Лабораторна робота No6

Застосування numpy

Мета роботи: отримати поглиблені навички роботи з numpy; дослідити поняття лінійної регресії та градієнтного спуску.

Постановка задачі:

Ознайомтесь з теоретичним матеріалом. Створіть програму для обчислення лінійної регресії методом найменших квадратів та градієнтним спуском.

Теоретична довідка

Проста лінійна регресія - це метод аналізу взаємозв'язку між двома змінними, де одна змінна (незалежна) використовується для передбачення значень іншої (залежної) змінної, припускаючи лінійний зв'язок між ними. Модель виглядає так:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Де:

- Y - залежна змінна (яку ми намагаємось передбачити).
- X - незалежна змінна (на основі якої ми робимо прогноз).
- β_0 - зсув (intercept), константа або початкова точка.
- β_1 - коефіцієнт нахилу (slope).
- ε – шум (випадкова похибка), яка враховує невипадкову складову і рандомізовану природу даних.

Мета лінійної регресії - знайти оптимальні значення β_0 і β_1, такі, що модель найкраще підходить до наявних даних. Це робиться шляхом мінімізації функції втрат, яка вимірює різницю між спостережуваними значеннями Y і прогнозованими значеннями $\widehat{Y} = \beta_0 - \beta_1 X$.

Оцінка точності. Для оцінки точності моделі лінійної регресії можна використовувати метрики, такі як середньоквадратична помилка (MSE), середня або медіанна абсолютна помилка (MAE), коефіцієнт детермінації R-squared і інші. Чим менше значення MSE і MAE і чим ближче R-squared до 1, тим краща точність моделі.

Функція втрат MSE виглядає наступним чином:

$$L(\beta_0, \beta_1) = \frac{1}{n}\sum (Y_i - \beta_0 - \beta_1 X_i)^2,$$

а MAE в свою чергу так:

$$L(\beta_0, \beta_1) = \frac{1}{n}\sum |Y_i - \beta_0 - \beta_1 X_i|.$$

Загальна задача лінійної регресії - мінімізація функції втрат в залежності від параметрів, виглядає вона наступними чином:

$$\underset{\beta_0,\beta_1}{arg\min}(L(\beta_0, \beta_1)) = \underset{\beta_0,\beta_1}{arg\min}\frac{1}{n}\sum (Y_i - \beta_0 - \beta_1 X_i)^2$$

Метод найменших квадратів (МНК) використовується для знаходження оптимальних значень β^0 і β^1, які мінімізують функцію втрат $L(\beta^0, \beta^1)$. Це робиться шляхом обчислення часткових похідних функції втрат по β_0 і β_1 і встановлення їх рівними нулю:

$$\frac{\partial L}{\partial \beta_0} = -2\frac{1}{n}\sum(Y_i - \beta_0 - \beta_1 X_i) = 0$$

$$\frac{\partial L}{\partial \beta_1} = -2\frac{1}{n}\sum X_i(Y_i - \beta_0 - \beta_1 X_i) = 0$$

Вирішивши ці рівняння, отримуємо оцінки для β_0 та β_1:

$$\beta_0 = \hat{Y} - \beta_1 \hat{X}$$

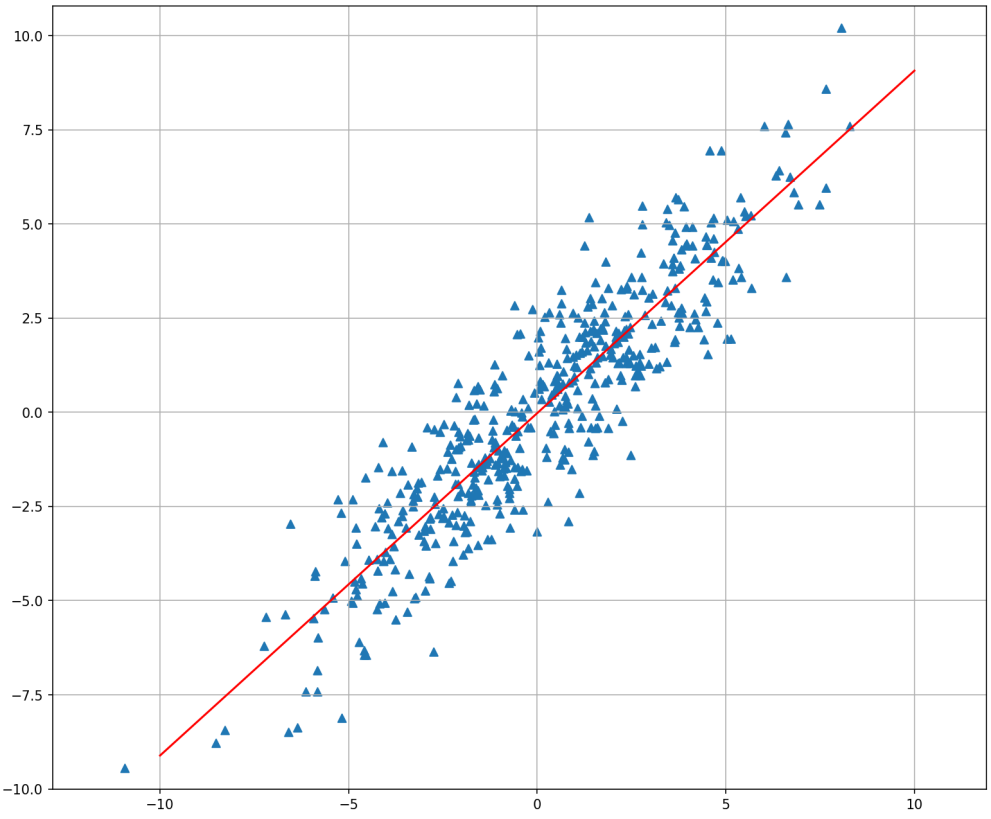
$$\beta_1 = \frac{\sum(X_i - \hat{X})(Y_i - \hat{Y})}{\sum(X_i - \hat{X})^2},$$

де $\hat{X}$ – середнє значення X, а $\hat{Y}$ – середнє значення Y.

Після знаходження цих параметрів ми можемо побудувати **регресійну лінію**:

$$Y = \beta_0 - \beta_1 X$$

Ця лінія представляє лінійний зв'язок між X і Y, і її можна використовувати для прогнозування значень Y для нових значень X.



Багатовимірна лінійна регресія - це розширення простої лінійної регресії на випадок, коли є більше однієї незалежної змінної. Модель виглядає так: $Y = \beta^0 + \beta^1 X^1 + \beta^2 X^2 + \cdots + \beta_k X_k + \varepsilon$, де $X^1, X^2, \ldots, X_k$ - різні незалежні змінні, $\beta^0, \beta^1, \beta^2, \ldots, \beta_k$ - параметри моделі, ε – шум.

Аналогічно до простої лінійної регресії, для багатовимірної лінійної регресії також використовується метод найменших квадратів (МНК) для знаходження оптимальних значень параметрів, які мінімізують функцію втрат

$$L(\beta_0, \ldots, \beta_n) = \frac{1}{n}\sum (Y_i - \hat{Y})^2$$

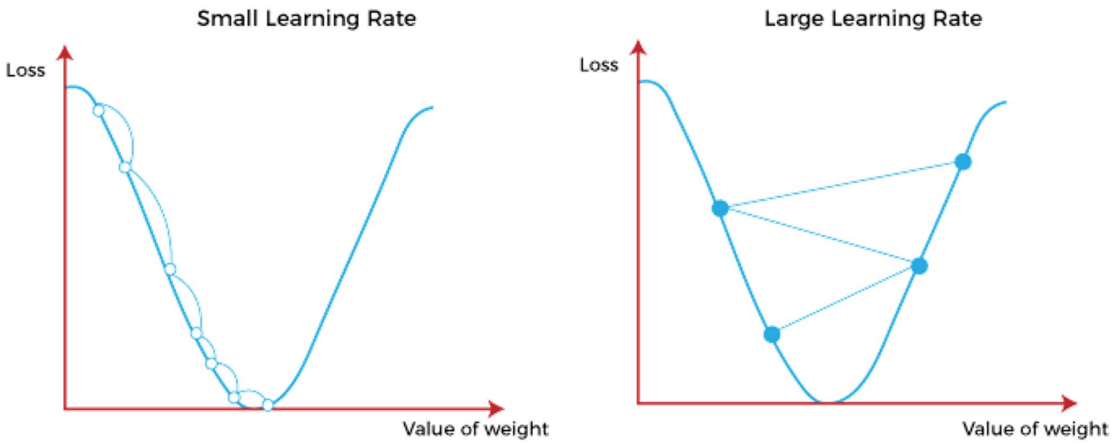
Побудова прогнозів. Після навчання моделі багатовимірної лінійної регресії, можна використовувати її для прогнозування значень залежної змінної на основі нових значень незалежних змінних. Для цього підставляємо дані для прогнозу X в формулу з отриманими коефіцієнтами β_k. Далі проводиться оцінка прогнозу (MSE, MAE,…).

Перехресна перевірка: Це метод оцінки точності моделі, при якому дані розділяються на навчальний і тестовий набори. Модель навчається на навчальному наборі і перевіряється на тестовому наборі, щоб оцінити її узагальнюючу здатність і уникнути перенавчання. Щоб отримати більш надійну оцінку, перехресну перевірку зазвичай повторюють декілька разів з різними поділами на навчальний і тестовий набори. Наприклад, у п'ятикратній перехресній перевірці (5-fold cross-validation), дані розділяються на п'ять частин, і модель тренується та тестується п'ять разів, використовуючи різні комбінації навчальних і тестових наборів. Перехресна перевірка допомагає виявити, чи модель перенавчена або недонавчена і чи можна її використовувати для прогнозування нових даних. Вона є важливим інструментом для оцінки загальної ефективності моделі та вдосконалення її параметрів.

Важливо: метод найменших квадратів (МНК) дозволяє оцінити лінійну регресію одразу, за одну ітерацію (ми аналітично знайшли формули для мінімізації функції втрат $L(\beta_0, \beta_1)$ шляхом прямого обчислення β_0 та β_1 з даних, які нам одразу відомі). Однак існує багато інших методів, які працюють ітеративно (наприклад, метод градієнтного спуску).

Метод градієнтного спуску (найшвидшого спуску) – ітеративний метод оптимізації (мінімізації) деякої функції. Для знаходження локального мінімуму функції здійснюються кроки, пропорційні протилежному значенню градієнта в поточній точці (початкова точка обирається навмання). Величина кроку (learning rate) – відстань, на яку змінюються параметри кожної ітерації. Вона може змінюватись в процесі навчання на найоптимальнішу для поточної ітерації, однак в класичному алгоритмі вона обирається константною на початку і не змінюється. До вибору величини кроку слід ставитись відповідально: вибравши найменш можливу величину кроку, кількість ітерацій до знаходження мінімуму значно зросте, а вибравши занадто велику – алгоритм почне перестрибувати мінімум на кожній ітерації, так і не знайшовши його.

Існують два підходи для реалізації алгоритму навчання градієнтного спуску: пакетний (batch) та стохастичний (stochastic). У першому підході вибірка розглядається цілком на кожній ітерації, а зсув виконується по усередненому значенні по всій вибірці, у другому – випадково обирається підвибірка і ваги оновлюються на ній, а обрана підвибірка видаляється з загальної. Переваги стохастичного спуску очевидні – оцінювати мільйонні вибірки на кожні з сотень ітерацій ресурсоємно, якщо вибірка навпаки, занадто мала, можна її дозаповнити (Bootstrapping). Але стохастичний спуск має і свої недоліки: повільна збіжність і потенційні застрягання в локальних мінімумах функції.



У випадку лінійної регресії, будемо розглядати пакетний градієнтний спуск з постійним кроком навчання. Мінімізувати будемо функцію втрат MSE $L(\beta_0, \beta_1)$. Вона опукла і має один глобальний та локальний мінімум, має похідні в кожній точці, тому методом градієнтного спуску ми точно отримаємо правильний результат.

<u>Алгоритм виглядає наступним чином</u>:

1. Ініціалізуємо початкові значення β_0^0 та β_1^0, визначаємо learning_rate, а також кількість ітерацій (або умову зупинки замість цього, наприклад, мінімально допустиму похибку прогнозу)
2. Обчислюємо поточні прогнози для кожного рядка: $\check{y}_i = \beta_0^i + \beta_1^i x_i$
3. Обчислюємо часткові похідні по кожному з параметрів ($\frac{\partial L}{\partial \beta_0}$ та $\frac{\partial L}{\partial \beta_1}$):

$$\frac{\partial L}{\partial \beta_0} = -2\frac{1}{n}\sum(y_i - \check{y}_i)$$
$$\frac{\partial L}{\partial \beta_1} = -2\frac{1}{n}\sum x_i(y_i - \check{y}_i)$$

4. Оновлюємо параметри, використовуючи знайдені часткові похідні та коефіцієнт навчання:

$$\beta_0^{i+1} = \beta_0^i - learning\_rate * \frac{\partial L}{\partial \beta_0}$$
$$\beta_1^{i+1} = \beta_1^i - learning\_rate * \frac{\partial L}{\partial \beta_1}$$

5. Повторюємо кроки 2-4 визначену кількість ітерацій (або до умови зупинки).

У випадку лінійної регресії градієнтний спуск не може дати оцінку краще, ніж одразу знайдена аналітично методом найменших квадратів. Тому, його використання неефективне. Окрім того, в природі рідко виникають ситуації, коли дані мають лінійно визначені закономірності. В реальних завданнях із складнішими моделями або функціями втрат може бути важко або навіть неможливо знаходити аналітичний розв'язок. В таких випадках ітераційні методи, такі як градієнтний спуск, можуть бути єдиною реальною альтернативою. Окрім того, з МНК аналітично обчислити розв'язок може бути обчислювально витратно, особливо з великими наборами даних. У таких випадках ітераційні методи можуть бути більш ефективними за рахунок використання градієнтів та часткових похідних. Маючи багатомільйонну вибірку даних, стохастичний градієнтний спуск може впоратись на порядки швидше, ніж МНК. Нарешті, гнучка зміна кроку градієнту під час кожної ітерації може запобігти псуванню оцінки помилковими даними і викидами.

Завдання 1 (2б):

1. Згенеруйте двовимірні дані (x, y) за допомогою numpy.random : бажано, щоб розподіл точок був навколо деякої наперед заданої прямої ($y = kx + b$) для подальшого аналізу результатів.
2. Напишіть функцію, яка реалізує метод найменших квадратів для пошуку оптимальних оцінок $\hat{k}$ та $\hat{b}$.
3. Порівняйте знайдені параметри з оцінкою `np.polyfit(x,y,1)` (оцінка полінома степеню 1 методом найменших квадратів), та з початковими параметрами прямої (якщо такі є).
4. Відобразіть на графіку знайдені оцінки лінії регресії (вашої та numpy). Якщо ви генерували вхідні дані навколо лінії, відобразіть також її.

Завдання 2 (2б):

1. Напишіть функцію, яка реалізує метод градієнтного спуску для пошуку оптимальних оцінок $\hat{k}$ та $\hat{b}$. Визначіть оптимальні вхідні параметри: `learning_rate, n_iter`
2. Додайте отриману лінію регресії на загальний графік
3. Побудуйте графік похибки від кількості ітерацій, зробіть висновки
4. Порівняйте отримані результати з результатами попереднього завдання

Корисні посилання

1. https://uk.wikipedia.org/wiki/Проста_лінійна_регресія
2. https://uk.wikipedia.org/wiki/Градієнтний_спуск
3. https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html
4. https://numpy.org/doc/stable/reference/arrays.html
5. McKinney Wes. 2012. Python for data analysis (1st. ed.). O'Reilly Media, Inc.