Міністерство освіти і науки України
Луцький національний технічний університет

DATAMINING

Конспект лекцій для здобувачів
першого (бакалаврського) рівня вищої освіти
освітньої програми «Штучний інтелект та аналіз масивів даних»
спеціальності «Прикладна математика»
денної форми навчання

Луцьк 2025

УДК 004.8(075.8)

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ
Директор бібліотеки_____ Н. Поліщук

Рекомендовано до видання вченою радою факультету архітектури, будівництва та дизайну ЛНТУ,
протокол № ___ від «__» _____ 2025 року.

Голова вченої ради факультету ФАБД _____ О. Андрійчук

Розглянуто і схвалено на засіданні кафедри прикладної математики та механіки ЛНТУ,
протокол №___ від «___» _____ 2025 року.

Завідувачка кафедри прикладної математики та механіки _____ О. Мікуліч

Укладач: _____К. Вавринюк, асистент кафедри прикладної математики та механіки ЛНТУ

Рецензент: _____ кандидат технічних наук, доцент О. Приходько

Відповідальний за випуск: _____ О. Мікуліч, доктор технічних наук, професор, завідувачка кафедри прикладної математики та механіки ЛНТУ

DataMining: конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Штучний інтелект та аналіз масивів даних» спеціальності «Прикладна математика» денної форми навчання/ уклад. К. Вавринюк Луцьк: ЛНТУ, 2025. – 84 с.

Конспект лекцій для дисципліни «DataMining». Призначений для студентів освітньої програми «Штучний інтелект та аналіз масивів даних» спеціальності «Прикладна математика» денної форми навчання.

ЗМІСТ

ЛЕКЦІЯ 1

Вступ до Data Science. Філософія та інструментарій дослідницького аналізу даних (EDA)

Чи чули ви фразу "дані – це нова нафта". Чому вона стала такою популярною саме зараз? Відповідь криється у трьох глобальних зрушеннях: по-перше, вибухоподібне зростання обсягів даних (Big Data), які генерує цифровий світ; по-друге, шалене зростання обчислювальних потужностей, що дозволяє нам ці дані обробляти; і по-третє, розвиток складних алгоритмів. Саме на перетині цих трьох доріг і народилася професія Data Scientist.

Але що це означає на практиці? Це означає, що ми живемо в унікальний час, коли можна знаходити відповіді на складні питання, спираючись не лише на інтуїцію, а й на об'єктивні докази, приховані в даних. Наше сьогоднішнє завдання – навчитися бути свого роду детективами або перекладачами: перекладати мову сирих даних на мову людських знань та бізнес-рішень. А нашим головним методом для цього стане дослідницький аналіз даних, або EDA.

1. Визначення ключових понять

Data Science – це комплексна дисципліна, що поєднує статистику, комп'ютерні науки та знання предметної області для вилучення знань та інсайтів із даних. Це не лише про моделі, а про весь процес: від постановки питання до впровадження рішення.

Data Mining – це підрозділ Data Science, що фокусується на застосуванні алгоритмів для виявлення неочевидних патернів у великих наборах даних.

Машинне навчання – це набір інструментів та алгоритмів, які Data Science використовує для створення прогнозних моделей.

1.2. Життєвий цикл проєкту: ітеративний підхід OSEMN

Будь-який аналітичний проєкт – це не хаотичний пошук, а структурована експедиція. Модель OSEMN описує її ключові етапи. Це не жорстка інструкція, а гнучка філософія, адже ми постійно повертаємося на попередні кроки з новими ідеями та знаннями.

Класична модель OSEMN описує етапи, але важливо розуміти, що це циклічний та ітеративний, а не лінійний процес.

Починається все з отримання даних. Це етап визначення та збору сировини. Тут ми ставимо собі стратегічні питання: які саме дані нам потрібні для відповіді на наше бізнес-питання? Чи є вони в наших базах даних, чи їх потрібно купувати, чи, можливо, збирати з відкритих джерел? Надійність джерела – ключовий фактор успіху.

Отримані дані майже ніколи не бувають ідеальними. Це підводить нас до етапу очищення, який є фундаментом всього аналізу. Це кропітка, але вкрай важлива робота. Ми стикаємося з "брудом" у даних: пропущеними значеннями, помилками вводу (напр., "Чоловік", "чол.", "М"), неконсистентними форматами дат, а також аномальними викидами, які можуть спотворити результати. Наше завдання, як у реставратора, – акуратно очистити дані, не пошкодивши їхню структуру та зміст.

Коли дані готові, починається найцікавіший, творчий етап – дослідження, або EDA. Це вільний пошук, діалог з даними. Ми не намагаємося підтвердити заздалегідь відому відповідь. Навпаки, ми з відкритим розумом шукаємо будь-які закономірності, аномалії чи несподівані зв'язки. Саме на цьому етапі народжуються найцінніші гіпотези, які потім можна перевірити.

Маючи на руках гіпотези та кристально чисті дані, ми можемо переходити до моделювання. Тут ми використовуємо інструменти машинного навчання, щоб побудувати математичну модель. Це може бути

модель регресії для прогнозування числа (напр., ціни квартири) або модель класифікації для прогнозування категорії (напр., чи поверне клієнт кредит).

І нарешті, фінальний акорд – інтерпретація. Модель може дати нам точний прогноз, але без інтерпретації він не має цінності. На цьому етапі ми перетворюємо числа та графіки на історію – зрозумілу, переконливу історію для бізнесу. Ми пояснюємо, що означають наші результати, чому ми їм довіряємо і які конкретні дії можна вжити на їх основі. Це мистецтво сторітелінгу на основі даних.

2.Екосистема Python для аналізу даних

Для навігації цим складним процесом нам потрібен набір потужних інструментів. У світі Python таким набором є наукова екосистема.

Jupyter Notebook – це наша інтерактивна лабораторія. Його унікальність у тому, що він дозволяє поєднувати в одному документі живий код, який можна виконувати, його результати, форматований текст, математичні формули та візуалізації. Це створює ідеальне середовище для досліджень та відтворюваності результатів.

Фундаментом цієї екосистеми є NumPy, бібліотека для швидких числових обчислень. Вона надає нам ефективні структури даних (масиви) і є двигуном для багатьох інших інструментів.

Нашим головним робочим столом є Pandas. Він вводить ключовий об'єкт – DataFrame, який можна уявити як таблицю з даними. Але це не просто таблиця; це гнучка структура з потужним індексом, що дозволяє нам легко отримувати доступ до даних, фільтрувати їх, змінювати, групувати та аналізувати. Pandas – це, по суті, наша мова для спілкування з табличними даними.

Але щоб побачити приховані патерни, нам потрібні "окуляри" та "мікроскоп". Це наші інструменти для візуалізації – Matplotlib та Seaborn. Matplotlib – це низькорівнева, фундаментальна бібліотека, що дає тотальний

контроль над кожним пікселем графіка. Seaborn – це високорівнева, декларативна надбудова. "Декларативна" означає, що ми описуємо, що ми хочемо побачити (наприклад, "покажи мені залежність між рахунком та чайовими, розфарбувавши точки за статтю"), а Seaborn сам піклується про те, як це найкраще намалювати. Саме тому він ідеальний для швидкого дослідницького аналізу.

3. Філософія EDA. Як ставити питання до даних

Дослідницький аналіз – це не просто виконання команд. Це структурований процес постановки запитань, керований цікавістю та скептицизмом. Він розгортається від простих спостережень до складних гіпотез. Існує два основні типи аналізу: одномірний та двомірний

3.1. Одномірний аналіз (Univariate Analysis)

Все починається з одномірного аналізу, де ми вивчаємо кожну змінну окремо, ніби знайомлячись з кожним персонажем нашої історії, щоб зрозуміти її характеристики.

Для категоріальних змінних, наприклад, день тижня, ми хочемо знати частоту появи кожної категорії.

Для числових змінних (сума рахунку, вік), ми хочемо зрозуміти дві ключові речі: її центр та її розкид.

Центр – це типове значення. Середнє арифметичне – це простий і зрозумілий показник, але він дуже чутливий до аномально великих значень (викидів).

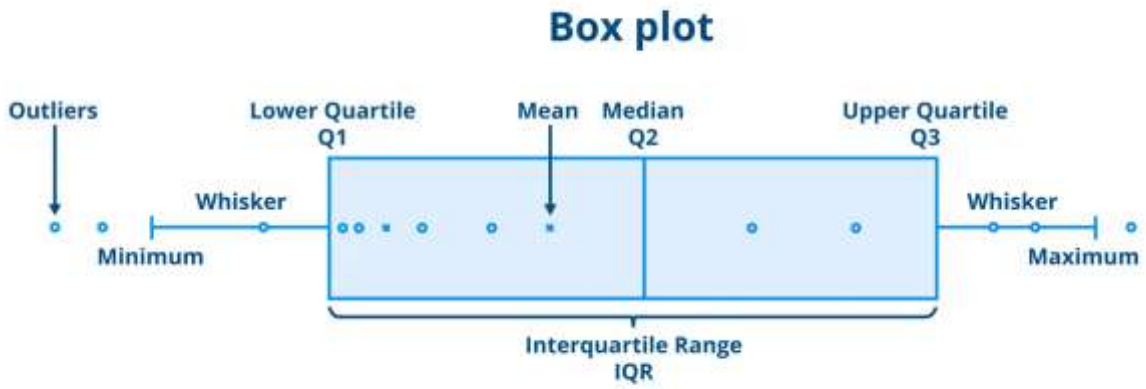
Уявіть, що до вашої групи з 9 людей із зарплатою $1000 приєднався один мільйонер. Середня зарплата різко зросте і перестане відображати реальність для більшості. Саме тому аналітики часто надають перевагу медіані – значенню, що знаходиться рівно посередині відсортованого ряду, на яке викиди майже не впливають.

Розкид показує, наскільки дані мінливі. Стандартне відхилення інтуїтивно можна розуміти як "типову відстань" від середнього значення.

Щоб побачити це все візуально, ми використовуємо гістограми, які показують форму розподілу, та коробчасті діаграми (boxplots).

Boxplot – це геніальний винахід, що на маленькому просторі показує медіану, 50% "типових" даних (міжквартильний розмах) та ідентифікує потенційні аномалії-викиди.

Boxplot – це статистичний інструмент візуалізації, який компактно зображує розподіл числових даних, показуючи медіану, нижній і верхній квантилі (перший та третій), а також викиди та діапазон даних за допомогою "вусів". Він дозволяє швидко порівнювати показники, виявляти закономірності та оцінювати симетрію або зміщення розподілу даних.



Компоненти Box plot:

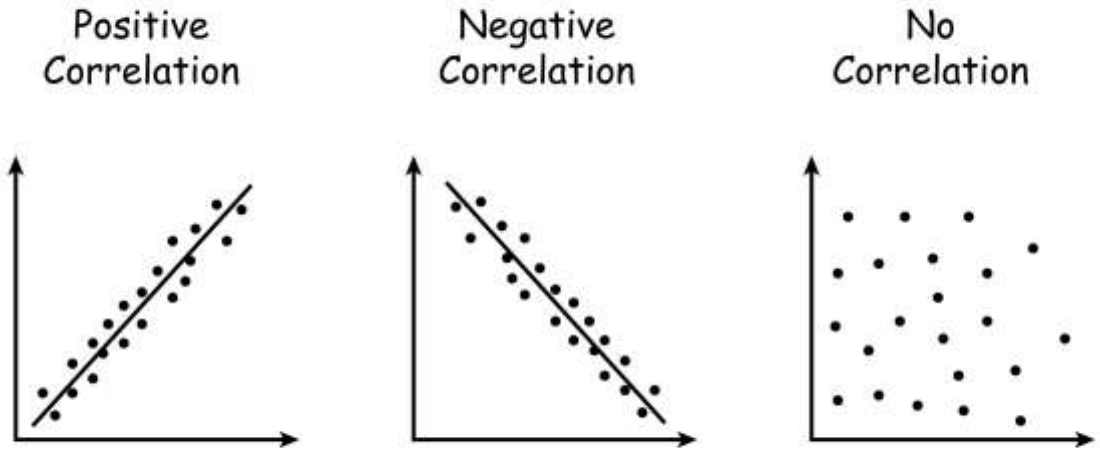
- коробка (box) центральна частина графіка, що містить 50% даних, де:
  - нижня межа коробки: перший квантиль (q1), 25-й процентиль;
  - верхня межа коробки: третій квантиль (q3), 75-й процентиль;
  - лінія всередині коробки: медіана (q2), 50-й процентиль;
- вуса (whiskers): лінії, що виходять з коробки:
  - нижня вуса: показує мінімальне значення в межах, що не є викидом;
  - верхня вуса: показує максимальне значення в межах, що не є викидом;

— викиди (outliers): дані, які знаходяться за межами вусів. Вони позначаються окремими символами, наприклад, хрестиком.

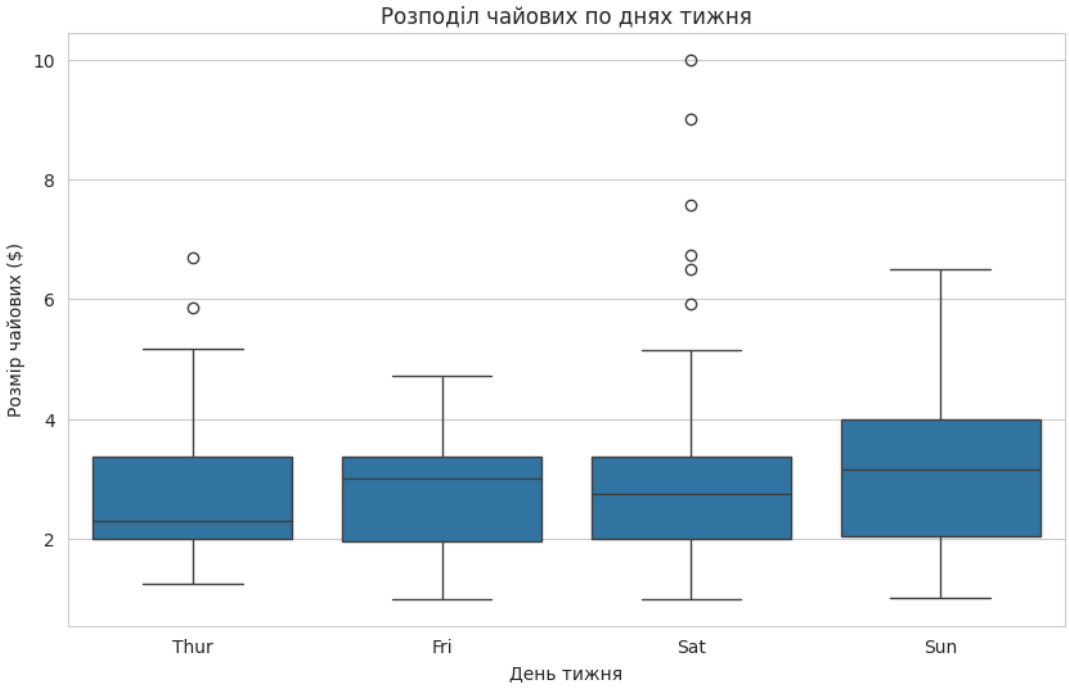
3.2. Двомірний аналіз

Після знайомства з кожним "персонажем" окремо, ми переходимо до двомірного аналізу, досліджуємо зв'язки між парами змінних вивчаючи їхні взаємовідносини.

Як пов'язані дві числові змінні, наприклад, сума рахунку та розмір чайових? На це питання найкраще відповідає діаграма розсіювання - scatterplot. Вона дозволяє нам візуально оцінити напрямок (позитивний чи негативний) та силу зв'язку(наскільки точки щільно групуються навколо уявної лінії).



А як числова змінна відрізняється між різними категоріями, наприклад, розмір чайових у різні дні тижня? Тут ми знову використовуємо коробчасті діаграми, розміщуючи їх поруч для легкого порівняння медіан та розкидів у кожній групі.


Розподіл чайових по днях тижня
Розмір чайових ($)
10
8
6
4
2
Thur
Fri
Sat
Sun
День тижня

ЛЕКЦІЯ 2

Робота з "брудними" даними: пропуски та викиди

У минулій лекції ми говорили, що дані – це нова нафта. Сьогодні ми заглибимося в цю аналогію. Сира нафта, щойно видобута з землі, є малокорисною. Щоб перетворити її на пальне, потрібен складний процес очищення та переробки. З даними абсолютно та сама історія. Сирі дані, які ми отримуємо, майже завжди містять "домішки", які роблять їх непридатними для якісного аналізу та моделювання.

Сьогодні ми зосередимося на двох найпоширеніших типах таких "домішок": пропущені значення та викиди.

Пропущені значення (NaN) – це "дірки" в наших даних, де інформація відсутня.

Викиди (Outliers) – аномальні, екстремальні значення, що сильно вибиваються із загальної картини.

Наша мета – навчитися бути майстрами "нафтопереробного заводу даних", ідентифікувати ці проблеми та застосовувати правильні стратегії для їх вирішення.

1. Порожнеча в даних: Глибоке розуміння та робота з NaN

Пропущені значення в даних – це не просто технічна незручність. Це відсутній фрагмент історії, який ми намагаємося зрозуміти. Перш ніж "ремонтувати" ці прогалини, ми, як справжні детективи, повинні зрозуміти їхню природу. Відповідь на питання "*чому* ці дані відсутні?" визначає всю подальшу стратегію роботи з ними.

Чому дані зникають?

У статистиці прийнято розрізняти три основні механізми виникнення пропусків. Це не просто академічна класифікація; це фундаментальна основа для прийняття правильних аналітичних рішень.

MCAR (Missing Completely At Random) – повністю випадкові пропуски. Це "ідеальний" сценарій для аналітика, хоча й найрідкісніший. Пропуск є повністю випадковим, якщо ймовірність його виникнення не залежить ані від самого значення, яке могло б бути на його місці, ані від будь-яких інших даних у нашому датасеті. Це справді випадкова помилка.

Наприклад: людина заповнює анкету і випадково, через неуважність, пропускає одне питання. Оскільки такий пропуск не несе в собі прихованої інформації, найпростіша стратегія – видалення рядків з такими пропусками (якщо їхня кількість незначна) – не призведе до систематичного викривлення результатів.

MAR (Missing At Random) – випадкові пропуски. Це значно поширеніший і цікавіший випадок. Пропуск вважається випадковим, якщо ймовірність його виникнення залежить від іншої, наявної в нас інформації, але не від самого пропущеного значення. Іншими словами, в наших даних є "ключі" або "підказки", які пояснюють, чому виник пропуск.

Наприклад: у медичному опитуванні ми бачимо, що чоловіки значно частіше пропускають питання про психологічний стан, ніж жінки. Пропуск у колонці "психологічний стан" не є повністю випадковим, він пов'язаний зі значенням у колонці "стать". Це добра новина, оскільки ми можемо використати цю залежність для більш "розумного" заповнення пропусків.

MNAR (Missing Not At Random) – невипадкові пропуски. Це найскладніший і найнебезпечніший тип пропусків. Тут ймовірність пропуску залежить від самого значення, яке ми не бачимо. Сам факт пропуску несе в собі важливу інформацію.

Наприклад: люди з дуже високими або, навпаки, дуже низькими доходами найчастіше відмовляються вказувати свій дохід в опитуваннях. Якщо ми просто заповнимо ці пропуски середнім доходом по вибірці, ми систематично завищимо оцінку для бідних і занизимо для багатих, що призведе до фундаментально неправильних висновків. Робота з MNAR

вимагає глибокого знання предметної області та часто – застосування складних моделей для оцінки самого механізму пропуску.

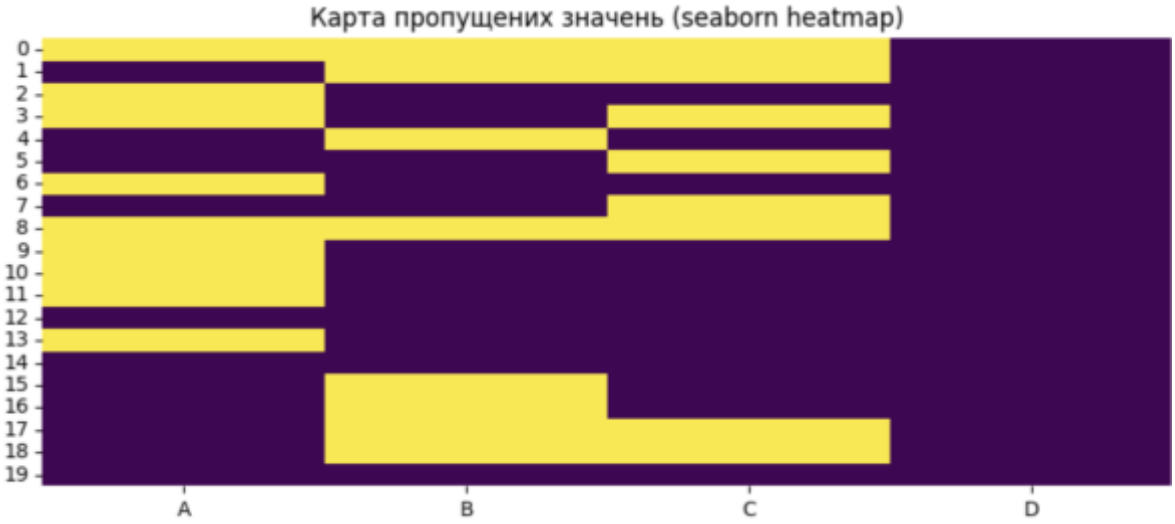
Практичний інструментарій

Озброївшись теорією, перейдемо до практики. Наш робочий процес складається з кількох логічних кроків.

Крок 1: Детекція – Пролити світло на порожнечу

Перш за все, нам потрібно оцінити масштаб проблеми. Найпростіший спосіб – викликати df.isnull().sum(), щоб отримати кількість пропусків у кожній колонці. Але для повнішої картини, особливо у великих датасетах, дуже корисною є візуалізація.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Створюємо приклад датафрейму з пропусками
np.random.seed(42)
df = pd.DataFrame({
    "A": np.random.choice([1, 2, np.nan], size=20),
    "B": np.random.choice([5, 6, 7, np.nan], size=20),
    "C": np.random.choice([10, np.nan], size=20),
    "D": np.random.randint(1, 100, size=20)
})
# Табличка з кількістю пропусків
missing_count = df.isnull().sum()
# Карта пропусків через seaborn
plt.figure(figsize=(10, 4))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Карта пропущених значень (seaborn heatmap)')
plt.show()
```



Цей код створює теплову карту, де кожна жовта лінія вказує на пропущене значення. Це дозволяє миттєво побачити не тільки обсяг, але й патерни пропусків. Наприклад, якщо жовті лінії групуються разом, це може вказувати на залежність між пропусками в різних колонках.

Крок 2: Ампутація – Стратегія видалення (.dropna())

Це найрадикальніший метод. Його варто застосовувати обережно, коли ви впевнені, що не втратите цінну інформацію.

Коли це виправдано?

- коли пропусків дуже мало (умовно, < 5%);
- коли пропуски є повністю випадковими (mcar);
- коли певна колонка або рядок майже повністю порожні і не несуть інформації.

Як це зробити гнучко? Метод .dropna() має корисні параметри:

- axis=0 (за замовчуванням) видаляє рядки з пропусками;
- axis=1 видаляє колонки з пропусками;
- thresh=N залишає тільки ті рядки/колонки, в яких є щонайменше N непропущених значень.

Крок 3: Реставрація – Стратегії імпьютації (.fillna())

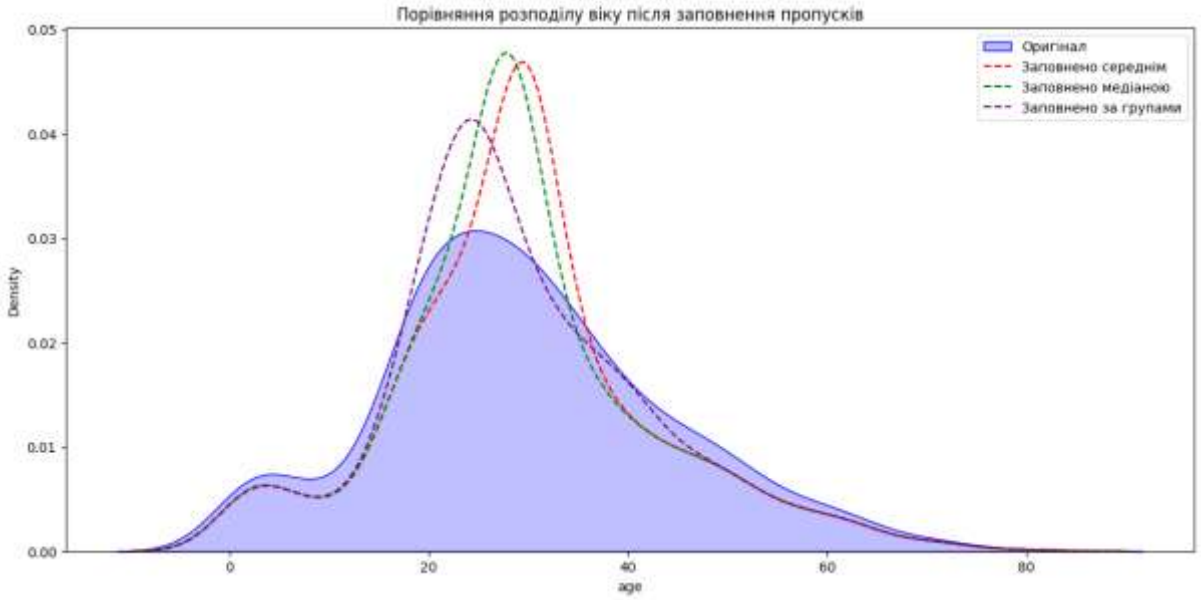
Це більш тонкий та рекомендований підхід, який полягає в заміні NaN на обґрунтовані значення.

Прості методи (для швидкого аналізу):

Заповнення середнім (.mean()): df['age'].fillna(df['age'].mean()). Даний метод зазвичай застосовується для числових даних без сильних викидів, коли розподіл близький до симетричного.

Заповнення медіаною (.median()): df['age'].fillna(df['age'].median()). Найкращий вибір за замовчуванням для числових даних. Медіана стійка до викидів і добре працює з асиметричними даними (ціни, доходи).

Заповнення модою (.mode()[0]): df['city'].fillna(df['city'].mode()[0]). Єдиний логічний вибір для категоріальних даних. Заповнюємо найпопулярнішим значенням.

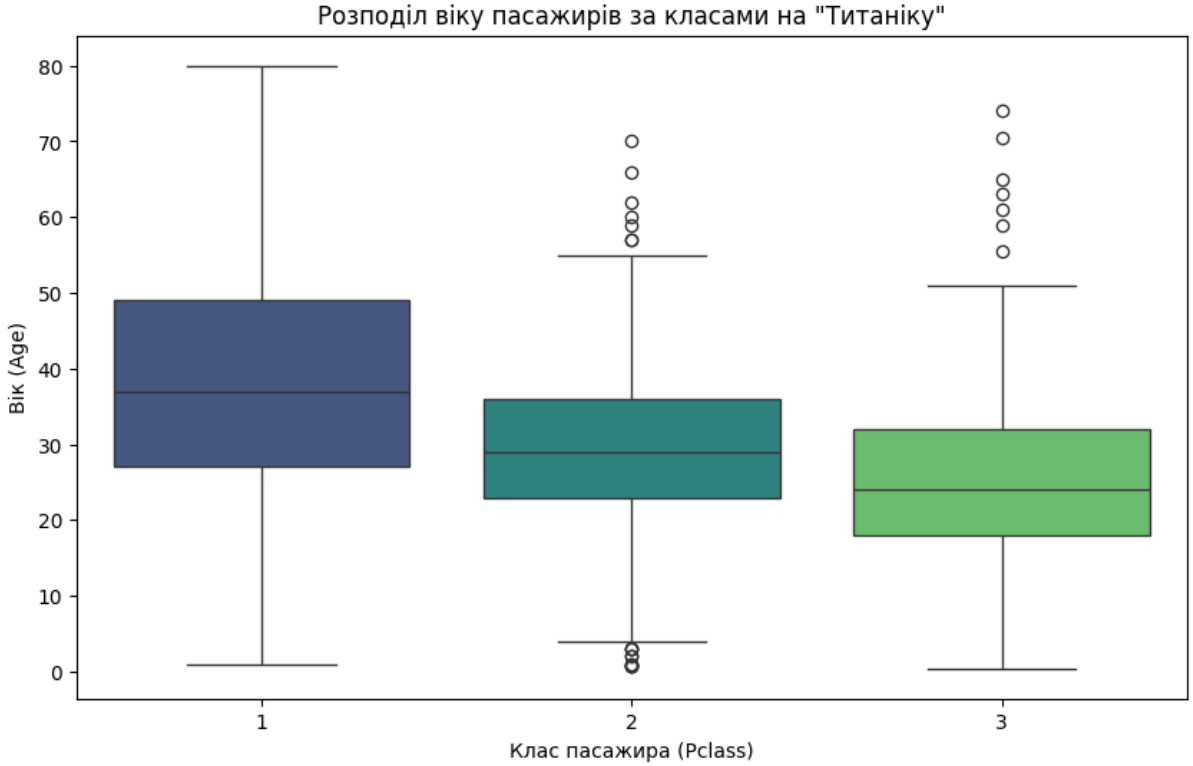


Просунутий метод: групова імпьютація Це значно точніший підхід, який враховує структуру даних. Замість того, щоб використовувати загальну статистику, ми розраховуємо її для кожної підгрупи окремо, **тобто** заповнюємо пропущений вік пасажира "Титаніка" не загальним середнім віком, а середнім віком пасажирів його ж класу.

Як реалізувати? За допомогою потужної комбінації groupby() та transform():

Заповнюємо NaN в 'age' медіанним віком для кожної групи 'pclass'

```
df['age']=df['age'].fillna(df.groupby('pclass')['age'].transform('median'))
```



Цей однорядковий код робить складну річ: для кожного NaN він знаходить, до якої групи pclass належить цей рядок, обчислює медіану age для цієї конкретної групи і підставляє її.

1-й клас: медіанний вік пасажирів першого класу був найвищим, близько 37 років. Це вказує на те, що пасажири цього класу були переважно зрілими та заможними людьми. Розмах віку тут також значний, але менше, ніж у 3-му класі.

2-й клас: пасажири другого класу були в середньому молодшими, з медіанним віком близько 29 років. Розподіл віку тут найбільш щільний, що свідчить про меншу вікову різноманітність у порівнянні з іншими класами.

3-й клас: медіанний вік у третьому класі був найнижчим – приблизно 24 роки. Цей клас мав найбільший діапазон вікових груп, включаючи велику кількість дітей та молодих людей, що, ймовірно, пов'язано з тим, що цілі сім'ї емігрували в пошуках кращого життя.

Кожна "коробка" на діаграмі показує інтерквартильний розмах (від 25-го до 75-го перцентиля), лінія всередині – медіану (50-й перцентиль), а "вуса" простягаються до мінімального та максимального значень, за винятком викидів, які позначені окремими точками.

Існують ще складніші методи, такі як KNN-імпьютація (заповнення на основі "найближчих сусідів") або регресійна імпьютація (побудова моделі для прогнозування пропущених значень), які ви зможете опанувати з часом.

Не існує єдиного "правильного" способу роботи з пропусками. Вибір стратегії – це завжди компроміс, який вимагає від аналітика розуміння як статистичних властивостей методів, так і бізнес-контексту даних.

Екстремуми в даних: Ідентифікація та приборкання викидів (Outliers)

Якщо пропущені значення – це "дірки" в наших даних, то викиди – це "гори" та "прірви". Це значення, які настільки відрізняються від решти, що викликають підозру. Робота з ними – це тонке мистецтво, що вимагає не лише технічних навичок, але й інтуїції та глибокого розуміння контексту.

Викид (Outlier) – це спостереження, яке лежить на аномальній відстані від інших значень у випадковій вибірці. Однак не всі викиди однакові. Ключове питання, яке ми маємо собі поставити: це помилка чи інсайт?

Причини виникнення.

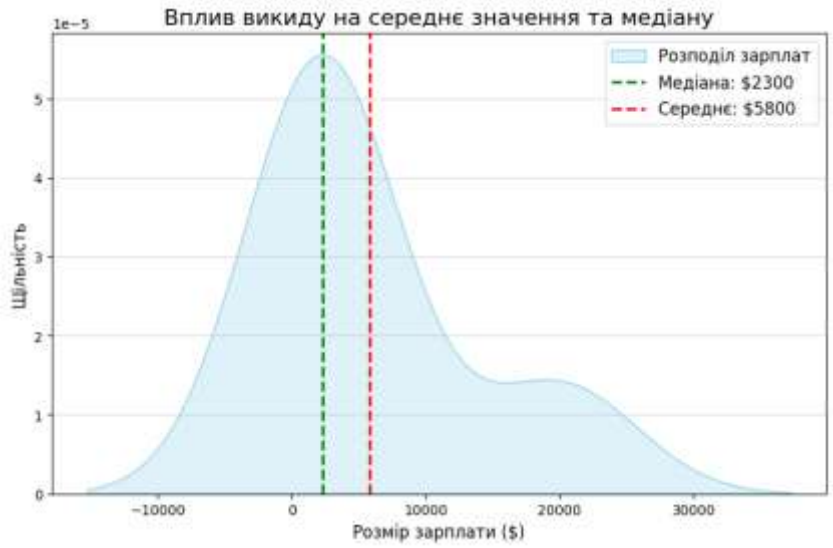
Помилки. Це може бути помилка при введенні даних (наприклад, зарплата $5,000,000 замість $50,000), збій вимірювального обладнання, або

помилка при передачі даних. Такі викиди є "шумом" і підлягають корекції або видаленню.

Реальні, але рідкісні події. Це найцікавіший випадок. Рахунок у ресторані на $1000 – це не помилка, а реальна подія (можливо, банкет). Транзакція на величезну суму може бути сигналом шахрайства. Найвищий бал на іспиті може вказувати на геніального учня. Такі викиди є "сигналом" і містять надзвичайно цінну інформацію.

Чому вони небезпечні? Викиди мають руйнівний вплив на багато статистичних показників та моделей, оскільки вони порушують основні припущення про розподіл даних.

Спотворення статистики. Вони сильно "тягнуть" на себе середнє значення, роблячи його нерепрезентативним. Уявіть набір зарплат: [2000, 2200, 2300, 2500, 20000]. Медіана (серединне значення) тут $2300, що добре описує типову зарплату. А середнє значення – $5800, що є абсолютно неінформативним для більшості людей у групі. Викиди також "роздувають" стандартне відхилення.



На графіку щільності ми бачимо два основні піки: один, щільний, де згруповані чотири зарплати, та інший, далекий, де знаходиться одна зарплата у $20,000.

Медіана ($2300) – зелена лінія. Вона знаходиться прямо в центрі основної групи зарплат. Це серединне значення, і йому байдуже, наскільки далеко знаходиться крайня точка.

Середнє значення ($5800) – червона лінія. Вона значно зміщена вправо, у бік аномально високої зарплати. Це відбувається тому, що середнє значення – це "центр мас" усього набору даних. Викид у $20,000, як важкий вантаж, "перетягує" середнє на себе.

Чому так відбувається?

Медіана – це робастний показник. Вона стійка до викидів, оскільки для її розрахунку важливий лише порядок чисел, а не їхня величина. Вона просто вказує на "середину" відсортованого списку.

Середнє значення – чутливе до кожного числа в наборі. Воно враховує величину кожного значення, тому навіть один екстремальний показник може кардинально змінити результат, роблячи його менш репрезентативним для типового значення у вибірці.

Цей приклад чудово ілюструє, чому для аналізу даних, де можливі сильні відхилення (як зарплати, ціни на нерухомість тощо), медіана часто є більш надійним показником центральної тенденції, ніж середнє арифметичне.

Вплив на моделі.

Чутливі моделі: алгоритми, що базуються на відстанях (як K-Means) або на сумі квадратів помилок (як Лінійна регресія), надзвичайно чутливі до викидів. Один екстремум може повністю змінити лінію регресії або перетягнути на себе центр кластера.

Стійкі моделі: алгоритми на основі дерев (Дерева рішень, Random forest) набагато стійкіші до викидів, оскільки вони ділять дані на групи за порогами, і для них не так важливо, наскільки далеко знаходиться екстремальне значення.

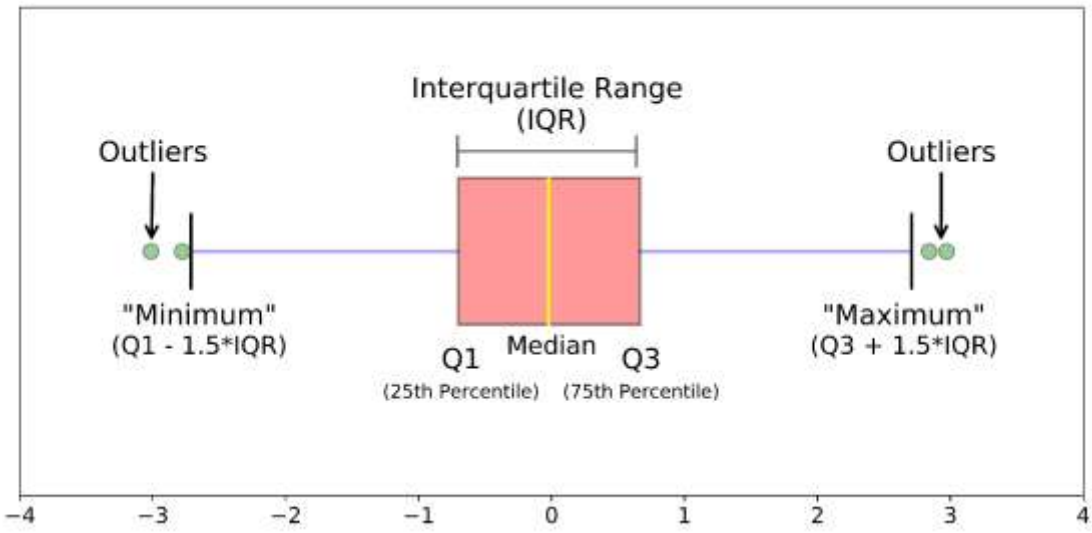
Детекція та нейтралізація викидів

Наш процес роботи з викидами складається з двох основних етапів: спочатку ми їх знаходимо, а потім вирішуємо, що з ними робити.

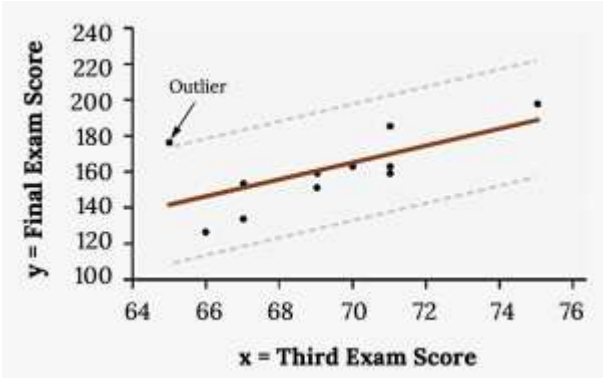
Крок 1: Пошук аномалій (Детекція)

Візуальні методи (інтуїтивний підхід).

Коробчаста діаграма (boxplot). Вона спеціально розроблена для ідентифікації викидів – вони відображаються як окремі точки за межами "вусів". Це дозволяє швидко оцінити наявність та кількість екстремумів.



Діаграма розсіювання (scatterplot). Незамінна для пошуку викидів у контексті двох змінних. Точка може не бути викидом по кожній осі окремо, але їхня комбінація може бути аномальною (наприклад, людина з ростом 2 метри і вагою 40 кг).



Статистичні методи (математичний підхід):

Правило міжквартильного розмаху (IQR Rule). Це формалізований підхід, який використовується для побудови "вусів" у boxplot. Він є надійним, оскільки базується на медіані та стійкий до самих викидів.

Алгоритм:

1. розраховуємо квантилі: Q1 = df['column'].quantile(0.25) та Q3 = df['column'].quantile(0.75).
2. розраховуємо IQR: IQR = Q3 - Q1.
3. визначаємо межі: lower_bound = Q1 - 1.5 * IQR та upper_bound = Q3 + 1.5 * IQR.
4. знаходимо викиди: df[(df['column'] < lower_bound) | (df['column'] > upper_bound)].

Метод Z-оцінки (Z-score). Цей метод вимірює, на скільки стандартних відхилень кожна точка віддалена від середнього. Добре працює для даних, розподіл яких є близьким до нормального ("дзвоноподібного"). Зазвичай, значення з Z-оцінкою > 3 або < -3 вважаються викидами.

Це менш надійний метод для асиметричних даних, оскільки середнє та стандартне відхилення самі по собі чутливі до викидів.

Крок 2: Прийняття рішення (Обробка)

Перш ніж видаляти чи змінювати викид, поставте собі питання:

— Що це за подія?

— Чи може вона бути реальною?

— Який її контекст?

Можливо, цей викид – найцінніша частина вашого датасету.

Видалення: найпростіший, але й найнебезпечніший метод. Його варто застосовувати, лише якщо ви на 100% впевнені, що викид є наслідком помилки вводу або вимірювання. Необдумане видалення реальних, хоч і рідкісних, подій може призвести до хибних висновків.

Трансформація: якщо ваші дані сильно асиметричні (мають "довгий хвіст" з екстремальних значень), можна застосувати математичну трансформацію, щоб "стиснути" цей хвіст.

Логарифмічна трансформація (np.log1p): дуже популярний метод. Вона сильно зменшує великі значення, але слабко впливає на малі, роблячи розподіл більш симетричним і зменшуючи вплив викидів.

Обрізання або "Вінсоризація" (Capping/Winsorizing): це м'який підхід, який не видаляє викиди, а "підтягує" їх до решти даних. Всі значення, що виходять за межі, встановлені за правилом IQR, замінюються на значення цих меж.

Розраховуємо межі (як у методі IQR)

upper_bound = Q3 + 1.5 * IQR

lower_bound = Q1 - 1.5 * IQR

Замінюємо все, що виходить за межі

df['column_capped'] = df['column'].clip(lower = lower_bound, upper = upper_bound)

Цей метод зберігає кількість даних і значно зменшує вплив екстремумів на середнє та стандартне відхилення.

Бінаризація (Binning): іноді числову змінну можна перетворити на категоріальну. Наприклад, замість точного віку можна створити групи: "0-18", "19-35", "36-50", "51+". У такому випадку аномальний вік "200" просто потрапить в останню групу "51+", і його екстремальність буде нейтралізована.

Очищення даних – це не просто технічна процедура, це глибоко аналітичний процес. Він вимагає від нас не лише знання коду, але й критичного мислення та розуміння контексту даних.

Пропуски – це не просто порожні клітинки, а сигнал, який потрібно правильно інтерпретувати.

Викиди – це не завжди "погані" дані; іноді саме вони є найцікавішими (наприклад, у задачах виявлення шахрайства).

Грамотна робота з "брудними" даними – це запорука побудови надійних та точних аналітичних моделей.

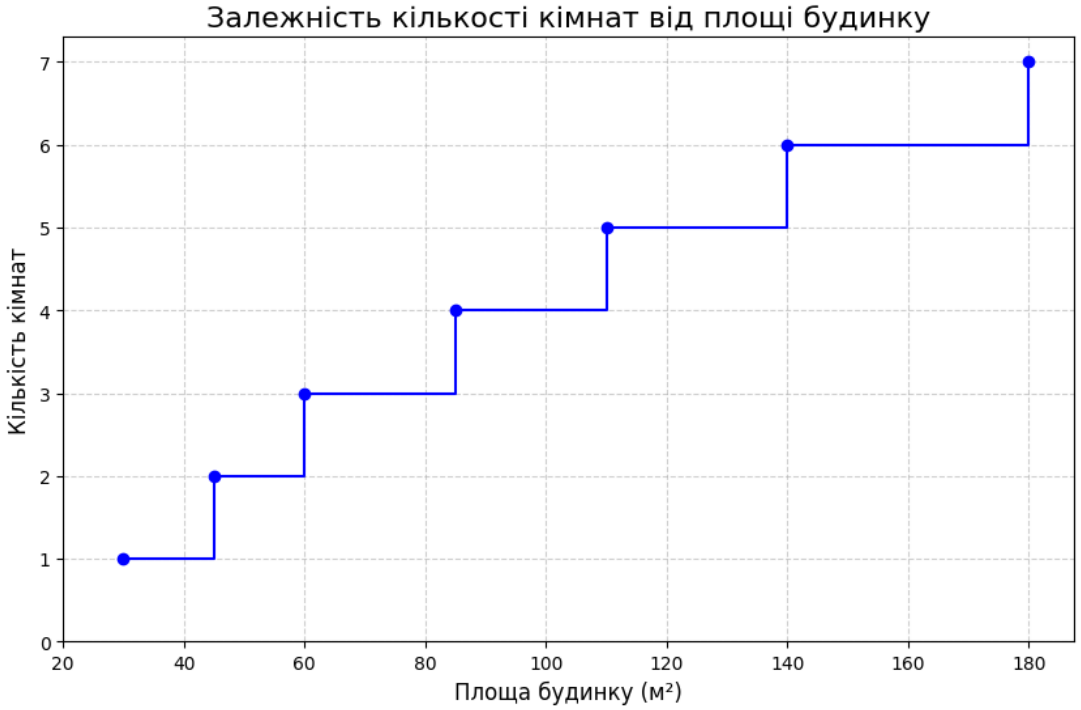
ЛЕКЦІЯ 3

Мистецтво створення нових даних. Feature Engineering

1. Що таке "ознака" та "простір ознак"?

Як ми вже знаємо, ознака – це окрема характеристика нашого об'єкта. Сукупність усіх можливих значень цих ознак створює багатовимірний простір ознак (feature space).

Уявіть собі простий 2D-графік, де по осі X – площа будинку, а по осі Y – кількість кімнат.



Кожен будинок у нашому датасеті – це одна точка в цьому просторі. Завдання моделі – провести в цьому просторі лінію або поверхню, яка найкращим чином розділяє, наприклад, дорогі та дешеві будинки.

Головна ідея інженерії ознак – так трансформувати цей простір, щоб об'єкти різних класів стали легше розділити. Ми можемо додавати нові осі (нові ознаки) або змінювати існуючі, щоб "розтягнути" простір у потрібних місцях.

2. Чому інженерія ознак – це 80% успіху моделі?

Покращення продуктивності: це не просто покращення, а часто стрибок у якості. Добре сконструйована ознака може дати моделі "еврику" – раптове розуміння прихованої закономірності.

Підвищення гнучкості моделі: інженерія ознак дозволяє простим моделям (як лінійна регресія) знаходити складні, нелінійні залежності. Наприклад, сама по собі широта може не казати нічого про ціну, але комбінація широти та довготи може вказувати на престижний район.

Використання експертних знань: це ваш найпотужніший інструмент. Якщо ви аналізуєте медичні дані, лікар підкаже, що співвідношення холестерину до ліпопротеїнів важливіше за їхні окремі значення. Якщо аналізуєте продажі – маркетолог знає, що ознака “день_тижня_п'ятниця” може бути ключовою.

Зменшення обчислювальної складності: створення однієї сильної ознаки (наприклад, площа) замість двох слабших (довжина, ширина) може зменшити кількість даних без втрати інформації та прискорити навчання.

3. Поглиблені техніки для категоріальних даних. Label Encoding vs. Ordinal Encoding

Кодування мітками (Label Encoding) та порядкове кодування (Ordinal Encoding) часто плутають, але вони слугують різним цілям, залежно від природи даних. Ключова відмінність полягає в тому, чи є осмисленим числове співвідношення між категоріями.

Коротко кажучи, порядкове кодування використовується для ознак, які мають внутрішній порядок, тоді як кодування мітками найкраще залишити для цільової змінної або як загальний термін для перетворення міток на числа.

Приклад Label Encoding для цільової змінної: якщо ми прогнозуємо категорію (так/ні, кіт/собака, колір - червоний, синій, зелений), ми кодуємо її числами (0/1/2).

Колір	Код
червоний	0
синій	1
зелений	2

Порядкове кодування (Ordinal Encoding) – використовується, коли ваші категоріальні дані мають чіткий, осмислений порядок або ранг. Воно перетворює категорії на цілі числа, які зберігають цю природну ієрархію. Призначення: Зберегти інформацію про ранжування категорій.

Приклад Ordinal Encoding: коли категорії мають чіткий порядок. Наприклад: ["низький", "середній", "високий"] → [0, 1, 2]. Тут порядок має сенс.

Для номінальних даних, де порядку немає, класичне кодування мітками вводить модель в оману.

One-Hot Encoding (OHE) та його проблеми

One-Hot Encoding (OHE) – це метод кодування категоріальних (якісних) ознак у числовий формат, який використовується в машинному навчанні та аналізі даних. Ідея полягає в тому, щоб перетворити кожну категорію на окрему бінарну (0/1) ознаку.

Метод	**Для чого підходить**	**Приклад даних**	**Як кодує**
Label Encoding	Цільові змінні	y = ["кіт","собака","папуга"]	[0,1,2]

Ordinal Encoding	Ознаки з порядком	["низький","середній","високий"]	[0,1,2]
One-Hot Encoding	Ознаки без порядку	["червоний","синій","зелений"]	[1,0,0], [0,1,0], [0,0,1]

Пастка фіктивних змінних. Коли ми створюємо N стовпців для N категорій, виникає проблема мультиколінеарності – один стовпець можна ідеально виразити через інші (наприклад, якщо is_Red=0 і is_Green=0, то ми точно знаємо, що is_Blue=1). Це може викликати проблеми у деяких моделях.

Яке рішення? Завжди видаляти один зі створених стовпців. Більшість бібліотек роблять це автоматично за допомогою параметра drop_first=True.

Проблема високої кардинальності. Що робити, якщо у нас 10,000 унікальних поштових індексів? Створювати 10,000 стовпців – погана ідея.

Рішення 1: об'єднання рідкісних категорій. Усі індекси, що зустрічаються менше 10 разів, можна об'єднати в одну категорію "Інше".

Рішення 2: цільове кодування. Дуже потужна техніка. Ми замінюємо кожну категорію на середнє значення цільової змінної для цієї категорії. Наприклад, категорію "Київ" можна замінити на середню ціну квартири в Києві.

Ризик. Високий ризик перенавчання (overfitting), оскільки ми використовуємо інформацію про правильну відповідь для кодування ознак. Потребує обережної реалізації (наприклад, крос-валідації).

4. Розширені техніки для числових даних

Ви вже бачили, що для категоріальних ознак (nominal/ordinal) є багато варіантів кодування. Але числові ознаки (вік, дохід, кількість клієнтів і т. д.) теж потребують спеціальної обробки, і тут у гру вступають так звані розширені техніки для числових даних.

Це набір методів попередньої обробки, які застосовуються до числових ознак, щоб:

- покращити якість моделей;
- прискорити навчання алгоритмів;
- зменшити вплив шуму та викидів;
- забезпечити коректне порівняння ознак (коли вони в різних шкалах).

Основні техніки

Дискретизація: стратегії поділу

Рівноширинна дискретизація (Equal-width binning): діапазон значень (від min до max) ділиться на N інтервалів однакової ширини. Часто допомагає у деревоподібних моделях. Проте, якщо в даних є викиди, більшість значень може потрапити в один-два інтервали, а інші будуть майже порожніми.

Квантильна дискретизація (Quantile binning): дані діляться так, щоб у кожному з N інтервалів була однакова кількість спостережень. Дана техніка краще працює з асиметричними даними.

Масштабування ознак (Feature Scaling). Багато алгоритмів (KNN, SVM, лінійні моделі, нейронні мережі) чутливі до масштабу даних. Якщо ознака вік змінюється від 20 до 70, а дохід – від 5000 до 100 000, модель буде вважати дохід набагато важливішим просто через більший діапазон значень.

Нормалізація (Min-Max Scaling): Перетворює дані так, щоб вони знаходились у діапазоні [0, 1].

Формула:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Коли використовувати: Коли ви точно знаєте мінімальні та максимальні межі даних і хочете їх зберегти. Чутлива до викидів.

Стандартизація (Z-score Standardization): Перетворює дані так, щоб вони мали середнє значення 0 і стандартне відхилення 1.

Формула:

$$X_{std} = \frac{X - \mu}{\sigma}$$

Це найпоширеніший метод, він не обмежує дані певним діапазоном, що робить його стійкішим до викидів.

Трансформації для роботи з асиметрією

Багато моделей краще працюють, якщо дані мають нормальний розподіл. Якщо ваша гістограма має "довгий хвіст" в один бік (асиметрія), це можна виправити.

Логарифмічна трансформація: np.log1p(x) – дуже ефективна для зменшення правосторонньої асиметрії (характерно для доходу, цін, кількості переглядів). log1p використовується замість log для коректної обробки нульових значень.

Трансформація Бокса-Кокса: автоматично знаходить найкращу степеневу трансформацію, щоб зробити дані максимально нормальними.

5. Створення ознак "з повітря"

Найбільш творча частина – створення абсолютно нових ознак.

Взаємодія ознак: комбінування двох або більше ознак. Наприклад, у нас є “висота_стелі” та “кількість_вікон”. Окремо вони можуть бути не дуже інформативними. Але їх добуток “висота_стелі” * “кількість_вікон” може створити нову ознаку "відчуття простору та світла", яка добре корелює з ціною.

Поліноміальні ознаки: дозволяють лінійним моделям знаходити нелінійні зв'язки. Ми додаємо до даних їхні степені (x^2,x^3) та комбінації ($x_1 \cdot x_2$).

Приклад: ціна будинку може рости не лінійно з площею, а швидше (квадратично). Додавши ознаку площа^2, ми допоможемо моделі це врахувати.

Ознаки з дати та часу: одна колонка з датою може стати джерелом багатьох ознак: рік, місяць, день тижня, номер тижня в році.

Бінарні ознаки: “Чи вихідний”, “Чи кінець місяця”, “Чи святковий день”

Інженерія ознак – це ітеративний процес. Ви створюєте нову ознаку, тестуєте модель, дивитеся на результат, аналізуєте помилки і повертаєтеся назад, щоб створити ще кращі ознаки. Знання предметної області + креативність + аналіз даних = потужна модель.

ЛЕКЦІЯ 4

Вступ до навчання з учителем

Навчання з учителем (Supervised Learning) – це найпоширеніший тип машинного навчання. Його головна ідея полягає в тому, що ми навчаємо модель на даних, які вже мають правильні відповіді.

Уявіть, що ви навчаєте дитину розрізняти тварин за картками. Ви показуєте картку із зображенням кота (це вхідні дані, або ознаки X) і кажете: "Це кіт" (це правильна відповідь, або цільова змінна y). Після перегляду сотень таких карток дитина вчиться самостійно розпізнавати котів на нових зображеннях.

Модель машинного навчання працює так само: вона аналізує тисячі прикладів (X, y), знаходить у них закономірності і вчиться прогнозувати y для нових, небачених раніше X.

У навчанні з учителем є два основні типи задач:

- Регресія (Regression) – коли ми прогнозуємо неперервне числове значення: ціну квартири, температуру завтра, кількість продажів.
- Класифікація (Classification) – коли ми прогнозуємо категорію (клас): чи є лист спамом, розпізнати тварину на фото, діагностувати хворобу.

Поняття регресії та інтуїція простої лінійної регресії

Регресія – це задача прогнозування числового значення. Сьогодні ми розглянемо її найпростіший випадок – просту лінійну регресію.

Проста лінійна регресія – це метод, що дозволяє змоделювати лінійну залежність між однією ознакою-предиктором (незалежною змінною X) та цільовою змінною (y).

Головна ідея: знайти таку пряму лінію, яка найкраще описує зв'язок між нашими даними.

Уявімо, що ми хочемо спрогнозувати оцінку студента на іспиті (y), знаючи лише кількість годин, які він готувався (X). Ми можемо нанести дані

на графік, де кожна точка – це один студент. Інтуїтивно зрозуміло, що через ці точки можна провести пряму, яка покаже загальний тренд: чим більше годин підготовки, тим вища оцінка.

Ця "найкраща" пряма і є нашою моделлю лінійної регресії. Математично ця лінія описується знайомим усім рівнянням:

$$Y=\beta_0+\beta_1X+\epsilon$$

Де:

- Y – цільова змінна, яку ми прогнозуємо (оцінка).
- X – наша ознака-предиктор (години підготовки).
- β_0 та β_1 – коефіцієнти моделі, які ми намагаємося знайти.
- ε – похибка (error), що відображає випадковість та фактори, які ми не врахували (настрій студента, складність білету тощо).

Як модель знаходить "найкращу" лінію?

Логічне питання: як саме алгоритм обирає цю єдину "найкращу" лінію серед нескінченної кількості можливих?

Для цього використовується Метод найменших квадратів. Його ідея дуже проста:

1. Алгоритм "приміряє" безліч різних ліній до наших даних.
2. Для кожної лінії він вимірює відстань (похибку ε) від кожної реальної точки даних до цієї лінії.
3. Щоб уникнути взаємного скасування додатних і від'ємних похибок, він бере квадрат кожної відстані.
4. Далі він сумує всі ці квадрати.

"Найкращою" лінією вважається та, для якої ця сума квадратів похибок є мінімальною. Тобто, це лінія, яка проходить "найближче" до всіх точок одночасно.

Інтерпретація коефіцієнтів моделі

Коефіцієнти – це серце нашої моделі. Вони показують, як саме X впливає на Y.

β_1 – Нахил (Slope)

Коефіцієнт нахилу β_1 – це найважливіший коефіцієнт. Він показує, на скільки в середньому зміниться Y, якщо X збільшиться на одну одиницю.

- Інтерпретація: Якщо ми отримали $\beta_1 = 5$, це означає, що кожна додаткова година підготовки в середньому підвищує оцінку на іспиті на 5 балів.
- Якщо β_1 позитивний – зв'язок прямий (чим більше X, тим більше Y).
- Якщо β_1 негативний – зв'язок обернений (чим більше X, тим менше Y).

β_0 – Перетин (Intercept)

Коефіцієнт перетину β_0 (також відомий як вільний член) – це прогнозоване значення Y, коли X дорівнює нулю.

- Інтерпретація: якщо ми отримали $\beta_0 = 40$, це означає, що прогнозована оцінка студента, який готувався 0 годин, становить 40 балів.
- Важливо: інтерпретувати β_0 варто з обережністю. Часто значення X=0 не має фізичного сенсу або знаходиться далеко за межами наших даних (наприклад, вага людини при зрості 0 см). У таких випадках β_0 є лише математичною константою, що допомагає лінії краще "сісти" на дані.

Метрика якості R^2 (Коефіцієнт детермінації)

Після того, як ми побудували нашу модель, нам потрібно зрозуміти, наскільки вона хороша. Головна метрика для цього в регресії – R-квадрат (R^2).

R-квадрат показує, який відсоток варіації (мінливості) цільової змінної Y пояснюється нашою моделлю за допомогою змінної X.

- Діапазон: R^2 завжди знаходиться в межах від 0 до 1 (або від 0% до 100%).
- Інтерпретація:

- $R^2 = 0.75$ означає, що 75% мінливості оцінок студентів пояснюється кількістю годин їхньої підготовки. Решта 25% припадає на інші фактори (похибку ε).
- $R^2 = 0$ означає, що наша модель не пояснює нічого.
- $R^2 = 1$ означає, що наша модель ідеально пояснює всі дані.

Чим ближче R^2 до 1, тим краще наша модель описує дані.

Про що варто пам'ятати: Припущення лінійної регресії 📝

Лінійна регресія – потужний, але простий інструмент. Вона добре працює, коли виконуються певні припущення щодо даних.

1. Залежність між X та Y має бути схожою на пряму лінію. Якщо дані утворюють криву, лінійна регресія буде поганою моделлю.
2. Похибки (ε) для різних спостережень не повинні бути пов'язані між собою.
3. Розкид похибок має бути приблизно однаковим для всіх значень X.

Перевірка цих припущень є важливим кроком у побудові якісної моделі, але це вже тема для більш поглибленого вивчення.

Від простої до множинної регресії

Ми розглянули просту лінійну регресію, де на результат впливала лише одна ознака (X). Але в реальному житті на оцінку студента впливає не тільки час підготовки, а й, наприклад, його середній бал за семестр (X_2), кількість відвіданих лекцій (X_3) тощо.

Модель, яка враховує декілька ознак, називається множинною лінійною регресією. Її рівняння виглядає так:

$$Y=\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_nX_n+\epsilon$$

Це дозволяє будувати значно складніші та точніші моделі, але основна ідея та інтерпретація коефіцієнтів залишаються дуже схожими.

ЛЕКЦІЯ 5

Будуємо реалістичні моделі

Ми навчилися передбачати одну річ за допомогою іншої. Наприклад, ми можемо приблизно передбачити вагу людини за її зростом. Це – проста регресія. Але в реальному світі на вагу впливає не лише зріст, а й вік, стать, харчування, рівень активності. Щоб врахувати всі ці фактори, нам потрібен потужніший інструмент.

Ускладнення моделі: Множинна регресія

Від простого до складного. Уявіть, що ви готуєте страву. У простій регресії ми б сказали: "Смак страви (Y) залежить тільки від кількості солі (X1)". Це дуже спрощено.

Множинна регресія – це як повноцінний рецепт. Смак страви (Y) залежить від кількості солі (X1), перцю (X2), часу приготування (X3) та температури духовки (X4). Кожен інгредієнт робить свій внесок у фінальний результат.

Множинна регресія – це статистичний метод, який дозволяє нам передбачити значення однієї змінної (залежної), використовуючи декілька інших змінних (незалежних).

Давайте розберемо її формулу на простому прикладі: прогнозування ціни будинку.

Ціна будинку (Y) може залежати від:

- його площі в квадратних метрах (X1).
- кількості спальних кімнат (X2).
- віку будинку в роках (X3).

Формула буде виглядати так:

$$\text{Ціна } (Y) = \beta_0 + \beta_1 \cdot X_1 + \beta_2 \cdot X_2 + \beta_3 \cdot X_3 + \epsilon$$

Давайте розшифруємо кожен елемент:

— Y (ціна) – це те, що ми хочемо передбачити. Наша цільова змінна.

- X_1, X_2, X_3 (площа, к-сть кімнат, вік) – це наші "фактори впливу" або предиктори.
- $\beta_1, \beta_2, \beta_3$ (коефіцієнти) – це "вага" або "сила впливу" кожного фактора.
  - β_1 показує, на скільки доларів в середньому зросте ціна будинку, якщо ми збільшимо його площу на один квадратний метр (за умови, що кількість кімнат і вік залишаться незмінними).
  - β_2 показує, як зміниться ціна при додаванні однієї спальної кімнати.
  - β_3 ймовірно, буде від'ємним. Він покаже, на скільки дешевшає будинок з кожним роком його існування.
- β_0 (вільний член або базова ціна) – це теоретична ціна будинку, який має 0 кв. метрів, 0 кімнат і 0 років. Практичного сенсу він часто не має, але математично він потрібен як відправна точка для нашої моделі.
- ϵ (помилка) – це "фактор непередбачуваності". Сюди входить все, що ми не врахували: якість ремонту, вид з вікна, настрій продавця. Наша мета – побудувати модель так, щоб ця помилка була якомога меншою.

Головна мета множинної регресії – знайти такі числові значення для $\beta_0, \beta_1, \beta_2, \ldots$, які найкраще описують зв'язок між нашими факторами (X) і тим, що ми прогнозуємо (Y).

Проблема мультиколінеарності

Уявіть, що ви слухаєте групу і намагаєтеся зрозуміти, хто з музикантів який внесок робить у загальну мелодію.

Ви можете легко розрізнити їхній внесок. А тепер уявіть, що гітарист і клавішник почали грати одну й ту саму мелодію в унісон. Тепер, якщо мелодія звучить голосніше, ви не можете точно сказати, хто з них додав гучності – гітарист чи клавішник? Їхній вплив змішався.

Мультиколінеарність – це те саме, але для змінних у регресії. Це ситуація, коли дві (або більше) незалежні змінні (X) сильно пов'язані між собою. Вони ніби "дублюють" інформацію одна одної.

Простий приклад: ми хочемо спрогнозувати споживання пального автомобілем (Y). Ми беремо два фактори:

- X1 – об'єм двигуна в літрах.
- X2 – потужність двигуна в кінських силах.

Очевидно, що потужність двигуна сильно залежить від його об'єму. Це і є мультиколінеарність.

Чому це погано?

1. "Плутанина" коефіцієнтів. Модель не може розібратися, кому "приписати" вплив на Y. Вона може сказати, що β1 (вплив об'єму) = 5.8, а β2 (вплив потужності) = -2.1. Знак "мінус" тут абсурдний – виходить, що чим потужніший двигун, тим менше пального він споживає? Це відбувається тому, що модель намагається "збалансувати" дублюючий вплив. Коефіцієнти стають ненадійними та неінтерпретованими.
2. Нестабільність моделі. Якщо ви додасте або видалите всього кілька рядків даних, коефіцієнти можуть кардинально змінитися (наприклад, з +5.8 на -10.3). Це ознака дуже хиткої моделі.

Як це виявити? Подивитися на кореляцію. Побудуйте таблицю, яка показує, наскільки сильно кожна пара змінних пов'язана між собою. Якщо коефіцієнт кореляції між двома змінними близький до +1 або -1 (наприклад, 0.9), це червоний прапорець.

Рішення:

- видалити зайве: просто приберіть одну зі змінних, що сильно корелюють, якщо потужність та об'єм двигуна дублюють одна одну, залиште щось одне;
- об'єднати: іноді можна створити нову, одну змінну, яка об'єднує в собі інформацію з декількох. Наприклад, замість "зріст" і "вага" можна створити "індекс маси тіла".

Важливість розділення даних: тренувальна та тестова вибірки

Уявіть собі студента, який готується до іспиту. У нього є збірник із 100 задач з відповідями.

Поганий студент бере всі 100 задач, зазубрює правильні відповіді і на питання викладача відповідає ідеально. Його результат на цих 100 задачах – 100%. Але коли на іспиті йому дають нову, 101-шу задачу, якої він не бачив, він не може її розв'язати. Він не *навчився* розв'язувати задачі, він їх *запам'ятав*.

Хороший студент бере 80 задач і вчиться на них, розбираючи логіку розв'язання. Потім він бере решту 20 задач (відповіді на які він не підглядав!) і намагається розв'язати їх самостійно. Це і є його "іспит". Результат на цих 20 задачах показує, наскільки добре він насправді зрозумів матеріал.

У машинному навчанні все точно так само.

Весь набір даних – це наш збірник із 100 задач.

Тренувальна вибірка (Training Set) – це ті 80 задач, на яких ми "навчаємо" нашу модель. Модель дивиться на ці дані і підбирає найкращі коефіцієнти (β), щоб мінімізувати помилку.

Тестова вибірка (Test Set) – це ті 20 задач, які модель ніколи не бачила. Ми використовуємо їх, щоб чесно перевірити, наскільки добре модель працює з новими даними. Це її фінальний "іспит".

Якщо ми будемо навчати і тестувати модель на одних і тих самих даних, ми отримаємо занадто оптимістичну, неправдиву оцінку її якості. Модель може просто "запам'ятати" правильні відповіді для тренувальних даних (це називається перенавчання або overfitting), але буде безпорадною в реальних умовах.

Завжди розділяйте дані. Зазвичай у співвідношенні 80/20 або 70/30.

Метрики помилок: Як виміряти, наскільки ми неправі?

Отже, наша модель пройшла "іспит" на тестових даних. Вона зробила прогнози. Тепер нам треба оцінити її роботу в числах. Як це зробити?

Уявімо, ми прогнозували ціни на 3 будинки. Ось результати:

Реальна ціна (y_i)	Прогноз моделі ($\hat{y}_i$)	Помилка ($y_i - \hat{y}_i$)
$100,000	$110,000	-$10,000
$200,000	$195,000	+$5,000
$150,000	$165,000	-$15,000

Як узагальнити ці помилки?

1. MAE (Mean Absolute Error). Це найпростіший і найчесніший спосіб.

1. Беремо помилки: -10000, +5000, -15000.
2. Ігноруємо знаки (беремо модуль): 10000, 5000, 15000. Нам не важливо, в який бік помилилася модель, важливо – наскільки.
3. Знаходимо середнє: (10000+5000+15000)/3=30000/3=10000.

MAE = $10,000. Інтерпретація: "В середньому, наша модель помиляється на $10,000". Просто і зрозуміло.

2. MSE (Mean Squared Error). Ця метрика працює інакше: вона дуже сильно "карає" за великі помилки.

1. Беремо помилки: -10000, +5000, -15000.
2. Підносимо кожну до квадрату:
    - (−10000)2=100,000,000
    - (+5000)2=25,000,000
    - (−15000)2=225,000,000 Зверніть увагу, як сильно зросла остання, найбільша помилка!
3. Знаходимо середнє:

(100,000,000+25,000,000+225,000,000)/3=350,000,000/3≈116,666,667.

MSE ≈ 116,666,667. Інтерпретація: це число важко інтерпретувати, бо це "квадратні долари". Тому саму по собі MSE використовують рідко, але вона є важливим кроком до наступної метрики.

3. RMSE (Root Mean Squared Error). Це просто корінь з попереднього результату. Ми робимо це, щоб повернутися до нормальних одиниць вимірювання (доларів).

1. Беремо MSE: 116,666,667.
2. Добуваємо квадратний корінь: $\sqrt{116,666,667} \approx 10,801$.

RMSE ≈ $10,801. Інтерпретація: "Типова помилка нашої моделі становить близько $10,801".

Головна відмінність MAE від RMSE:

- MAE ($10,000) – це просто середня помилка. Всі помилки для неї "рівні".
- RMSE ($10,801) – завжди трохи більша за MAE. Ця різниця тим більша, чим більше в наших даних аномально великих помилок. RMSE ніби каже нам: "Так, в середньому помилка може й 10,000, але будь обережний, іноді трапляються значно більші промахи!"

Якщо для вашої задачі одна велика помилка набагато гірша, ніж десять маленьких (наприклад, прогноз попиту на ліки), то краще орієнтуватися на RMSE. Якщо всі помилки однаково неприємні – MAE буде чудовим показником.

ЛЕКЦІЯ 6

Вступ до класифікації. Логістична регресія

До цього моменту ми з вами вчилися передбачати числа – ціну будинку, кількість продажів, температуру. Це називається регресією. Але що, як нам потрібно передбачити не число, а категорію? Наприклад, чи є лист спамом, чи ні? Чи пройде студент іспит, чи завалить? Чи поверне клієнт кредит? Для таких задач існує інший тип машинного навчання – класифікація.

Поняття класифікації. Відмінність від регресії

Класифікація – це процес присвоєння об'єкту однієї з кількох заздалегідь визначених міток або класів. Простими словами, модель вчиться відповідати на питання "Хто?" або "Що?".

Приклади задач класифікації:

- Медицина: визначити за аналізами, чи є у пацієнта захворювання (класи: "хворий", "здоровий").
- Банкінг: вирішити, чи видавати кредит клієнту (класи: "схвалити", "відхилити").
- Email: відфільтрувати спам (класи: "спам", "не спам").
- Розпізнавання зображень: визначити, що зображено на фото (класи: "кіт", "собака", "автомобіль").

Ключова відмінність від регресії полягає у типі вихідних даних, які ми прогнозуємо.

- Регресія – прогнозує неперервне числове значення.
    - *Приклад:* Якою буде ціна будинку? ($150,000, $210,500, ...).
- Класифікація – прогнозує дискретну мітку (клас).
    - *Приклад:* Чи є цей лист спамом? (Так/Ні).

Логістична регресія як базовий класифікатор

А чому б не використати для класифікації вже знайому нам лінійну регресію?

Уявіть, що ми кодуємо класи числами: "не спам" = 0, "спам" = 1. Якщо ми побудуємо лінійну регресію, вона може видати нам прогноз 0.9 (що логічно), але також може видати 1.5 або -0.2. Що означають ці числа? Як їх інтерпретувати? Пряма лінія не підходить для задачі, де відповідь має бути чітко обмежена.

Нам потрібна функція, яка "стискає" будь-який результат у зрозумілий діапазон **від 0 до 1**.

Інтуїція роботи логістичної регресії

Логістична регресія, попри слово "регресія" у назві, є моделлю класифікації. Вона працює у два етапи.

Етап 1 – лінійний. Всередині вона працює дуже схоже на лінійну регресію. Вона так само обчислює суму впливів усіх ознак з їхніми коефіцієнтами: $z=\beta_0+\beta_1X_1+\beta_2X_2+\dots$ Результат (z) може бути будь-яким числом: великим, малим, додатним, від'ємним.

Етап 2 – логістичний. Ось тут і відбувається магія. Результат z пропускається через спеціальну логістичну функцію (сигмоїду), яка перетворює його на ймовірність.

S-подібна крива (Сигмоїда). Сигмоїда – це математична функція, яка будь-яке вхідне число перетворює на значення в діапазоні від 0 до 1.

Її формула виглядає так:

$$\sigma(z)=1+e-z_1$$

Але важливіша не формула, а її форма та властивості: що б ми не подали на вхід (хоч -1000, хоч +500), вихід завжди буде між 0 та 1. Вихід сигмоїди можна трактувати як ймовірність того, що об'єкт належить до класу "1".

Модель розраховує лінійну комбінацію z для нового листа. Наприклад, отримали z=2.5. Це число z=2.5 подається на вхід сигмоїди: σ(2.5)=1+e−2.51≈0.92. Результат: 0.92. Ми інтерпретуємо це так: "ймовірність того, що цей лист є спамом, становить 92%".

Прийняття рішення

Отримавши ймовірність, нам потрібно ухвалити фінальне рішення. Для цього ми встановлюємо поріг відсікання (threshold), який зазвичай дорівнює 0.5.

- Якщо ймовірність ≥ 0.5 – відносимо об'єкт до класу "1" (наприклад, "спам").
- Якщо ймовірність < 0.5 – відносимо об'єкт до класу "0" (наприклад, "не спам").

Таким чином, логістична регресія не просто каже "так" чи "ні", а дає нам впевненість у своєму рішенні у вигляді ймовірності, що робить її дуже потужним та інтерпретованим інструментом.

Логістична регресія бере потужність лінійної регресії, але за допомогою "магічного перетворювача" – S-подібної кривої – адаптує її для задач класифікації, видаючи на виході зрозумілу ймовірність. Це робить її ідеальним першим кроком у світ класифікації.

Як оцінити якість класифікатора?

Для регресії ми використовували MAE та RMSE, бо вимірювали відстань від прогнозу до реального числа. У класифікації ми не можемо виміряти "відстань" між "котом" і "собакою". Нам потрібні інші, більш хитрі метрики.

Точність (Accuracy) – це найпростіша метрика, яка відповідає на питання: "Який відсоток відповідей моделі був правильним?".

$$\text{Accuracy} = \frac{\text{Кількість правильних прогнозів}}{\text{Загальна кількість прогнозів}}$$

Модель проаналізувала 100 імейлів. 95 з них вона класифікувала правильно (спам як спам, не спам як не спам). Її точність = 95 / 100 = 0.95 або 95%.

Проте точність може бути оманливою на незбалансованих даних. Уявіть, що ми створюємо тест на рідкісну хворобу, яка є лише в 1% людей. Якщо створити "дурну" модель, яка завжди каже "пацієнт здоровий", її точність буде 99%! Вона буде права у 99 випадках зі 100. Але при цьому вона буде абсолютно марною, бо пропустить усіх хворих.

Для глибшого аналізу нам потрібна матриця плутанини.

2. Матриця плутанини (Confusion Matrix) – це таблиця, яка детально показує, де саме наша модель мала рацію, а де "заплуталася". Розглянемо на прикладі класифікації спаму:

- True Positive (TP). Модель правильно визначила спам. (Прогноз: "Спам", Реальність: "Спам").
- True Negative (TN). Модель правильно визначила звичайний лист. (Прогноз: "Не спам", Реальність: "Не спам").
- False Positive (FP). Модель назвала спамом звичайний лист. Це дуже погано –важливий лист може потрапити у спам.
- False Negative (FN). Модель пропустила спам і назвала його звичайним листом. Це неприємно, але менш критично.

Матриця плутанини є основою для розрахунку більш просунутих метрик.

Precision та Recall –баланс між помилками

Ці дві метрики дозволяють зрозуміти характер помилок моделі.

Precision (точність прогнозу). З усіх листів, які модель назвала спамом, яка частка справді була спамом?

Формула:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Коли застосовується. Коли ціна False Positive (FP) дуже висока. Ми хочемо бути максимально впевненими у своїх "позитивних" прогнозах. Краще пропустити спам (низький Recall), ніж помилково відправити у спам важливий лист (високий Precision).

Recall (повнота або чутливість). З усіх реальних спам-листів, яку частку модель змогла знайти?

Формула:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Коли застосовується. Коли ціна False Negative (FN) дуже висока. Наприклад, у медицині при діагностиці раку. Нам життєво важливо знайти всіх, хто хворий (високий Recall), навіть якщо ми помилково відправимо кількох здорових людей на додаткове обстеження (низький Precision).

4. F1-Score –гармонійний баланс

Часто нам потрібен баланс між Precision та Recall. F1-Score –це гармонійне середнє цих двох метрик. Вона дає єдине число, яке показує загальну ефективність моделі.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-Score високий тільки тоді, коли обидві метрики (Precision та Recall) високі.

Оцінка моделі класифікації – це не просто погляд на загальну точність (Accuracy). Важливо розуміти тип помилок, які робить модель. Матриця плутанини, Precision, Recall та F1-Score дають нам глибоке розуміння сильних та слабких сторін нашого класифікатора, дозволяючи вибрати найкращу модель для конкретної задачі.

ЛЕКЦІЯ 7

Мистецтво бачити правду за цифрами: Чому 99% точності – це може бути провал, і як обрати правильний шлях.

У світі даних є одна метрика, яка одночасно є найпопулярнішою і найнебезпечнішою. Майстер ілюзій, що здатна перетворити повний провал на гучну перемогу – точність (accuracy).

Сьогодні ми станемо детективами. Наше завдання – навчитися бачити крізь туман оманливих цифр, проводити "розтин" помилок нашої моделі та обирати інструмент, що веде не просто до красивого звіту, а до реальної користі для бізнесу.

1. Ілюзія успіху: історія про Ігоря та його 99% модель

Ігор молодий дата-саєнтист, який отримав перше серйозне завдання: створити модель для виявлення шахрайства з кредитними картками. Тиждень роботи, і ось він – результат! Модель протестована на 10 000 транзакціях. Ігор запускає скрипт оцінки і бачить на екрані магічне число: Accuracy = 99%.

«Неймовірно! – думає він. – Я геній! Це ж майже ідеальний результат». Він уже готується повідомити керівництву про свій тріумф. Але щось його турбує. Він вирішує копнути глибше.

З'ясовується, що зі 10 000 транзакцій лише 100 були шахрайськими (1%). А його "геніальна" модель просто навчилася бути ледарем: на будь-яку транзакцію вона відповідала "Все ОК, це не шахрайство".

Вона правильно вгадала 9900 легальних операцій.

Але вона пропустила ВСІ 100 шахрайських.

Точність 99%, а користь для бізнесу – нуль. Модель виявилася корисною, як сонцезахисні окуляри вночі. Вона не виконала свою єдину важливу функцію – ловити злодіїв.

Золоте правило детектива даних: якщо ви бачите високу точність на незбалансованих даних – вмикайте червону сирену. Ймовірно, вас намагаються обдурити.

2. Що насправді приховує наша модель?

Щоб зрозуміти, де саме ховається брехня, нам потрібен хірургічний інструмент – Матриця помилок (Confusion Matrix). Це не просто таблиця, це чесна розмова про всі успіхи та провали нашої моделі.

Уявімо, що наша модель – це молодий лікар, який ставить діагноз: "хворий" чи "здоровий".

	Вердикт лікаря: "Хворий"	Вердикт лікаря: "Здоровий"
Пацієнт насправді Хворий	TP (True Positive) *Ідеальний діагноз!* Життя врятовано.	FN (False Negative) *Фатальна помилка!* "Йдіть додому, ви здорові". Хвороба прогресує, час втрачено.
Пацієнт насправді Здоровий	FP (False Positive) *Фальшива тривога!* "У вас серйозна хвороба". Стрес, непотрібні ліки, зайві витрати.	TN (True Negative) *Все правильно.* Здорова людина щасливо йде додому.

Кожна клітинка в цій матриці має свою ціну, і ця ціна вимірюється не тільки в грошах, а й у людських долях.

- Ціна FN (пропустити хворого): може коштувати життя.
- Ціна FP **(**поставити хибний діагноз): коштує нервів, часу та грошей.

Тепер очевидно, що ці помилки нерівноцінні. Тому нам потрібні метрики, які це враховують.

3. Precision vs. Recall: Кого слухати – перфекціоніста чи рятівника?

Уявіть, що у вас є два радники, які оцінюють роботу вашого "лікаря"-моделі.

Радник №1: Precision (Перфекціоніст)

Його девіз: "Краще промовчати, ніж сказати дурницю!"

Чого він боїться найбільше: звинуватити невинного (FP).

На що він дивиться:

$$\text{Precision} = \frac{TP}{TP + FP}$$

(З усіх, кого ми назвали хворими, скільки *дійсно* були хворими?)

Коли його слухати? Коли ціна фальшивої тривоги захмарна.

Приклад: спам-фільтр. Для Перфекціоніста відправити лист із пропозицією роботи вашої мрії у спам (FP) – це професійне самогубство. Він краще пропустить 10 рекламних листів, аніж заблокує один важливий.

Радник №2: Recall (Рятівник)

Його девіз: "Краще перестрахуватися, ніж потім шкодувати!"

Чого він боїться найбільше: пропустити когось у біді (FN).

На що він дивиться:

$$\text{Recall} = \frac{TP}{TP + FN}$$

(З усіх, хто *реально* був хворим, скількох нам вдалося знайти?)

Коли його слухати? Коли ціна пропуску є катастрофічною.

Приклад: діагностика раку. Рятівник готовий підняти на вуха всю лікарню через найменшу підозру. Для нього відправити 10 здорових людей на повторний аналіз – це прийнятна ціна за те, щоб не пропустити жодного хворого.

Ці двоє ніколи не домовляться. Якщо ви змусите модель бути обережнішою, щоб підвищити Precision, вона почне пропускати сумнівні випадки, і впаде Recall. Якщо ви накажете їй хапати всіх підозрілих, щоб підвищити Recall, вона нахапає купу невинних, і впаде Precision.

F1-score (Дипломат)

Коли вам потрібен баланс, на сцену виходить третій радник – F1-score. Він – досвідчений дипломат.

Його мета: знайти компроміс, який задовольнить і Перфекціоніста, і Рятівника.

Його інструмент:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-score – це не просте середнє, а гармонійне середнє. Його фокус у тому, що він не любить крайнощів. Якщо один із показників (Precision чи Recall) буде дуже низьким, F1-score теж буде низьким. Він змушує обох радників працювати добре, а не когось одного.

4. Ваш компас у світі метрик: Як зробити правильний вибір?

Отже, як не загубитися? Вибір метрики – це ваш стратегічний компас. І його стрілка має вказувати не на "найвищий показник", а на головну мету бізнесу.

Перед початком роботи поставте собі ці три питання:

1. У чому полягає перемога? (Знайти всіх шахраїв? Не турбувати чесних клієнтів?)
2. Що є "позитивним" результатом? (Виявлення хвороби, знайдений спам, факт шахрайства).
3. Яка помилка страшніша?
   - `Страшніше звинуватити невинного (FP)? → Ваш вибір – PRECISION.` *(Спам-фільтри, дуже таргетована реклама)*.
   - `Страшніше пропустити винного (FN)? → Ваш вибір – RECALL.` *(Медицина, виявлення дефектів, антифрод-системи)*.

- Обидві помилки однаково погані? → Ваш вибір – F1-SCORE. *(Більшість збалансованих завдань, де потрібен компроміс)*.

ЛЕКЦІЯ 8

Інтуїтивне пояснення роботи дерев рішень. Поняття "ентропії" та "приросту інформації". Переваги та недоліки.

Настав час для глибокого занурення у світ дерев рішень. Це один із небагатьох алгоритмів машинного навчання, логіку якого можна не просто зрозуміти, а буквально побачити. Уявіть, що ви лікар, який намагається діагностувати хворобу. Ви не робите всі аналізи одразу, а ставите послідовні запитання: "Чи є у пацієнта температура?", "Якщо так, чи є кашель?", "Якщо кашлю немає, чи є висип?". Кожне таке запитання звужує коло можливих діагнозів. Саме за таким принципом і працює дерево рішень.

Дерево рішень – це непараметрична керована модель машинного навчання, яка використовується для задач класифікації та регресії. Модель прогнозує значення цільової змінної, вивчаючи прості правила прийняття рішень, виведені з характеристик даних.

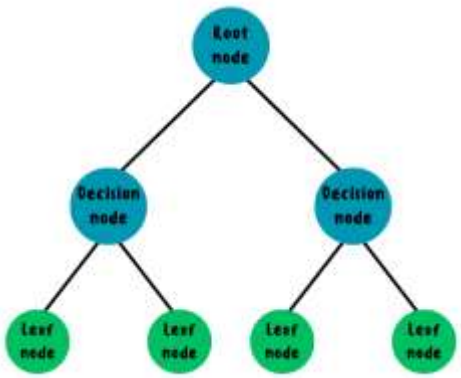
Давайте розберемо його структуру на прикладі бізнесу: прогнозування, чи поверне клієнт банку кредит.

Корінь дерева (Root Node) – це початкова точка, яка містить усі ваші дані. Наприклад, 1000 клієнтів, з яких 700 повернули кредит, а 300 – ні. Наше завдання – знайти ознаку, яка найкраще розділить цих клієнтів.

Вузол рішення (Decision Node) – це точка, де дані розбиваються на підгрупи. Кожен вузол представляє певне запитання. Наприклад, "Чи є у клієнта стабільний дохід > 15 000 грн?".

Гілки (Branches) – це шляхи, що ведуть від вузла рішення, і представляють відповіді на запитання ("Так" або "Ні").

Листки (Leaf Nodes / Terminal Nodes) – це кінцеві вузли, які більше не розбиваються. Вони містять кінцевий прогноз. Наприклад, листок "Повернув кредит" (якщо в цій групі 95% клієнтів повернули) або "Не повернув кредит".



Процес побудови дерева називається рекурсивним розбиттям. Алгоритм починає з кореня і послідовно розбиває дані на менші й менші підгрупи, на кожному кроці обираючи те запитання (ту ознаку), яке робить ці підгрупи якомога "чистішими". Але що таке "чистота" з погляду математики?

Ентропія: Від хаосу до порядку

В теорії інформації ентропія – це міра невизначеності або непередбачуваності в наборі даних. Формула для ентропії виглядає так:

$$E(S) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

Де:

- S – набір даних.
- c – кількість класів (у нашому випадку 2: "повернув" і "не повернув").
- p_i – імовірність (частка) елементів класу i в наборі S.

Максимальна ентропія (максимальний хаос): Уявімо, що в нашій групі з 100 клієнтів 50 повернули кредит і 50 ні. Імовірність кожного результату p_1=0.5 та p_2=0.5. Ентропія буде дорівнювати 1. Це повна невизначеність.

Нульова ентропія (повний порядок): Тепер уявіть, що ми виділили групу зі 100 клієнтів, і всі 100 повернули кредит. Імовірність $p_1=1$, а $p_2=0$. Ентропія буде дорівнювати 0. Невизначеності немає, результат очевидний.

Головна мета алгоритму – на кожному кроці розбиття максимально зменшити ентропію. Він хоче перейти від змішаних груп до чистих, де більшість об'єктів належить до одного класу.

Приріст інформації: Пошук "золотого" запитання

Отже, як алгоритм вибирає найкраще запитання з-поміж десятків можливих ("кредитна історія", "вік", "дохід", "наявність роботи")? Він використовує показник, який називається приріст інформації.

Приріст інформації – це різниця між ентропією *до* розбиття і середньозваженою ентропією *після* розбиття.

$$\text{Information Gain}(S, A) = E(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} E(S_v)$$

Своїми словами, це працює так:

1. Порахувати початкову ентропію. Розраховуємо ентропію для всього набору даних (нашого кореневого вузла).
2. Перебрати всі можливі запитання. Для кожної ознаки (наприклад, "Кредитна історія") алгоритм "вдає", що він робить розбиття.
3. Розрахувати ентропію для нащадків. Він рахує ентропію для кожної новоутвореної групи (наприклад, група з "хорошою" історією і група з "поганою").
4. Розрахувати середньозважену ентропію. Він знаходить середнє значення ентропії після розбиття, враховуючи розмір кожної нової групи.
5. Обчислити приріст інформації: Віднімає середньозважену ентропію (крок 4) від початкової (крок 1).

Він обирає те запитання (ту ознаку), яке дає найвищий показник приросту інформації. Це означає, що саме це запитання найкраще

"очистило" дані та зменшило невизначеність. І цей процес повторюється для кожного нового вузла, поки не буде досягнуто критерію зупинки (наприклад, усі елементи в групі одного класу або досягнуто максимальної глибини дерева).

Переваги та недоліки дерев рішень

Переваги

Виняткова інтерпретованість: Це головна причина їх популярності. У критично важливих сферах, як-от медицина (чому пацієнту призначено саме ці ліки?) або банківська справа (чому клієнту відмовили у кредиті?), регулятори вимагають прозорих моделей. Дерево рішень надає чіткі, зрозумілі правила, які можна пояснити будь-кому.

Мінімальна підготовка даних: Алгоритму байдуже на масштабування чи нормалізацію даних. Він працює з даними "як є".

Вбудований відбір ознак: Оскільки дерево завжди обирає ознаки з найбільшим приростом інформації, менш важливі ознаки просто не будуть використані для розбиття, що є формою автоматичного відбору найвпливовіших факторів.

Обробка різних типів даних: Легко працює з числовими (вік, дохід) та категоріальними (стать, місто) змінними одночасно.

Недоліки

Схильність до перенавчання (Overfitting): Це "Ахіллесова п'ята" дерев рішень. Якщо не обмежувати ріст дерева, воно стане надзвичайно складним і глибоким. Воно ідеально "запам'ятає" всі особливості та навіть шум у навчальних даних.

- Компроміс між зміщенням та дисперсією (Bias-Variance Tradeoff): Просте, неглибоке дерево має високе зміщення (bias) – воно може не вловити всі закономірності в даних. Дуже глибоке дерево має високу

дисперсію (variance) – воно надто чутливе до навчальних даних і погано узагальнює на нових.

- Методи боротьби:
    - Обрізка (Pruning): Спочатку будується повне дерево, а потім з нього видаляються гілки, які дають мало приросту інформації.
    - Встановлення максимальної глибини: Заборонити дереву рости глибше певного рівня.
    - Мінімальна кількість зразків у вузлі: Не дозволяти розбиття вузла, якщо в ньому менше певної кількості об'єктів.

Нестабільність: Незначні зміни в даних (навіть кілька рядків) можуть кардинально змінити структуру всього дерева. Цю проблему вирішують ансамблеві методи (наприклад, Випадковий ліс), які будують сотні різних дерев і усереднюють їхні прогнози.

Дерева рішень є одним із фундаментальних алгоритмів у машинному навчанні. Вони імітують людський процес прийняття рішень, розбиваючи складну проблему на низку простих запитань. Використовуючи ентропію для вимірювання хаосу та приріст інформації для вибору найкращого запитання, вони будують прозору та зрозумілу модель. Попри схильність до перенавчання, їхня інтерпретованість робить їх незамінним інструментом у багатьох галузях.

ЛЕКЦІЯ 9

"Мудрість натовпу": ідея ансамблів. Беггінг та випадкові підпростори ознак. Random Forest

Мудрість натовпу

Уявіть, що вам потрібно вгадати кількість цукерок у великій банці. Якщо ви запитаєте одну людину, її відповідь, ймовірно, буде далекою від істини. Але якщо ви запитаєте сотню людей і усередните їхні відповіді, результат, найімовірніше, буде напрочуд близьким до правильного. Це і є "мудрість натовпу": колективна думка групи людей часто є точнішою, ніж думка одного, навіть найрозумнішого, експерта.

У машинному навчанні цей принцип лежить в основі ансамблевих методів. Замість того, щоб покладатися на одну, нехай і складну, модель, ми створюємо "натовп" з багатьох простіших моделей і об'єднуємо їхні "думки" для отримання значно точнішого та надійнішого результату.

Ансамбль (Ensemble) – це метод машинного навчання, який комбінує прогнози кількох окремих моделей (їх називають базовими або слабкими учнями) для створення однієї сильної моделі.

Основна ідея полягає в тому, що окремі моделі можуть робити різні помилки. Коли ми об'єднуємо їхні прогнози, ці помилки взаємно компенсуються. Головна умова успіху ансамблю – різноманітність моделей. Якщо всі моделі однакові й помиляються однаково, ніякого покращення не буде.

Двома ключовими техніками для створення різноманітних моделей є беггінг та метод випадкових підпросторів.

Беггінг (від Bootstrap aggregating) – це техніка, яка створює різноманітність моделей, навчаючи їх на різних підвибірках вихідних даних.

Як це працює?

1. Bootstrap (Створення підвибірок): з оригінального набору даних (розміром N) ми створюємо багато (наприклад, 100) нових наборів даних такого ж розміру N. Це робиться шляхом випадкового вибору з поверненням. Це означає, що деякі об'єкти з початкового набору можуть потрапити в одну підвибірку кілька разів, а інші – не потрапити зовсім.
2. Навчання моделей: На кожній з цих унікальних підвибірок ми навчаємо окрему базову модель (наприклад, дерево рішень). Оскільки кожна модель бачила трохи інший набір даних, вони виходять різними.
3. Aggregating (Об'єднання прогнозів):
    - Для задач класифікації кінцевий результат визначається голосуванням більшості. Кожна модель "голосує" за свій клас, і перемагає той клас, що набрав найбільше голосів.
    - Для задач регресії кінцевий результат – це усереднення прогнозів усіх моделей.

Головна перевага беггінгу – він ефективно зменшує дисперсію (variance), тобто схильність моделі до перенавчання. Модель стає стабільнішою і краще узагальнює дані, які не бачила раніше.

Метод випадкових підпросторів ознак (Random Subspaces)

Ця техніка додає ще один рівень випадковості та різноманітності. Замість того, щоб давати кожній моделі доступ до всіх ознак (стовпців) у даних, ми дозволяємо їй використовувати лише їх випадкову підмножину.

Наприклад, якщо у нас є 100 ознак, одна модель може будувати свої прогнози, використовуючи лише випадково обрані 10 ознак, інша – іншу випадкову десятку, і так далі.

Навіщо це потрібно? Якщо в даних є одна або дві дуже сильні ознаки, більшість базових моделей (особливо дерева рішень) будуть покладатися саме на них. Це зробить моделі дуже схожими одна на одну, і "мудрість

натовпу" не спрацює. Метод випадкових підпросторів змушує моделі шукати менш очевидні, але все ж корисні патерни в різних наборах ознак, що значно підвищує їхню різноманітність і, як наслідок, точність ансамблю.

Випадковий ліс (Random Forest) – це один із найпотужніших та найпопулярніших алгоритмів машинного навчання, який є ідеальним поєднанням ідей дерев рішень, беггінгу та методу випадкових підпросторів.

Базовий учень у Random Forest – це завжди дерево рішень.

Як працює Random Forest

1. Створюється "ліс" з N дерев рішень.
2. Для кожного дерева виконується беггінг: створюється своя унікальна підвибірка даних (bootstrap sample).
3. Кожне дерево навчається на своїй підвибірці. Але є ключова відмінність: при побудові дерева, на кожному кроці (в кожному вузлі), коли потрібно обрати найкращу ознаку для розділення, дерево розглядає не всі доступні ознаки, а лише їх випадкову підмножину (метод випадкових підпросторів).
4. Дерева в лісі ростуть до максимальної глибини без "обрізки" (pruning), що робить кожне окреме дерево схильним до перенавчання, але в ансамблі це не є проблемою.
5. Фінальний прогноз отримується шляхом голосування (для класифікації) або усереднення (для регресії) результатів усіх дерев у лісі.

Чому Random Forest такий популярний

Висока точність. Це один з найточніших "готових" алгоритмів. Він часто показує чудові результати без тривалого налаштування гіперпараметрів.

Стійкість до перенавчання. Завдяки беггінгу та випадковості у виборі ознак, ансамбль в цілому є дуже стійким до перенавчання, навіть якщо окремі дерева є перенавченими.

Ефективність. Добре працює на великих наборах даних і з великою кількістю ознак.

Оцінка важливості ознак. Алгоритм дозволяє оцінити, які ознаки зробили найбільший внесок у точність моделі. Це дуже корисно для аналізу даних та відбору найважливіших змінних.

Простота у використанні. Потребує мінімальної попередньої обробки даних (наприклад, не вимагає масштабування ознак).

Random Forest – це яскравий приклад того, як об'єднання багатьох простих і "неідеальних" моделей, кожна з яких дивиться на дані під своїм унікальним кутом, створює надзвичайно потужний, точний і надійний інструмент для вирішення складних задач машинного навчання.

ЛЕКЦІЯ 10

Вступ до навчання без учителя (Unsupervised Learning)

На відміну від навчання з учителем, де ми мали "правильні відповіді" (мітки) для наших даних, тут ми працюємо з нерозміченими даними. Головна мета – знайти приховані закономірності, структури та зв'язки без будь-яких попередніх підказок.

Уявіть, що вам дали величезну коробку з різними фруктами, але ніхто не сказав, де яблука, де апельсини, а де банани. Ваше завдання – розсортувати їх за схожими ознаками: кольором, формою, розміром. Це і є суть навчання без учителя.

Основні задачі, які вирішує цей підхід:

- Кластеризація (Clustering): групування схожих об'єктів.
- Зменшення розмірності (Dimensionality Reduction): стиснення даних без втрати важливої інформації.
- Пошук аномалій (Anomaly Detection): виявлення нетипових об'єктів.

Поняття кластеризації

Кластеризація – це процес розбиття набору даних на групи, які називаються кластерами. Основний принцип полягає в тому, щоб об'єкти всередині одного кластера були максимально схожими один на одного, а об'єкти з різних кластерів – максимально відмінними.

Схожість або відмінність між об'єктами вимірюється за допомогою метрики відстані, найпопулярнішою з яких є Евклідова відстань. Чим менша відстань між двома точками, тим вони більш схожі.

Кластеризація застосовується в багатьох сферах:

- Маркетинг: сегментація клієнтів за купівельною поведінкою для персоналізованих пропозицій.
- Біологія: класифікація видів рослин або тварин на основі їхніх характеристик.

- Аналіз документів: групування новинних статей за темами.
- Комп'ютерний зір: сегментація зображень для розпізнавання об'єктів.
- Алгоритм K-Means: серце кластеризації

Одним із найпопулярніших і найпростіших алгоритмів кластеризації є K-Means (К-середніх). Його мета – розділити n об'єктів на K кластерів, де K – це наперед задане число. Алгоритм працює ітеративно і складається з трьох основних кроків.

Крок 1: Ініціалізація центроїдів

Спочатку ми повинні визначити, скільки кластерів (K) ми хочемо отримати. Потім ми випадковим чином обираємо K точок з нашого набору даних. Ці точки стають початковими центроїдами – уявними центрами наших майбутніх кластерів.

Проблема: Невдалий вибір початкових центроїдів може призвести до поганих результатів.

Рішення: Існують розумніші методи ініціалізації, як-от k-means++, що намагається обрати початкові центроїди якнайдалі один від одного.

Крок 2: Присвоєння (Assignment Step)

На цьому етапі ми проходимо по кожній точці даних і вимірюємо відстань від неї до кожного з K центроїдів. Кожна точка присвоюється до найближчого центроїда. В результаті всі дані розподіляються по K групах.

Крок 3: Оновлення (Update Step)

Тепер, коли ми маємо нові кластери, ми перераховуємо положення центроїдів. Новий центроїд – це центр мас (середнє арифметичне) всіх точок, що належать до його кластера.

Ітераційний процес: Кроки 2 і 3 повторюються доти, доки центроїди не перестануть суттєво змінювати своє положення. Це означає, що алгоритм знайшов стабільне рішення, і кластери сформовані.

Проблема вибору кількості кластерів "K"

Найбільший виклик у використанні K-Means – це вибір правильної кількості кластерів K. Якщо обрати занадто мале K, ми можемо об'єднати в одну групу різні за своєю природою дані. Якщо занадто велике – розділити одну логічну групу на кілька дрібних.

Як же знайти оптимальне K?

Метод ліктя (Elbow Method). Ми запускаємо K-Means для різної кількості кластерів (наприклад, від 1 до 10) і для кожного K обчислюємо суму квадратів відстаней від кожної точки до центроїда її кластера (цей показник називається WCSS – Within-Cluster Sum of Squares).

Потім ми будуємо графік, де по осі X відкладено кількість кластерів K, а по осі Y – відповідне значення WCSS.

Зі збільшенням K WCSS завжди буде зменшуватися. Нас цікавить точка, де графік утворює "лікоть" – після неї зменшення WCSS стає незначним. Ця точка і є кандидатом на оптимальне значення K.

Коефіцієнт силуету (Silhouette Coefficient). Цей метод оцінює, наскільки добре кожна точка "вписується" у свій кластер порівняно з іншими кластерами. Значення коефіцієнта лежить в діапазоні від -1 до 1.

Значення, близькі до 1, означають, що об'єкт добре кластеризований.

Значення, близькі до 0, вказують на те, що об'єкт знаходиться на межі між двома кластерами.

Значення, близькі до -1, свідчать, що об'єкт, ймовірно, потрапив не в той кластер.

Ми можемо обчислити середній коефіцієнт силуету для різних значень K і обрати те, для якого цей показник є максимальним.

Навчання без учителя – це потужний інструмент для аналізу даних, коли ми не маємо готових відповідей. Алгоритм K-Means є чудовою відправною точкою для задач кластеризації, але завжди варто пам'ятати про важливість правильного вибору кількості кластерів K, використовуючи для цього спеціальні методи оцінки.

ЛЕКЦІЯ 11

Тема:Прокляття розмірності та PCA

Уявімо, що ми хочемо описати квартиру. Одна ознака – це її площа (1D простір). Дві ознаки – площа та кількість кімнат (2D). Десять ознак – площа, кількість кімнат, відстань до метро, вік будинку, якість ремонту тощо (10D). Здається, що більше ознак, то кращий опис. Але тут і починаються проблеми.

1. Експоненційне зростання "порожнього" простору

Давайте проведемо уявний експеримент.

1D (лінія): У вас є відрізок довжиною 1 метр. Ви кидаєте на нього 10 камінців. Вони лежать досить щільно.

2D (квадрат): Тепер у вас квадрат 1x1 метр. Ті ж 10 камінців розкидані по значно більшій площі. Відстань між ними в середньому зросла.

3D (куб): Тепер це куб 1x1x1 метр. 10 камінців у величезному просторі виглядають дуже самотньо. Вони стали ще далі один від одного.

А тепер уявіть 100-вимірний "гіперкуб". Наші дані в ньому будуть настільки рідкісними, що практично весь простір буде порожнім. Це означає, що для будь-якої точки даних її сусіди будуть знаходитись на величезній відстані.

Наслідок: Моделі машинного навчання, які покладаються на локальну близькість (як-от k-NN, кластеризація), починають працювати погано. Якщо всі точки однаково далекі одна від одної, як знайти "найближчих" сусідів?

2. Концентрація відстаней

Це ще один дивний ефект багатовимірного простору. Уявіть, що ви стоїте в центрі 100-вимірної кімнати. Виявляється, що відстань до найближчої стіни і до найдальшого кута буде майже однаковою.

У багатовимірному просторі відносна різниця між відстанню до найближчої та найдальшої точки для будь-якого спостереження прямує до нуля. Усі точки стають "однаково далекими". Це робить поняття відстані майже безглуздим для порівняння.

3. Перенавчання (Overfitting) – Аналогія зі студентом

Уявіть двох студентів, які готуються до іспиту.

Студент А (Проста модель, мало "ознак"): намагається зрозуміти основні принципи та формули. Він не запам'ятовує кожну задачу з підручника, а вчить методи їх розв'язання. На іспиті він може розв'язати навіть ті задачі, яких ніколи не бачив. Це узагальнення.

Студент Б (Складна модель, багато "ознак"): зазубрює відповіді на всі 500 задач з підручника. Він не розуміє логіки. Якщо на іспиті йому дадуть одну з цих 500 задач, він дасть ідеальну відповідь. Але якщо задача буде хоч трохи змінена, він не зможе її розв'язати. Це перенавчання.

Коли модель має занадто багато ознак (вимірів), вона стає схожою на студента Б. Вона має стільки "гнучкості", що може просто "прокласти" ідеальну межу між класами в тренувальних даних, ігноруючи загальну тенденцію. Вона запам'ятовує шум і випадковості, а не закономірність.

Метод головних компонент (PCA) – інтуїція через аналогії

PCA – це не просто видалення стовпців з даними. Це розумний спосіб стиснути інформацію, створюючи нові, змістовніші ознаки.

Тінь від літака

Уявіть, що ви тримаєте в руках модель літака (це ваші багатовимірні дані). Ваша мета – отримати найкраще 2D зображення (тінь) цього 3D об'єкта на стіні.

Ви світите ліхтариком зверху. Тінь буде схожа на хрест (фюзеляж і крила). Ви втратили багато інформації про форму кабіни, хвоста.

Ви світите ліхтариком збоку. Тінь покаже профіль літака: ніс, кабіну, хвіст. Це вже набагато інформативніше.

Ви світите ліхтариком під певним кутом, і отримуєте тінь, яка показує і довжину фюзеляжу, і розмах крил. Ця тінь має найбільшу площу (найбільший "розкид", або варіативність) і найкраще передає суть форми 3D літака.

PCA – це алгоритм, який автоматично знаходить цей "найкращий кут" для проєкції, щоб тінь (дані в меншій розмірності) зберігала максимум інформації про вихідний об'єкт.

Перша головна компонента (PC1) – це напрямок, уздовж якого тінь є найдовшою (максимальна варіативність).

Друга головна компонента (PC2) – це наступний найкращий напрямок, перпендикулярний до першого.

Оцінка ресторанів

Уявіть, що ви аналізуєте дані про ресторани. У вас є 20 ознак:

- ціна_супу, ціна_стейку, ціна_десерту
- рейтинг_чистоти, рейтинг_інтерєру, рейтинг_вбиральні
- швидкість_обслуговування, привітність_офіціанта
- ... і так далі.

Багато з цих ознак корелюють. Ціни на страви пов'язані між собою. Рейтинги чистоти також. PCA може це виявити і створити нові, узагальнені ознаки:

Створення PC1: Алгоритм бачить, що ресторани, де дорогі супи, зазвичай мають дорогі стейки та десерти. Він об'єднує всі ці цінові ознаки в

одну нову ознаку, яку ми можемо назвати "Рівень цін". Це і є перша головна компонента. Вона пояснює найбільшу частину відмінностей між ресторанами.

Створення PC2: Далі PCA шукає наступну за важливістю групу корелюючих ознак. Наприклад, швидкість_обслуговування та привітність_офіціанта. Він об'єднує їх у другу компоненту – "Якість сервісу". Ця вісь буде перпендикулярна до осі "Рівень цін" (тобто, дорогий ресторан не обов'язково має хороший сервіс, і навпаки).

Створення PC3: Наступною може бути компонента "Атмосфера", що об'єднує рейтинг_інтерєру, чистоту тощо.

Результат: Замість 20 вихідних ознак ми тепер можемо описати кожен ресторан лише трьома новими, більш змістовними ознаками: "Рівень цін", "Якість сервісу" та "Атмосфера". Ми значно зменшили розмірність (з 20 до 3), втративши мінімум важливої інформації, і отримали ознаки, які легше інтерпретувати. Ми перейшли від сирих даних до сутностей.

ЛЕКЦІЯ 12

Історія аналізу ринкового кошика

У світі аналізу даних існує одна відома історія, яка чудово ілюструє силу аналізу ринкового кошика. Легенда свідчить, що у 1990-х роках аналітики однієї з великих мереж супермаркетів виявили дивну закономірність. По п'ятницях чоловіки, які купували підгузки, часто купували й пиво.

Здивовані? Пояснення виявилося простим. Молоді батьки, яких дружини відправляли до магазину за підгузками на вихідні, вирішували винагородити себе кількома пляшками пива. Виявивши цей зв'язок, магазин розмістив пиво поруч із підгузками. Результат? Продажі обох товарів значно зросли.

Ця історія є ідеальним прикладом аналізу ринкового кошика (Market Basket Analysis) – методу, що дозволяє знаходити взаємозв'язки між товарами у великих наборах даних про покупки. Мета цього аналізу – виявити так звані асоціативні правила (наприклад, "Якщо покупець купує підгузки, він також, імовірно, купить пиво").

Щоб виміряти силу та значущість цих асоціативних правил, аналітики використовують три ключові метрики: Support (Підтримка), Confidence (Впевненість) та Lift (Підйом).

Support (Підтримка) – це показник популярності товару або набору товарів. Вона показує, як часто певний товар (або комбінація товарів) зустрічається у всіх транзакціях.

Формула:

$$\text{Support}(A) = \frac{\text{Кількість транзакцій, що містять товар A}}{\text{Загальна кількість транзакцій}}$$

Для набору товарів {A, B}:

$$\text{Support}(A \cup B) = \frac{\text{Кількість чеків, де є і A, і B}}{\text{Загальна кількість чеків}}$$

Простими словами: Якщо у нас 1000 чеків, і в 100 з них є підгузки, то підтримка для підгузків становить 100/1000 = 10%. Якщо у 50 з цих чеків є і підгузки, і пиво, то підтримка для набору {підгузки, пиво} становить 50/1000 = 5%.

Confidence (Впевненість) – це умовна ймовірність покупки. Вона показує, наскільки часто товар B купують, якщо вже купили товар A. Це основний показник надійності правила "Якщо A, то B".

Формула:

$$\text{Confidence}(A \Rightarrow B) = \frac{\text{Support}(A \cup B)}{\text{Support}(A)}$$

Простими словами: Продовжуючи наш приклад: у 100 чеках є підгузки, і в 50 з них є також пиво. Впевненість правила "Якщо підгузки, то пиво" становить 50/100 = 50%. Це означає, що у 50% випадків, коли купують підгузки, у кошику також опиняється пиво.

Lift (Підйом) – це найцікавіша метрика. Вона показує, наскільки зростає ймовірність покупки товару B, якщо ми знаємо, що товар A вже куплений, порівняно з базовою ймовірністю покупки товару B.

Формула:

$$\text{Lift}(A \Rightarrow B) = \frac{\text{Confidence}(A \Rightarrow B)}{\text{Support}(B)}$$

Інтерпретація:

Lift > 1: товари A і B позитивно корелюють. Покупка A збільшує ймовірність покупки B. Наш випадок з пивом та підгузками.

Lift = 1: товари A і B незалежні. Покупка A ніяк не впливає на покупку B.

Lift < 1: товари A і B негативно корелюють. Покупка A зменшує ймовірність покупки B (наприклад, товари-замінники).

Припустимо, пиво загалом купують у 20% чеків (Support(Пиво) = 0.2). Наша впевненість правила "Якщо підгузки, то пиво" дорівнює 50% (Confidence = 0.5). Тоді Lift = 0.5 / 0.2 = 2.5. Це означає, що людина, яка купує підгузки, у 2.5 рази більш схильна купити пиво, ніж середньостатистичний покупець. Це сильний зв'язок!

Алгоритм Apriori

Як знайти всі ці цікаві правила у мільйонах чеків? Перебирати всі можливі комбінації товарів – це надто довго. Тут на допомогу приходить алгоритм Apriori.

Apriori – це класичний алгоритм для пошуку наборів товарів, що часто зустрічаються (frequent itemsets), у базі даних транзакцій. Його основний принцип базується на так званій "властивості Apriori":

Якщо набір товарів зустрічається часто, то всі його підмножини також повинні зустрічатися часто. І навпаки: якщо якась підмножина товарів зустрічається рідко, то будь-який більший набір, що її містить, також буде рідкісним.

Це дозволяє "відсікати" величезну кількість непопулярних комбінацій на ранніх етапах, значно прискорюючи процес.

Кроки алгоритму Apriori:

Встановити мінімальну підтримку (min_support). Це поріг, який ми вважаємо "частою" покупкою. Наприклад, min_support = 2% (тобто нас цікавлять комбінації товарів, які зустрічаються хоча б у 2% чеків).

Крок 1: Пошук частих наборів з 1-го елемента.

Скануємо всі транзакції та рахуємо підтримку для кожного окремого товару.

Відкидаємо всі товари, чия підтримка менша за min_support. Отримуємо список частих товарів (L1).

Крок 2: Пошук частих наборів з 2-х елементів.

Генеруємо кандидатів у часті пари (C2), комбінуючи елементи з L1.

Скануємо базу даних і рахуємо підтримку для кожної пари з C2.

Відкидаємо пари, чия підтримка менша за min_support. Отримуємо список частих пар (L2).

Повторення процесу.

Продовжуємо генерувати кандидатів з (k+1) елементів на основі частих наборів з k елементів (L_k), рахувати їхню підтримку і відфільтровувати за min_support.

На кожному кроці використовуємо властивість Apriori: якщо якась k-елементна підмножина кандидата не входить до L_k, ми одразу відкидаємо цього кандидата, не рахуючи його підтримку.

Зупинка. Алгоритм зупиняється, коли більше неможливо згенерувати нових частих наборів.

Генерація асоціативних правил.

З отриманих частих наборів (наприклад, {хліб, молоко, масло}) ми генеруємо правила (напр., "Якщо {хліб, молоко}, то {масло}").

Для кожного правила розраховуємо Confidence і Lift та відбираємо ті, які задовольняють заданим порогам (наприклад, min_confidence > 50%).

Аналіз ринкового кошика – це потужний інструмент, що дозволяє бізнесу краще розуміти поведінку клієнтів. Метрики Support, Confidence та Lift дають змогу кількісно оцінити знайдені закономірності, а алгоритм Apriori є ефективним методом для їх виявлення у великих масивах даних. Знаючи, які товари купують разом, магазини можуть оптимізувати розкладку товарів, створювати ефективні акційні пропозиції та персоналізувати рекомендації, що в кінцевому підсумку веде до збільшення продажів і лояльності клієнтів.

ЛЕКЦІЯ 13

Специфіка даних, що залежать від часу. Поняття тренду, сезонності та залишків. Прості методи прогнозування: ковзне середнє.

1. Унікальна природа даних, що залежать від часу

На відміну від стандартних наборів даних, де ми можемо перемішувати рядки без втрати інформації (наприклад, анкета з опитуванням клієнтів), у часових рядах порядок є найважливішим елементом. Кожне спостереження нерозривно пов'язане з попереднім та наступним. Ця властивість називається темпоральною залежністю або автокореляцією.

Автокореляція – це, по суті, міра того, наскільки значення ряду в певний момент часу пов'язане зі значеннями в попередні моменти.

Приклад: температура повітря сьогодні о 14:00 сильно залежить від температури о 13:00, і значно менше – від температури тиждень тому.

Саме ця залежність дозволяє нам будувати прогнози. Ми вивчаємо патерни з минулого, щоб передбачити майбутнє. Основна мета аналізу часових рядів – не просто описати дані, а й зрозуміти їхню внутрішню структуру для подальшого прогнозування.

Ключові завдання аналізу часових рядів:

1. опис: виявлення основних властивостей ряду (тренд, сезонність);
2. пояснення: побудова моделі, яка описує поведінку даних;
3. прогнозування: передбачення майбутніх значень на основі побудованої моделі;
4. управління: використання прогнозів для прийняття рішень (наприклад, оптимізація запасів на складі на основі прогнозу продажів).

2. Декомпозиція часового ряду: Розбираємо дані на складові

Декомпозиція – це процес розкладання часового ряду на його фундаментальні компоненти. Це дозволяє нам краще зрозуміти фактори, що впливають на дані, та побудувати точніші моделі.

Тренд – це не просто "напрямок", а довгострокова, повільна еволюція середнього рівня ряду. Він відображає фундаментальні зміни в системі, яку ми аналізуємо.

Типи трендів:

- лінійний: ряд зростає або спадає з постійною швидкістю. (наприклад, стабільне зростання обсягу продажів на 100 одиниць щороку);
- експоненційний: ряд зростає або спадає все швидше з часом. (наприклад, поширення вірусу або зростання популярності нової технології);
- поліноміальний: тренд має більш складну форму, зі зміною напрямку.

Причини виникнення трендів:

- технологічні зміни: поява інтернету спричинила спадний тренд у продажах друкованих газет;
- демографічні фактори: зростання населення міста створює висхідний тренд цін на нерухомість;
- економічні політики: зміни в податковому законодавстві можуть створити тренд у споживчих витратах.

Сезонність – це патерн, який повторюється через фіксований та відомий проміжок часу.

Ключова відмінність від циклічності полягає у тому, що сезонність має фіксовану і передбачувану тривалість (24 години, 7 днів, 12 місяців). Ми знаємо, коли очікувати наступний пік. Наприклад: продажі сонцезахисного крему щороку досягають піку в липні. Циклічність має змінну і непередбачувану тривалість. Приклад: економічні цикли рецесії та зростання, які можуть тривати від 2 до 10 років без чіткої закономірності.

Залишки – це те, що залишається після того, як ми "витягли" з даних тренд і сезонність. Це непередбачувана, стохастична (випадкова) частина.

Аналіз залишків є критично важливим етапом. Якщо наша модель хороша, залишки повинні нагадувати "білий шум":

- середнє значення близьке до нуля.
- відсутня автокореляція (вони не залежать один від одного).
- дисперсія постійна (немає періодів високої та низької волатильності).

Якщо в залишках простежується якась структура (наприклад, вони систематично позитивні в певні періоди), це означає, що наша модель неповна і не врахувала якийсь важливий патерн.

Моделі декомпозиції

Адитивна модель: $Y_t=T_t+S_t+R_t$

- Коли амплітуда сезонних коливань є приблизно однаковою незалежно від рівня тренду.
- Щомісячна кількість опадів. Влітку опадів завжди приблизно на 50 мм більше, ніж взимку, незалежно від того, чи був рік загалом посушливим чи дощовим.

Мультиплікативна модель: $Y_t=T_t\times S_t\times R_t$

- Коли амплітуда сезонних коливань зростає або зменшується разом із трендом.
- Квартальний дохід авіакомпанії. У святковий четвертий квартал дохід завжди приблизно на 20% вищий, ніж у середньому. Якщо загальний дохід компанії зростає з року в рік, то абсолютне значення цих 20% також буде зростати.

3. Прості методи прогнозування: Метод ковзного середнього (Moving Average, MA)

Це базовий метод, який часто використовується не стільки для точного прогнозування, скільки для **згладжування** ряду, щоб краще побачити тренд.

Просте ковзне середнє (Simple Moving Average, SMA). Кожен новий прогноз – це просто середнє значення k останніх спостережень. Метод "забуває" все, що було до цього "вікна".

Вибір розміру вікна (k) – це компроміс

Маленьке k (напр., k=3):

- Перевага: прогноз швидко реагує на зміни.
- Недолік: дуже чутливий до випадкових викидів та шуму. Згладжування слабке.

Велике k (напр., k=12):

- Перевага: сильно згладжує ряд, добре показує довгострокову тенденцію.
- Недолік: сильно "запізнюється". Якщо тренд різко змінює напрямок, прогноз ще довго буде "рухатися за інерцією".

Детальний приклад: Припустимо, маємо дані про відвідуваність сайту за 10 днів. Розрахуємо 4-денне ковзне середнє (k=4) для прогнозу.

День (t)	Відвідувачі (Yt)	Розрахунок 4-MA	Прогноз (Y^t)
1	150	-	-
2	160	-	-
3	155	-	-
4	175	(150+160+155+175)/4 = 160.0	-
5	180	(160+155+175+180)/4 = 167.5	160.0
6	170	(155+175+180+170)/4 = 170.0	167.5

7	190	(175+180+170+190)/4 = 178.75	170.0
8	195	(180+170+190+195)/4 = 183.75	178.75
9	185	(170+190+195+185)/4 = 185.0	183.75
10	200	(190+195+185+200)/4 = 192.5	185.0
11	**?**	-	**192.5**

Як бачимо, значення ковзного середнього за день t стає прогнозом на день t+1. Наш прогноз на 11-й день – **192.5**.

Зважене ковзне середнє (Weighted Moving Average, WMA). Це вдосконалення простого MA, яке вирішує його головну проблему: однакове ставлення до старих і нових даних.

Ідея: Надамо більше ваги новішим спостереженням, оскільки вони, ймовірно, містять більше актуальної інформації про майбутнє.

Формула:

$$\hat{Y}_{t+1} = w_1 Y_t + w_2 Y_{t-1} + \cdots + w_k Y_{t-k+1}$$

де ваги wi є числами, сума яких дорівнює 1 $(\sum_{i=1}^{k} w_i = 1)$, і зазвичай $w_1 > w_2 > \cdots > w_k$.

Приклад (k=3): Припустимо, ми вирішили надати ваги: 0.5 (останнє значення), 0.3 (передостаннє), 0.2 (третє з кінця). Дані за останні 3 дні: 185, 200, 190.

Прогноз=(0.5×190)+(0.3×200)+(0.2×185)=95+60+37=192

Цей прогноз більш "агресивно" реагує на останні зміни, ніж простий середній.

Часові ряди мають унікальну структуру через темпоральну залежність.

Декомпозиція на тренд, сезонність та залишки – потужний інструмент аналізу, який дозволяє зрозуміти рушійні сили, що стоять за даними.

Ковзне середнє – це простий, але ефективний інструмент для згладжування даних та створення базових прогнозів. Він є чудовою відправною точкою.

Однак, для роботи з даними, що мають складні тренди та виражену сезонність, ковзне середнє є занадто примітивним. Воно не може прогнозувати сезонні піки чи адаптуватися до швидкої зміни тренду.

ЛЕКЦІЯ 14

Важливі "м'які" теми в машинному навчанні

Ми живемо в епоху, коли алгоритми стають невидимими архітекторами нашого суспільства. Вони впливають на те, які новини ми читаємо, кого беремо на роботу, які медичні діагнози ставимо та навіть на те, як працює система правосуддя. Саме тому розмова про машинне навчання (ML) виходить далеко за межі коду та математики. Етичні та соціальні аспекти – справедливість, упередженість, прозорість – стають центральними для створення технологій, які приносять користь, а не шкоду.

Глибоке занурення в проблему упередженості (Bias)

Упередженість у машинному навчанні – це не просто статистична похибка, а системне відхилення, яке може закріплювати та навіть посилювати існуючу соціальну нерівність. Це фундаментальна проблема, що виникає на кожному етапі життєвого циклу ML-моделі.

Розширена класифікація джерел упередженості:

Упередженість у даних: найпоширеніше джерело проблем.

Історична упередженість (Historical Bias). Дані відображають минулі стереотипи, навіть якщо вони вже не є актуальними або є неприйнятними.

Система для автоматичного скринінгу резюме навчається на даних компанії за останні 20 років. У цей період на керівних посадах було 90% чоловіків. Модель робить логічний, але несправедливий висновок, що стать є важливою ознакою для успішного керівника, і починає систематично занижувати рейтинг кандидаток-жінок, незалежно від їхньої кваліфікації.

Упередженість вибірки (Selection/Sampling Bias). Дані, на яких навчається модель, не є репрезентативними для популяції, на якій модель буде використовуватись.

Розробка моделі для діагностики раку шкіри. Для її навчання використовували тисячі фотографій уражень шкіри, але 95% з них були

зроблені на пацієнтах зі світлою шкірою. В результаті, модель показує високу точність на світлій шкірі, але катастрофічно помиляється на темній, оскільки прояви хвороби можуть виглядати інакше.

Упередженість вимірювання (Measurement Bias). Помилки вносяться на етапі збору даних через несправні сенсори, різницю в методах збору для різних груп або суб'єктивність оцінювача.

У різних лікарнях використовують різні тонометри для вимірювання тиску. Деякі з них систематично завищують показники. Модель, навчена на цих даних, може невірно оцінювати ризик серцево-судинних захворювань для пацієнтів з певних клінік.

Алгоритмічна упередженість (Algorithmic Bias): проблема криється в самій моделі або в тому, як ми її оптимізуємо.

Упередженість через спрощення. Алгоритм є надто простою моделлю складної реальності, і в процесі "спрощення" він ігнорує важливі нюанси, що призводить до дискримінації.

Упередженість через проксі-змінні (Proxy Bias). Модель може не використовувати захищену ознаку (наприклад, расу) напряму, але використовувати іншу змінну, яка сильно з нею корелює (наприклад, поштовий індекс, який часто пов'язаний з расовим складом району). Це призводить до прихованої дискримінації.

Банківська модель для оцінки кредитоспроможності не використовує расу клієнта. Однак вона використовує дані про район проживання. Історично, представники меншин частіше живуть у бідніших районах. Модель може "вивчити", що заявки з певних поштових індексів є більш ризикованими, що фактично відтворює расову дискримінацію.

Методи виявлення та пом'якшення упередженості

Боротьба з упередженістю – це активний процес.

Виявлення. Використання спеціальних метрик справедливості (наприклад, demographic parity, equal opportunity difference) для аудиту моделі. Аналіз помилок моделі для різних демографічних груп.

Пом'якшення

Pre-processing (на етапі даних): збалансування даних, аугментація даних для недостатньо представлених груп.

In-processing (на етапі навчання): включення обмежень справедливості безпосередньо у функцію втрат моделі. Модель штрафується не тільки за помилки, а й за неспраливі рішення.

Post-processing (на етапі прогнозування): коригування результатів моделі для різних груп, щоб досягти бажаних показників справедливості.

Архітектура довіри: Справедливість, Підзвітність, Прозорість (FAT)

FAT – це не просто гасла, а інженерні принципи, які мають бути інтегровані в процес розробки ML-систем.

Справедливість (Fairness)

Справедливість вимагає глибокого розуміння контексту. Те, що є справедливим в одній ситуації, може бути дискримінаційним в іншій.

- Індивідуальна справедливість: схожі індивіди мають отримувати схожі результати від моделі.
- Групова справедливість: різні групи (визначені за статтю, расою тощо) мають отримувати однакові результати в середньому.

Проблема компромісу (Trade-off): Часто неможливо одночасно оптимізувати всі метрики справедливості. Більше того, покращення справедливості може призвести до незначного зниження загальної точності моделі. Це бізнес-рішення, яке вимагає свідомого вибору: що для нас важливіше – максимальна точність чи уникнення дискримінації?

Підзвітність (Accountability)

Підзвітність означає створення ланцюжка відповідальності.

Хто відповідальний? Розробники, які створюють модель? Менеджери продукту, які визначають її цілі? Компанія, яка її впроваджує? Законодавство (наприклад, європейський GDPR або AI Act) намагається дати відповіді на ці питання, вводячи вимоги до документації, аудиту та управління ризиками.

Практичні кроки:

- Model Cards: створення "паспортів" для моделей, де чітко описано, на яких даних вона навчалася, які її обмеження, для яких груп вона може працювати некоректно.
- Внутрішні комітети з етики: спеціальні групи в компаніях, які оцінюють потенційні ризики нових ML-систем.
- Механізми апеляції: користувачі повинні мати можливість оскаржити рішення, прийняте алгоритмом, і отримати перегляд справи людиною.

Прозорість та Пояснюваність (Transparency & Explainability, XAI)

Довіра неможлива без розуміння. Прозорість – це ключ до налагодження діалогу між складними алгоритмами та людьми, на яких вони впливають.

"Білі скриньки" vs "Чорні скриньки"

Інтерпретовані моделі: лінійна регресія, логістична регресія, дерева рішень. Ми можемо легко подивитися на їхні коефіцієнти або правила і зрозуміти логіку їхньої роботи. Наприклад: "Ваш шанс на отримання кредиту зменшується на 5% за кожні $1000 боргу".

Складні моделі: глибокі нейронні мережі, градієнтний бустинг. Вони показують високу точність, але їхня внутрішня логіка прихована за мільйонами параметрів.

Інструменти XAI для "чорних скриньок":

Локальні пояснення – пояснюють, чому було прийнято *конкретне* рішення для *одного* випадку.

LIME (Local Interpretable Model-agnostic Explanations) створює просту, локальну модель навколо конкретного прогнозу, щоб імітувати поведінку "чорної скриньки" в цій точці.

SHAP (SHapley Additive exPlanations) використовує теорію ігор, щоб визначити внесок кожної ознаки в кінцевий прогноз. Показує, які фактори "штовхнули" прогноз вгору, а які – вниз.

Глобальні пояснення – допомагають зрозуміти загальну поведінку моделі. Наприклад, які ознаки є найважливішими для моделі в цілому.

Життя після model.fit(): деплоймент, моніторинг та підтримка

Створення моделі – це лише початок її шляху. Реальна цінність з'являється лише тоді, коли модель починає працювати в реальному світі. Цей процес описується практиками MLOps (Machine Learning Operations).

Розгортання (Deployment)

Деплоймент – це процес перетворення вашого коду з Jupyter Notebook на надійний програмний продукт.

Стратегії розгортання

REST API. Модель "загортається" у веб-сервіс (наприклад, за допомогою Flask або FastAPI) і надає прогнози через HTTP-запити.

Вбудована модель (Edge Deployment). Модель працює безпосередньо на пристрої користувача (смартфон, камера). Це забезпечує низьку затримку та не потребує постійного інтернет-з'єднання.

Потокова обробка (Streaming). Модель обробляє дані в реальному часі, які надходять безперервним потоком (наприклад, з датчиків IoT).

Інструменти: Docker для контейнеризації, Kubernetes для оркестрації, хмарні платформи (AWS, Google Cloud, Azure) для керування інфраструктурою.

Моніторинг та Підтримка (Monitoring & Maintenance)

Реальність не статична. Модель, яка ідеально працювала в день запуску, може стати абсолютно неточною через місяць. Моніторинг – це система раннього сповіщення про проблеми.

Що саме ми моніторимо?

Дрейф даних: розподіл нових, вхідних даних починає відрізнятися від даних, на яких модель навчалася.

- Дрейф коваріат – змінюється розподіл вхідних ознак (наприклад, середня сума покупки клієнтів зросла).
- Дрейф концепції – змінюється сам зв'язок між вхідними ознаками та цільовою змінною (наприклад, через економічну кризу низький дохід більше не є надійним показником дефолту за кредитом).

Деградація моделі – ключові метрики якості моделі (точність, F1-score тощо) починають падати з часом. Це прямий наслідок дрейфу.

Операційні метрики – час відповіді моделі, навантаження на сервер, кількість помилок. Модель може бути точною, але якщо вона відповідає 5 секунд замість 100 мілісекунд, вона є непридатною для багатьох завдань.

Цикл зворотного зв'язку. Моніторинг – це не пасивне спостереження. Його результати є тригером для дій: збір нових даних, їх розмітка, перенавчання моделі та її повторне розгортання. Це безперервний цикл, який забезпечує актуальність та надійність ML-системи.

СПИСОК ЛІТЕРАТУРИ

1. McKinney W. Python for Data Analysis. 3rd ed. Boston : O'Reilly Media, 2022. 582 p. URL: https://wesmckinney.com/book/
2. Вавринюк К. Аналіз впливу вибору ознак на точність моделей класифікації. «Проблеми та перспективи реалізації та впровадження міждисциплінарних наукових досягнень»: збірник наукових праць з матеріалами IX Міжнародної наукової конференції, м.Луцьк, 13 червня, 2025р. / Міжнародний центрнаукових досліджень. – Вінниця: ТОВ УКРЛОГОС Груп 2025. С. 251–252.
3. Burkov A. The Hundred-Page Machine Learning Book. 2019. 150 p. URL: http://themlbook.com/
4. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 3rd ed. Boston : O'Reilly Media, 2022. 856 p. URL: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/
5. Печончик Н.Р., Приходько О. С. Порівняльний аналіз моделей машинного навчання для прогнозування міцності бетону// Міжвузівський збірник «Наукові нотатки». – Луцьк, 2025. – № 84. – С. 273–278. – DOI: https://doi.org/10.36910/775.24153966.2025.84.26
6. James G., Witten D., Hastie T., Tibshirani R. An Introduction to Statistical Learning: with Applications in R. 2nd ed. Springer, 2021. 607 p. (Springer Texts in Statistics). URL: https://www.statlearning.com/
7. Knaflic C. N. Storytelling with Data: A Data Visualization Guide for Business Professionals. New Jersey : John Wiley & Sons, 2015. 288 p. URL: https://www.storytellingwithdata.com/books
8. Hyndman R. J., Athanasopoulos G. Forecasting: Principles and Practice. 3rd ed. Melbourne : OTexts, 2021. URL: https://otexts.com/fpp3/
9. O'Neil C. Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy. New York : Crown, 2016. 272 p. URL:https://www.sfu.ca/~palys/ONeil-2018-WeaponsOfMathDestruction.pdf

DataMining: конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Штучний інтелект та аналіз масивів даних» спеціальності «Прикладна математика» денної форми навчання/ уклад. К. Вавринюк Луцьк: ЛНТУ, 2025. – 84 с.

Комп'ютерний набір К. Вавринюк
Редактор К. Вавринюк

Підп. до друку « ___ »_____ 2025 р. Формат 60x84/16. Папір офс.
Гарн. Таймс. Ум. друк. арк. 5,25.
Тираж 50 прим.

Інформаційно-видавничий відділ
Луцького національного технічного університету
43018, м. Луцьк, вул. Львівська, 75
Друк – ІВВ ЛНТУ