**Модуль 3.1: Класичне машинне навчання**

**Класичне машинне навчання**

Нарешті ми прийшли до основної частини роботи Data Scientist'а - побудова моделі/гіпотези та тренування моделі на наших даних. І тепер ми познайомимось з основною бібліотекою для машинного навчання scikit-learn.

Встановити її дуже просто, як завжди скористуємось командою pip:

pip install scikit-learn

Для інших можливостей встановлення цієї бібліотеки рекомендуємо користуватись [офіційною сторінкою](https://scikit-learn.org/stable/install.html) (<https://scikit-learn.org/stable/install.html>) даної бібліотеки.

**Навчання з учителем. Регресія**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B7-%D1%83%D1%87%D0%B8%D1%82%D0%B5%D0%BB%D0%B5%D0%BC-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%96%D1%8F)

Якщо на сайті бібліотеки scikit-learn ми зайдемо на [посібник користувача](https://scikit-learn.org/stable/user_guide.html) (https://scikit-learn.org/stable/user\_guide.html), то побачимо, що список алгоритмів починається з блоку **supervised learning** - навчання з учителем. Ми пам'ятаємо, що це комплекс алгоритмів з розміченими даними, тобто для кожного набору вхідних даних вже є визначена правильна відповідь. А однією з основних задач supervised learning є задача **регресії.**

Задача регресії в машинному навчанні полягає у побудові моделі(гіпотези), котра по суті є математичною функцією, яка прогнозує значення неперервної змінної (відомої як цільова змінна) на основі вхідних даних (відомих як ознаки).

Наприклад, якщо ви маєте дані про площу будинку, кількість спалень, вік будинку, то задача регресії полягає у побудові функції, яка може прогнозувати ціну на будинок на основі цих ознак. В цьому випадку, ціна будинку є цільовим значенням, а площа, кількість спалень і вік будинку є вхідними змінними чи ознаками.

Метою регресійної моделі є знайти оптимальну функцію, яка буде добре підходити до вхідних даних та зможе добре прогнозувати значення цільової змінної для нових даних. Для досягнення цієї мети можна використовувати різні алгоритми машинного навчання, такі як лінійна регресія, дерева рішень, ансамблі дерев, нейронні мережі та інші.

**Лінійна регресія**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%BB%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%96%D1%8F)

Найпростішим алгоритмом для задачі регресії звичайно буде **лінійна модель** - тобто математична функція цієї моделі буде лінійною, а її графік відповідно буде лінією. Розберемо це на прикладі.

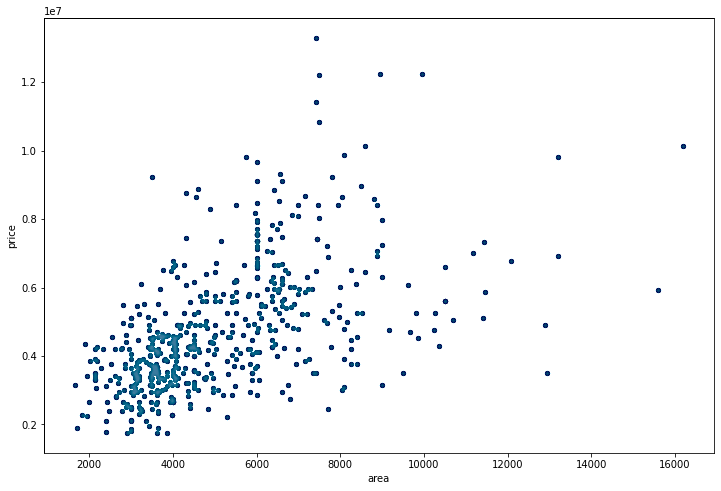
**Лінійна регресія з однією змінною**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%BB%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%96%D1%8F-%D0%B7-%D0%BE%D0%B4%D0%BD%D1%96%D1%94%D1%8E-%D0%B7%D0%BC%D1%96%D0%BD%D0%BD%D0%BE%D1%8E)

Припустимо, що у нас є задача регресії для передбачення цін на будинок в залежності від площі. Тобто у нас є лише одна ознака - площа, а спрогнозувати ми хочемо вартість, до того ж наша гіпотеза буде лінійною:

𝑦=𝑘𝑥+𝑏*y*=*kx*+*b*

Зобразимо наші дані на графіку залежності ціни на будинок від його площі



Отже, нашою задачею буде знаходження прямої (лінії), котра буде найкраще описувати залежність між вище згаданими величинами. Для повноцінного занурення у модель даного алгоритму введемо такі позначення:

* 𝑥(𝑖)*x*(*i*) - ознака (**feature**) 𝑖*i*-ого прикладу (sample, example) з нашого датасету, також відома яка вхідна змінна
* 𝑦(𝑖)*y*(*i*) - цільове значення для 𝑖*i*-ого прикладу, чи вихідне значення (**target**, output value)
* (𝑥(𝑖),𝑦(𝑖))(*x*(*i*),*y*(*i*)) - 𝑖*i*-ий навчальний приклад
* (𝑥(𝑖),𝑦(𝑖))(*x*(*i*),*y*(*i*)), де 𝑖=1...𝑚*i*=1...*m* - множина навчальних даних, чи **training set**, а 𝑚*m* - кількість прикладів у цій множині

Тепер ми можемо сформулювати гіпотезу лінійної регресії:

ℎ(𝑥)=𝑤0+𝑤1𝑥*h*(*x*)=*w*0​+*w*1​*x*

А в процесі навчання нашої моделі ми повинні знайти такі коефіцієнти (ваги) 𝑤0,𝑤1*w*0​,*w*1​​, щоб значення ℎ(𝑥(𝑖))*h*(*x*(*i*))ℎ(𝑥(𝑖))*h*(*x*(*i*))​ були якомога ближчі до 𝑦(𝑖)*y*(*i*). А для того, щоб оцінити, наскільки добре прогнозує наша модель, нам потрібно ввести функцію помилок.

**Функція втрат**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F-%D0%B2%D1%82%D1%80%D0%B0%D1%82)**Функція втрат/помилок** (loss/error/cost function) - це функція, яка вимірює рівень помилок алгоритму машинного навчання, який навчається на тренувальних даних. Отже, завдяки даній функції ми можемо оцінити, наскільки добре працює наша модель, а точніше, наскільки сильно вона помиляється.

У процесі тренування моделі машинного навчання, алгоритм приймає вхідні дані та виконує певні обчислення, щоб зробити прогноз. Очікувані результати прогнозу порівнюються зі справжніми значеннями цільової змінної, які відомі для вхідних даних. Чим більша різниця між прогнозованим та справжнім значеннями, тим вище значення функції втрат.

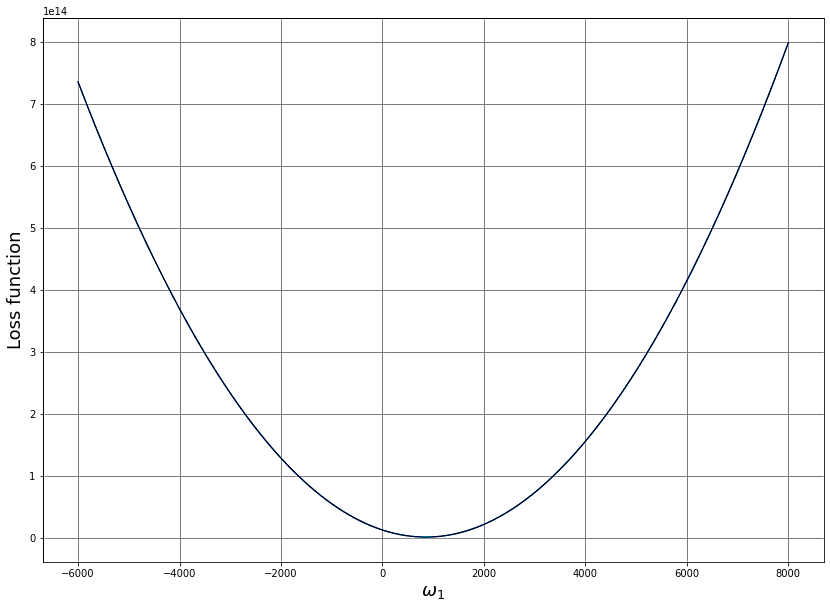
В якості міри помилки не може бути обрано відхилення від прогнозу 𝐽(𝑤0,𝑤1)(𝑖)=ℎ(𝑥(𝑖))−𝑦(𝑖)*J*(*w*0​,*w*1​)(*i*)=*h*(*x*(*i*))−*y*(*i*), оскільки воно може мати як і позитивне, так і негативне значення, а в такому випадку мінімум функції втрат не буде досягатись при правильній відповіді ℎ(𝑥(𝑖))=𝑦(𝑖)*h*(*x*(*i*))=*y*(*i*). З іншого боку, можна взяти абсолютне значення відхилення 𝐽(𝑤0,𝑤1)(𝑖)=∣ℎ(𝑥(𝑖))−𝑦(𝑖)∣*J*(*w*0​,*w*1​)(*i*)=∣*h*(*x*(*i*))−*y*(*i*)∣, але дана функція не є диференційованої у кожній точці. Тому в якості помилки візьмемо квадрат відхилення 𝐽(𝑤0,𝑤1)(𝑖)=(ℎ(𝑥(𝑖))−𝑦(𝑖))2*J*(*w*0​,*w*1​)(*i*)=(*h*(*x*(*i*))−*y*(*i*))2. Це дуже хороша функція - вона тягла, опукла та диференційовна, що є дуже важливим для навчання нашої моделі. А для повної функції втрат просумуємо помилки для кожного навчального прикладу та візьмемо середнє значення, тому цю функцію часто називають **середньоквадратичною** функцією втрат:

𝐽(𝑤0,𝑤1)=12𝑚∑𝑖=1𝑚(ℎ(𝑥(𝑖))−𝑦(𝑖))2*J*(*w*0​,*w*1​)=2*m*1​∑*i*=1*m*​(*h*(*x*(*i*))−*y*(*i*))2

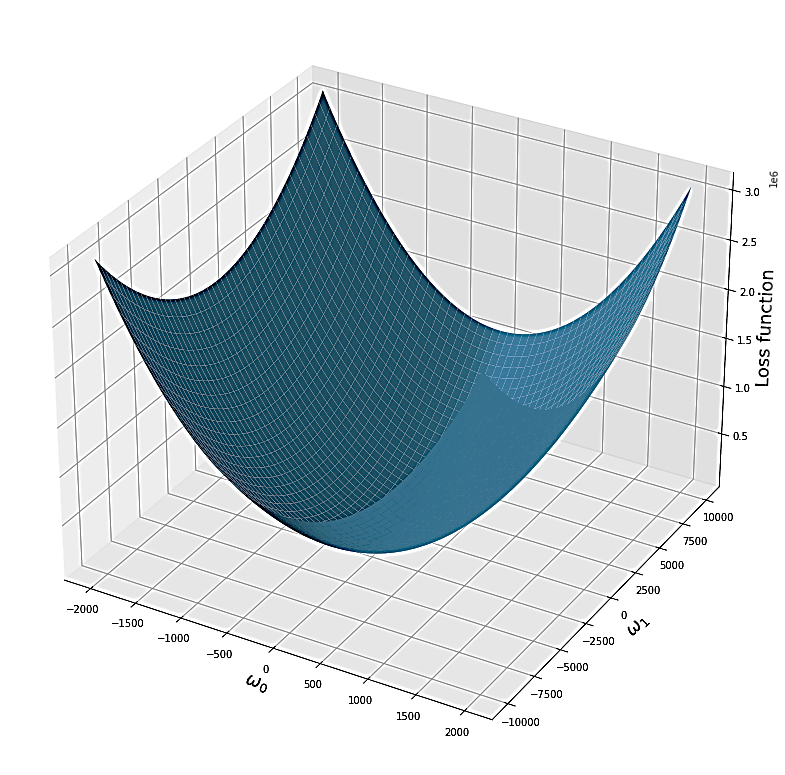
де 𝑚*m* - кількість навчальних прикладів у нашому датасеті. Оскільми менше значення функції втрат відповідає тому, що наша модель менше помиляється, то нашим завданням під час навчання моделі є знаходження таких 𝑤0,𝑤1*w*0​,*w*1​​, щоб значення функції втрат було мінімальним:

𝑤0∗,𝑤1∗=min⁡𝑤0,𝑤1𝐽(𝑤0,𝑤1)*w*0∗​,*w*1∗​=min*w*0​,*w*1​​*J*(*w*0​,*w*1​)

Для кращого розуміння подивимось на графік функції втрат. Давайте візьмемо 𝑤0=0*w*0​=0 і зобразимо графік залежності функції помилок від 𝑤1*w*1​:



Як ми бачимо, дана функція має чіткий мінімум, і саме те значення 𝑤1*w*1​​, при якому функція втрат досягає свого мінімуму нас і цікавить. Проте якщо ми побудуємо графік залежності функції помилок від обох коефіцієнтів, то будемо мати ідентичну ситуацію:



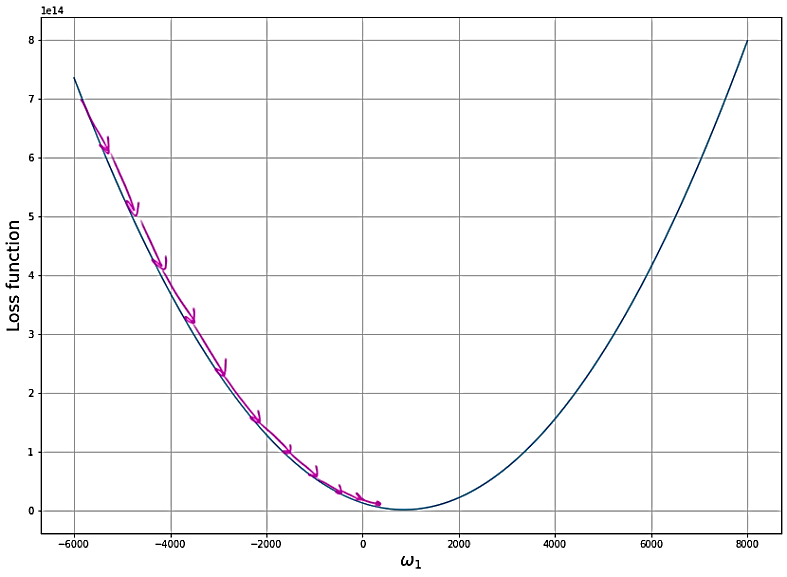
**Градієнтний спуск**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%B3%D1%80%D0%B0%D0%B4%D1%96%D1%94%D0%BD%D1%82%D0%BD%D0%B8%D0%B9-%D1%81%D0%BF%D1%83%D1%81%D0%BA)А щоб знайти такі коефіцієнти 𝑤0,𝑤1*w*0​,*w*1​​, що відповідають мінімальному значенню функції втрат, ми повинні скористатись якимось алгоритмом оптимізації. І основним таким методом є [градієнтний спуск](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D1%96%D1%94%D0%BD%D1%82%D0%BD%D0%B8%D0%B9_%D1%81%D0%BF%D1%83%D1%81%D0%BA) (<https://uk.wikipedia.org/wiki>/Градієнтний\_спуск ).

Градієнтний спуск (Gradient Descent) - це алгоритм оптимізації, який використовується для знаходження мінімуму (або максимуму) функції шляхом ітеративного зменшення (або збільшення) значення функції в напрямку найшвидшого спаду (або зростання) її значення.

В контексті машинного навчання, градієнтний спуск використовується для оптимізації параметрів 𝑤0,𝑤1*w*0​,*w*1​ моделі шляхом мінімізації функції втрат (loss function). Основна ідея полягає у тому, щоб визначити напрямок найшвидшого зменшення функції втрат і змінювати параметри моделі в цьому напрямку. Цей процес повторюється доти, поки значення функції втрат не стане достатньо малим або досягне заданого ліміту ітерацій.

Візуалізувати роботу цього алгоритму можна досить просто:

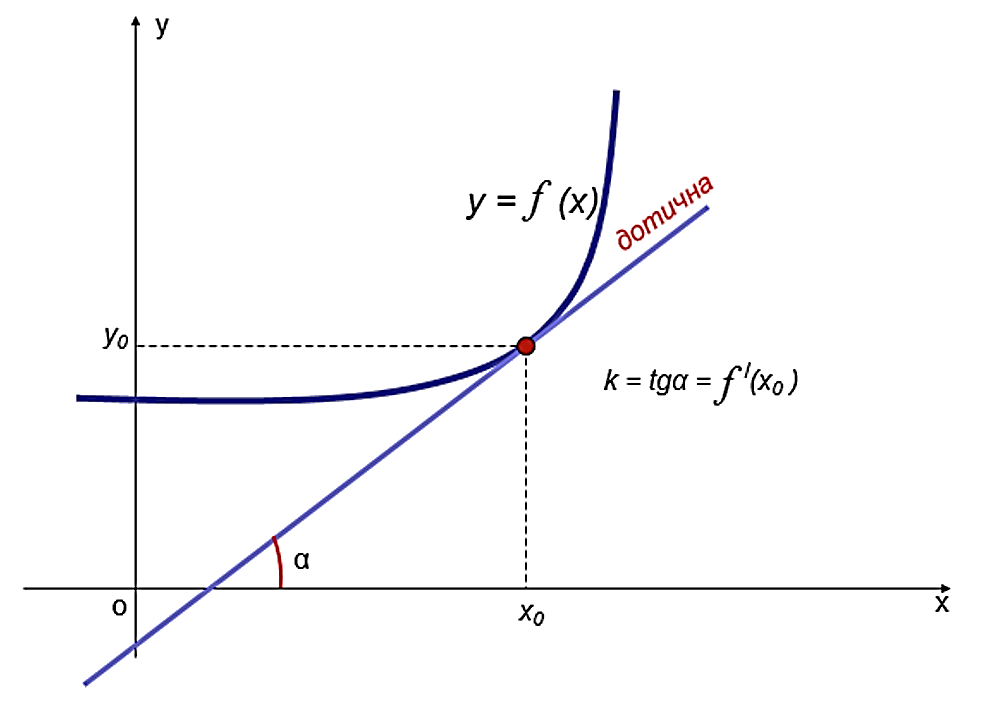


А щоб визначити напрямок зростання (або ж зменшення) функції втрат, ми повинні згадати, що таке [похідна](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%85%D1%96%D0%B4%D0%BD%D0%B0) функції у точці. Нехай у нас функція 𝑓*f* визначена у точці 𝑥0*x*0​​, візьмемо довільну точку 𝑥*x* в околі точки 0*x*0​​, то відношення приросту функції Δ𝑓=∣𝑓(𝑥)−𝑓(𝑥0)∣Δ*f*=∣*f*(*x*)−*f*(*x*0​)∣ до приросту аргумента Δ𝑥=∣𝑥−𝑥0∣Δ*x*=∣*x*−*x*0​∣ буде значенням похідної функції 𝑓*f* у точці 𝑥0*x*0​:

𝑓′(𝑥0)=𝑑𝑓𝑑𝑥=lim⁡Δ𝑥→0Δ𝑓Δ𝑥*f*′(*x*0​)=*dxdf*​=limΔ*x*→0​Δ*x*Δ*f*​

​

а графічно це буде відповідати значенню тангенсу кута нахилу дотичної до графіка функції 𝑓*f* у точці 𝑥0*x*0​



а отже значення похідної показує нам, чи зростає функція - 𝑓′(𝑥0)>0*f*′(*x*0​)>0, або ж спадає - 𝑓′(𝑥0)<0*f*′(*x*0​)<0. Проте у нас функція втрат залежить від двох параметрів 𝑤0,𝑤1*w*0​,*w*1​, тому нам потрібен градієнт функції - вектор часткових похідних по кожному з параметрів:

∇𝐽(𝑤0,𝑤1)=(∂𝐽(𝑤0,𝑤1)∂𝑤0∂𝐽(𝑤0,𝑤1)∂𝑤1)∇*J*(*w*0​,*w*1​)=(∂*w*0​∂*J*(*w*0​,*w*1​)​∂*w*1​∂*J*(*w*0​,*w*1​)​​)

Тоді однією ітерацією роботи алгоритму градієнтного спуску буде таке оновлення значень параметрів 𝑤0,𝑤1*w*0​,*w*1​​:

𝑤0=𝑤0𝑝𝑟𝑒𝑣−𝛼∂𝐽(𝑤0,𝑤1)∂𝑤0*w*0​=*w*0*prev*​−*α*∂*w*0​∂*J*(*w*0​,*w*1​)​

𝑤1=𝑤1𝑝𝑟𝑒𝑣−𝛼∂𝐽(𝑤0,𝑤1)∂𝑤1*w*1​=*w*1*prev*​−*α*∂*w*1​∂*J*(*w*0​,*w*1​)​

де 𝛼*α* - коефіцієнт навчання (leatning rate), який відповідає за те, наскільки великий крок буде робити алгоритм на кожній ітерації. Візуально це вплине на довжину стрілочки на зображенні роботи градієнтного спуску.

Тепер нам потрібно знайти градієнт нашої середньоквадратичної функції втрат. Для цього ми повинні згадати декілька правил диференціювання.

**Ланцюжкове правило (Chain rule)**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%BB%D0%B0%D0%BD%D1%86%D1%8E%D0%B6%D0%BA%D0%BE%D0%B2%D0%B5-%D0%BF%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D0%BE-chain-rule)Ланцюжкове правило диференціювання - це правило, яке дозволяє знайти похідну складеної функції, використовуючи похідні складових функцій та їх взаємозв'язок.

Формула ланцюжкового правила має вигляд:

𝑑𝑓(𝑔(𝑥))𝑑𝑥=𝑑𝑓𝑑𝑔⋅𝑑𝑔𝑑𝑥*dxdf*(*g*(*x*))​=*dgdf*​⋅*dxdg*​

де 𝑓*f* та 𝑔*g* - це функції залежні від змінної 𝑥*x*.

Також нам потрібно знати похідні простих функції, для цього просто скористатись [таблицею похідних](https://uk.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8F_%D0%BF%D0%BE%D1%85%D1%96%D0%B4%D0%BD%D0%B8%D1%85) (<https://uk.wikipedia.org/wiki/> Таблиця\_похідних).

**Навчання нашої моделі**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F-%D0%BD%D0%B0%D1%88%D0%BE%D1%97-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%96)Щоб навчити, натренувати нашу модель, ми повинні застосувати алгорим градієнтного спуску для нашої лінійної регресії. Тому тепер ми обчислимо градієнт помилки для 𝑖*i*-ого прикладу:

𝐽(𝑤0,𝑤1)(𝑖)=(ℎ(𝑥(𝑖))−𝑦(𝑖))2=(𝑤0+𝑤1𝑥(𝑖))−𝑦(𝑖))2*J*(*w*0​,*w*1​)(*i*)=(*h*(*x*(*i*))−*y*(*i*))2=(*w*0​+*w*1​*x*(*i*))−*y*(*i*))2

∂𝐽(𝑤0,𝑤1)(𝑖)∂𝑤0=2⋅(𝑤0+𝑤1𝑥(𝑖))−𝑦(𝑖))⋅1=2(ℎ(𝑥(𝑖))−𝑦(𝑖))∂*w*0​∂*J*(*w*0​,*w*1​)(*i*)​=2⋅(*w*0​+*w*1​*x*(*i*))−*y*(*i*))⋅1=2(*h*(*x*(*i*))−*y*(*i*))

∂𝐽(𝑤0,𝑤1)(𝑖)∂𝑤1=2⋅(𝑤0+𝑤1𝑥(𝑖))−𝑦(𝑖))⋅𝑥(𝑖)=2(ℎ(𝑥(𝑖))−𝑦(𝑖))𝑥(𝑖)∂*w*1​∂*J*(*w*0​,*w*1​)(*i*)​=2⋅(*w*0​+*w*1​*x*(*i*))−*y*(*i*))⋅*x*(*i*)=2(*h*(*x*(*i*))−*y*(*i*))*x*(*i*)

Тоді градієнт для функції втрат буде мати такий вигляд:

∂𝐽(𝑤0,𝑤1)∂𝑤0=12𝑚∑𝑖=1𝑚2(ℎ(𝑥(𝑖))−𝑦(𝑖))=1𝑚∑𝑖=1𝑚(ℎ(𝑥(𝑖))−𝑦(𝑖))∂*w*0​∂*J*(*w*0​,*w*1​)​=2*m*1​∑*i*=1*m*​2(*h*(*x*(*i*))−*y*(*i*))=*m*1​∑*i*=1*m*​(*h*(*x*(*i*))−*y*(*i*))

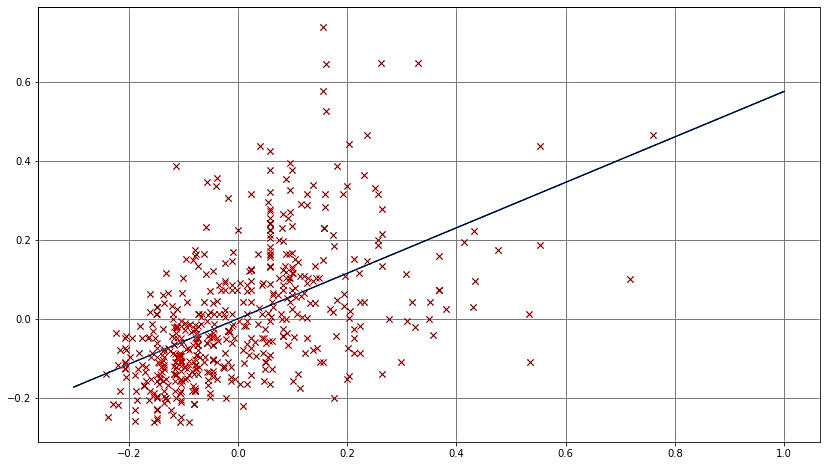
∂𝐽(𝑤0,𝑤1)∂𝑤1=1𝑚∑𝑖=1𝑚(ℎ(𝑥(𝑖))−𝑦(𝑖))𝑥(𝑖)∂*w*1​∂*J*(*w*0​,*w*1​)​=*m*1​∑*i*=1*m*​(*h*(*x*(*i*))−*y*(*i*))*x*(*i*)

Тепер однією ітерацією градієнтного спуску, а відповідно й одним кроком навчання буде така дія:

𝑤0=𝑤0𝑝𝑟𝑒𝑣−𝛼⋅1𝑚∑𝑖=1𝑚(ℎ(𝑥(𝑖))−𝑦(𝑖))*w*0​=*w*0*prev*​−*α*⋅*m*1​∑*i*=1*m*​(*h*(*x*(*i*))−*y*(*i*))

𝑤1=𝑤1𝑝𝑟𝑒𝑣−𝛼⋅1𝑚∑𝑖=1𝑚(ℎ(𝑥(𝑖))−𝑦(𝑖))𝑥(𝑖)*w*1​=*w*1*prev*​−*α*⋅*m*1​∑*i*=1*m*​(*h*(*x*(*i*))−*y*(*i*))*x*(*i*)

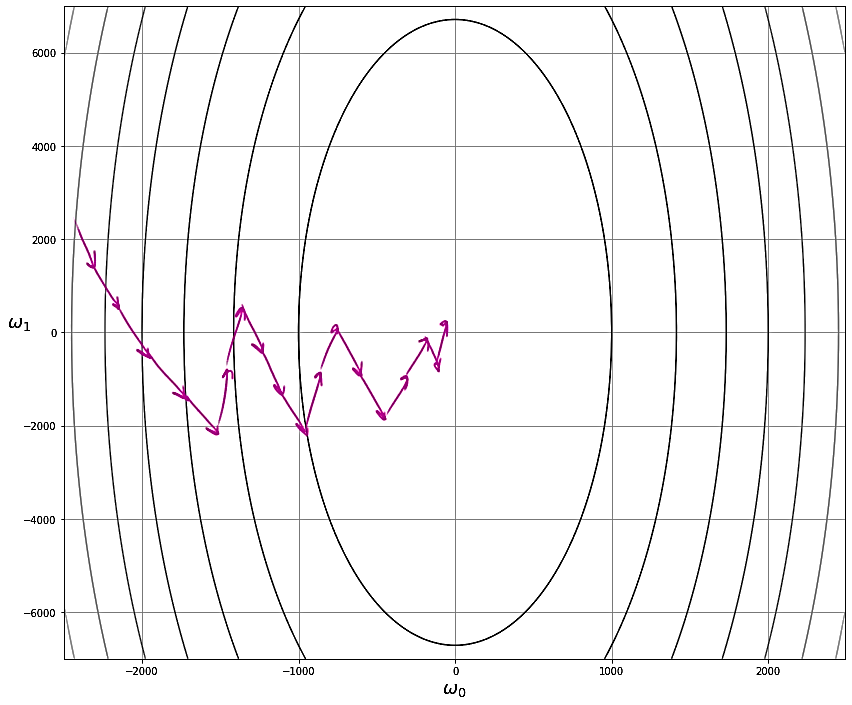
Провівши достатню кількість ітерації ми можемо знайти оптимальні значення наших вагів (параметрів) 𝑤0,𝑤1*w*0​,*w*1​​. Візуалізуємо результат тренування побудувавши лінію, яку ми отримали під час навчання:



**Нормалізація даних**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)Якщо у нас виникає така ситуація, що різні ознаки, та й цільова змінна мають значення різних порядків - наприклад значення площ наших будинків коливається у межах 60-400 м2м2, то їхня вартість вже буде вимірюватись у мільйонах гривень. Якщо ж ми зобразимо контурний графік нашої функції втрат, то побачимо, що вона є не симетричною відносно різних параметрів. А значить, градієнти по різних параметрах будуть мати різне значення, і при одному кроці градієнтного спуску ми по одному параметру будемо швидко рухатись до мінімума функції втрат, а для іншого параметру ці кроки будуть маленькими.

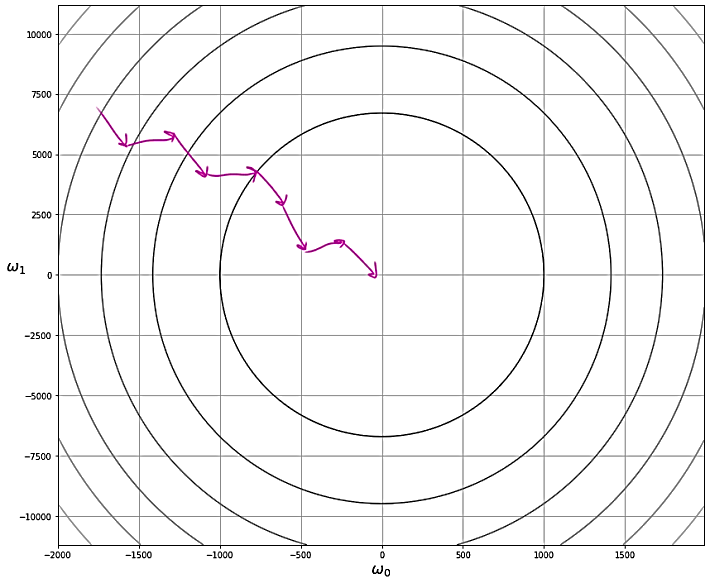
Така ситуація може призвести до того, що наша модель буде дуже довго навчатись, і ми будемо змушені робити велику кількість ітерацій. Візуалізуємо такий випадок:



Щоб уникнути такої ситуації, ми повинні **перемасштабувати** наші ознаки, щоб вони мали значення одного порядку, а ще ми **нормалізуємо** наші дані по середньому значенню, щоб всі значення наших ознак були в межах від −1−1 до 11. Математично ми повинні провести таку математичну операцію:

𝑥𝑖𝑛𝑜𝑟𝑚:=𝑥𝑖−𝜇𝑖𝑠𝑖*xinorm*​:=*si*​*xi*​−*μi*​​

де 𝜇𝑖*μi*​ це середнє значення 𝑖*i*-ої ознаки, а 𝑠𝑖*si*​​ - діапазон значень 𝑚𝑎𝑥−𝑚𝑖𝑛*max*−*min* цієї ж ознаки, часто замість діапазону використовують стандартне відхилення 𝜎𝑖*σi*​​. В такому випадку ми зробимо функцію втрат більш симетричною та навчання буде відбуватись швидше:



Більше варіантів трансформації вхідних даних можна подивитись на офіційному сайті scikit-learn у відповідному [розділі](https://scikit-learn.org/stable/modules/preprocessing.html#standardization-or-mean-removal-and-variance-scaling)

(https://scikit-learn.org/stable/modules/preprocessing.html#standardization-or-mean-removal-and-variance-scaling).

**Лінійна регресія з багатьма змінними**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%BB%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%96%D1%8F-%D0%B7-%D0%B1%D0%B0%D0%B3%D0%B0%D1%82%D1%8C%D0%BC%D0%B0-%D0%B7%D0%BC%D1%96%D0%BD%D0%BD%D0%B8%D0%BC%D0%B8)Ми розібрались як працює лінійна регресія на прикладі залежності ціни на будинок лише від його площі. Проте, якщо ми маємо також інформацію про кількість поверхів, кількість кімнат, наявність парковки, прибудинкової території і т.д., та хочемо ці ознаки (фічі) також використати для побудови більш складної та більш точної моделі, то ми можемо скористатись лінійною регресією з багатьма змінними. Тоді наші дані будуть мати вже трішки інший вигляд:

* 𝑥𝑗(𝑖)*xj*(*i*)​- 𝑗*j*-та ознака (фіча) 𝑖*i*-ого прикладу (sample, example) з нашого датасету
* 𝑥⃗(𝑖)*x*(*i*) - вектор ознак для 𝑖*i*-ого прикладу, і тепер це буде наш вхідний вектор значень
* 𝑚*m* - кількість прикладів у нашому датасеті
* 𝑛*n* - кількість ознак для прикладів у датасеті

Тоді наша гіпотеза буде мати такий вигляд:

ℎ(𝑥⃗)=𝑤0+𝑤1𝑥1+𝑤2𝑥2+𝑤3𝑥3+⋯+𝑤𝑛𝑥𝑛*h*(*x*)=*w*0​+*w*1​*x*1​+*w*2​*x*2​+*w*3​*x*3​+⋯+*wn*​*xn*​

​

Якщо ми введемо додаткову ознаку 𝑥0=1*x*0​=1, тоді ми можемо ввести позначення вектора вагів та повний вектор ознак:

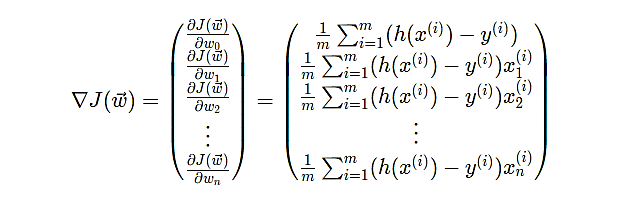
𝑥⃗=(1𝑥1𝑥2𝑥3⋮𝑥𝑛)𝑤⃗=(𝑤0𝑤1𝑤2𝑤3⋮𝑤𝑛)*x*=​1*x*1​*x*2​*x*3​⋮*xn*​​​*w*=​*w*0​*w*1​*w*2​*w*3​⋮*wn*​​​

​

Тоді нашу гіпотезу можна переписати у векторному вигляді, оскільки це буде просто скалярний добуток вектору вагів на вектор ознак:

ℎ(𝑥⃗)=𝑤⃗⋅𝑥⃗*h*(*x*)=*w*⋅*x*

Для знаходження оптимальних значень наших коефіцієнтів ми знову скористаємось методом градієнтного спуску, проте тепер вектор-градієнт функції втрат також буде мати більше елементів:



Ну а відповідно одна ітерація навчання, чи крок нашого градієнтного спуску буде мати такий вигляд:

𝑤⃗=𝑤⃗𝑝𝑟𝑒𝑣−𝛼∇𝐽(𝑤⃗)*w*=*wprev*−*α*∇*J*(*w*)

Якщо ми введемо ще матрицю значень нашого датасету, то взагалі можемо робити векторні обчислення, замість сумування циклами:

𝑋=[1𝑥1(1)𝑥2(1)𝑥3(1)⋯𝑥𝑛(1)1𝑥1(2)𝑥2(2)𝑥3(2)⋯𝑥𝑛(2)1𝑥1(3)𝑥2(3)𝑥3(3)⋯𝑥𝑛(3)⋮⋮⋮⋮⋱⋮1𝑥1(𝑚)𝑥2(𝑚)𝑥3(𝑚)⋯𝑥𝑛(𝑚)]*X*=​111⋮1​*x*1(1)​*x*1(2)​*x*1(3)​⋮*x*1(*m*)​​*x*2(1)​*x*2(2)​*x*2(3)​⋮*x*2(*m*)​​*x*3(1)​*x*3(2)​*x*3(3)​⋮*x*3(*m*)​​⋯⋯⋯⋱⋯​*xn*(1)​*xn*(2)​*xn*(3)​⋮*xn*(*m*)​​​

​

яка по суті і є таблицею наших даних, наприклад у pandas, тоді матимемо такі формули для обчислень:

∇𝐽(𝑤⃗)=1𝑚𝑋𝑇(𝑋𝑤⃗−𝑦⃗)∇*J*(*w*)=*m*1​*XT*(*Xw*−*y*​)

А векторні обчислення виконуються набагато ефективніше, ніж цикли.

**Normal equation**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#normal-equation)

Давайте трохи поміркуємо. Якщо ми перепишемо нашу гіпотезу у матричному вигляді:

ℎ⃗(𝑋)=𝑋𝑤⃗*h*(*X*)=*Xw*

ми можемо порівняти отримані значення з вектором цільових значень, і таким чином отримати матричне рівняння 𝑋𝜔=𝑦*Xω*=*y*. Ми можемо розв'язати дане матричне рівнняння завдяки силі лінійної алгебри і отримати таке аналітичне рішення:

𝑤⃗∗=(𝑋𝑇𝑋)−1𝑋𝑇𝑦⃗*w*∗=(*XTX*)−1*XTy*​

​

Вираз праворуч носить назву [псевдооберненої матриці](https://uk.wikipedia.org/wiki/%D0%9F%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D0%BE%D0%B1%D0%B5%D1%80%D0%BD%D0%B5%D0%BD%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F). Було б чудово, якби в цьому місці можна було закінчити оповідання. Однак, на жаль, на практиці справа йде зовсім не так гладко, як може здатися. Для знаходження рішення необхідно обчислювати зворотну матрицю на що буде потрібно порядку 𝑘3*k*3 операцій при квадратній матриці розміру 𝑘*k*. На реальних даних це досить невигідно. Крім того, якщо матриця погано обумовлена, ми не можемо використовувати чисельні методи обернення. Тому вирішення задачі мінімізації функції помилки знаходять за допомогою методів оптимізації, як градієнтний спуск.

**Розв'язання задачі лінійної регресії за допомогою sklearn**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D1%80%D0%BE%D0%B7%D0%B2%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D1%96-%D0%BB%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%BE%D1%97-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%96%D1%97-%D0%B7%D0%B0-%D0%B4%D0%BE%D0%BF%D0%BE%D0%BC%D0%BE%D0%B3%D0%BE%D1%8E-sklearn)

У цій бібліотеці є безліч різних корисних механізмів. У тому числі і для вирішення задачі лінійної регресії. Нам знадобиться інстанс класу LinearRegression, який можна навчити передавши у метод fit навчальну вибірку. Далі навчена модель вже може робити передбачення, для цього потрібно скористатися методом predict.

Приклад

import numpy as np

from sklearn.linear\_model import LinearRegression

X = np.array([

[1, 1],

[1, 2],

[2, 2],

[2, 3]

])

y = np.dot(X, np.array([1, 2])) + 3

regressor = LinearRegression().fit(X, y)

print(regressor.predict(np.array([[3, 5]])))

Як бачите sklearn допоміг нам умістити всю наведену вище математику в кількох рядках коду. Зазначимо лише, що у нашому "іграшковому" прикладі ми вручну згенерували навчальну вибірку - матриці X та y. Насправді ж ці дані ми взяли б звідкись ззовні.

**Замість висновку**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main#%D0%B7%D0%B0%D0%BC%D1%96%D1%81%D1%82%D1%8C-%D0%B2%D0%B8%D1%81%D0%BD%D0%BE%D0%B2%D0%BA%D1%83)

У цьому розділі ми познайомилися із завданням лінійної регресії, а також способах її вирішення. Далі ми побачимо як після деяких перетворень розглянута задача регресії може бути адаптована для вирішення задачі класифікації.