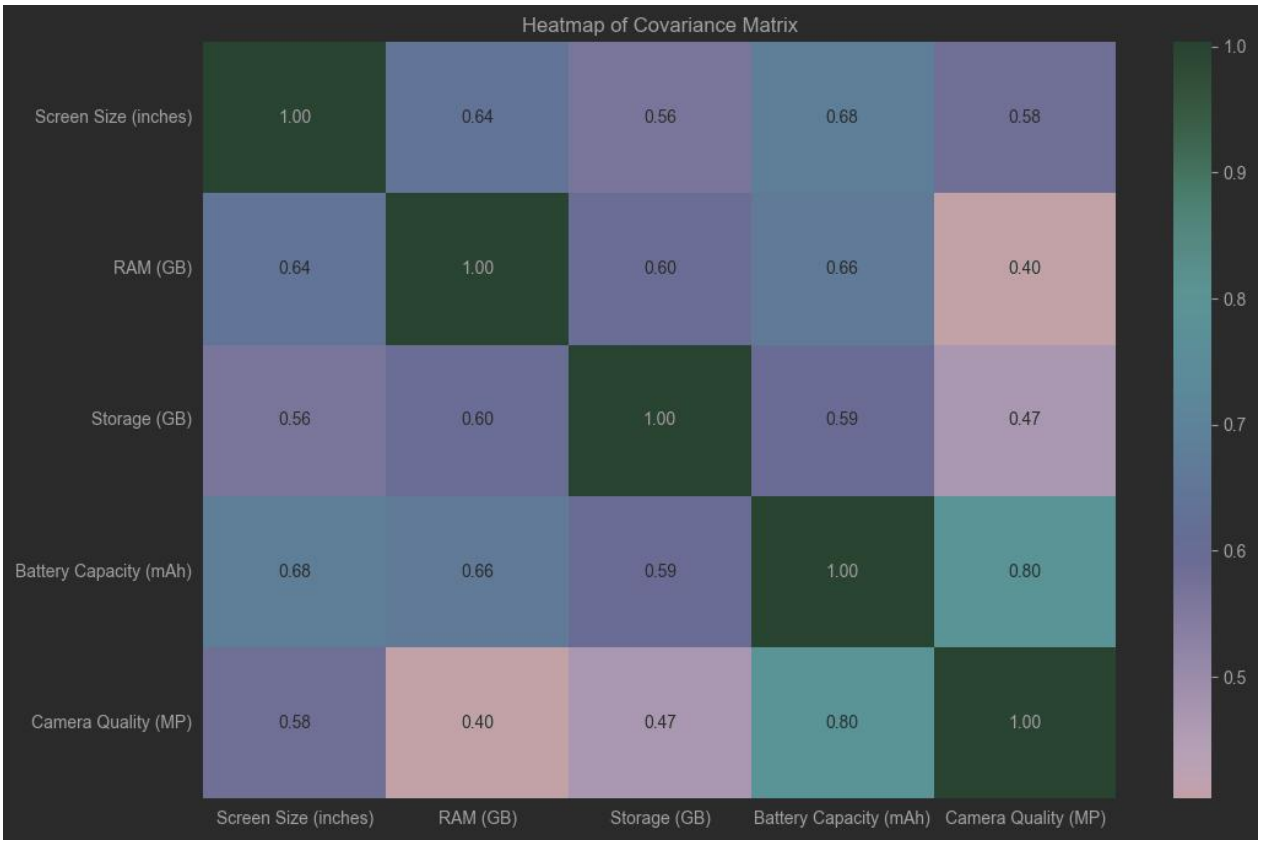
```
def dldw_L2(X, y, W, b, alpha):
    return -2 * X.T @ (y - (W @ X.T + b)) / X.shape[0] + 2 * alpha * W


# Градієнт функції втрат з L2 регуляризацією по зміщенню b
2 usages
def dldb_L2(X, y, W, b):
    return -2 * np.sum(y - (W @ X.T + b)) / X.shape[0]
```

Тепер до цієї цільової функції та цих градієнтів за допомогою градієнтного спуску, аналогічно як і в L1 регуляризації, знайдемо оптимальне W та b.

Важливо зауважити, що підбір параметру регуляризації є важливим етапом у побудові моделі з регуляризацією, такою як L1 або L2. Цей параметр alpha контролює вплив регуляризації на модель. Він визначає, наскільки сильно великі значення ваг можуть бути штрафовані, тобто наскільки сильно регуляризація впливає на загальний процес навчання. Мною було проведено досліди для визначення найкращого регуляризаційного параметру alpha та було обрано значення alpha = 0.1.

Аналіз та інтерпретація отриманих результатів

Для аналізу отриманих мною результатів спочатку розглянемо детально мій датасет. Я використала для цього теплову мапу для коваріаційної матриця, таким чином я оціню наскільки зростають одні змінні разом з іншими. Ми можемо використати це для визначення кореляції між різними ознаками та їхнього впливу на цільову змінну. Це може допомогти зрозуміти, які ознаки є найбільш істотними для нашої моделі.



Давайте розглянемо декілька залежностей: між battery capacity і camera quality (0.8), це висока позитивна кореляція, що означає, що збільшення ємності батареї супроводжується збільшенням якості камери. Іншими словами, пристрої з більшою ємністю батареї, ймовірно, мають кращу якість камери. Кореляція між RAM і storage (0.6): це також позитивна кореляція, але трохи менша за попередню. Це означає, що збільшення обсягу оперативної пам'яті (RAM) супроводжується збільшенням обсягу зберігання. Це може бути очікуваним, оскільки більша оперативна пам'ять часто використовується для виконання більш складних програм і зберігання більшого обсягу даних. Якщо аналізувати screen size (inches) і RAM (0.64): це означає, що збільшення розміру екрану супроводжується збільшенням оперативної пам'яті. Це може бути пов'язане з тим, що пристрої з більшим екраном, ймовірно, призначені для виконання більш складних задач, що потребують більше оперативної пам'яті.

Після розгляду мого датасету, можемо перейти до результатів мого алгоритму, оцінка моєї моделі буде здійснено на основі точності передбачень на тестових

даних та значення функції втрат. Для валідації моделі було використано середньоквадратичну похибку(MSE) та було виміряно час навчання моделі.
Спочатку розглянемо лінійну регресію з використанням градієнтного спуску. Для вибору оптимального початкового наближення було проведено декілька експерементів:

W(початкове)	b(початкове)	Time(s)	MSE
0,6*X.shape[1]	0.6	0.75	0.00017
0,5*X.shape[1]	0,5	0.48	0.00017
0,4*X.shape[1]	0.4	0.39	0.00016

Після проведення експерементів для усіх задач було обрано таке початкове наближення W = 0,4*X.shape[1], b=0.4
Після навчання моделі лінійної регресії з використанням градієнтного спуску, я отримала наступні результати:

Параметри моделі

Використовуючи початкове наближення W=0,4*X.shape[1], b=0.4 параметри моделі, отримані після кількох ітерацій, показують, як вектор ваг (W) і зсув (b) змінюються в процесі навчання. З часом ці параметри зближаються до оптимальних значень, що мінімізують функцію втрат.

W	b	Time(s)	MSE	Кількість ітерацій
[0.39927231 0.39975442 0.34860604 -0.08077392 0.37834573]	0.4	0.39	0.0001646	316

```
Iteration 314 | W = [ 0.39927218  0.39975284  0.34870943 -0.08082581  0.37840288]| b = 0.12254850677497102
Iteration 315 | W = [ 0.39927224  0.39975358  0.34866095 -0.08080049  0.37837608]| b = 0.12252600543991901
Iteration 316 | W = [ 0.39927231  0.39975442  0.34860604 -0.08077392  0.37834573]| b = 0.12249623241364592
Loss on test set: 0.0001646110469162720
Час виконання:  0.3919978141784668  секунд
```

Функція втрат

Значення функції втрат на тестовому наборі даних: 0.0001646110469162720

Аналіз параметрів моделі

Параметри моделі, отримані після навчання, вказують на значимість кожної з ознак у передбаченні ціни мобільного телефону. Зокрема, ваги для кожної ознаки показують, наскільки сильний вплив цієї ознаки на кінцеву ціну.

Аналіз ваг для різних характеристик моделі при таких значеннях параметрів показує, що збільшення розміру екрану на одиницю пов'язане зі збільшенням ціни мобільного телефону на приблизно 0.3993 одиниці, а збільшення оперативної пам'яті на один гігабайт додає приблизно 0.3998 одиниць до ціни. Обсяг внутрішньої пам'яті також впливає на ціну, проте трохи менше, ніж розмір екрану і оперативна пам'ять, але все одно є значущим - збільшення внутрішньої пам'яті на один гігабайт пов'язане з підвищенням ціни на приблизно 0.3486 одиниці. Значення негативного коефіцієнта для ємності батареї (-0.08077392) вказує на те, що збільшення ємності батареї на одну одиницю пов'язане зі зменшенням ціни на приблизно 0.0808 одиниці. Нарешті, збільшення якості камери на одиницю додає приблизно 0.3783 одиниць до ціни, підкреслюючи важливість цієї характеристики для споживачів.

Значення функції втрат є дуже малим (0.00016461104691627203), що свідчить про високу точність моделі. Це означає, що різниця між передбаченими цінами та фактичними цінами на тестовому наборі даних є мінімальною. Низьке значення функції втрат вказує на те, що модель добре навчилася наданим даним і може точно передбачати ціни на мобільні телефони.

Отримані результати показують, що лінійна регресія з використанням градієнтного спуску є ефективним методом для передбачення цін на мобільні телефони на основі їх характеристик. Ці результати також підкреслюють важливість вибору правильних параметрів для алгоритму градієнтного спуску, таких як крок навчання і кількість ітерацій, для забезпечення стабільної і точної збіжності. Подальше покращення результатів можна досягти шляхом застосування більш складних моделей або використання регуляризації для запобігання перенавчанню і поліпшення узагальнюючої здатності моделі.

L1 регуляризація, відома також як Lasso регресія, сприяє отриманню моделей, де деякі ваги стають нульовими. Це дозволяє зменшити складність моделі і відфільтрувати менш важливі ознаки. У моїх результатах видно, що деякі ваги стали дуже близькими до нуля, що вказує на їхню менш значущу роль у прогнозуванні ціни мобільного телефону. Після проведення навчання моделі лінійної регресії з використанням L1 регуляризації, я отримала наступні результати

W	b	MSE	Time(s)	Кількість ітерацій	Штраф(alpha)
[-6.06547721e-08, -4.26716362e-08, -3.26700543e-07, -1.35673562e-01, -5.01955502e-07]	0.2140	0.0143	0.49	302	0,1

```
Iteration 301 | W = [ 4.77188356e-07  4.95110592e-07  2.17555126e-07 -1.35673879e-01
  3.75757270e-08]| b = 0.21406395284113566
Iteration 302 | W = [-6.06547721e-08 -4.26716362e-08 -3.26700543e-07 -1.35673562e-01
 -5.01955502e-07]| b = 0.21406373264496392
Loss on test set: 0.014300004076274119
Час виконання:  0.44402456283569336  секунд
```

Параметри моделі після використання L1 регуляризації вказують на те, що деякі з характеристик, таких як розмір екрану, обсяг оперативної пам'яті та якість камери, мають дуже малі значення ваг, навіть близькі до нуля. Це може вказувати на те, що ці характеристики не є дуже важливими для моделі або на те, що вони слабко корелюють з цільовою змінною. Особливо важливим є вага для ємності батареї, яка має значення приблизно -0.1357. Це вказує на те, що збільшення ємності батареї на одну одиницю пов'язане зі зменшенням ціни мобільного телефону на приблизно 0.1357 одиниць. Загальна втрата на тестовому наборі даних становить 0.0143, що є більш високим значенням порівняно з результатом, отриманим без регуляризації. Це вказує на те, що L1 регуляризація призвела до меншої узгодженості моделі з тестовими даними, що може бути зумовлено перенавчанням моделі або недостатньою регуляризацією.

Якщо аналізувати **L2 регуляризацію,** то було отримано такі результати:

W	b	MSE	Кількість ітерацій	Time(s)	Штраф
[-3.83e-06, 9.35e-06, 5.39e-04, 6.98e-05, -4.492e-05]	0.05	0.0001611	17	0.020	0,1

```
Iteration 16 | W = [-3.74700742e-06  9.42783018e-06  5.41275295e-04  1.23769898e-04
 -4.44389758e-05]| b = 0.05785281529696829
Iteration 17 | W = [-3.83218196e-06  9.35411555e-06  5.39477300e-04  6.98343180e-05
 -4.49267975e-05]| b = 0.05793345351857461
Loss on test set: 0.00016115249545998987
Час виконання:  0.022999048233032227  секунд
```

Після використання L2 регуляризації, ваги коефіцієнтів вектора W стали значно меншими порівняно з результатами без регуляризації. Це свідчить про те, що L2 регуляризація ефективно зменшує величину ваг, уникаючи великих значень, що можуть призвести до перенавчання моделі. Значення зсуву (b) також змінилося на досить мале значення, 0.05, що може свідчити про невелику кореляцію між зсувом та цільовою змінною. Середньоквадратична помилка (MSE) на тестовому наборі даних становить 0.0001611, що є невеликим значенням, вказуючи на добру узгодженість моделі з тестовими даними. Також, кількість ітерацій для досягнення збіжності складає лише 17, що є досить невеликою кількістю та свідчить про ефективність оптимізації з використанням L2 регуляризації, оскільки без регуляризації кількість була 302.

Таким чином, результати після використання L1 та L2 регуляризації свідчать про те, що для мого датасету використання L1 регуляризації не дало покращення, бо MSE та час виконання збільшились порінюючи з лінійною регресією без використання регуляризацій. Проте використання L2 регуляризації дало покращення в точності, і дало значне покращення часу виконання.

Висновки

У рамках цього дослідження я провела аналіз та моделювання даних з набору Mobile Price Prediction.csv, що містить інформацію про мобільні телефони та їх характеристики. Використовуючи методи лінійної регресії з різними видами регуляризації - L1 та L2, було досліджено ефективність моделей у передбаченні цін на мобільні телефони.

Лінійна регресія з використанням градієнтного спуску є ефективним методом для передбачення цін на мобільні телефони на основі їх характеристик. Результати показали високу точність моделі та добру узгодженість з тестовими даними.

У моєму випадку, L1 регуляризація не принесла покращення у точності моделі, а навіть призвела до збільшення середньоквадратичної помилки (MSE) та збільшення часу виконання. Це може бути пов'язано з тим, що деякі ознаки у нашому датасеті можуть бути менш важливими, і використання L1 регуляризації призвело до видалення цих ознак, що знизило точність моделі. З іншого боку, використання L2 регуляризації принесло покращення в точності моделі і значне зменшення часу виконання. Це свідчить про те, що у моєму датасеті більше підходить саме L2 регуляризація, яка дозволяє зменшити ваги ознак, уникаючи великих значень, що можуть призвести до перенавчання моделі, а також полегшити обчислювальне завантаження.

Отже, дослідження підтвердило, що використання різних видів регуляризації в поєднанні з градієнтним спуском у лінійній регресії є важливим і ефективним методом для покращення якості прогнозів та узагальнення моделі.

Список літератури

Linear Regression using Gradient Descent, 2018 https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931

Linear Regression in Machine learning, 2024, https://www.geeksforgeeks.org/ml-linear-regression/

Linear Regression with Gradient Descent,2020, Rukshan Manorathna, https://www.researchgate.net/publication/342163679_Linear_Regression_with_Gradient_Descent

Private Gradient Descent for Linear Regression, 2024, **https://arxiv.org/html/2402.13531v1**

Linear Regression in Machine Learning **https://www.javatpoint.com/linear-regression-in-machine-learning**

Додаток

```
def fib(n):
    i = 1
    j = i
    c = 1
    while c < n:
        i, j = j, i + j
        c += 1
    return i
# find index of fib number
def get_iteration(vars):
    # j число фібоначі
    # i попереднє число фіб
    i, j, c = 1, 1, 0
    while i <= vars:
        i, j = j, i + j
        c += 1
    return c
def fib_method(f, a, b, eps):
    vars = round((b - a) / eps)  # знаходить найменше потрібне число фібоначі
    n = get_iteration(vars) + 1  # знаходить номер того фіб числа
    x1 = a + (fib(n - 2) / fib(n)) * (b - a)
    x2 = a + (fib(n - 1) / fib(n)) * (b - a)
    fx1 = f(x1)
    fx2 = f(x2)
    k = 0
    iterations = 0  # зберігатиме кількість ітерацій
    while abs(b - a) >= eps:
        iterations += 1
        if fx1 <= fx2:
            b, x2 = x2, x1
            fx2 = fx1
            x1 = a + (fib(n - k - 3) / fib(n - k - 1)) * (b - a)
            fx1 = f(x1)
        else:
            a, x1 = x1, x2
            fx1 = fx2
            x2 = a + (fib(n - k - 2) / fib(n - k - 1)) * (b - a)
            fx2 = f(x2)
        k += 1
        print(f"Iteration {iterations}: a = {a}, b = {b}, x1 = {x1}, x2 = {x2},
f(x1) = {fx1}, f(x2) = {fx2}")

    return (a + b) / 2, iterations

#%%

import csv
from sklearn import preprocessing

data_url = "Mobile Price Prediction.csv"
def read(data_url):
    with open(data_url, 'r') as f:
        reader = csv.DictReader(f)
        data = [row for row in reader]
    return data


def receive(data):
    feature_names = [key for key in data[0].keys() if key  in [
        'Screen Size (inches)', 'RAM (GB)', 'Storage (GB)',
        'Battery Capacity (mAh)', 'Camera Quality (MP)']]
    print("here",feature_names)
    features = [[float(row[key]) for key in feature_names] for row in data]
```

```
    price = [float(row['Price ($)']) for row in data]
    return preprocessing.normalize(np.array(features)),
preprocessing.normalize([np.array(price)])[0]
import csv
import time
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

def Loss(X, y, W, b):
    return np.sum((y - (W @ X.T + b)) ** 2) / X.shape[0]


def dldw(X, y, W, b):
    return -2 * X.T @ (y - (W @ X.T + b)) / X.shape[0]


def dldb(X, y, W, b):
    return -2 * np.sum(y - (W @ X.T + b)) / X.shape[0]


def gradient_descent(X, y, W, b, max_iterations=10000):
    itr = 0
    beta = get_beta(X, y, W, b)
    while np.abs(Loss(X, y, W, b) - Loss(X, y, W - beta * dldw(X, y, W, b),b -
beta * dldb(X, y, W, b))) > 1e-9 and itr < max_iterations:
        itr += 1
        beta = get_beta(X, y, W, b)
        W = W - beta * dldw(X, y, W, b)
        b = b - beta * dldb(X, y, W, b)
        print( f"Iteration {itr} | W = {W}| b = {b}")

    return W, b

def get_beta(X, y, W, b):
    epsilon = 1e-5
    def approximation(beta):
        return Loss(X, y, W - beta * dldw(X, y, W, b), b - beta * dldb(X, y, W,
b))

    beta, _ = fib_method(approximation, 0, 10, epsilon)
    return beta

def plot_scatter(X, y, W, b):
    plt.scatter(X[:, 0], y)
    plt.plot(X[:, 0], W[0] * X[:, 0] + b, color='red', label='Linear Regression')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.show()
print("## Using gradient descent ##")
X, y = receive(read(data_url))
print(X)
print(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

W = np.array([0.4] * X.shape[1])
b = 0.4
start_time = time.time()
W, b = gradient_descent(X_train, y_train, W, b)
print("Loss on test set:", Loss(X_test, y_test, W, b))
end_time = time.time()
# Predicting on test set
y_pred_test = W @ X_test.T + b
execution_time = end_time - start_time
```

```
print("Час виконання: ", execution_time, " секунд")

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, color='blue', label='Фактичні значення vs.
Передбачені значення')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
label='Ідеальна узгодженість')
plt.title('Графік фактичних значень проти передбачених значень')
plt.xlabel('Фактичні значення')
plt.ylabel('Передбачені значення')
plt.legend(loc='upper left')
plt.show()
```

L1 regularization:

```
def Loss_L1(X, y, W, b, alpha):
    return np.sum((y - (W @ X.T + b)) ** 2) / X.shape[0] + alpha *
np.sum(np.abs(W))


def dldw_L1(X, y, W, b, alpha):
    return -2 * X.T @ (y - (W @ X.T + b)) / X.shape[0] + alpha * np.sign(W)


def dldb_L1(X, y, W, b, alpha):
    return -2 * np.sum(y - (W @ X.T + b)) / X.shape[0]


def gradient_descent_L1(X, y, W, b, alpha, max_iterations=10000, tol=1e-9):
    itr = 0
    losses = []
    while itr < max_iterations:
        loss = Loss_L1(X, y, W, b, alpha)
        beta = get_beta_L1(X, y, W, b, alpha)
        W_new = W - beta * dldw_L1(X, y, W, b, alpha)
        b_new = b - beta * dldb_L1(X, y, W, b,alpha)
        new_loss = Loss_L1(X, y, W_new, b_new, alpha)
        if np.abs(loss - new_loss) < tol:
            break
        W, b = W_new, b_new
        print(f"Iteration {itr} | W = {W}| b = {b}")
        losses.append(new_loss)
        itr += 1
    return W, b, losses


# для пошуку бета яке мінімізує функцію на певному кроці
def get_beta_L1(X, y, W, b, alpha):
    golden = GoldenSectionSearch()
    eps = 1e-5

    def next_approx(beta):
        return Loss_L1(X, y, W - beta * dldw_L1(X, y, W, b, alpha), b - beta *
dldb_L1(X, y, W, b, alpha), alpha)

    beta, *other = golden.search(next_approx, 0, 10, eps)
    return beta


X, y = receive(read(data_url))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print("*******Using gradient descent with L1 regularization*******")
```

```
alpha = 0.1
W = np.array([0.4] * X.shape[1])
b = 0.4
start_time = time.time()
W, b ,losses = gradient_descent_L1(X_train, y_train, W, b, alpha)
print("Loss on test set:", Loss_L1(X_test, y_test, W, b, alpha))
end_time = time.time()
execution_time = end_time - start_time
print("Час виконання: ", execution_time, " секунд")
# Прогнозування на тестовому наборі
y_pred_test = W @ X_test.T + b
```

L2 regularization:

```
# Функція втрат з L2 регуляризацією
def Loss_L2(X, y, W, b, alpha):
    return np.sum((y - (W @ X.T + b)) ** 2) / X.shape[0] + alpha * np.sum(W ** 2)


# Градієнт функції втрат з L2 регуляризацією по вагам W
def dldw_L2(X, y, W, b, alpha):
    return -2 * X.T @ (y - (W @ X.T + b)) / X.shape[0] + 2 * alpha * W


# Градієнт функції втрат з L2 регуляризацією по зміщенню b
def dldb_L2(X, y, W, b):
    return -2 * np.sum(y - (W @ X.T + b)) / X.shape[0]


def gradient_descent_L2(X, y, W, b, alpha, max_iterations=10000, tol=1e-9):
    itr = 0
    losses = []
    while itr < max_iterations:
        loss = Loss_L2(X, y, W, b, alpha)
        beta = get_beta_L2(X, y, W, b, alpha)
        W_new = W - beta * dldw_L2(X, y, W, b, alpha)
        b_new = b - beta * dldb_L2(X, y, W, b)
        new_loss = Loss_L2(X, y, W_new, b_new, alpha)
        if np.abs(loss - new_loss) < tol:
            break
        W, b = W_new, b_new
        print(f"Iteration {itr} | W = {W}| b = {b}")
        losses.append(new_loss)
        itr += 1
    return W, b, losses


# для пошуку бета яке мінімізує функцію на певному кроці
def get_beta_L2(X, y, W, b, alpha):
    gss = GoldenSectionSearch()
    epsilon = 1e-5

    def next_approx(beta):
        return Loss_L2(X, y, W - beta * dldw_L2(X, y, W, b, alpha), b - beta *
dldb_L2(X, y, W, b), alpha)

    beta, _ = gss.search(next_approx, 0, 10, epsilon)
    return beta


print("*******Using gradient descent with L2 regularization*******")
X, y = receive(read(data_url))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
alpha = 0.1
W = np.array([0.4] * X.shape[1])
b = 0.4
start_time = time.time()
W, b, losses = gradient_descent_L2(X_train, y_train, W, b, alpha)
print("Loss on test set:", Loss_L2(X_test, y_test, W, b, alpha))
end_time = time.time()
y_pred_test = W @ X_test.T + b

execution_time = end_time - start_time
print("Час виконання: ", execution_time, " секунд")
```