ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ УПРАВЛІННЯ

МОРОЗОВ В.В., ТІМІНСЬКИЙ О.Г., ЄРЕМЕНКО Б.М.

АНАЛІТИКА ПРИРОДНОМОВНИХ ТЕКСТІВ

Методичні вказівки
до виконання лабораторних та практичних робіт

Київ – 2025

Рецензенти:

д.т.н., професор, завідувач кафедри інформаційних технологій Київського національного університету будівництва та архітектури Тетяна Гончаренко;

к.т.н., доцент, доцент кафедри інтелектуальних технологій Київського національного університету імені Тараса Шевченка, Оксана Золотухіна.

Рекомендовано до друку Вченою радою факультету інформаційних технологій, протокол №7 від “29” грудня 2025 р.

Морозов Віктор Володимирович, Тімінський Олександр Георгійович, Єременко Богдан Михайлович.

Аналітика природномовних текстів: методичні вказівки до виконання лабораторних та практичних робіт / Морозов В. В., Тімінський О. Г., Єременко Б. М. – К. : КНУ імені Тараса Шевченка, 2025. – 30 с.

Методичні вказівки призначені для студентів освітнього рівня магістр другого року навчання освітньо-наукової програми “Інформаційна аналітика та впливи” (галузь знань 12 “Інформаційні технології”, спеціальність F3 (122) “Комп’ютерні науки”).

Видається в авторській редакції.

ЗМІСТ

ВСТУП

Навчальна дисципліна “Аналітика природномовних текстів” присвячена вивченню актуальної тематики обробки природної мови за допомогою інформаційних технологій і використання такої обробки у різних галузях застосування. За словами дослідниці Елізабет Лідді: “Обробка природної мови - це комп'ютеризований підхід до аналізу тексту, що базується на низці теорій та наборі технологій. Ця галузь не має одного загальноприйнятого визначення, адже вона перебуває у стані постійних досліджень та розробок”[5].

Метою дисципліни “Аналітика природномовних текстів” є сформувати у студентів цілісне уявлення про дослідження природних текстових даних в інформаційній аналітиці, наукові засади, технології, методи, функції, типи, види і основні напрямки їх використання в інформаційно-аналітичній діяльності.

Навчальна дисципліна “Аналітика природномовних текстів” викладається у третьому семестрі для студентів освітнього рівня магістр другого року навчання освітньо-наукової програми “Інформаційна аналітика та впливи” (галузь знань 12 “Інформаційні технології”, спеціальність F3 (122) “Комп’ютерні науки”).

Робочою навчальною програмою дисципліни передбачено 16 годин для лабораторних робіт та 16 годин для практичних робіт, 2 години з яких виділяється для захисту лабораторних і практичних.

До лабораторних і практичних робіт додаються короткі теоретичні відомості, формулюються завдання для виконання, надаються контрольні запитання для перевірки набутих знань і навичок.

ПРАКТИЧНІ РОБОТИ

Практична робота №1.

Основні техніки попередньої обробки природномовного тексту.

Мета: Ознайомлення студентів із базовими методами попередньої обробки текстових даних, зокрема токенізацією, лематизацією, видаленням стоп-слів і нормалізацією тексту. Формування практичних навичок використання бібліотек spaCy та NLTK для очищення та підготовки текстів до подальшої аналітики.

Час проведення: 2 год.

Теоретичні відомості

Токенізація тексту – це процес розбиття тексту на менші одиниці, які називаються токенами. Зазвичай токени – це слова, розділові знаки, числа, речення тощо. Токенізація є першим кроком у багатьох задачах обробки природної мови (NLP) [2, 8, 10].

Розглянемо речення: “У 2025 році, після довгих досліджень, вчені представили нову мовну модель”.

Після токенізації отримаємо результат:

[“У”, “2025”, “році”, “,”, “після”, “довгих”, “досліджень”, “,”, “вчені”, “представили”, “нову”, “мовну”, “модель”, “.”].

Видалення стоп-слів

Стоп-слова – це найбільш вживані, але малозначущі слова (наприклад: "і", "в", "не", "це", "є") [2, 11]. Їх видаляють, щоб зменшити шум.

Приклад:

[“2025”, “році”, “довгих”, “досліджень”, “вчені”, “представили”, “нову”, “мовну”, “модель”].

Стемінг

Стемінг – техніка спрощення слова до кореня без урахування граматики**.**

Приклад:

[“2025”, “рік”, “довг”, “дослідж”, “вчен”, “представ”, “нов”, “мовн”, “модел”]

Лематизація

Лематизація – процес зведення слова до ***лексичної основи (леми)*** з урахуванням частини мови. Є більш точною технікою, ніж стемінг, але більш ресурсоємною.

Для зменшення розмірності ознакового простору та уніфікації текстових одиниць використовують методи нормалізації: стемінг та лематизацію [3, 10, 11].

Порівняння стемінгу та лематизації подано у табл. 1.

Приклад:

[“2025”, “рік”, “довгий”, “дослідження”, “вчений”, “представити”, “новий”, “мовний”, “модель”].

Таблиця 1. Порівняння методів нормалізації тексту

Слово	Стемінг	Лематизація
2025	2025	2025
дослідження	дослідж	дослідження
вчені	вчен	вчений
представили	представ	представити
довгих	довг	довгий
мовну	мовн	мовний
рік	рік	рік
модель	модел	модель

Завдання для обов’язкового виконання:

1. Виконати токенізацію тексту з використанням бібліотеки spaCy та NLTK. Порівняти результати токенізації.
2. Видалити зайві символи та стоп-слова:
3. Виконати лематизацію та стемінг. Порівняти результати. Словам, що не мають нормалізованих форм, задати їх.
4. Сформувати звіт до практичної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Контрольні запитання

1. Для чого необхідно виконувати попередню обробку тексту?
2. Які інструменти використовуються для первинної обробки тексту?
3. Що таке токенізація, і які існують її типи?
4. У чому різниця між стемінгом і лематизацією?
5. Для чого використовують видалення стоп-слів, і які можливі недоліки?

Рекомендовані джерела: [2, 3, 8, 10, 11].

Практична робота №2.
POS-тегування та синтаксичний аналіз.

Мета: Набуття навичок використання інструментів автоматичного визначення частин мови (POS-тегування) та аналізу синтаксичної структури речень (синтаксичний аналіз). Формування навичок роботи з бібліотеками spaCy та NLTK для виконання морфологічного аналізу тексту та побудови дерев синтаксичних залежностей.

Час проведення: 2 год.

Теоретичні відомості

Тег частини мови (part-of-speech tag) вказує, до якої частини мови в цьому конкретному реченні відноситься конкретне слово (воно може бути іменником, дієсловом, прислівником тощо). В залежності від контексту одне й те саме слово може виступати у ролі різних частин мови. У бібліотеці spaCy теги частин мови нерідко містять і детальну інформацію про токени. Наприклад, в інформації про дієслово можуть бути вказані наступні ознаки: час (минуле, сьогодення або майбутнє); число – однина або множина [2, 3].

Виділення тегів частин мови для дієслів може допомогти з визначенням наміру користувача, коли токенізації та лематизації недостатньо.

Задачу ускладнює той факт, що при лематизації дієслова зводяться до форм інфінітива, через що складно визначити їх роль у реченні.

У такій ситуації стають у нагоді теги частин мови. До основних частин мови належать: іменник, займенник, визначник, прикметник, дієслово, прислівник, числівник; службові частини мови: прийменник, сполучник, артикль. У бібліотеці spaCy представлені ті самі категорії, а також ще декілька для позначення укрупнених частин мови (coarse-grained parts of speech) – символів, розділових знаків тощо. Всі категорії доступні у вигляді фіксованого набору тегів через атрибути Token.pos і Token.pos_[10, 11].

Крім того, spaCy надає теги для уточнених частин мови (fine-grained parts of speech) з більш детальною інформацією про токені, де вказані морфологічні ознаки, час дієслів і типи займенників.

В табл. 2. наведені деякі поширені теги частин мови, що використовуються у spaCy.

Таблиця 2. Опис морфологічних тегів бібліотеки spaCy

TAG	POS	Опис
NN	NOUN	Іменник, однина
NNS	NOUN	Іменник, множина
NNP	PROPN	Власна назва
PRP	PRON	Особовий займенник
PRP$	DET	Присвійний займенник
VB	VERB	Дієслово в інфінітиві
VBD	VERB	Дієслово, минулий час
VBN	VERB	Дієслово у формі дієприкметника
VBP	VERB	Дієслово, теперішній час
JJ	ADJ	Прикметник
JJR	ADJ	Ступінь порівняння
RB	ADV	Прислівник
MD	AUX	Допоміжне дієслово
POS	PART	Частка (не, би, ж, хай...)
PUNCT	PUNCT	Розділовий знак
SYM	SYM	Символ або спеціальний знак

Синтаксичні залежності

Мітки синтаксичної залежності (syntactic dependency labels), так само, як і леми і теги частин мови – це лінгвістичні ознаки, що надаються бібліотекою spaCy об’єктам Token, які утворюють в об’єкті Doc текст. Наприклад, мітка залежності dobj відповідає прямому додатку (direct object). Відповідне синтаксичне відношення можна проілюструвати спрямованою дугою, як показано на рис 1.

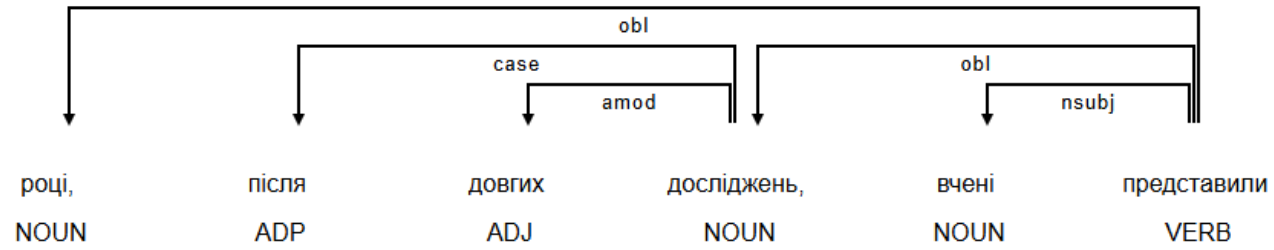

Рис. 1. — Візуалізація синтаксичних залежностей у реченні за допомогою spaCy

У кожного слова в реченні є тільки один головний елемент. Відповідно, будь-яке слово може бути дочірнім по відношенню лише до одного головного елементу. Зворотне твердження справедливо не завжди: одне і те саме слово може взагалі не виступати у ролі головного елемента, або відігравати цю роль як в одній, так і в декількох парах. Останнє означає, що у головного елемента є кілька дочірніх. Таким чином, мітка залежності завжди надається дочірньому елементу пари.

Одне і те саме слово в реченні може брати участь в декількох синтаксичних відносинах. В табл.3. наведено деякі мітки залежностей.

Таблиця 3. Основні мітки синтаксичних залежностей у spaCy

Мітка	Назва	Опис
nsubj	nominal subject	Підмет
obj	object	Прямий додаток
obl	oblique nominal	Обставина (місце, час, засіб тощо)
advmod	adverbial modifier	Прислівникове означення
amod	adjectival modifier	Прикметникове означення
case	case marker	Прийменник
cc	coordinating conjunction	Сполучник сурядності
root	Root	Головне слово речення

Мітка ROOT позначає токен, головним елементом для якого є він сам. Зазвичай таку мітку Spacy надає смисловому дієслову речення (тобто дієслову, що становить основу присудка). У кожному повному реченні повинні бути дієслово з тегом ROOT і підмет з тегом nsubj. Інші елементи не є обов’язковими.

Завдання для обов’язкового виконання:

1. Виконати POS-тегування тексту з використанням spaCy та NLTK для визначення частин мови. Порівняти результати POS-тегування для різних речень.
2. Проаналізувати синтаксичні залежності:
   - Визначити підмет, присудок та об’єкти в реченнях.
   - Визначити залежності між словами в реченні.
   - Визначити іменовані сутності.
   - Побудувати графічне представлення синтаксичних зв’язків.
3. Порівняти результати для текстів з різними граматичними структурами (науковий, художній, новини). Проаналізувати, як змінюються залежності та POS-теги.
4. Проаналізувати можливі помилки у тегуванні та синтаксичному аналізі.
5. Сформувати звіт до практичної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Контрольні запитання

1. Що таке POS-тегування, і для чого воно використовується?
2. Які основні частини мови визначаються під час POS-тегування? Наведіть приклади.
3. Що таке синтаксичний аналіз, і які його основні завдання?
4. Що таке дерево залежностей, і як воно використовується в NLP?

Рекомендовані джерела: [2, 3, 10, 11].

Практична робота №3.
Робота з об’єктами-контейнерами у spaCy.

Мета: Набуття навичок роботи з об’єктами-контейнерами бібліотеки *spaCy* для подальшої обробки текстів. Отримання навичок розбиття тексту на речення, роботи з іменниковими групами, синтаксичними зв’язками, та аналізу відносини між словами за допомогою дерев залежностей.

Час проведення: 2 год.

Теоретичні відомості

Об’єкти-контейнери spaCy

Об’єкти-контейнери (container objects) призначені для організації кількох елементів в одній групі. Це можуть бути колекції токенів або речень, а також набори міток, що стосуються одного об’єкта. Наприклад, об’єкт Token у spaCy є контейнером для набору міток, таких як частини мови, що відносяться до конкретного токену в тексті. Об’єкти-контейнери в spaCy відтворюють структуру текстів природної мови, де текст складається з речень, а кожне речення містить токени.

Найбільш поширені об’єкти-контейнери в spaCy – це Token, Span і Doc, які є окремими токенами, фразами/реченнями та усім текстом відповідно. Один контейнер може містити інші: наприклад, об’єкт Doc містить об’єкти Token [3, 10, 11].

Індексування токенів в об’єкті Doc

Об’єкт Doc містить колекцію об’єктів Token, створених у процесі токенізації вхідного тексту. Токени можна отримувати за індексами, що відповідають їх порядку в тексті. Індексація токенів починається з нуля, тому індекс останнього токена дорівнює довжині документа мінус один. Щоб отримати токени з об’єкта Doc, їх потрібно помістити до списку Python і ітеративно проходити через кожен токен.

Контейнер doc.sents

Зазвичай синтаксичні мітки, що присвоєні токенам, мають значення лише у контексті речення, в якому цей токен зустрічається. Наприклад, інформація про те, чи є слово іменником або дієсловом, має значення лише у межах конкретного речення, що містить це слово. За допомогою властивості doc.sents об'єкта Doc можна розділити текст на окремі речення.

Контейнер doc.noun_chunks

Властивість doc.noun_chunks об'єкта Doc дозволяє пройти по іменникових фразах. Іменникова фраза (noun chunk) – це фраза, головним елементом якої є іменник.

Завдання для обов'язкового виконання:

1. Виконати розбиття тексту на речення різними техніками (з використанням doc.sents та власним методом). У тексті мають бути речення, що закінчуються різними знаками пунктуації. Порівняти результати.
2. Вивести іменникові групи з використанням doc.noun_chunks.
3. Побудувати список синтаксичних залежностей кожного слова (через Token.children).
4. Визначити головні слова для підметів і об'єктів (через Token.head).
5. Візуалізувати дерево залежностей.
6. Сформувати звіт до практичної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Контрольні запитання

1. Що таке об'єкти-контейнери в spaCy? Наведіть приклади.
2. Як spaCy організує дані для обробки тексту ?
3. Як у spaCy виділити речення з тексту?
4. Що таке іменникові групи і як їх можна отримати в spaCy?
5. Як працює об'єкт doc.noun_chunks, і які переваги дає його використання?

Рекомендовані джерела: [3, 11].

Практична робота №4.
Векторизація тексту.

Мета: Ознайомитись з найпростішими методами векторизації різних типів текстової інформації, навчитись виконувати порівняння фрагментів тексту.

Час проведення: 4 год.

Теоретичні відомості

Векторизація – це процес перетворення тексту в числове представлення (вектор), яке можна обробляти алгоритмами машинного навчання. Мета векторизації – подати зміст слів, речень або документів у формі, що придатна для обчислень [2, 3, 6, 10].

Щоб аналізувати або класифікувати текст, потрібно представити його у вигляді векторів, які зберігають певні характеристики слів чи речень, до прикладу: частоту термів, контекст, значення або семантику. Основним критерієм класифікації методів векторизації є врахування контексту. Найпоширенішими методами векторизації, що не враховують контекст, є метод мішка слів та метод TF-IDF.

Метод «мішка слів».

Створюється словник усіх термінів, що зустрічаються у корпусі текстів, а кожен документ подається як вектор частот термінів.

Розглянемо три речення:

“Кіт сидить на ліжку”.

“Собака лежить на ліжку”.

“Кіт сидить на ліжку, бо кіт любить ліжко”.

Після токенізації усіх слів, отримаємо словник:

{“кіт”, “сидить”, “на”, “ліжку”, “собака”, “лежить”, “бо”, “любить”, “ліжко”}.

Кожне речення представляється як вектор розміром 8, по одному числу на кожне слово зі словника (табл. 4).

Таблиця 4. Побудова частотного словника та векторів для корпусу текстів

Слово	Речення 1	Речення 2	Речення 3
кіт	1	0	1
сидить	1	0	1
на	1	1	1
ліжко	1	1	2
собака	0	1	0
лежить	0	1	0
бо	0	0	1
любить	0	0	1

Таким чином, отримаємо три вектори речень:

“Кіт сидить на ліжку”: [1;1;1;1;0;0;0;0],

“Собака лежить на ліжку”: [0;0;1;1;1;1;0;0],

“Кіт сидить на ліжку, бо кіт любить ліжко”: [1;1;1;2;0;0;1;1].

Цей приклад ілюструє, як «мішок слів» фіксує лексичну присутність термінів, але не враховує їх порядок, граматичну функцію або контекст**.**

Метод TF-IDF.

Це статистичний метод, який оцінює важливість слова у документі відносно всього корпусу. Суть методу полягає в тому, що він визначає важливість слова, оцінює, наскільки часто воно зустрічається в документі (term frequency) та наскільки воно унікальне в колекції документів (inverse document frequency) на основі показника TF-IDF, який розраховується за формулою:

$$TF - IDF = TF \cdot IDF,$$

де TF (Term Frequency) – частота слова в документі:

$$TF(t,d) = \frac{f_{t,d}}{\sum_{k=1}^{n} f_{k,d}}$$

де $f_{t,d}$ – кількість появ слова t у документі d,

$\sum_{k=1}^{n} f_{k,d}$ – загальна кількість слів у документі.

IDF (Inverse Document Frequency) – це логарифмічна обернена частота, з якою слово зустрічається в колекції документів. IDF обчислюється як загальна кількість документів, поділена на кількість документів, які містять слово.

$$IDF = log\ \frac{|D|}{|(d_i \supset t_i)|}$$

де $|D|$ – кількість документів колекції;

$|(d_i \supset t_i)|$ – кількість документів, в яких зустрічається слово t_i.

Розглянемо 2 документи:

- ***Документ 1:*** “кіт спить на ліжку”.
- ***Документ 2:*** “пес грає на дворі”.

Лематизуємо усі слова та побудуємо словник: {кіт, спить, на, ліжко, пес, грати, двір}.

Подамо кількість появ слова у кожному документі у вигляді табл.5:

Таблиця 5. Частота появи слів у корпусі документів

Слово	Документ 1	Документ 2
кіт	1	0
спить	1	0
на	1	1
ліжко	1	0
пес	0	1
грати	0	1
двір	0	1

Розраховані значення показника TF для кожного слова наведено у табл. 6.

Таблиця 6. Результати розрахунку показників частоти термінів (TF)

Слово	TF (D1)	TF (D2)
кіт	0.25	0
спить	0.25	0
на	0.25	0.25
ліжко	0.25	0
пес	0	0.25
грати	0	0.25
двір	0	0.25

Результати обчислення IDF для кожного слова наведено у табл. 7.

Таблиця 7. Значення IDF для унікальних слів

Слово	n	IDF
кіт	1	lg(2/1) = 0.301
спить	1	0.301
на	2	lg(2/2) = 0
ліжку	1	0.301
пес	1	0.301
грає	1	0.301
дворі	1	0.301

Підсумкові значення TF-IDF, що визначають важливість кожного слова, подано у табл. 8.

Таблиця 8. Фінальні ваги TF-IDF для термінів у документах D1 та D2

Слово	TF-IDF (D1)	TF-IDF (D2)
кіт	0.0753	0
спить	0.0753	0
на	0	0
ліжку	0.0753	0
пес	0	0.0753
грає	0	0.0753
дворі	0	0.0753

У результаті ми отримаємо векторне представлення кожного документа:

Документ 1: [0.0753; 0.0753; 0; 0.0753; 0; 0; 0; 0].

Документ 2: [0; 0; 0; 0;0; 0.0753; 0.0753; 0.0753].

У подальшому для порівняння документів між собою можна використати метод косинусної подібності.

Завдання для обов'язкового виконання:

1. Виконати векторизацію декількох текстів методом TF-IDF, реалізованими власноруч та вбудованими методами spaCy.
2. Виконати порівняння текстів, проаналізувати результати.
3. Порівняти результати роботи власного методу та методу spaCy.
4. Виконати векторизацію слів вбудованими методами spaCy.
5. Виконати порівняння слів, проаналізувати результати.
6. Сформувати звіт до практичної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Контрольні запитання

1. Що таке метод TF-IDF? Як він працює при векторизації тексту?
2. Як працює метрика косинусної подібності при порівнянні векторів текстів?
3. Що таке векторизація слів, і чим вона відрізняється від векторизації тексту?
4. Як працює векторизація слів?
5. Які методи векторизації тексту існують для слів і речень, і чим вони відрізняються?
6. Як контекст впливає на векторизацію слів і текстів?

Рекомендовані джерела: [2, 3, 6, 10].

Практична робота №5.
Сентимент аналіз (визначення тональності тексту).

Мета: Ознайомити студентів із методами аналізу сентименту тексту, навчити працювати з лексичними методами та моделями машинного навчання.

Час проведення: 4 год.

Теоретичні відомості

Сентимент-аналіз (англ. *sentiment analysis*) – це процес автоматичного визначення емоційного тону тексту: чи є він позитивним, негативним або нейтральним.

Широко застосовується:

- в аналізі відгуків;
- у соціальних медіа;
- у журналістиці;
- у фінансовій сфері (аналіз новин, прогноз поведінки ринку).

До визначення емоційного забарвлення тексту можна використовувати велику кількість методів – від елементарної косинусної подібності до складних моделей машинного навчання. Загалом їх можна поділити на лексичні методи та машинне навчання.

Лексичні методи базуються на словниках, де кожному слову ставиться у відповідність певна емоційна вага (полярність). Найбільш поширеними є VADER (Valence Aware Dictionary and sEntiment Reasoner) та TextBlob [3, 9, 10].

Методи на основі машинного навчання використовують попередньо навчені моделі, які враховують контекст слів. Найпоширенішими з них на разі є моделі Transformers.

Завдання для обов’язкового виконання:

1. Провести обробка тексту.
2. Здійснити аналіз сентименту за лексичним методом (VADER).
3. Здійснити аналіз полярності тексту(TextBlob).
4. Використати трансформери для сентимент-аналізу.

5. Порівняти методи.

Порівняльна характеристика роботи використаних методів аналізу тональності наведена у табл. 9.

Таблиця 9. Результати аналізу тональності відгуків

Текст	VADER	TextBlob	Transformers
“I love this movie! It was amazing”.	0.85	0.9	POSITIVE
“This product is awful. Don't buy it”.	-0.76	-0.7	NEGATIVE
“It works. Nothing special, just okay”.	0.05	0.1	NEUTRAL (або POSITIVE/NEGATIVE з низьким score)

6. Побудувати графіки розподілу сентименту.

Можна використати гістограми або кругові діаграми для кожного методу (кількість позитивних / негативних / нейтральних відгуків, розподіл полярностей).

7. Сформувати звіт до практичної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Контрольні запитання

1. Які методи аналізу сентименту існують?
2. Як працює VADER, та чим він відрізняється від інших методів?
3. Як можна оцінити якість аналізу сентименту?
4. У яких випадках трансформери працюють краще за лексичні методи?

Рекомендовані джерела: [3, 9, 10].

ЛАБОРАТОРНІ РОБОТИ

Лабораторна робота №1.
Робота з LLM через API та локальні моделі.

Мета: Ознайомитись із великими мовними моделями (LLM), навчити запускати їх локально та через API, порівнювати якість та швидкість відповідей.

Час проведення: 4 год.

Велика мовна модель (LLM – Large Language Model) – це модель, заснована переважно на глибокому навчанні, натренована на великих обсягах текстових даних і орієнтована на обробку текстових даних [1, 4, 9]. Така модель здатна:

- продовжувати текст,
- відповідати на запитання,
- генерувати інструкції, коди, діалоги.

LLM бувають локальними та хмарними. Доцільність використання локальних або хмарних сервісів залежить від наявних ресурсів, вимог до конфіденційності даних та інших вимог (табл.10)

Таблиця 10. Порівняльна характеристика локальних та хмарних сервісів великих мовних моделей (LLM)

Вимога	Локальна модель	Хмарна модель
Місце виконання	локальний комп'ютер (або локальний сервер)	Віддалений сервер (OpenAI, Hugging Face тощо)
Контроль над даними	Повний	Частковий (текст передається до хмари)
Ресурси	Потрібен достатній обсяг ОЗП / GPU	Модель працює на сторонніх серверах
Затримка	Може бути нижча (залежно від моделі)	Може залежати від інтернет-з'єднання
Простота використання	Вимагає встановлення та налаштування	Легко запускати через API
Конфіденційність	Повна	Потрібна обережність з приватними даними

Для використання хмарних LLM потрібно використовувати API ключ.

API (Application Programming Interface) – це спосіб звернення до LLM. Найбільш розповсюдженими API-платформами є OpenAI API, Hugging Face Inference API, Google Gemini API.

Завдання для обов'язкового виконання:

1. Ознайомитись з LLM, трансформерами, локальними та хмарними моделями.
2. Запустити локальну LLM:

- Оберіть модель із відкритим кодом (наприклад, Mistral-7B, LLaMA, GPT2).
- Здійсніть запуск з використанням бібліотеки Python та через інтерфейси.
- Переконайтесь, що LLM відповідає на запити.
- Проведіть тести.

3. Запуск LLM через API:

- Зареєструйтесь на платформі (наприклад, OpenAI, Hugging Face).
- Отримайте токен доступу (API ключ).
- Зробіть запит через Python.

4. Порівняти результати роботи моделей.

Для порівняння результатів роботи моделей варто використовувати однакові запити. Оберіть критерії порівняння, наприклад якість тексту (змістовність, логіка, мова), швидкість відповіді, контрольованість (чи слухає модель інструкції). Результати порівняння подати в вигляді таблиці.

5. Сформувати звіт до лабораторної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Очікувані результати

- сформовані два приклади відповідей (через API та локальну модель);
- таблиця порівняння: якість, швидкість, довжина тексту;
- висновок щодо переваг кожного підходу.

Контрольні запитання

1. Що таке LLM, і як вони працюють?
2. Які основні відмінності між локальними моделями та API?
3. Які плюси та мінуси використання локальних LLM?
4. Чому одні API можуть бути повільнішим за інші API?
5. Як можна оцінити якість відповіді LLM?

Рекомендовані джерела: [1, 4, 9].

Лабораторна робота №2.
Тонке налаштування LLM.

Мета: Ознайомитися з можливостями Fine-tuning LLM на спеціалізованих текстових даних з використанням LoRA (Low-Rank Adaptation) або PEFT, адаптувати LLM для конкретних доменів (медицина, юриспруденція, фінанси), аналізувати якість fine-tuning та тестувати отриману модель.

Час проведення: 4 год.

Теоретичні відомості

Тонке налаштування (Fine-tuning) – це донавчання попередньо натренованої великої мовної моделі (LLM) на новому, вузькоспеціалізованому корпусі даних. Метою тонкого налаштування є адаптація моделі до конкретного стилю, задачі або галузі. Інструмент є доволі ефективним, але має низку проблем:

- Потребує великих обчислювальних ресурсів (GPU/TPU, багато пам’яті).
- Призводить до перезапису знань (катастрофічне забування).
- Потребує збереження великої кількості ваг.

Зараз для Fine-tuning LLM використовуються методи LoRA (Low-Rank Adaptation) або PEFT, які дозволяють виконувати тонке налаштування з урахуванням вищезазначених проблем.

LoRA (Low-Rank Adaptation) – це метод адаптації LLM без зміни всіх ваг моделі. Основною ідеєю є заміна повного оновлення ваг додатковими матрицями-адаптерами низького рангу, які навчаються. При цьому базова модель залишається незмінною, а зміни є компактними та зберігаються окремо [1, 4, 9].

Основними параметрами LoRA вважають:

- r – розмір матриці адаптації;
- alpha–масштабування.

Для Fine-tuning великих мовних моделей в Python використовується бібліотека PEFT, що підтримує роботу як з LoRA, так і з іншими PEFT-методами.

Для кількісної оцінки результатів донавчання моделі використовуються метрики, що базуються на ймовірнісному прогнозуванні токенів. Оскільки LLM є генеративною системою, її якість визначається здатністю мінімізувати невизначеність при побудові тексту. найбільш поширеними метрики базуються на визначені втрат та перплексії.

Для кількісної оцінки процесу fine-tuning LLM застосовуються метрики, засновані на ймовірнісному прогнозуванні токенів. Оскільки мовні моделі є генеративними, їхня якість визначається здатністю мінімізувати невизначеність під час генерації тексту. Найбільш поширеними метриками є функція втрат (loss) та перплексія (perplexity).

Функція втрат (Loss) є основною метрикою під час навчання LLM. У задачах fine-tuning зазвичай використовується cross-entropy loss, яка вимірює відхилення між передбаченими модельними ймовірностями та реальними токенами у навчальних даних. Очікується, що під час fine-tuning значення loss поступово зменшуватиметься, що свідчить про адаптацію моделі до доменних даних.

Водночас зростання loss на валідаційному наборі за умови його зменшення на тренувальному може вказувати на перенавчання та потребу корекції параметрів навчання. Слід зазначити, що значення loss не дозволяє безпосередньо оцінити якість згенерованого тексту.

Perplexity характеризує рівень невизначеності моделі під час прогнозування наступного токена. Після успішного fine-tuning очікується суттєве зниження значень perplexity на цільових доменних даних, що свідчить про кращу адаптацію моделі до специфічної термінології та синтаксичних конструкцій. Перплексія є зручною метрикою для кількісного порівняння стану моделі до та після fine-tuning, проте також не відображає повною мірою стилістичні та семантичні характеристики відповідей.

У зв’язку з цим для повної оцінки результатів fine-tuning необхідно поєднувати кількісні метрики з порівняльним якісним аналізом відповідей моделі.

Завдання для обов’язкового виконання:

1. Ознайомитись з Fine-tuning LLM, LoRA, Hugging Face.
2. Підготувати дані (Очистка, токенізація, форматування датасету).

3. Налаштувати Fine-tuning.
4. Протестувати модель.
5. Виконати порівняльний якісний аналіз відповідей моделей “до” і “після” Fine-tuning (табл. 11).
6. Виконати оцінку процесу Fine-tuning (loss, perplexity).

До навчання потрібно зробити кілька контрольних запитів та зберегти відповіді моделі. Після донавчання виконати ті ж запити та здійснити порівняльний аналіз отриманих відповідей за такими характеристиками, як стиль, точність та доменна специфіка, тощо. Результати варто представити у вигляді табл.11 зі стислими характеристиками результатів до та після навчання.

Таблиця 11. Характеристики популярних відкритих великих мовних моделей

Запит	До навчання	Після навчання
“Що таке векторизація?”	Загальний опис	Більш точний, адаптований
“Поясни регуляризацію простими словами”	Сухе визначення	Влучне, навчальне пояснення

7. Сформувати звіт до лабораторної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Очікувані результати

- натренована модель на невеликому корпусі;
- порівняльний якісний аналіз відповідей моделі “до” і “після” Fine-tuning (табл. 11);
- звіт із фрагментами виводу моделі та показниками процесу Fine-tuning (loss, perplexity) в графічному вигляді.

Контрольні запитання

1. Що таке Fine-tuning, і чим він відрізняється від Prompt Engineering?
2. Що таке LoRA, і чому він зменшує витрати на навчання?
3. Як можна оцінити покращення якості відповіді LLM після навчання?
4. У яких випадках Fine-tuning потрібен, а коли достатньо оптимізувати запити?
5. Як впливає обсяг тренувальних даних на точність LLM?

Рекомендовані джерела: [1, 4, 9].

Лабораторна робота №3.
Розширення LLM (LangChain + Retrieval-Augmented Generation).

Мета: Ознайомитись із способами використання LangChain для обробки великих текстових документів у LLM. Навчитись Інтегрувати LLM у систему пошуку інформації, використовувати FAISS для збереження та пошуку текстів, налаштовувати RAG.

Час проведення: 4 год.

Теоретичні відомості

LangChain – це фреймворк для створення застосунків, що базуються на великих мовних моделях. LangChain реалізує стандартний інтерфейс для великих мовних моделей та пов'язаних технологій, що дозволяє виконувати багатоетапні ланцюжки запитів, звертатися до баз даних, API, файлів, використовувати зовнішні джерела знань для підсилення LLM.

Одним із основних способів інтеграції зовнішніх знань у LLM є метод Retrieval-Augmented Generation [1, 9].

RAG (Retrieval-Augmented Generation) – це метод, який дозволяє генерувати відповіді на базі додаткової інформації з зовнішніх джерел.

Послідовність дій при використанні даного методу узагальнено зводиться до наступного:

1. Користувацький запит перетворюється у вектор.
2. У системі здійснюється векторний пошук у базі знань (наприклад, у текстах, документах).
3. Знайдені фрагменти додаються до запиту.
4. LLM генерує відповідь з урахуванням доданої інформації.

Завдання для обов'язкового виконання:

1. Ознайомлення з LangChain, FAISS, RAG.
2. Розбиття тексту на фрагменти.

- Завантажити власний або готовий текст (статтю, інструкцію тощо).

- Розбити його на чанки за допомогою вбудованих методів, або вручну.
- Зібрати чанки у список об’єктів.

3. Створення векторної бази для пошуку. Протестувати індексацію та пошук.

- Векторизувати фрагменти за допомогою відповідних методів.
- Створити індекс (наприклад, FAISS).
- Протестувати пошук.

4. Провести інтеграцію з LLM (налаштування Retrieval-Augmented Generation).

Для цього необхідно побудувати ланцюг дій:

- Користувач створює запит.
- Система виконує пошук у векторній БД.
- Результати додаються до запиту.
- LLM генерує відповідь.

5. Сформувати звіт до лабораторної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Очікувані результати

- побудована RAG-система, що здатна виконувати пошук інформації у базі знань;
- приклад результатів пошуку й відповідей LLM;
- висновок про точність і релевантність відповідей моделі.

Контрольні запитання

1. Що таке LangChain і для чого його використовують?
2. Чому FAISS швидше за звичайний пошук у тексті?
3. Як працює Retrieval-Augmented Generation (RAG)?
4. У яких випадках пошук у документах покращує відповіді LLM?
5. Як можна оцінити ефективність RAG?

Рекомендовані джерела: [1, 4, 12].

Лабораторна робота №4.
Створення власної LLM-платформи.

Мета: Агрегувати навички, набуті у попередніх лабораторних роботах (1–3), для створення повноцінної LLM-системи: запуск fine-tuned моделі, інтеграція через LangChain та розгортання у вигляді веб-застосунку. А також ознайомитись з способами оптимізації LLM (квантування, дистиляція моделей).

Час проведення: 4 год.

Ця лабораторна робота поєднує результати трьох попередніх робіт і має інтегральний характер для практичного засвоєння дисципліни. Однак, перед об'єднанням частин в одне ціле, необхідно виконати оптимізацію LLM. Для цього є два найпоширеніших способи: квантування та дистиляція моделей.

Дистиляція LLM – це процес зменшення розміру базової моделі (також відома, як вчитель) до розміру меншої моделі (також відома, як студент), прогнози якої мають максимально відповідати прогнозам базової моделі. Причому менша модель працюватиме швидше, використовуватиме менше ресурсів, але якість прогнозів буде нижчою.

Квантування (quantization) LLM— метод оптимізації нейронних мереж, що полягає у зменшенні розрядності числового представлення параметрів моделі з метою зниження використання пам'яті та прискорення відповідей моделі [4, 10].

Завдання для обов'язкового виконання:

1. Налаштування середовища.
2. Інтеграція fine-tuned LLM (Завантаження навченої моделі, запуск через API).
3. Виконання квантування та дистиляції моделі.
4. Підключення LangChain.
5. Розгортання веб-інтерфейсу.
6. Сформувати звіт до лабораторної роботи.

Допускається формування звіту у Jupyter Notebook або в аналогах.

Очікувані результати

- створений веб-інтерфейс для генерації відповідей LLM;
- оптимізована LLM (дистиляція + квантування);
- порівняння базової та оптимізованої LLM (швидкість, якість, стабільність роботи) у табличному вигляді.

Контрольні запитання

1. Які основні компоненти LLM-платформи?
2. Як можна оптимізувати швидкість відповідей LLM?
3. Як можна розгорнути веб-застосунок у хмарі?
4. Що таке дистиляція моделі?
5. Для чого потрібно виконувати квантування моделі?

Рекомендовані джерела: [1, 4, 9, 10, 12].

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Pai S. Designing Large Language Model Applications: A Holistic Approach / Suhas Pai. – O'Reilly Media, 2023. – 88 p.
2. Manning C. D. An Introduction to Information Retrieval / Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. – Cambridge University Press, 2009. – 581 p.
3. Rao D. Natural Language Processing with PyTorch / Delip Rao and Brian McMaha. – O'Reilly, 2019. – 254 p.
4. Webber E. Pretrain Vision and Large Language Models in Python / Emily Webber. – Packt Publishing, 2023. – 475 p.
5. Liddy, E.D. 2001. Natural Language Processing. In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc.
6. Pennington J. GloVe: Global Vectors for Word Representation [Електронний ресурс] / J. Pennington, R. Socher, C. Manning. – 2014. – Режим доступу: https://www.researchgate.net/publication/284576917_Glove_Global_Vectors_for_Word_Representation.
7. Deep Communicating Agents for Abstractive Summarization [Електронний ресурс] / C.Asli, B. Antoine, H. Xiaodong, C. Yejin. – 2018. – Режим доступу: https://arxiv.org/pdf/1803.10357.pdf.
8. Domingo M. How Much Does Tokenization Affect Neural Machine Translation? [Електронний ресурс] / Miguel Domingo. – 2018. – Режим доступу: https://arxiv.org/pdf/1812.08621.pdf.
9. Ashish Vaswani. Attention is all you need [Електронний ресурс] / A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. – 2017. – Режим доступу: https://arxiv.org/pdf/1706.03762.
10. Обробка природної мови [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра, магістра за спец. 122 (F3) Комп'ютерні науки та 124 Системний аналіз (F4 Системний аналіз та наука про дані) / М.З. Згуровський

– Київ : КПІ ім. Ігоря Сікорського, 2025. – Режим доступу: https://ela.kpi.ua/items/333a9853-832f-4fce-abe1-6ad74506e023.

11. Web ресурс бібліотеки spaCy / Режим доступу: https://spacy.io.
12. Web ресурс LangChain / Режим доступу: https://www.langchain.com.