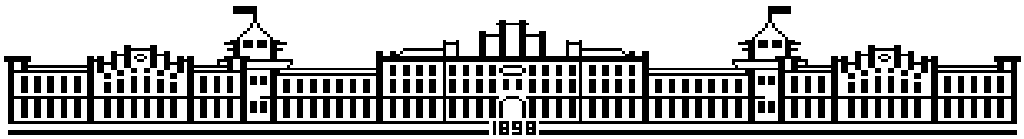


Міністерство освіти та науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Кафедра інформаційних систем на технологій

Методичні вказівки до комп’ютерних практикумів з дисципліни **«Технології машинного навчання»**

Підготував: асистент кафедри ІСТ
Коломоєць Сергій Олексійович

Київ 2024

Зміст

Інформація про практичні заняття

Кількість практичних занять – 9. В рамках курсу необхідно виконати і захистити 4 роботи, що відповідають темам курсу. Робота може виконуватись в бригадах (не більше 3 студентів). Максимальна кількість балів за практику – 50.

№	**Тема роботи**	Максимальна кількість балів
1	Метрики якості задач класифікації	13
2	Функції помилок (втрат) у машинному навчанні	13
3	Вступ до опрацювання природної мови	12
4	Згорткові мережі та робота з зображеннями	12

Практика "Технології машинного навчання"

Увійти в конференцію Zoom
https://us02web.zoom.us/j/89752819748?pwd=PNak00HHSUj19fm7SLqkpIVPKp2iiq.1

Ідентифікатор конференції: 897 5281 9748
Код доступу: 212940

Порядок захисту роботи

Студент для захисту попередньо надсилає оформлений звіт на пошту викладачу serhii.o.kolomoiets@ukr.net (файл називати у форматі «ТМН-[група]-Лаб[номер роботи]-ПІБ»). Тема повідомлення така ж як назва файлу. В процесі захисту студент або бригада демонструє програмну реалізацію та відповідає на запитання.

Комп’ютерний практикум № 1.

Метрики якості задач класифікації

Мета роботи: отримати знання основних метрик якості бінарної класифікації і варіантів тонкого налаштування алгоритмів класифікації.

Короткі теоретичні відомості

Приклади алгоритмів класифікації

Для розрахунку метрик якості в задачі машинного навчання зі вчителем необхідні тільки дві величини: вектори дійсних і передбачених значень. Дійсні значення – це мітки класів у тренувальній і тестовій вибірці; алгоритм класифікації повертає передбачені.

Розглянемо кілька моделей прикладів векторів, які отримаємо на виході.

Нехай, в нашій задачі дійсні значення складають вектор з нулів і одиниць, а передбачені вектори лежать в інтервалі [0, 1] (де число означає ймовірність віднести приклад до класу "1"). Такі пари векторів будемо візуалізувати так, як представлено на рисунку 1:

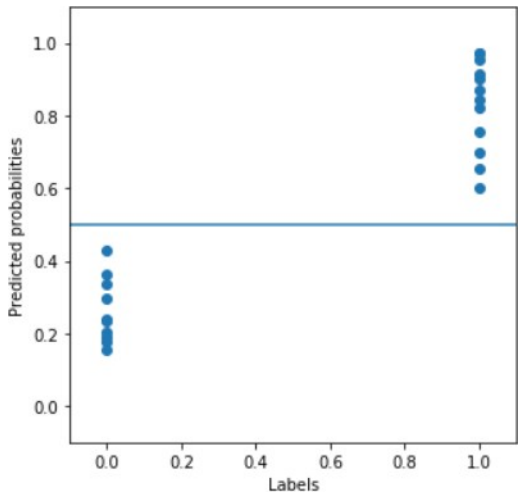


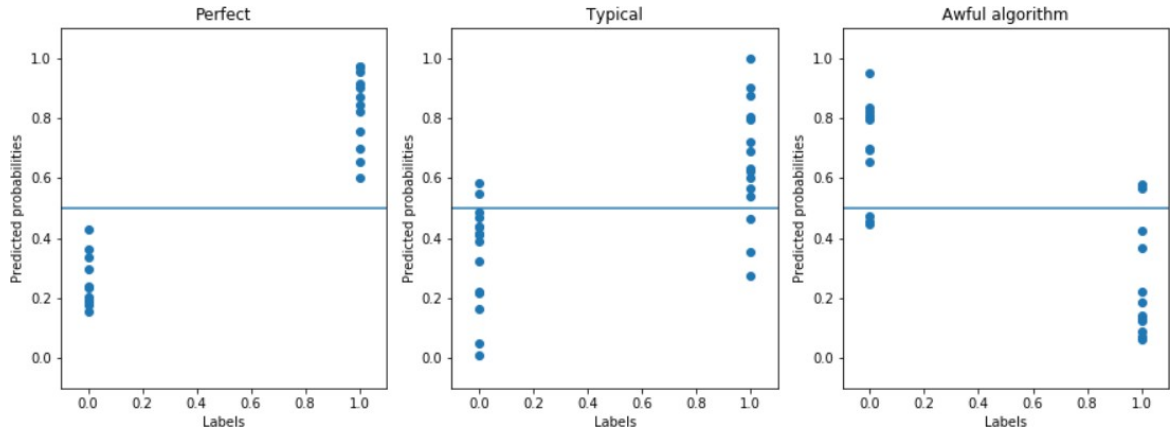
Рисунок 1 – Приклад дійсних (по осі абсцис) і передбачених (по осі ординат) значень

Щоб зробити фінальне передбачення (віднести приклад до класу "0" або "1"), потрібно встановити поріг T (горизонтальна лінія на рис. 1): всім об'єктам з передбаченим значенням вище T буде присвоєно клас "1", іншим – "0".

Найкраща ситуація: поріг T абсолютно вірно розділяє передбачені ймовірності за двома класами. Для прикладу з рис. 1 ідеальні інтервали ймовірностей можна отримати шляхом вибору порогу $T = 0,5$.

Найчастіше ймовірнісні інтервали накладаються один на одного (рисунок 2б) – тоді доводиться підбирати поріг уважно.

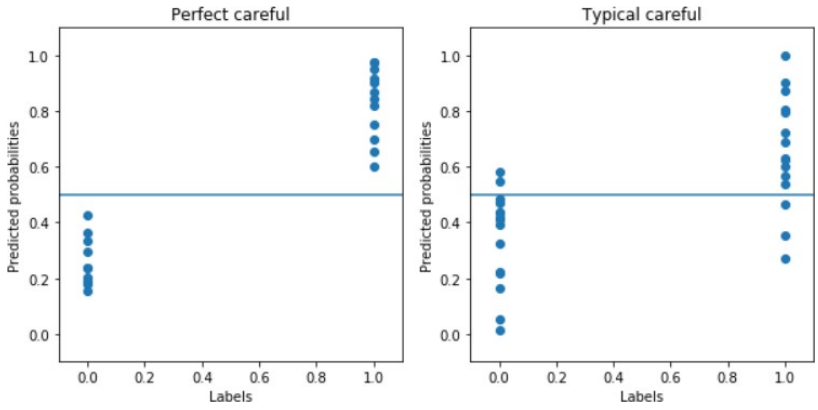
Невірно навчений алгоритм робить зворотнє: він ставить ймовірності об'єктів класу "0" вище ймовірностей прикладів класу "1" (рисунок 2в). У такій ситуації слід перевірити, чи не були переплутані мітки "0" і "1" при отриманні тренувальної вибірки.



а б в

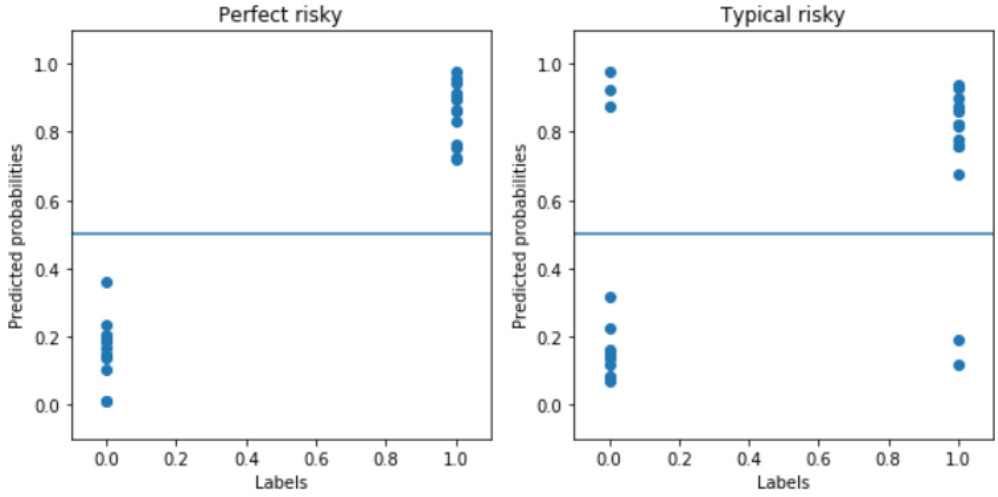
Рисунок 2 – Приклади дійсного і передбаченого векторів для ідеального (а), типового (б) і невірно навченого (в) алгоритмів

Алгоритми можуть бути обережними і видавати значення, не надто віддалені від $0,5$ (рис. 3), або можуть брати на себе ризик по отриманню значень, близьких до нуля і одиниці (рис. 4).



а б

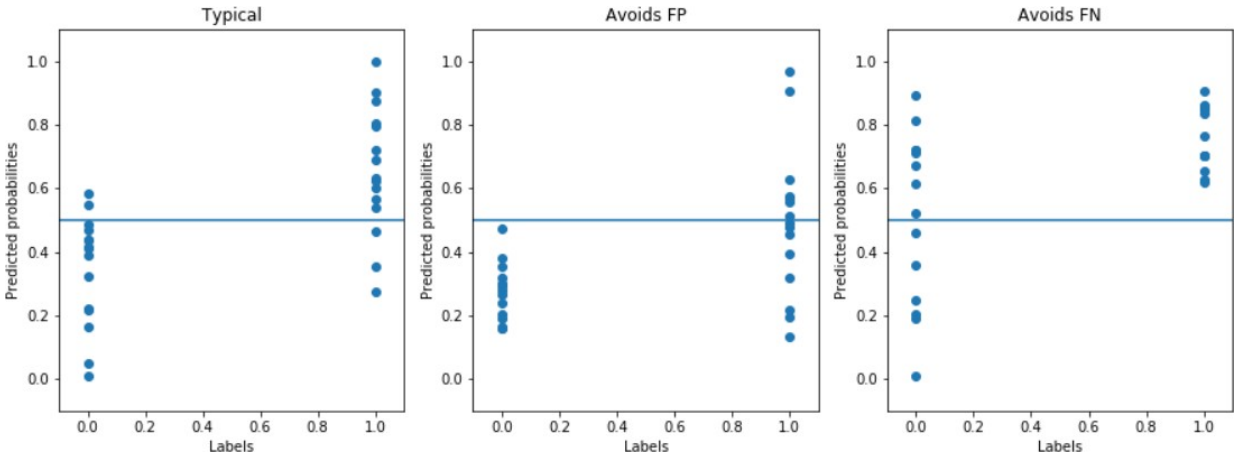
Рисунок 3 – Приклади дійсного і передбаченого векторів для ідеального (а) і типового (б) обережного алгоритму



а б

Рисунок 4 – Приклади дійсного і передбаченого векторів для ідеального (а) і типового (б) алгоритму ризику (ризикуючий алгоритм)

Інтервали ймовірностей можуть бути зміщені. Наприклад, якщо небажані помилки I роду (false-positive), то алгоритм буде видавати значення в середньому ближче до нуля. Аналогічно, для уникнення помилок II роду (false-negative) частіше необхідно отримувати на виході моделі ймовірності вище 0,5. На рисунку 5 представлені приклади векторів в даних ситуаціях.



а б в

Рисунок 5 – Приклади дійсного і передбаченого векторів для типового (а), що уникає помилок I роду (б) і уникає помилок II роду (в) алгоритмів

Precision ma recall. Accuracy

Дані метрики розраховуються після бінаризації передбачених значень порогом *T*. На рисунку 6 представлені типи об'єктів в залежності від пар дійсних і передбачених значень.

	Передбачений клас		
Реальний клас		1	0
	1	True Positive	False Negative
	0	False Positive	True Negative

Рисунок 6 – Типи об'єктів: True, False – вірно і невірно передбачений клас об'єкта відповідно; Positive, Negative – передбачений клас об'єкта "1" ("Yes") і "0" ("No") відповідно

Найбільш проста і відома метрика – accuracy. Вона показує частку вірно передбачених прикладів:

$$Accuracy = \frac{Tp + Tn}{Tp + Tn + Fp + Fn}$$

де

$Tp - True\ Positive,$

$Tn - True\ Negative,$

$Fp - False\ Positive,$

$Fn - False\ Negative.$

Precision і recall також широко поширені. Перша метрика показує наскільки вірно передбачені об'єкти, яким була виставлена "1", а друга – точність передбачення прикладів, в дійсності відносяться до класу "1" (рисунок 7).

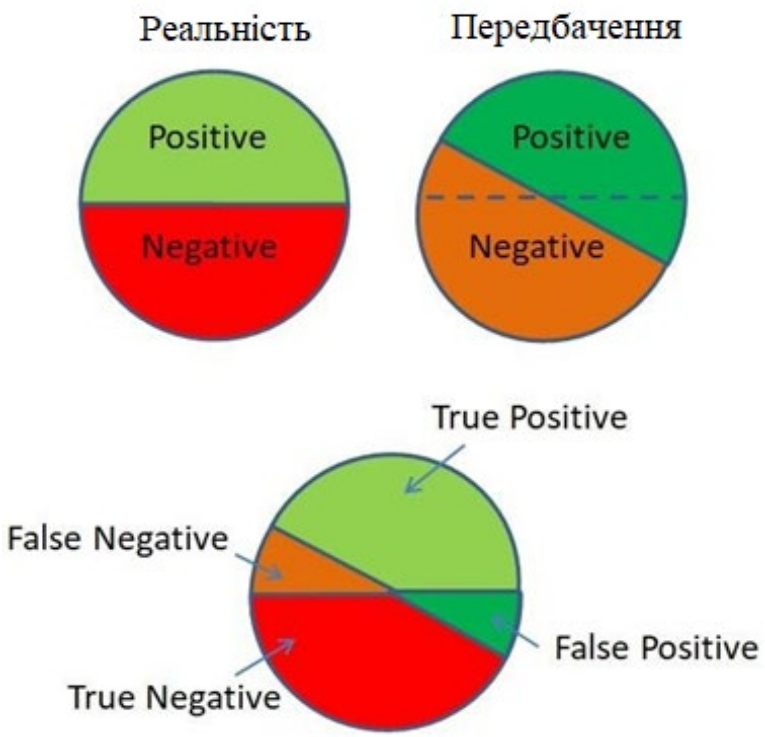


Рисунок 7 – Візуалізація типів помилок бінарної класифікації

Розрахунок даних метрик здійснюється за такими формулами:

$$Precision = \frac{Tp}{Tp + Fp}$$

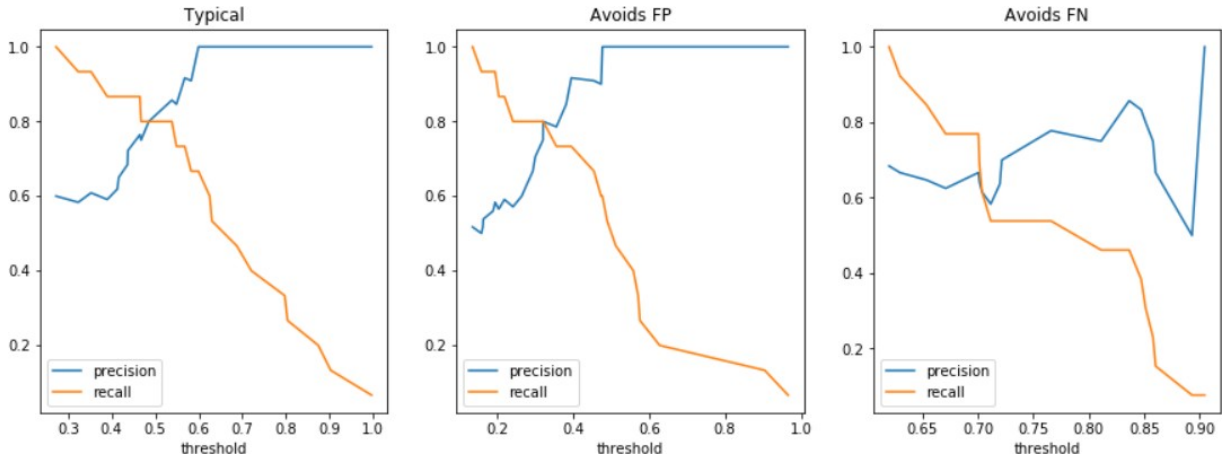
$$Recall = \frac{Tp}{Tp + Fn}$$

За допомогою всіх трьох метрик можна з легкістю визначати випадки добре і погано навчених алгоритмів класифікації. До того ж, так як можливі значення лежать в інтервалі [0, 1], то їх легко інтерпретувати.

З даних метрик нічого не можна дізнатися про самі значення ймовірностей об'єктів; можна визначити тільки, яка їх частка лежить не по той бік від порогу T.

Метрика accuracy в рівній мірі штрафує алгоритм за наявність помилок I і II родів. У той же час використання пари precision і recall дозволяє чітко встановлювати співвідношення між родами помилок: дані метрики використовуються для контролю Fp и Fn помилок відповідно.

На рисунку 8 представлені метрики precision і recall в залежності від порогу T для пар векторів з рисунка 5.



а б в

Рисунок 8 – Precision і recall при різних значеннях порога T для типового (а), уникнення помилок I роду (б) і уникнення помилок II роду (в) алгоритмів

У міру зростання значення порога T ми отримуємо менше Fp і більше Fn помилок, згідно з тим фактом, що одна з кривих піднімається, а друга падає вниз. Виходячи з даної закономірності, можна вибрати оптимальний поріг, при якому метрики precision і recall знаходяться в прийнятному інтервалі значень. Якщо такий поріг не був знайдений, слід знайти інший алгоритм для навчання.

Необхідно зазначити, що прийнятні значення precision і recall визначаються прикладним характером задачі. Наприклад, у разі вирішення проблеми, чи є у пацієнта

розглянута хвороба ("0" - здоровий, "1" – хворий), Fn помилки вкрай небажані, звідси значення метрики recall виставляється ≈ 0,9. Ми можемо сказати пацієнтові, що він хворий, і подальшою діагностикою виявити помилку, що набагато краще в порівнянні з ігноруванням реально наявної хвороби.

F1-score

Очевидний недолік пари precision-recall в тому, що ми розраховуємо дві метрики: при порівнянні алгоритмів неясно, як використовувати обидві відразу. Рішенням даної проблеми є метрика F1-score:

$$\text{F1-score} = 2\frac{precision * recall}{precision + recall}$$

F1 метрика буде дорівнює одиниці тільки тоді, коли $precision = 1$ і $recall = 1$ (тобто в ідеальному алгоритмі).

Досить складно ввести в оману за допомогою F1-score: якщо одна зі складових близька до 1, а друга показує низькі значення (а таку ситуацію легко отримати згідно рис. 8), то F1 метрика буде далека від ідеального значення. Дану метрику важко оптимізувати, так як для цього потрібна як висока точність, так і збалансованість між пологами помилок.

Для пар векторів, представлених на рисунках 5а, 5б, 5в, значення F1-score при T = 0,5, склало 0,828, 0,636 і 0,765 відповідно. Значення метрики для другого і третього прикладів, де одне значення з пари precision-recall дорівнювало одиниці, виявилися менше в порівнянні з першим збалансованим випадком.

Описані метрики легко інтерпретувати, але при цьому ми не беремо до уваги більшу частину інформації, яку одержуємо з виходу моделі. У деяких завданнях потрібні ймовірності в чистому вигляді (тобто без їх бінаризації). Наприклад, при виставленні ставки на виграш футбольної команди потрібно знати не клас, до якого буде віднесений об'єкт, а ймовірність його віднесення. Перед бінаризацією передбачень може бути корисно розглянути вектор ймовірностей на присутність будь-яких закономірностей.

Логістична функція помилок (log_loss)

Метрика визначає середню розбіжність між ймовірностями віднесення об'єктів до конкретних класів та їх дійсними класами:

$$log\_loss = -\frac{1}{n}\sum_{i=1}^{n}[actual_i * \log(predicted_i) + (1 - actual_i) * \log(1 - predicted_i)],$$

де $actual_i$ – дійсний клас i-го об’єкту, $predicted_i$ – ймовірність відношення і-го об’єкту до класу “1”, n – кількість об’єктів. Функцію необхідно мінімізувати.

Далі можна бачити значення функції помилок для раніше використаних пар векторів.

Алгоритми, різні за якістю (рис. 2):

Ідеальний – 0,249

Типовий – 0,465

Невірно навчений – 1,527

Обережний і ризикуючий алгоритми (рис. 3 і 4):

Ідеальний обережний – 0,249

Ідеальний ризикуючий – 0,171

Типовий обережний – 0,465

Типовий ризикуючий – 0,614

Різні схильності алгоритмів до Fp і Fn помилок (рис. 5):

Уникаючий Fp помилок – 0,585

Уникаючий Fn помилок – 0,589

Як і попередні метрики, *log_loss* може розрізняти добре і погано навчені алгоритми, але при цьому значення метрики важко інтерпретувати: вона не може досягти нуля і не має обмеження зверху. Таким чином, навіть для абсолютно точного алгоритму, якщо поглянути на його значення логістичної функції помилок, неможливо буде сказати, що він ідеальний.

З іншого боку, метрика робить відмінності між обережним і ризикуючим алгоритмами. Як видно з рис. 3б і 4б, число помилкових прикладів при порозі

бінаризації $T = 0,5$ для типової обережної і ризикує моделей приблизно рівні, а в разі ідеальних алгоритмів (рис. 3а, 4а) помилок немає зовсім. Однак ризикуючий алгоритм вносить більше ваги в метрику *log_loss* за невірно підібрані передбачення в порівнянні з обережним, якщо модель є типовою, і менше, якщо модель абсолютно точна.

Таким чином, *log_loss* чутлива до ймовірностей, близьких як до 0 і 1, так і до 0,5.

Fp і Fn помилки метрика виявити не може, але нескладно перейти до більш узагальненої версії функції помилок, де можна підняти штраф для того чи іншого роду помилок. Для цього додамо комбінацію невід'ємних і даючих в сумі одиницю коефіцієнтів при ймовірнісних членах функції. Наприклад, якщо потрібно більше штрафувати Fp помилки:

$$weighted\_log\_loss(actual, predicted) =$$

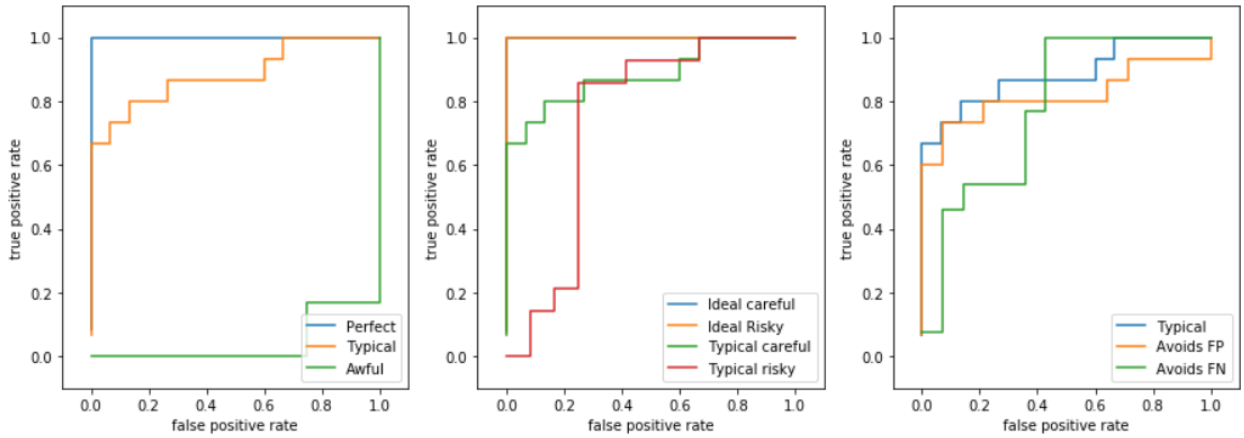
$$= -\frac{1}{n}\sum_{i=1}^{n}[0,3 * actual_i * \log(predicted_i) + 0,7 * (1 - actual_i) * \log(1 - predicted_i)].$$

Якщо алгоритм видає високе значення ймовірності, а об'єкт насправді належить до класу "0", то перший член функції буде дорівнює нулю, а другий буде розрахований з урахуванням його більшої ваги.

ROC та AUC

При побудові ROC-кривої (Receiver Operating Characteristic) піддається варіюванню поріг бінаризації і розраховуються деякі величини, що залежать від кількості *Fp* і *Fn* помилок. Ці параметри підбираються таким чином, що в разі наявності порога для ідеального поділу класів ROC-крива буде проходити через певну точку – лівий верхній кут квадрата [0, 1] x [0, 1]. На додаток до цього, крива завжди починається в лівому нижньому і закінчується в правому верхньому куті. Для того, щоб охарактеризувати криву чисельно, використовується метрика AUC (Area Under the Curve) – площа під ROC-кривою.

Далі візуалізовані ROC-криві для раніше використаних пар дійсних і передбачених векторів (рисунок 9).



а б в

Рисунок 9 – ROC-криві для пар векторів, які представлені на рисунках 2 (а), 3 и 4 (б), 5 (в)

Значения AUC, соответствующие кривым на рис. 9, представлены ниже.

Алгоритмы, разные по качеству (рис. 2):

Ідеальний – 1,000

Типовий – 0,884

Невірно навчений – 0,042

Обережний і ризикуючий алгоритми (рис. 3 и 4):

Ідеальний обережний – 1,000

Ідеальний ризикуючий – 1,000

Типовий обережний – 0,884

Типовий ризикуючий – 0,738

Різні схильності алгоритмів до Fp і Fn помилок (рис. 5):

Уникаючий Fp помилок – 0,819

Уникаючий Fn помилок – 0,780

Чим більше об'єктів у вибірці, тим більш гладкими будуть криві (хоча при наближенні буде видно, що вони все ще ступінчасті).

Як і очікувалося, криві ідеальних алгоритмів проходять через верхній лівий кут. Також на рис. 9а жовтою лінією позначена типова ROC-крива (в більшості випадків крива не може досягти даного кута).

Метрика AUC у ризикуючої моделі значно нижча в порівнянні з обережним алгоритмом, хоча в разі абсолютно точних моделей різниці між їх ROC-кривими і AUC метриками немає (рис. 9б). Таким чином, для ідеального алгоритму немає сенсу віддаляти один від одного точки, що відносяться до різних передбаченим класам.

При наявності більшого числа Fp або Fn помилок на кривих буде спостерігатися зміщення (рис. 9в). Але за значенням AUC його неможливо визначити (в окремому випадку криві можуть бути симетричні щодо діагоналі (0, 1) – (1, 0)).

Після побудови кривої зручно вибирати поріг бінаризації, що задовольняє обмеженням по частках помилок I або II роду. Певне значення порогу відповідає одній точці на ROC-кривій. Якщо ми хочемо уникнути Fp помилок, то слід вибрати точку якомога ближче до лівої сторони квадрата, якщо небажані Fn помилки – ближче до верхньої сторони. Всі проміжні точки відповідають за деякі більш збалансовані співвідношення між родами помилок.

Частина з розглянутих метрик якості класифікації (наприклад, log_loss) може бути перенесена на завдання, де об’єкти відносяться до більш ніж 2 класів. У разі, якщо важко узагальнити формулу метрики на кілька класів, таке завдання представляють у вигляді набору підзадач бінарної класифікації, а узагальнену метрику отримують шляхом деякого усереднення (micro - або macro-середнє) метрик підзадач.

На практиці завжди корисно візуалізувати вектори передбачених алгоритмом значень з метою зрозуміти, які помилки робить модель при різних порогах і як використовувана метрика реагує на них.

Завдання

1) Завантажте файл bioresponse.csv з папки “Лаб 1” за посиланням https://drive.google.com/drive/folders/1SYLvcaJ5XZQL2socPuGc6466AoEsrLue?usp=sharing
2) Навчіть 4 класифікатора, щоб передбачити поле "Activity" (біологічна відповідь молекули) з набору даних "bioresponse.csv":

   - дрібне дерево рішень;

   - глибоке дерево рішень;

   - випадковий ліс на дрібних деревах;

   - випадковий ліс на глибоких деревах.
3) Розрахуйте наступні метрики, щоб перевірити якість ваших моделей:

   - частка правильних відповідей (accuracy);

   - точність;

   - повнота;

   - $F1\text{-}score$;

   - *log-loss*.
4) Побудуйте precision-recall і ROC-криві для ваших моделей.
5) Навчіть класифікатор, який уникає помилок II роду і розрахуйте для нього метрики якості.

Комп’ютерний практикум № 2

Функції помилок (втрат) у машинному навчанні

Мета роботи: отримати знання і критерії застосування основних використовуваних у сучасному машинному навчанні функцій помилок (функцій втрат).

Короткі теоретичні відомості

Функція втрат - функціонал, що оцінює величину розбіжності між істинним значенням параметра, що оцінюється, і модельною оцінкою цього параметра.

Що таке функції втрат? (Приклад) Припустимо, ви знаходитесь на вершині гори і вам потрібно спуститися вниз. Як ви вирішите, куди йти? Потрібно: озирнутися навколо, щоб побачити всі можливі шляхи; відкинути ті, що ведуть вгору (тому, що ці шляхи насправді коштуватимуть вам більше енергії і зроблять ваше завдання ще складнішим); нарешті, піти тим шляхом, який, на вашу думку, має найбільший нахил вниз. Ця інтуїція, з якою ви оцінювали свої рішення - це саме те, що забезпечує функція втрат. Функція втрат зіставляє рішення з пов’язаними з ними витратами. Рішення піднятися вгору по схилу буде коштувати нам енергії і часу. Рішення спуститися вниз принесе нам вигоду. Тому воно має негативну вартість.

В алгоритмах керованого машинного навчання ми хочемо мінімізувати помилку для кожного навчального прикладу в процесі навчання. Це робиться за допомогою деяких стратегій оптимізації, таких як градієнтний спуск. І ця помилка походить від функції втрат.

Задача регресії

Задача регресії - прогноз на основі вибірки об'єктів з різними ознаками. На виході має вийти чисельне значення (2, 35, 76.454), наприклад ціна квартири, вартість цінного паперу через півроку, очікуваний дохід магазину на наступний місяць, якість вина під час сліпого тестування.

Вибираючи функцію втрат для задач регресії, слід вирішити, яку саме властивість умовного розподілу хочемо відновити. Найчастіші варіанти:

$$L(y, f) = (y - f)^2$$

– Gaussian loss (L_2), Це класична умовна середня, найчастіший і найпростіший варіант. Якщо немає жодної додаткової інформації або вимог до стійкості моделі - використовуйте його.

$$L(y, f) = |y - f|$$

– Laplacian loss (L_1), Визначає умовну медіану. Медіана, як ми знаємо, більш стійка до викидів, тому в деяких завданнях ця функція втрат є кращою, оскільки вона не так сильно штрафує великі відхилення, ніж квадратична функція.

$$L(y, f) = \begin{cases} (1-\alpha)|y - f| & \text{при } y - f \leq 0 \\ \alpha|y - f| & \text{при } y - f > 0 \end{cases}$$

– Quantile loss (L_q), Якби ми, припустимо, захотіли не умовну медіану з L_1, а умовну 75% - квантиль, ми б скористалися цим варіантом з $\alpha = 0{,}75$. Можна помітити, що ця функція асиметрична і більше штрафує спостереження, які опиняються на потрібній нам стороні квантилі.

Графіки перерахованих функцій наведені на рисунку 1.

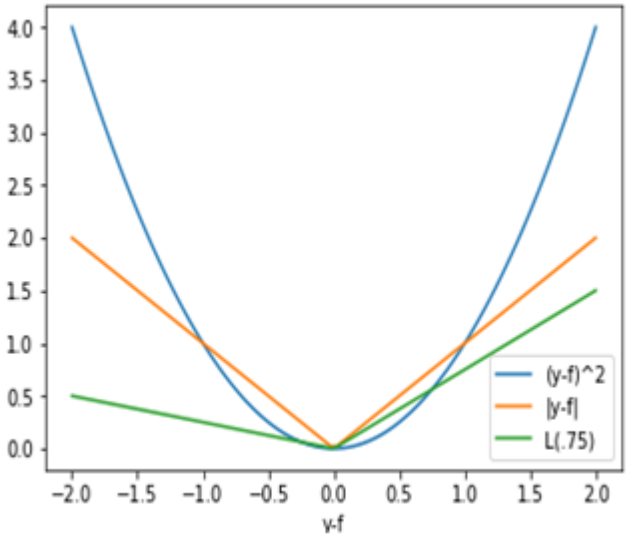


Рис. 1 – Функції втрат для задач регресії

*приклад

Давайте спробуємо скористатися L_q функцією втрат на наших «гральних» даних, намагаючись відновити умовну 75%-квантиль косинуса. Зберемо все разом:

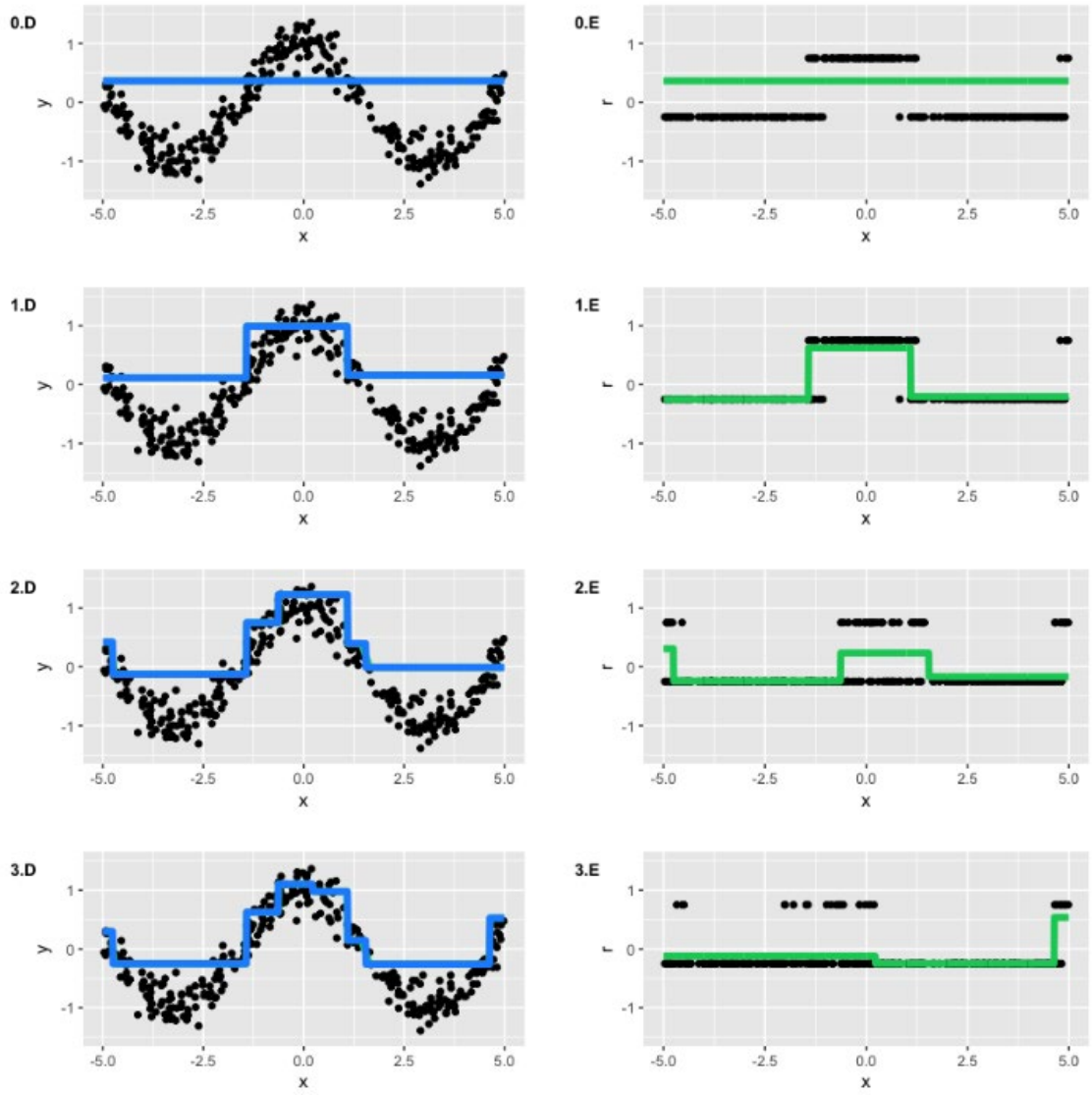
- Гральні дані $\{(x_i, y_i)\}_{i=1,\ldots,300}$

- Число ітерацій M = 3
- Квантильна функція втрат $L_{0.75}(y, f) = \begin{cases} 0.25 \cdot |y - f|, & \text{if } y - f \leq 0 \\ 0.75 \cdot |y - f|, & \text{if } y - f > 0 \end{cases}$
- Градієнт $L_{0.75}$ (y, f) - індикаторна функція, зважена нашим α = 0.75$. Ми будемо навчати дерева чомусь, дуже схожому на класифікатор:

$$r_i = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}(x)} =$$
$$= \alpha I(y_i > \hat{f}(x_i)) - (1-\alpha) I(y_i \leq \hat{f}(x_i)), \quad \text{for } i = 1, \ldots, 300$$

- Дерева рішень як базові алгоритми h(x);
- Гіперпараметри дерев рішень: глибина дерев дорівнює 2;

У нас є очевидне початкове наближення - просто взяти потрібну нам квантиль y. Однак, про оптимальні коефіцієнти ρ_t нам нічого не відомо, тож скористаємося стандартним line search. Подивимося, що в нас вийшло:



Незвично бачити, що за фактом ми навчаємо щось дуже несхоже на звичайні залишки - на кожній ітерації r_i приймають тільки два можливих значення. Однак, результат роботи Gradient Boosting Machine (GBM) досить схожий на нашу вихідну функцію.

Якщо залишити алгоритм навчатися на цьому іграшковому прикладі, ми отримаємо майже такий самий результат, що і з квадратичною функцією втрат, зміщений на ≈ 0.135. Але якби ми шукали квантилі вище 90%, могли б виникнути обчислювальні труднощі. А саме, якщо співвідношення числа точок вище потрібної квантилі буде занадто малим (як незбалансовані класи), модель не зможе якісно навчитися. Про такі нюанси варто замислюватися, розв'язуючи нетипові задачі.

Задача класифікації

Задача класифікації - отримання категоріальної відповіді на основі набору ознак. Має кінцеву кількість відповідей (як правило, у форматі «так» або «ні»): чи є на фотографії кіт, чи є зображення людським обличчям, чи хворий пацієнт на рак.

При класифікації через принципово іншу природу розподілу цільової змінної оптимізуються не самі мітки класів, а їх log-правдоподібність. Найбільш відомі варіанти таких класифікаційних функцій втрат зображені на рисунку 2.

Математичні вирази для розрахунку цих функцій наступні:

$$L(y, f) = \ln(1 + \exp(-yf))$$

– Logistic loss (Bernoulli loss). Штрафуються навіть коректно передбачені мітки класів, але, оптимізуючи цю функцію втрат, можна покращувати класифікатор, навіть якщо всі спостереження передбачені вірно. Це найбільш стандартна і часто використовувана функція втрат у бінарній класифікації.

$$L(y, f) = \exp(-yf)$$

– Adaboost loss; має більш жорсткий експоненціальний штраф на помилки класифікації і використовується рідше.

Для використання в задачах класифікації використовується також функція крос-ентропії, яка визначає міру розбіжності між двома ймовірнісними розподілами. Якщо крос-ентропія велика, різниця між двома розподілами велика, а якщо крос-ентропія мала, розподіли схожі один на одного:

$$H(P, Q) = -\sum P(x) \cdot \log_2 Q(x),$$

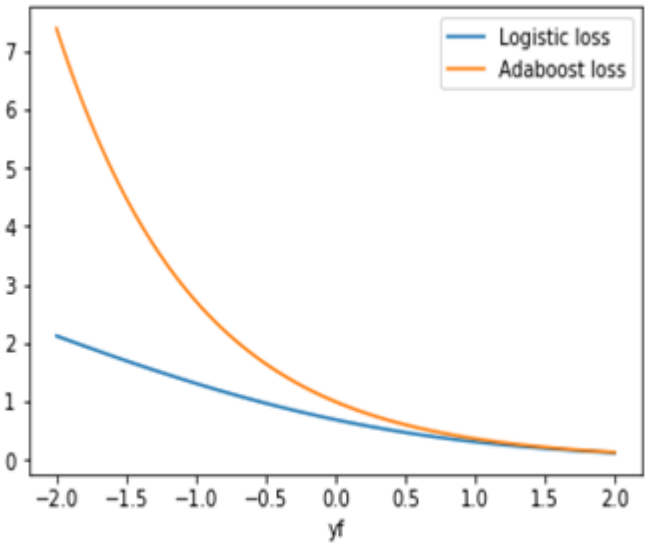


Рис. 2 – Функції Logistic loss і Adaboost loss

де P – розподіл істинних відповідей, А Q-розподіл ймовірностей прогнозів моделі.

У разі бінарної класифікації формула для розрахунку крос-ентропії має наступний вигляд:

$$L = -\mathrm{y} \cdot \log_2 p + (1 - y) \cdot \log_2 (1 - p),$$

де y – двійковий індикатор (0 або 1) того, чи є мітка класу правильною класифікацією для поточного спостереження; p – прогнозована ймовірність класу, визначена моделлю класифікації.

При бінарній класифікації кожна передбачена ймовірність порівнюється з фактичним значенням класу (0 або 1) та обчислюється оцінка, яка штрафує ймовірність пропорційно до величини відхилення від очікуваного значення. Конфігурація вихідного рівня моделі являє собою один вузол із сигмоїдальною функцією активації:

$$f(p_i) = \frac{1}{1 + \exp(-p_i)}$$

Щоб класифікувати об'єкт як один із кількох класів, задача формулюється як передбачення ймовірності того, що приклад належить кожному класу. У разі коли класів багато ($M > 2$) береться сума значень логарифмічних функцій втрат для кожного прогнозу класів, що спостерігаються – «categorical cross-entropy»:

$$CE = -\sum_{c=1}^{M} y_{o,c} \cdot \log_2 p_{o,c}.$$

Для розрахунку зваженого вектора ймовірностей віднесення об'єкта до конкретних класів використовується функція активації «softmax» - узагальнення логістичної функції багатовимірного випадку.

$$f(p_i) = \frac{\exp(p_i)}{\sum_{j=1}^{M} \exp(p_j)}.$$

Коли кожен з об'єктів повинен класифікуватися однозначно, що найчастіше і потрібно, можна відкинути доданки, які є нульовими через цільові індикатори y. Тоді:

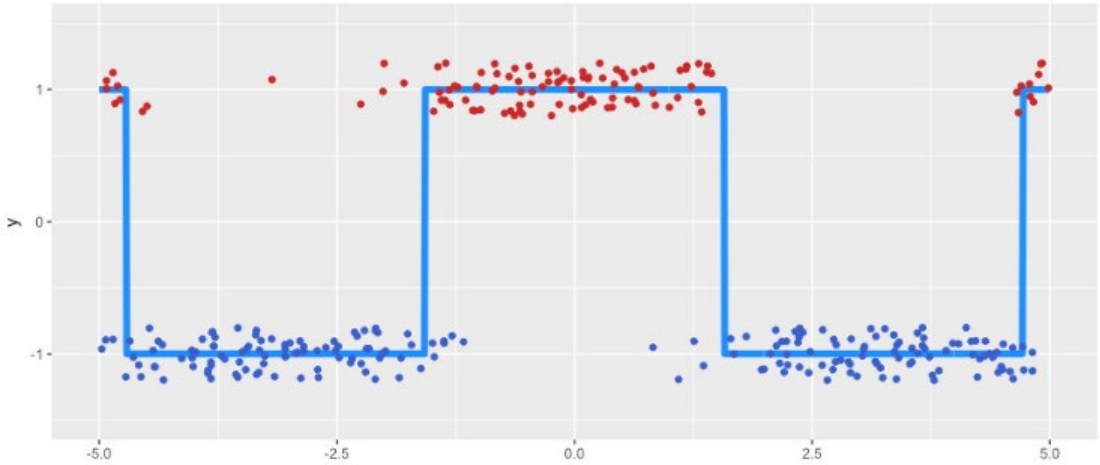
$$CE = -\log_2 \frac{\exp(p_i)}{\sum_{j=1}^{M} \exp(p_j)},$$

де p_i – результат оцінки належності до відповідного класу.

*приклад

Через принципово іншу природу розподілу цільової змінної, будемо передбачати й оптимізувати не самі мітки класів, а їхню log-правдоподібність. Для цього переформулюємо функції втрат над перемноженими передбаченнями та істинними мітками y * f . (Дивимось на рис.2).

Згенеруємо нові «гральні» дані для завдання класифікації. За основу візьмемо наш зашумлений косинус, а як класи цільової змінної використовуватимемо функцію sign. Нові дані мають такий вигляд (jitter-шум додано для наочності):



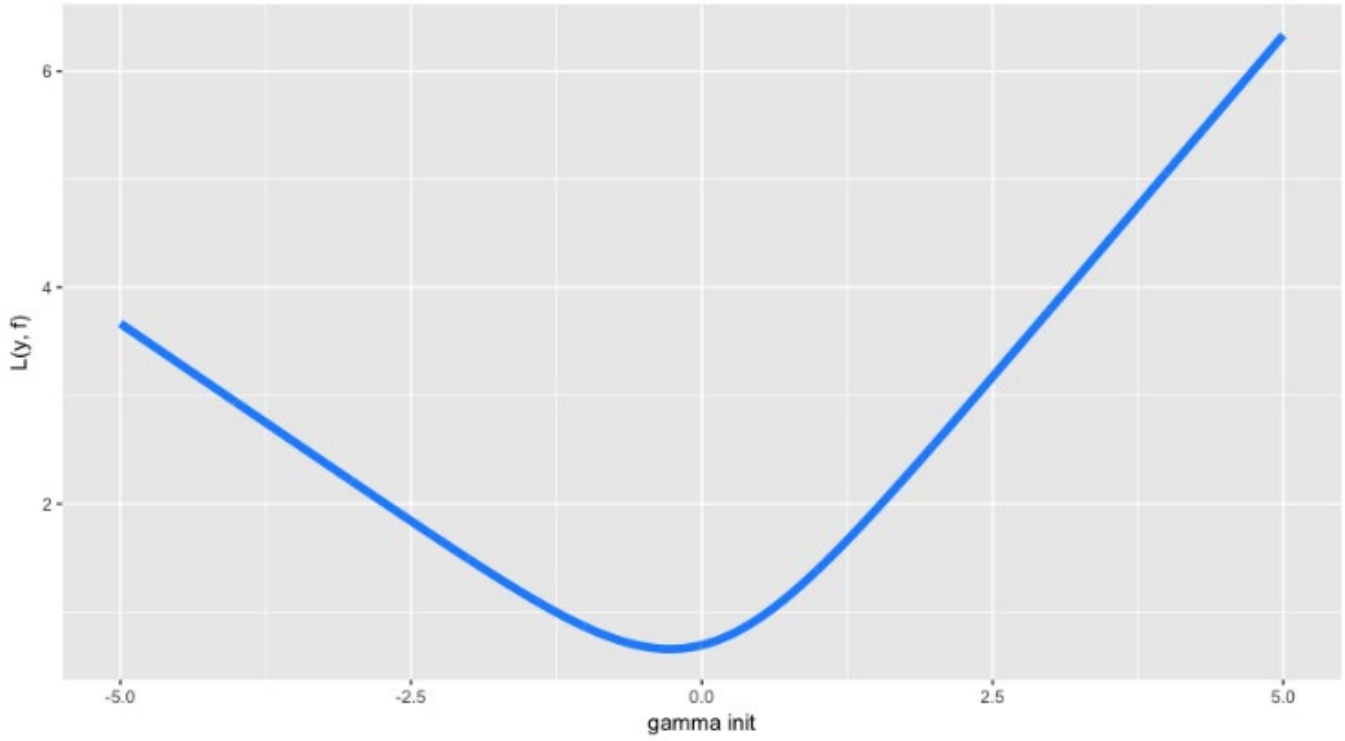
Скористаємося Logistic loss, щоб подивитися, що ж ми насправді бустимо. Як і раніше, зберемо разом те, що будемо вирішувати:

- Гральні дані $\{(x_i, y_i)\}_{i=1,\ldots,300}, y_i \in \{-1, 1\}$
- Число ітерацій M = 3;
- Як функція втрат - Logistic loss, її градієнт рахується так:

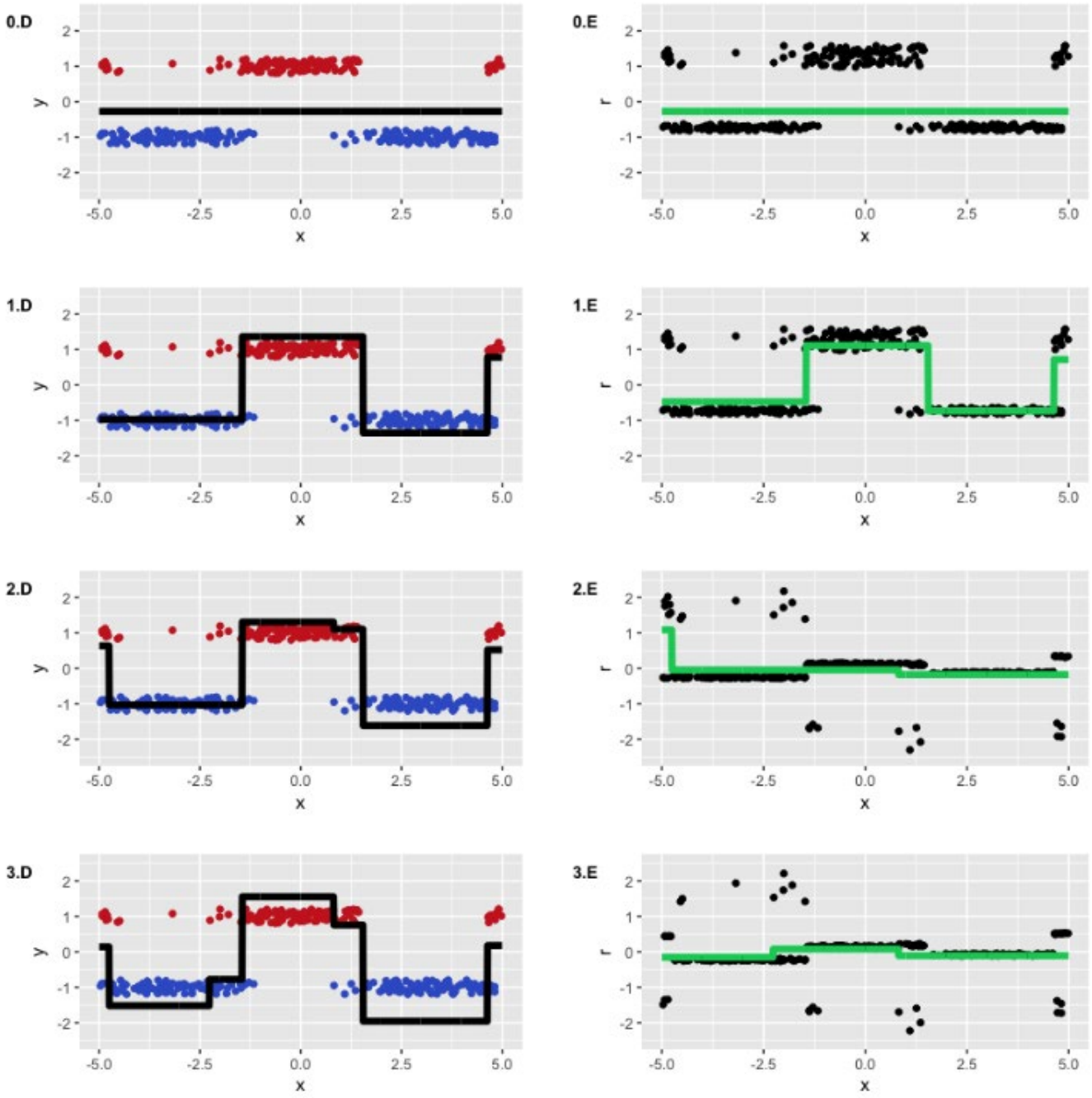
$$r_i = \frac{2 \cdot y_i}{1 + exp(2 \cdot y_i \cdot \hat{f}(x_i))}, \quad \text{for } i = 1, \ldots, 300$$

- Дерева рішень як базові алгоритми h(x);
- Гіперпараметри дерев рішень: глибина дерев дорівнює 2 ✓;

Цього разу з ініціалізацією алгоритму все трохи складніше. По-перше, наші класи незбалансовані і розділені в пропорції приблизно 63% на 37%. По-друге, аналітичної формули для ініціалізації для нашої функції втрат невідомо. Тож будемо шукати $\hat{f}_0 = \gamma$ пошуком:



Оптимальне початкове наближення знайшлося в районі -0.273. Можна було здогадатися, що воно буде негативним (нам вигідніше передбачати всіх найпопулярнішим класом), але формули точного значення, як ми вже сказали, немає. Запустимо GBM і подивимося, що ж насправді відбувається:



Алгоритм спрацював успішно, відновивши поділ наших класів. Можна побачити, як відокремлюються «нижні» області, в яких дерева більше впевнені в коректному передбаченні негативного класу, і як формуються дві сходинки, де класи були перемішані. На псевдо-залишках видно, що в нас є досить багато коректно класифікованих спостережень, і якась кількість спостережень із великими помилками, які з’явилися через шум у даних. Якось виглядає те, що насправді передбачає GBM у завданні класифікації (регресія на псевдо-залишках логістичної функції втрат).

Завдання

6) Завантажте дані за посиланням https://archive.ics.uci.edu/ml/datasets/banknote+authentication

7) Реалізувати модель логістичної регресії з наступними функціями втрат:

а) Logistic loss

б) Adaboost loss

в) binary crossentropy

8) Візуалізувати криві навчання моделі бінарної класифікації у вигляді динаміки зміни кожної з функцій помилок п.2 на тренувальній та тестовій вибірках.

9) Порівняти якість класифікації за метрикою accuracy у кожному з трьох модифікацій алгоритму.

Комп’ютерний практикум № 3

Вступ до опрацювання природної мови

Мета роботи: здобуття студентом навичок реалізації базових методів обробки природної мови, включно з попереднім опрацюванням тексту, формуванням «мішка слів» («bag-of-words»), виділенням стоп-слів і найважливіших слів у документі, створенням тематичних моделей.

Короткі теоретичні відомості

Завдання обробки природної мови — одне з найбільш затребуваних у сучасному машинному навчанні. Вони охоплюють такі галузі, як машинний переклад, анотування текстів, класифікація документів, генерація тексту, text-to-image і text-to-video задачі, побудова чат-ботів і діалогових систем та багато інших.

Розв’язання будь-якої задачі у сфері NLP (natural language processing) починається з попереднього опрацювання тексту, яке зазвичай містить у собі:

- Видалення всіх символів, що не належать до природної мови, з тексту: @, #, & тощо. (якщо це не суперечить меті дослідження);
- Токенізиція тексту, тобто, поділ його на окремі слова: слово = «токен»;
- Видалення стоп-слів, що трапляються в усіх без винятку текстах, які не несуть смислового навантаження для розв'язання поточного завдання;
- Приведення тексту до нижнього регістру, щоб модель розпізнавала такі слова, як, наприклад, «привіт» і «Привіт», однаково;
- Стеммінг (обрізка слова до його основи) і лематизація тексту (приведення слів до єдиної форми, наприклад, («гарний», «гарна», «гарне»)).

Щоб використовувати методи машинного навчання на текстових документах, потрібно перевести текстовий вміст (слова природною мовою) у числовий вектор ознак. Найбільш інтуїтивно зрозумілий спосіб зробити описане вище перетворення - це представити текст у вигляді набору слів, після чого приписати кожному слову в тексті унікальний цілочисельний індекс, який відповідає частоті його появи в документах навчальної вибірки. Для цього в кожному документі і слід порахувати

кількість уживань кожного слова w і зберегти це число в комірці X[i, j]. Це відповідатиме значенню ознаки j, де j - це індекс слова w у словнику.

Таке перетворення тексту на матрицю частот уживань слів має назву «мішку слів» (або «bag of words»). Воно передбачає, що ймовірність появи слова в різних документах однакова, і будує матрицю об’єкти-ознаки виходячи з припущення, що кількість ознак відповідає кількості унікальних слів у корпусі документів. «Мішок слів» найчастіше є високорозмірним розрідженим набором даних (з великою кількістю нулів).

D1 - “I am feeling very happy today”
D2 - “I am not well today”
D3 - “I wish I could go to play”

	I	am	feeling	very	happy	today	not	well	wish	could	go	to	play
D1	1	1	1	1	1	1	0	0	0	0	0	0	0
D2	1	1	0	0	0	1	1	1	0	0	0	0	0
D3	2	0	0	0	0	0	0	0	1	1	1	1	1

Рисунок 1. Приклад побудови матриці частот згадок унікальних слів для корпусу документів із трьох речень (джерело зображення: deeplearning.ai by Andrew Ng)

Однак, навіть якщо документи присвячені одній темі, у відносно довгих документах середня кількість слововживань буде вищою, ніж у коротких, що може призводити до некоректного налаштування моделей машинного навчання. Щоб уникнути потенційних невідповідностей, застосовують підхід, який називають TF-IDF – «term frequency - inverse document frequency», який дає змогу оцінити «важливість» слова для цього документа. Цей підхід дає змогу знизити вплив низькоінформативних слів і, навпаки, підвищити "вагу" важливих з погляду оцінки приналежності документа до певної тематики. Для кожного слова обчислюється така величина:

$$tf - idf\ (t, d, D) = tf(t, d) \times idf(t, D),$$

$$tf(t, d) = \frac{n_t}{\sum_k n_k},$$

$$idf(t, D) = log\ \frac{|D|}{|\{d_i \in D | t \in d_i\}|},$$

де - частота слова (токена) t у документі d, | D | - кількість документів у колекції; - кількість документів у колекції, в яких зустрічається слово t.

Велике значення TF-IDF отримуватимуть слова з високою частотою всередині конкретного документа і з низькою частотою використання в інших документах. Заповнюючи матрицю об’єкти-ознаки значеннями TF-IDF, можна ефективніше виокремлювати важливі «ознаки» (слова) і проводити якісніше налаштування моделей машинного навчання. Більш просунутими з погляду виділення семантики різних слів є сучасні рішення, отримані в результаті навчання на великих корпусах документів. До них належить, зокрема, інструмент word2vec, розроблений у 2013 році компанією Google, який набув широкого поширення. Word2vec обчислює векторне представлення слів, навчаючись на вхідних текстах. Це векторне подання ґрунтується на контекстній близькості: слова, що зустрічаються в тексті поруч з одними й тими самими словами (а отже, мають схожий сенс), матимуть близькі координати в багатовимірному просторі (норма відстані між їхніми векторними поданнями буде малою).

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)
Gender	−1	1	-0.95	0.97
Royal	0.01	0.02	0.93	0.95
Age	0.03	0.02	0.70	0.69
Food	0.09	0.01	0.02	0.01

Рисунок 2. Приклад формування векторного представлення слів у методі word2vec (джерело зображення: deeplearning.ai by Andrew Ng)

Сформовані в такий спосіб вектори дають змогу обчислювати «семантичну відстань» між словами: наприклад, слова «король» і «королева» перебуватимуть у цьому поданні близько одне до одного, а слова «король» і «яблуко» - далеко.

Завдання

1) Завантажити англійську книгу «Alice's Adventures in Wonderland» http://www.gutenberg.org/files/11/11-0.txt

2) Реалізувати пайплайн опрацювання обраного тексту англійською мовою, включно з усією необхідною попередньою обробкою тексту, включно з приведенням слів до нижнього регістру, видаленням стоп-слів, цифр/неалфавітних символів, знаків пунктуації.

3) Розділити текст на глави і в кожній главі відібрати Топ-20 слів за допомогою алгоритму TF-IDF.

4) Реалізувати LDA алгоритм і порівняти результати з отриманими раніше за допомогою TF-IDF. Зробити висновки про застосовність реалізованих підходів.

Комп’ютерний практикум №4

Згорткові мережі та робота з зображеннями.

Мета роботи: отримати навички реалізації згорткової мережі та методу перенесення навчання.

Короткі теоретичні відомості

Архітектура згорткової нейронної мережі

Під час спроби застосувати звичайну повнозв’язну нейронну мережу для оброблення зображень з метою розв'язання якої-небудь задачі, наприклад, класифікації зображень собак і кішок, ми зіткнемося з такими проблемами:

- Втрата важливої інформації під час перетворення зображення на вектор. Дійсно, під час подачі на вхід повнозв’язної мережі зображення у вигляді одновимірного вектора (наприклад, із застосуванням методу reshape() бібліотеки numpy) ми неминуче втратимо інформацію про взаємне розташування об'єктів на зображенні та їхню топологічну структуру;

- Обчислювальна неефективність: справді, при перетворенні одноканального зображення розміром усього лише 28 x 28 пікселів розмірність вхідного вектора буде (784,1), що з погляду повнозв’язної мережі відповідатиме 784 ознакам зображення. З урахуванням того, що для картинок розміром понад 1 Мб розмірність вхідного вектора перевищить 106, стає очевидною неефективність використання такого підходу.

У згортковій нейронній мережі основним складовим блоком є шар згортки, що складається з декількох (зазвичай від одиниць до кількох десятків) фільтрів, кожен з яких налаштовується в процесі навчання для виявлення характеристичних ознак на зображенні. Ці фільтри в режимі «ковзного вікна» проходять по всьому зображенню, записуючи результат згортки з елементами зображення у відповідну комірку вихідного зображення. Вихідне зображення в цьому разі називається «картою ознак», оскільки відповідає виділеним за допомогою цього фільтра ознакам. Сама операція згортки, яка реалізується з кожним фільтром у згортковому шарі, виражається в попарному перемноженні елементів фільтра з елементами вхідного зображення, причому розмір вікна в точності дорівнює розміру фільтра.

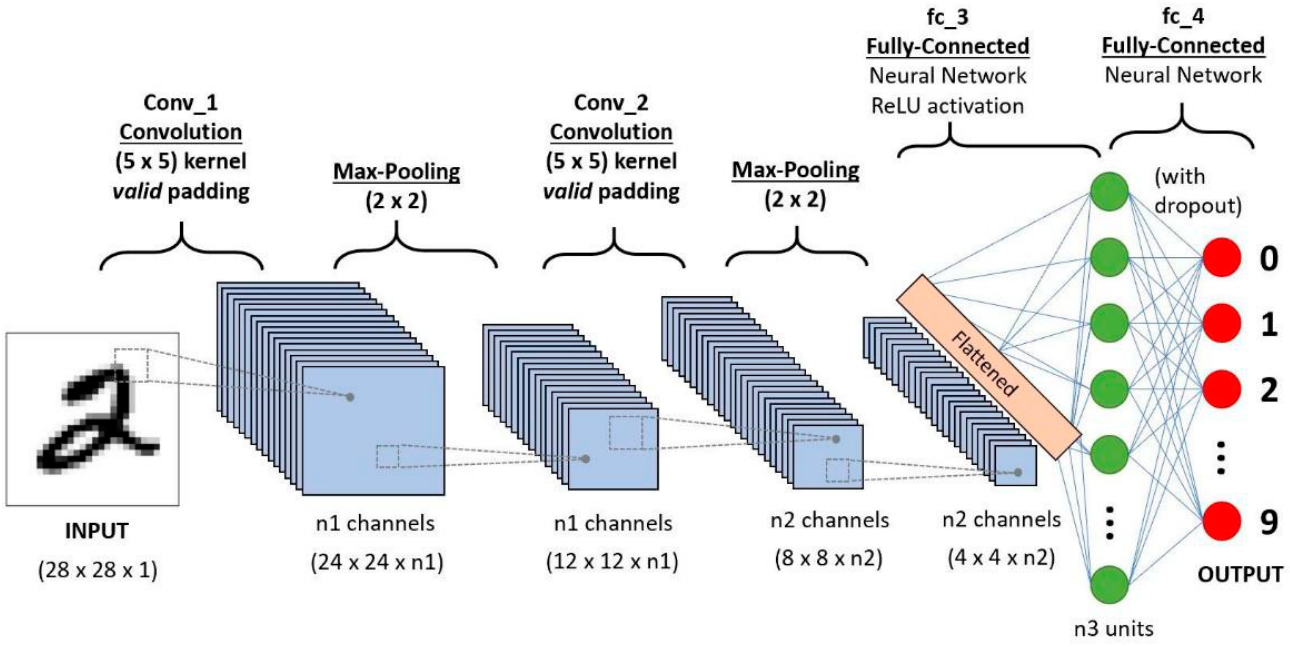


Рисунок 1. Приклад класичної архітектури згорткової нейронної мережі

Результат операції згортання можна виразити такою формулою:

$$(f * g)[m.n] = \sum_{k,l} f[m-k, n-k] * g[k,l],$$

де:

f - вихідна матриця зображення;

g - фільтр згортки.

Фільтри в перших шарах мережі активуватимуться базовими ознаками, на кшталт ліній, кутів, блобів, водночас під час руху до глибших шарів згорткової мережі фільтри в них виділятимуть дедалі складніші ознаки (див. рисунок 2). Розмір ядра зазвичай беруть у межах від 3x3 до 7x7. Якщо розмір ядра маленький, то воно не зможе виокремити будь-які ознаки, якщо занадто великий, то збільшується кількість зв'язків між нейронами. Кожен фільтр являє собою матрицю ваг, що налаштовуються в процесі навчання моделі: це одна з головних особливостей згорткової нейронної мережі, яка полягає в принципі «shared weights» (загальних ваг), що дає змогу скоротити кількість зв'язків і дає змогу знаходити ту саму ознаку в усій ділянці зображення.

При цьому залежно від методу опрацювання країв вихідної матриці результат згортки може бути меншим за вихідне зображення («valid»), такого самого розміру («same»). На практиці, щоб не втрачати інформацію з прикордонних пікселів зображення, частіше використовують згортку зі збереженням розміру вхідного зображення, для цього вихідну матрицю доповнюють одним або кількома шарами пікселів. Ця операція називається паддінг («padding») і описується параметром p, який відповідає за «товщину» доданого шару (див. рисунок 2). Найчастіше додаткові пікселі ініціалізуються нульовими значеннями.

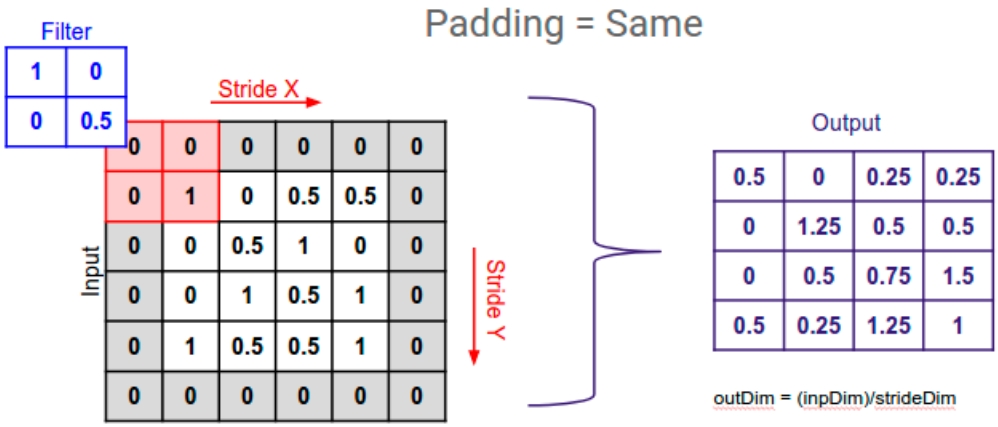


Рисунок 2. Ілюстрація операції «padding»

Розмір карти ознак можна обчислити за такою формулою:

$$n_{out} = \left\lceil \frac{n_{in} + 2p - k}{s} \right\rceil + 1,$$

де:

n_{in} - розмір вхідного зображення;

n_{out} - розмір вихідного зображення (карти ознак);

k - розмір фільтра;

p - розмір паддинга;

s - страйд.

За шаром згортки після застосування функції активації до отриманих карт ознак (у згорткових мережах - це найчастіше ReLU та її модифікації) майже завжди йде шар субдискретизації або пулінгу (див. рисунок 3). Завдання цього блоку - зменшення розмірності карт ознак попереднього шару. Це робиться для зниження числа параметрів моделі та щоб уникнути перенавчання. Також під час зменшення розмірності карт ознак ми здійснюємо перехід до іншого масштабу ознак, що зрештою дає змогу переходити від точок, ліній і плям на перших шарах до високорівневих ознак, які містять частини реальних об'єктів на останніх шарах мережі (див. рисунок 4). Розмір вікна пулінгу - зазвичай 2x2, при цьому застосовують два варіанти вибору значення у вікні: вибір максимального елемента (MaxPooling) або середнього елемента (AveragePooling), яке потім записують у відповідну комірку вихідної матриці.

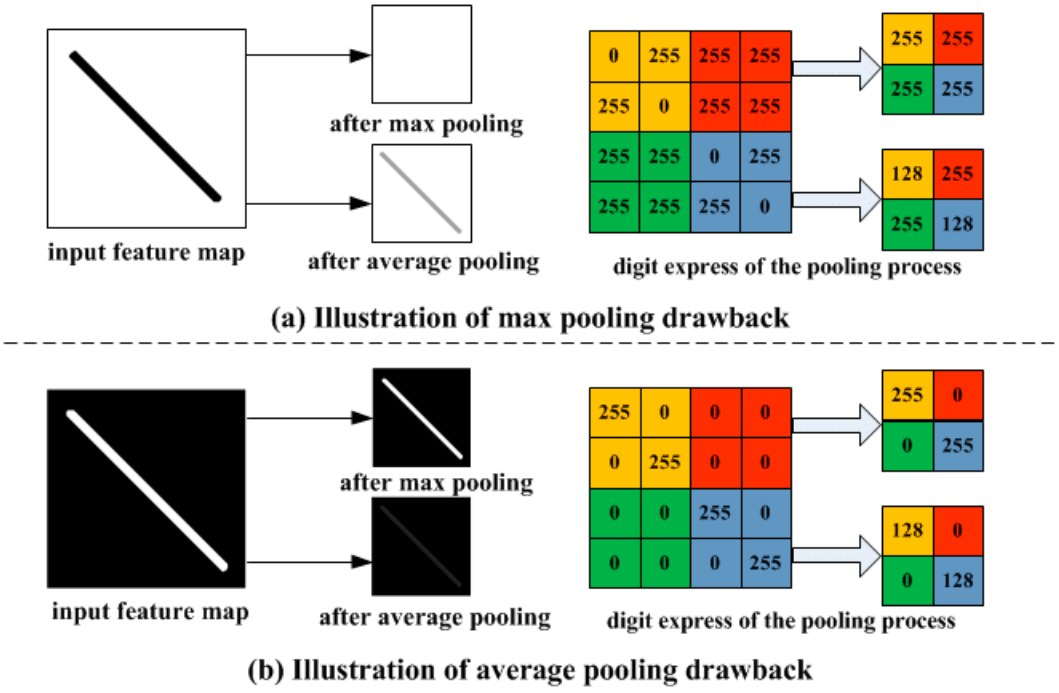


Рисунок 3. Ілюстрація операцій субдискретизації MaxPooling і AveragePooling

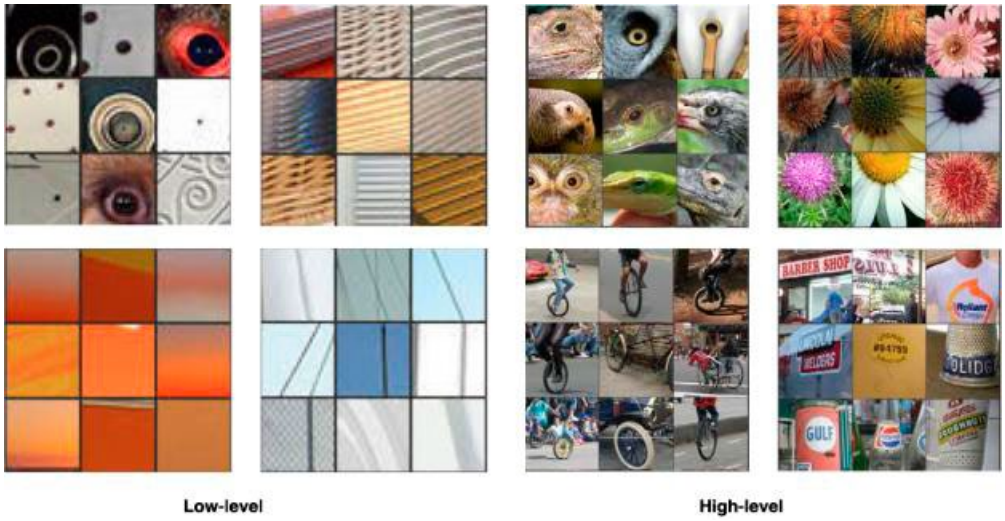


Рисунок 4. Ознаки низького рівня (а), що максимізують активацію нейронів на перших шарах згорткової мережі, і ознаки високого рівня (б) (що максимізують активацію нейронів на останніх шарах згорткової мережі)

За останні кілька років у завданнях класифікації зображень згорткові нейромережі домоглися точності, яку можна порівняти з точністю, що досягається людським мозком, а в деяких завданнях вони істотно перевершують людський мозок. Якщо перша велика нейромережа з групи університету Торонто, що показала результат, який перевершив найкращі моделі класичного комп'ютерного зору, містила трохи більше десятка шарів, то сучасні згорткові архітектури містять понад сотню шарів. Зрозуміло, для навчання таких глибоких мереж, що показують зараз state-of-the-art результати в задачах аналізу зображень, потрібен величезний обчислювальний час з використанням десятків GPU-процесорів.

Перенесення навчання

У реальних завданнях комп'ютерного зору часто трапляються ситуації, коли потрібно, наприклад, навчити класифікатор зображень з об’єктами, яких немає в базових датасетах, що використовувалися для навчання state-of-the-art моделей. У таких випадках можна використовувати вже налаштовані ваги моделей, навчених на великому наборі даних, як-от ImageNet, і потім здійснювати тонке налаштування додаткових параметрів на нові дані, що стосуються поточного завдання. Що більше нові дані відрізнятимуться від тих, на яких навчалася базова модель, то більше параметрів (шарів нейронної мережі) необхідно буде «перенавчити», щоб отримати хорошу точність у

новому домені даних. Інтуїція тут полягає в тому, що модель уже навчилася виділяти необхідні ознаки у великому наборі даних, її потрібно тільки «підналаштувати» для виконання конкретного завдання запам'ятовування нових високорівневих ознак цільового домену. Такий підхід має назву «перенесення навчання» («transfer learning») і широко застосовується в сучасному машинному навчанні. Метод дає змогу ефективно використовувати ваги «великих» моделей, для перенавчання яких знадобилося б використання істотних обчислювальних ресурсів. Більшість фреймворків глибокого навчання дає змогу завантажувати ваги попередньо навчених моделей для перевикористання і донавчання на цільовому домені даних. На рисунку 5 проілюстровано схему методу перенесення навчання.

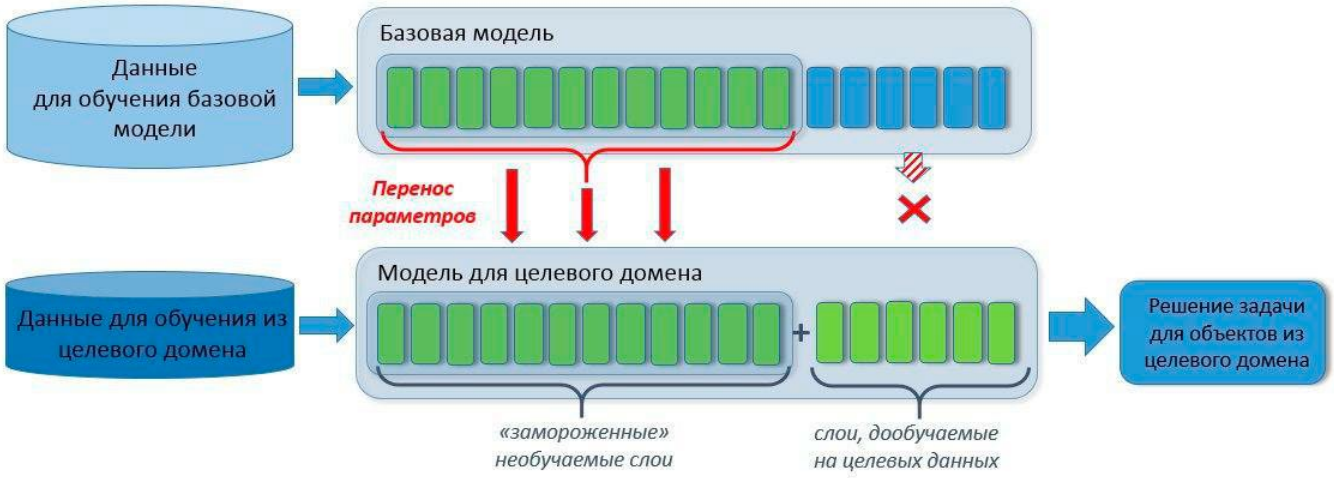


Рисунок 5. Схема перенесення навчання.

Завдання

1) Завантажте дані для навчання моделей:
https://drive.google.com/drive/folders/1nzVk4GOvKR6P87uPszUkKMPtaXV_wrZf

2) Побудуйте модель класифікації зібраних у датасеті зображень на собак і кішок.
   Для цього послідовно реалізуйте:
   а) повнозв'язну мережу з трьома прихованими шарами для класифікації зображень (на вхід - одновимірний вектор)
   б) згорткову нейронну мережу з двома блоками згортання і субдискретизації для тієї самої мети.

3) реалізуйте перенесення навчання для моделей VGG19 і ResNet, скориставшись вагами попередньо навчених моделей, «заморозивши» повнозв’язні шари і перенавчивши їх на нових даних.
4) Порівняйте продуктивність моделей.
5) Збільшіть число епох навчання для моделей (а) і (б) і побудуйте криві навчання, що демонструють явище перенавчання.

Література:

1) https://www.deeplearningbook.org/
2) https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53