**Модуль 3.2: Класичне машинне навчання**

**Перенавчання**

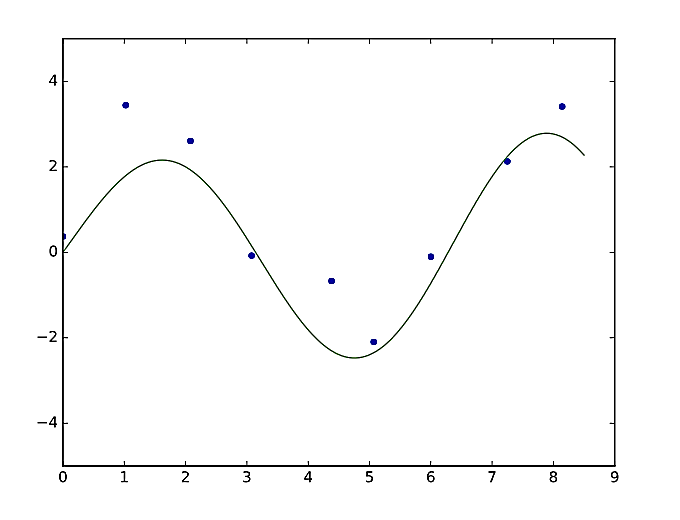
**Вступ**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%B2%D1%81%D1%82%D1%83%D0%BF)У цьому розділі ми обговоримо одне дуже небажане явище в машинному навчанні, яке псує якість збудованих моделей. Ми постараємося з'ясувати, через що воно з'являється, а головне як з ним боротися.

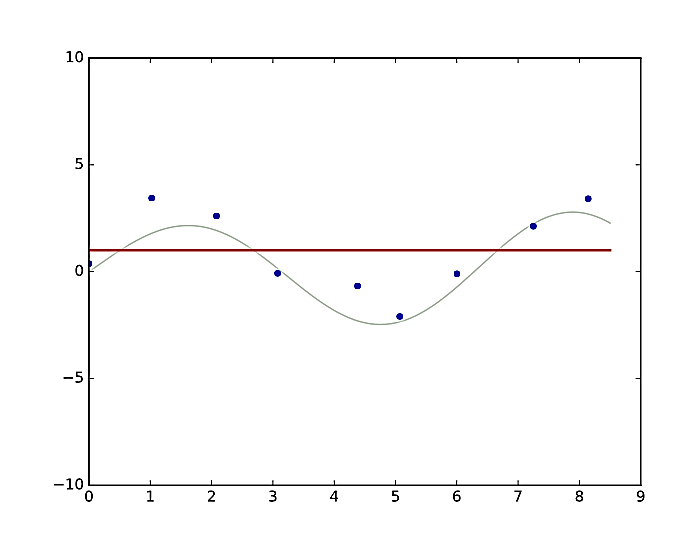
**Перенавчання**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%BF%D0%B5%D1%80%D0%B5%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F-1)Допустимо при розв'язанні задачі класифікації було побудовано певний алгоритм. Нехай це буде лінійний класифікатор, причому частка помилок на об'єктах з навчальної вибірки дорівнювала 0.2 така частка помилок є допустимою. Однак немає жодних гарантій, що така сама частка помилок буде досягатися на новій вибірці. Цілком може виникнути ситуація, що для нової вибірки **помилка** дорівнюватиме 0.9 або більше. Це означає, що алгоритм не зміг узагальнити навчальну вибірку. Іншими словами, він не зміг витягти з неї закономірності і застосувати їх для класифікації нових об'єктів. При цьому алгоритм підлаштувався саме під **навчальну** вибірку і показав хороші результати при навчанні **без** вилучення **істинної**закономірності. Явище з яким ми зіткнулися називається перенавчання.

Глибше зрозуміти проблему перенавчання можна на наступному прикладі. На графіку зображена істинна залежність та об'єкти навчальної вибірки:



Видно, що істинна залежність нелінійна. У моделі *a*(*x*)=*ω*0​​, після того, як вона буде налаштована на дані, на графіку виходить деяка горизонтальна крива, яка досить погано узагальнює інформацію про об'єкти з вибірки.

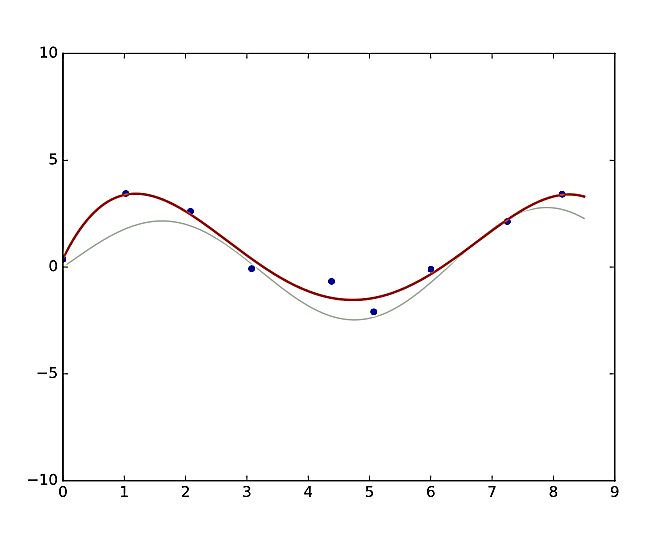


Має місце, так зване, **недонавчання**(underfitting). Хороший алгоритм не був побудований, оскільки сімейство алгоритмів занадто мало і з його допомогою неможливо вловити закономірність. Розглянемо безліч алгоритмів виду *a*(*x*)=*ω*0​+*ω*1​*x*. Така безліч набагато багатша, але і її не вистачає, щоб узагальнити подану нелінійну залежність. Якщо ми спробуємо навчити таку лінійну модель, то отримаємо трохи кращий результат, але все ж дуже далекий від прийнятного. У цьому випадку також матиме місце недонавчання.

Можна спробувати використати безліч багаточленів 4-го ступеня

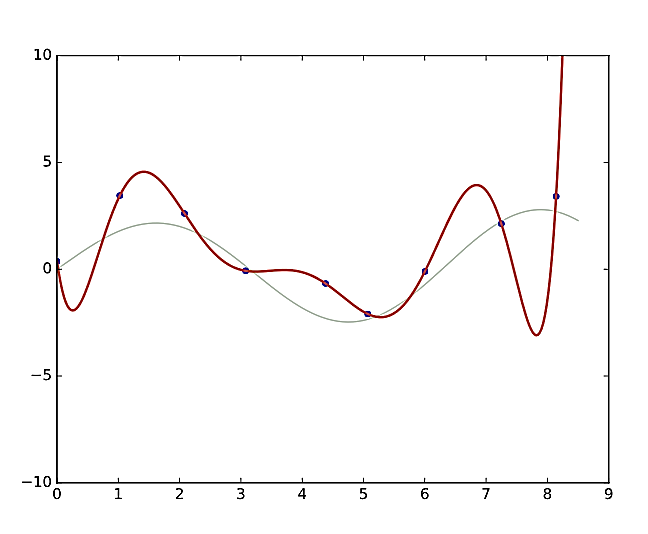
*a*(*x*)=*ω*0​+*ω*1​*x*+*ω*2​*x*2+…+*ω*3​*x*4

в цьому випадку після навчання крива буде досить добре описувати як навчальну вибірку, так і істинну залежність.



Видно, що якість алгоритму хороша, але ідеального збігу немає.

Виникає закономірне питання. А чи можна досягти поліпшень за рахунок використання багатшого сімейства алгоритмів? Що якщо ми ускладнимо модель і використаємо поліном 9-го ступеня?



Відновлена залежність дає ідеальні відповіді на всіх об'єктах навчальної вибірки, але при цьому в будь-якій іншій точці дуже сильно відрізняється від справжньої залежності. Ми зіткнулися з **перенавчанням**. Алгоритм дуже підлаштувався під навчальну вибірку і в результаті він дає погані відповіді на нових точках.

Таким чином, недонавчання — ситуація, коли алгоритм погано описує і навчальну вибірку, і нові дані. У цьому випадку алгоритм необхідно ускладнювати. У разі перенавчання дані з навчальної вибірки будуть описуватися добре, а нові - погано. Виявити перенавчання, використовуючи лише навчальну вибірку, неможливо. Справді, неперенавчений і перенавчені алгоритми добре її описуватимуть. Необхідно використати додаткові дані. Існує декілька підходів до виявлення перенавчання:

* Відкладена вибірка. Частина даних з навчальної вибірки не беруть участь у навчанні, щоб пізніше перевіряти на ній навчений алгоритм.
* Крос-валідація, дещо ускладнений метод відкладеної вибірки.
* Використовувати міри складності моделі.

**Регуляризація**

​Очевидно, перенавчання це дуже неприємне та небажане явище. Проте йому можна протистояти. Зараз ми поговоримо про регуляризацію — способі боротьби з перенавчанням у лінійних моделях.

Розглянемо деякі "симптоми", які можуть свідчити про факт перенавчання. Одним із сигналів про наявність перенавченості моделі є великі ваги при ознаках. Наприклад, у попередньому розділі під час навчання моделі

*a*(*x*)=*ω*0​+*ω*1​*x*+*ω*2​*x*2+…+*ω*9​*x*9

ваги вийшли дуже великими

*a*(*x*)=0.5+12458922*x*+43983740*x*2+…+2740*x*9

Інша ситуація, в якій можна зустрітися з перенавчанням - мультиколлінеарність. Так називається проблема, при якій ознаки у вибірці є лінійно залежними. Іншими словами, існують коефіцієнти α1,α2,…,αd*α*1​,*α*2​,…,*αd*​ такі, що для будь-якого об'єкту xi*xi*​​ з вибірки виконується:

⟨*α*,*xi*​⟩=*α*1​*xi*1​+*α*2​*xi*2​…+*αd*​*xid*​=0

Припустимо, було знайдено розв'язання задачі оптимізації:

*ω*∗=argmin*ω*​*n*1​∑*i*=1*n*​(⟨*ω*,*xi*​⟩−*yi*​)2

Тепер розглянемо якийсь інший вектор ваг, отриманий зрушенням у напрямку вектора α*α*:

*ωt*=*ω*∗+*tα*

В цьому випадку для елементів xi*xi*​ виконується співвідношення

⟨*ω*∗+*tα*,*xi*​⟩=⟨*ω*∗,*xi*​⟩+*t*⟨*α*,*xi*​⟩=⟨*ω*∗,*xi*​⟩

Отже ωt*ωt* також буде рішенням задачі оптимізації. Модель із параметрами ωt*ωt* буде також добре описувати дані у вибірці, як і вихідний алгоритм. Більше того, за наявності лінійної залежності між якимись ознаками рішеннями вихідної задачі оптимізації є безліч алгоритмів. Однак, багато з них мають великі ваги, і далеко не всі володіють гарною узагальнюючою здатністю. Перед нами черговий приклад перенавчання.

Як бачимо, великі ваги моделі можуть свідчити про перенавчання. Щоб боротися із цим замість мінімізації функціоналу помилки *Q*(*a*,*X*) мінімізують новий функціонал, що отримується додаванням **регуляризатора.** Найпростіший регуляризатор - квадратичний регуляризатор:

∣∣*ω*∣∣=∑*j*=1*d*​*ωj*2​

​

Зведемо завдання навчання моделі до мінімізації наступного виразу

(*ω*,*X*)+*λ*∣∣*ω*∣∣2→min*ω*​

​

Мінімізуючи цю функцію ми шукаємо такий вектор *ω* при якому величина *Q*(*ω*,*X*) приймає мінімальне значення і при цьому сам вектор *ω* має мінімально можливу довжину. Останнє гарантує нам відсутність великих ваг. Таким чином, під час навчання буде враховуватися також те, що не слід занадто сильно збільшувати ваги ознак.

**Коефіцієнт регуляризації**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%BA%D0%BE%D0%B5%D1%84%D1%96%D1%86%D1%96%D1%94%D0%BD%D1%82-%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D1%96%D1%97)Введений вище коефіцієнт *λ*, який стоїть перед регуляризатором, називається коефіцієнтом регуляризації. Чим більше *λ*, тим нижче складність моделі. Наприклад, при дуже великих його значеннях оптимально просто занулити всі ваги. У той же час, при занадто низьких значеннях *λ* високий ризик перенавчання, тобто модель стає занадто складною. Тому потрібно знайти певне оптимальне значення *λ*, досить велике, щоб не допустити перенавчання, і не дуже велике, щоб вловити закономірності в даних. Зазвичай *λ* підбирається на крос-валідації.

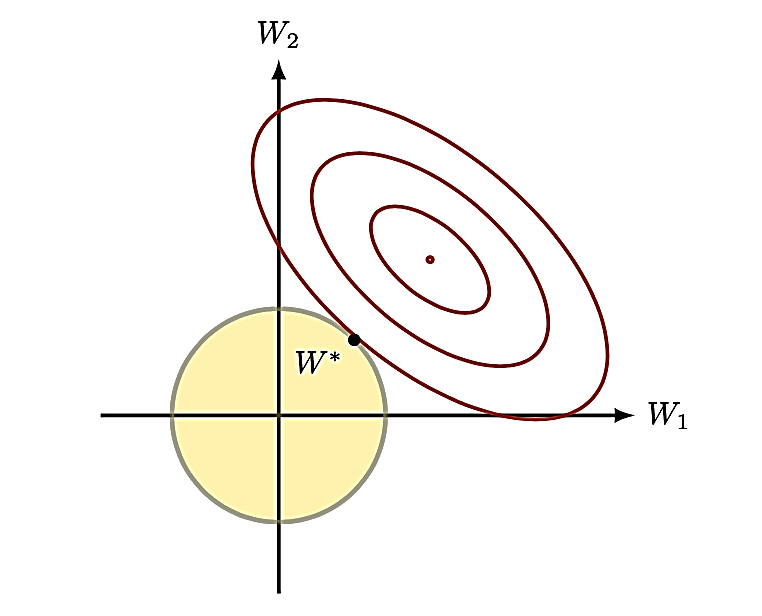
**Сенс регуляризації**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D1%81%D0%B5%D0%BD%D1%81-%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D1%96%D1%97)Щоб краще усвідомити сенс регуляризації, можна замість задачі оптимізації з квадратичним оптимізатором розглянути задачу умовної оптимізації:

{*Q*(*ω*,*X*)→min*ω*​∣∣*ω*∣∣2≤*C*​

​

Додавання регуляризатора вводить вимогу, щоб вирішення задачі мінімізації шукалося в деякій круглій області з центром у нулі.



Таким чином, вирішення задачі з регуляризатором не характеризуватиметься надто великими значеннями вагових коефіцієнтів.

**Види регуляризаторів**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%B2%D0%B8%D0%B4%D0%B8-%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80%D1%96%D0%B2)Розглянутий вище квадратичний регуляризатор (*L*2​​-регуляризатор) є гладким і опуклим, що дозволяє використовувати градієнтний спуск. Також існує *L*1​-регуляризатор:

∣∣*ω*∣∣1​=∑*j*=1*d*​∣*ωj*​∣

Який представляє собою *L*1​​-норму вектора ваг. Він уже не є гладким, а також має цікаву властивість. Якщо застосовувати такий регуляризатор, деякі ваги виявляються рівними нулю. Іншими словами, такий регуляризатор здійснює відбір ознак і дозволяє використовувати в моделі не всі ознаки, а лише найважливіші з них.

Розглянуті нами раніше класи [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) (<https://scikit-learn.org/stable/> modules/generated/sklearn.linear\_model.LogisticRegression.html) та [RidgeClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifier.html)

(<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model>. RidgeClassifier.html) використовують регуляризацію.

У класу RidgeClassifier він називається α*α* і за умовчанням дорівнює11**.**

Розглянемо приклад.

from sklearn.datasets import load\_breast\_cancer

from sklearn.linear\_model import RidgeClassifier

X, y = load\_breast\_cancer(return\_X\_y=True)

clf = RidgeClassifier(alpha=30.0).fit(X, y)

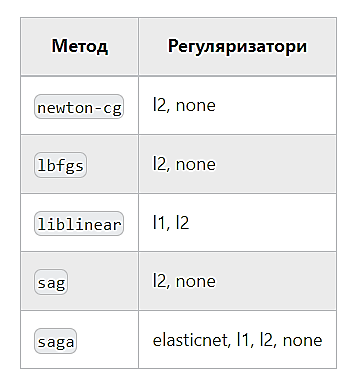
clf.score(X, y)

У класу LogisticRegression є параметр penalty, який задає наступні види регуляризації:

* None - регуляризація повністю відсутня
* "l2" - *L*2​ регуляризація, встановлена за замовчуванням
* "l1" - *L*1​ регуляризація
* "elasticnet" - використання комбінації обох *L*1​ та *L*2​ регуляризаторів

**Зауваження**

При використанні регулятора *L*1​ функція помилки перестає бути гладкою, тому ми більше не маємо можливості використовувати градієнтні методи оптимізації. Тому при виборі цього регуляризатора швидше за все доведеться додатково встановити параметр solver, який відповідає за вибір алгоритму мінімізації помилки. У наступній таблиці наведено список солверів та доступних для них регуляризаторів:



Приклад[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4)

from sklearn.datasets import load\_iris

from sklearn.linear\_model import LogisticRegression

X, y = load\_iris(return\_X\_y=True)

clf = LogisticRegression(penalty="l1", solver="liblinear", random\_state=0).fit(X, y)

clf.predict(X[:2, :])

clf.predict\_proba(X[:2, :])

clf.score(X, y)

**Оцінка якості алгоритмів**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%BE%D1%86%D1%96%D0%BD%D0%BA%D0%B0-%D1%8F%D0%BA%D0%BE%D1%81%D1%82%D1%96-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D0%B2)Давайте обговоримо яким чином можна оцінити якість роботи навченої моделі на **нових** даних. Як вже було сказано, перенавчання складно виявити, використовуючи лише навчальну вибірку. Дійсно, адекватний і перенавчений алгоритми будуть показувати хорошу якість на об'єктах навчальної вибірки. Зрозуміло з перенавчанням можна боротися за допомогою регуляризаторів. Однак, вони не дають відповіді на питання, наскільки добре алгоритм поведе себе на нових даних. Тому нам потрібні якісь ще підходи.

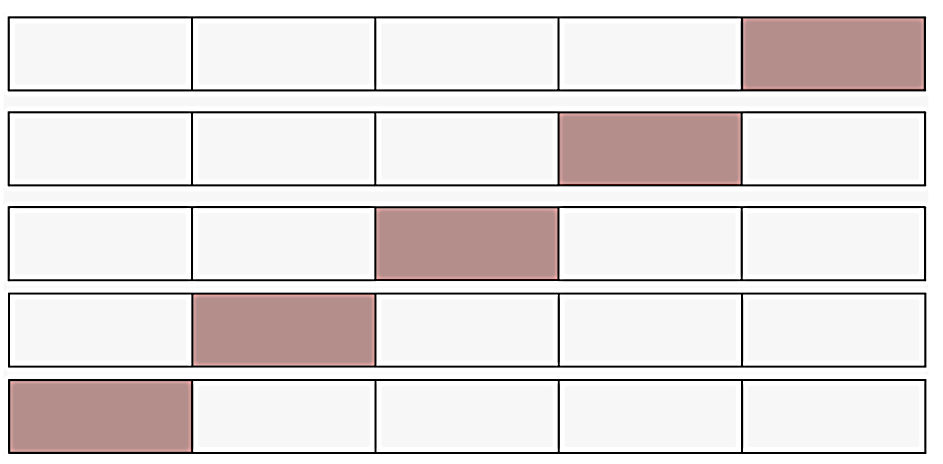
**Відкладена вибірка**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%B2%D1%96%D0%B4%D0%BA%D0%BB%D0%B0%D0%B4%D0%B5%D0%BD%D0%B0-%D0%B2%D0%B8%D0%B1%D1%96%D1%80%D0%BA%D0%B0)Найпростіший спосіб оцінити якість алгоритму - використання відкладеної вибірки. У цьому випадку слід розбити вибірку на дві частини: перша із двох частин буде використовуватися для навчання алгоритму, а друга - для оцінки його якості, у тому числі для знаходження частки помилок у задачі класифікації, MSE (середньоквадратичної помилки) у завданні регресії та інших заходів якості, залежно від специфіки завдання. Зазвичай вибірку, на якій відбувається навчання називають тренувальною, а ту, де далі тестується модель - тестовою. Виникає природне питання в якій пропорції проводити розбиття. Якщо взяти тестову вибірку надто маленькою, оцінка якості буде ненадійною, хоча навчальна вибірка майже збігається з повною вибіркою. В іншому випадку, якщо відкладена частина буде великою, оцінка якості буде надійною, але низька якість алгоритму може свідчити про недостатній обсяг навчальної частини вибірки. Зазвичай вибірку розбивають у співвідношеннях 70/30, 80/20. Перевагою відкладеної вибірки є те, що навчати алгоритм доводиться лише один раз, але при цьому результат сильно залежить від того, як було здійснено розбиття. Наприклад, нехай оцінюється вартість житла за деякими ознаками. І є особлива категорія житла - двоповерхові квартири. Якщо виявиться, що всі двоповерхові квартири, яких небагато, потрапили у відкладену вибірку, то після навчання алгоритм даватиме на них дуже погану якість, оскільки в навчальній вибірці таких об'єктів не було. Щоб вирішити цю проблему, можна використати наступний підхід: побудувати n*n* різноманітних розбиттів вибірки на 2 частини, для кожного розбиття знайти оцінку якості, а як підсумкову оцінку якості роботи алгоритму використовувати усереднене, за всіма розбиттями значення. Але і в цьому випадку, оскільки розбиття будуються випадково, немає жодних гарантій, що особливий об'єкт хоча б раз потрапить на навчання.

**Крос-валідація**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%BA%D1%80%D0%BE%D1%81-%D0%B2%D0%B0%D0%BB%D1%96%D0%B4%D0%B0%D1%86%D1%96%D1%8F)Більш системний підхід - крос валідація. У цьому випадку вибірка поділяється на k*k* блоків приблизно однакового розміру. Далі по черзі кожен із цих блоків використовується як тестовий, а всі інші - в якості навчальної вибірки. Після того, як кожен блок побуває в якості тестового, будуть отримані k*k* показників якості. В результаті усереднення виходить оцінка якості по крос-валідації. При цьому постає питання, яку кількість блоків використовувати. Якщо блоків мало, виходять надійні, але зміщені оцінки. У разі великого числа блоків, оцінки виходять незміщеними, але ненадійними, оскільки мають великий розкид. Конкретних рекомендацій щодо вибору *k* немає. Зазвичай обирають *k*=3,5,10. Чим більше *k*, тим більше разів доводиться навчати алгоритм. Тому на великих вибірках слід вибирати невеликі значення *k*, оскільки навіть при видаленні 1/3 ​вибірки даних, що залишилися буде достатньо для навчання.

Логіку розбиття можна проілюструвати на наступному малюнку.



Тут датасет складається із 5 умовних блоків. Ми можемо взяти останній блок як тестовий, а на інших навчити модель. Далі можна взяти передостанній блок як тестовий, а на решті провести навчання і так далі. Оцінку з крос-валідації можна визначити наступним чином:

*CV*5​(*a*(*x*),*X*)=51​∑*i*=15​*Q*(*a*(*x*),*Xi*​)

Де *Xi*​ - тренувальна вибірка, отримана на *i*-м розбитті.

Розглянемо приклад

import numpy as np

from sklearn.base import clone

from sklearn.model\_selection import StratifiedKFold

from sklearn.datasets import fetch\_openml

from sklearn.linear\_model import SGDClassifier

mnist = fetch\_openml("mnist\_784", version=1)

X, y = mnist["data"], mnist["target"]

y = y.astype(np.uint8)

X\_train, X\_test, y\_train, y\_test = X[:60000], X[60000:], y[:60000], y[60000:]

y\_train\_5 = (y\_train == 5)

y\_test\_5 = (y\_test == 5)

sgd\_clf = SGDClassifier(random\_state=42)

skfolds = StratifiedKFold(n\_splits=4, random\_state=42)

for train\_index, test\_index in skfolds.split(X\_train, y\_train\_5):

clone\_clf = clone(sgd\_clf)

X\_train\_folds = X\_train[train\_index]

y\_train\_folds = y\_train\_5[train\_index]

X\_test\_fold = X\_train[test\_index]

y\_test\_fold = y\_train\_5[test\_index]

clone\_clf.fit(X\_train\_folds, y\_train\_folds)

y\_pred = clone\_clf.predict(X\_test\_fold)

n\_correct = sum(y\_pred == y\_test\_fold)

print(n\_correct / len(y\_pred))

Часто дані у файлі записані у відсортованому вигляді за якоюсь ознакою. Тому завжди слід перемішувати вибірку перш ніж робити крос-валідацію. В іншому випадку алгоритм показуватиме погану якість і причина цього буде не так очевидна.

**Зауваження**

Є завдання, у яких вибірку не можна перемішувати. Це завдання передбачення майбутнього, наприклад прогноз курсу акцій. У цьому випадку потрібно особливо стежити за тим, як відбувається поділ вибірки.

**Гіперпараметри**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%B3%D1%96%D0%BF%D0%B5%D1%80%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B8)Гіперпараметрами називаються такі параметри алгоритмів, які не можуть бути отримані з навчальної вибірки при навчанні, тому їх треба підбирати шляхом багаторазового навчання алгоритму. Прикладами гіперпараметрів є:

* Параметр регуляризації *λ*
* Ступінь полінома в задачі регресії з сімейством алгоритмів, заданим безліччю поліномів деякого ступеня.

**Порівняння різних алгоритмів**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%BF%D0%BE%D1%80%D1%96%D0%B2%D0%BD%D1%8F%D0%BD%D0%BD%D1%8F-%D1%80%D1%96%D0%B7%D0%BD%D0%B8%D1%85-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D0%B2)Більш загальне завдання - порівняння різних алгоритмів:

* навчених з різними значеннями гіперпараметрів;
* які використовують різний спосіб регуляризації;
* налаштованих з використанням різного функціоналу помилки, наприклад середньоквадратичної помилки та середньої абсолютної помилки;
* які належать різним класам алгоритмів.

При порівнянні алгоритмів можна використовувати як відкладену вибірку, так і крос-валідацію, але при цьому слід дотримуватись обережності. Дійсно, нехай 100 алгоритмів порівнюються за якістю на відкладеній вибірці. Кожен із цих 100 алгоритмів, навчених на навчальній вибірці, тестується на відкладеній, і в результаті вибирається найкращий. Фактично на цьому етапі відкладена вибірка в певному сенсі стає навчальною. Виникає проблема перенавчання, оскільки з великої кількості алгоритмів вибирається той, який найкраще веде себе на відкладеній вибірці, краще підігнаний під неї.

**Удосконалена схема порівняння алгоритмів**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D1%83%D0%B4%D0%BE%D1%81%D0%BA%D0%BE%D0%BD%D0%B0%D0%BB%D0%B5%D0%BD%D0%B0-%D1%81%D1%85%D0%B5%D0%BC%D0%B0-%D0%BF%D0%BE%D1%80%D1%96%D0%B2%D0%BD%D1%8F%D0%BD%D0%BD%D1%8F-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D0%B2)Щоб боротися з цим, слід використати дещо вдосконалену схему оцінювання якості алгоритмів, а саме всі дані потрібно буде ділити на 3 частини: **навчання, валідація** та **контроль**. Кожен із тисячі алгоритмів буде навчений на навчальній вибірці, а його якість буде виміряна на валідаційній. Алгоритм із найкращою якістю буде перевірено на контрольній вибірці, щоб виключити перенавчання та перевірити алгоритм на адекватність. По суті саме контрольна вибірка відіграватиме роль нових даних. Якщо бажано використовувати крос-валідацію, дані слід розбити на 2 частини. Перша з них буде використовуватися для навчання алгоритмів та оцінки якості за допомогою крос-валідації, після чого найкращий алгоритм буде перевірено на адекватність на контрольній вибірці.

**Висновки**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%B2%D0%B8%D1%81%D0%BD%D0%BE%D0%B2%D0%BA%D0%B8)У цьому розділі ми обговорили основні підходи, які застосовуються у боротьбі з перенавчанням моделей машинного навчання.

**Навчальні ноутбуки**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-03/main-03#%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D1%96-%D0%BD%D0%BE%D1%83%D1%82%D0%B1%D1%83%D0%BA%D0%B8)[Стратегії крос-валідації](https://colab.research.google.com/drive/1cVBCl2rRXfCUCCyJhwZRJH3bPKhQZP_v?usp=sharing)

(https://colab.research.google.com/drive/1cVBCl2rRXfCUCCyJhwZRJH3bPKhQZP\_v?usp=sharing)

1. [Навчання лінійних моделей частина 1](https://colab.research.google.com/drive/11AHfR6q9Wuqae0-tdPtSazXaU5-Lzwu7?usp=sharing)

(https://colab.research.google.com/drive/11AHfR6q9Wuqae0-tdPtSazXaU5-Lzwu7?usp=sharing)

1. [Навчання лінійних моделей частина 2](https://colab.research.google.com/drive/1Y-aerV6uNK37aFk7xigWSrCTVQAcBXO-?usp=sharing)

(https://colab.research.google.com/drive/1Y-aerV6uNK37aFk7xigWSrCTVQAcBXO-?usp=sharing)