МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ПрАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНИЙ

Зав. кафедри _____

д.е.н., доц. С.І. Левицький

# МАГІСТЕРСЬКА ДИПЛОМНА РОБОТА

# РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ НЕЙРОМЕРЕЖЕВОГО МОДЕЛЮВАННЯ

Виконав
ст. гр. ІПЗ-312м _____ О.С. Подскребко

Керівник
завідувач кафедри
інформаційних технологій _____ С.І. Левицький

Запоріжжя

2024 р.

ПрАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ

Зав. кафедри _____

д.е.н., доц. С.І. Левицький

# З А В Д А Н Н Я

## НА МАГІСТЕРСЬКУ ДИПЛОМНУ РОБОТУ

студента гр. ІПЗ-312м, спеціальності 121 «Інженерія програмного забезпечення»
ОП «Інженерія програмного забезпечення»

Подскребка Олександра Сергійовича

1.Тема: Розробка програмних засобів реалізації нейромережевого моделювання затверджена наказом по інституту № 02-30 від 20.10.2023 р.

2. Термін здачі студентом закінченої роботи: 16.12.2023 р.

3. Перелік питань, що підлягають розробці

1. Проаналізувати вплив нейронних мереж на бізнес та людське життя;

2. Проаналізувати напрямки нейромережевого моделювання;

3. Надати рекомендації щодо апаратного та програмного забезпечення необхідного для розробки і використання нейронних мереж;

4. Реалізовувати задачу класифікації тексту класичними алгоритмами машинного навчання;

5. Побудувати моделі для вирішення задач обробки природної мови за допомоги нейронних мереж;

6. Розробити моделі класифікації зображень і нейромережеві моделі прогнозування часових рядів;

7. Розробити мікросервіс аналізу тональності тексту.

4. Календарний графік підготовки магістерської дипломної роботи

| № етапу | Зміст | Термін виконання | Готовність по графіку (%), підпис керівника | Підпис керівника про повну готовність етапу, дата |
|---|---|---|---|---|
| 1. | Формулювання теми магістерської дипломної роботи (збір практичного матеріалу за темою магістерської дипломної роботи) | 20.10.23 | | |
| 2. | I атестація I розділ магістерської дипломної роботи | 27.10.23 | | |
| 3. | II атестація II розділ магістерської дипломної роботи | 17.11.23 | | |
| 4. | III атестація III та IV розділ магістерської дипломної роботи, висновки та рекомендації, додатки, реферат, перевірка програмою «Антиплагіат» | 16.12.23 | | |
| 5. | Доопрацювання магістерської дипломної роботи, підготовка презентації, отримання відгуку керівника і рецензії | 08.01.23 | | |
| 6. | Попередній захист магістерської дипломної роботи | 09.01.23 | | |
| 7. | Подача магістерської дипломної роботи на кафедру | за 3 дні до захисту | | |
| 8. | Захист магістерської дипломної роботи | 17.01.24 | | |

Дата видачі завдання: 05.10.2023 р.

Керівник магістерської роботи _____ С.І. Левицький
(підпис) (прізвище, ініціали)

Завдання отримав до виконання _____ О.С. Подскребко
(підпис студента) (прізвище та ініціали)

# РЕФЕРАТ

Магістерська дипломна робота містить 94 сторінки, 7 таблиць, 60 рисунків, три додатки, 27 бібліографічних посилання.

Об’єктом дослідження є процес розробки програмних засобів реалізації нейромережевого моделювання.

Предметом дослідження є методи і практичні підходи до розробки програмних засобів реалізації нейромережевого моделювання.

В першому розділі проаналізовано предметну область. Досліджено вплив нейронних мереж на бізнес та людське життя. Розглянуті напрямки нейромережевого моделювання. Проведено аналіз до апаратного та програмного забезпечення для розробки нейронних мереж.

У другому розділі розглянуті основні бібліотеки для для розробки нейронних мереж. Проаналізовано інструменти реалізації нейронних мереж та особливості та відмінності згорткових та рекурентних нейронних мереж.

В третьому розділі реалізовано вирішення задачі класифікації тексту класичними алгоритмами машинного навчання. Продемонстровано способи вирішення задач обробки природної мови за допомоги нейронних мереж. Виконано реалізацію вирішення задач класифікації зображень і нейромережевих моделей прогнозування часових рядів. Розроблено мікросервіс аналізу тональності тексту.

Результатом роботи є розроблений мікросервіс аналізу тональності тексту,який може бути використаний спеціалістами різного профілю, такими як аналітики, маркетологи, служба підтримки і т.п.

НЕЙРОМЕРЕЖЕВЕ МОДЕЛЮВАННЯ, МАШИННЕ НАВЧАННЯ, АНАЛІЗ ТОНАЛЬНОСТІ ТЕКСТУ, FLASK, PYTORCH, NLP

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

| Слово/словосполучення | Скорочення | Умови використання |
|---|---|---|
| Natural Language Processing | NLP | У тексті |
| Long short-term memory | LSTM | У тексті |
| Gated Recurrent Units | GRU | У тексті |
| Середня абсолютна помилка | MAE | У тексті |
| Convolutional neural network | CNN | У тексті |
| Recurrent neural network | RNN | У тексті |
| Середньоквадратична помилка | RMSE | У тексті |
| Bidirectional Encoder Representations from Transformers | BERT | У тексті |
| Masked Language Modeling | MLM | У тексті |

# ВСТУП

Нейромережеве моделювання є потужним інструментом в галузі штучного інтелекту та машинного навчання. Існують багато прикладів ключових застосувань та важливостей реалізації нейромережевого моделювання.

Деяки типи нейромереж можуть використовуватися для класифікації об'єктів на зображеннях, аудіофайлах або текстових даних. Вони дозволяють прогнозувати майбутні значення на основі історичних даних. Це може бути застосовано в економіці, фінансах, метеорології та інших галузях для аналізу та прогнозування.

Нейромережі використовуються для автоматичного перекладу тексту між різними мовами. Вони можуть навчатися взаємодіяти зі складними лінгвістичними особливостями та контекстами.

Системи рекомендацій використовують нейромережі для аналізу користувацьких виборів та рекомендацій продуктів, фільмів, книг тощо.

Нейромережі можуть використовуватися для розв'язання завдань обробки природної мови, таких як визначення тональності тексту, класифікація такстів і навіть вирішувати регресійні задачі на основі текстових даних.

Нейромережі допомагають створювати роботів та автономні системи, які можуть вчитися взаємодіяти з оточенням, визначати об'єкти та приймати рішення на основі отриманих даних.

В медицині нейромережі використовуються для аналізу зображень медичних знімків, діагностики захворювань, розпізнавання патологій та розробки нових методів лікування.

Застосування нейромережевого моделювання поширюється на багато галузей і відіграє ключову роль в розвитку інтелектуальних систем та технологій.

Актуальність роботи полягає в створені мікросервісу для оцінки тональності тексту, такі системи є достатньо ефективними і затребуваними, так як можуть автоматизувати процеси відслідковування настроїв та тенденцій користувачів продукту, виявлення проблем та покращення обслуговування клієнтів, а також

такого роду мікросервіси можуть допомогти підвищити ефективність маркетингових кампаній.

Метою розробки є створення прототипу мікросервісу, який базується на застосуванні сучасних нейронних мереж і може допомогти у вирішення ряду актуальних завдань.

Об’єктом дослідження є процес розробки програмних засобів реалізації нейромережевого моделювання.

Предметом дослідження є методи і практичні підходи до розробки програмних засобів реалізації нейромережевого моделювання.

Для досягнення мети були поставлені і вирішені наступні завдання:

– проаналізовано вплив нейронних мереж на бізнес та людське життя;

– проаналізовано напрямки нейромережевого моделювання;

– надані рекомендації щодо апаратного та програмного забезпечення необхідного для розробки і використання нейронних мереж;

– реалізовано задачу класифікації тексту класичними алгоритмами машинного навчання;

– побудовано моделі для вирішення задач обробки природної мови за допомоги нейронних мереж;

– розроблено моделі класифікації зображень і нейромережеві моделі прогнозування часових рядів;

– розроблено мікросервіс аналізу тональності тексту.

Наукова новизна. Проведено аналіз застосування різноманітних методів машинного навчання, у тому числі і нейронних мереж для вирішення задач на основі даних зі складною структурою, таких як тестові дані, зображення, часові ряди. Визначено основні особливості розробки мікросервісів з використаням нейронних мереж, які являють собой зручни інструмент, який дозволяє автоматизувати деяку частину роботи аналітиків, маркетологів, служби підтримки і т.д.

Практична цінність роботи полягає в наступному. Під час виконання роботи був розроблений мікросервіс оцінки тональності тексту на базі нейронної мережі, який є універсальним і може бути застосований для автоматизації низькі задач різного роду спціалістів, а саме аналітиків, маркетологів, служби підтримки і т.п. Реалізований мікросервіс може працювати як на базі локального хосту, так і глобального.

Апробація теоретичних положень роботи відбулася 6 грудня 2023 року у ході XXV науково-практичної студентської конференції у ПрАТ «ПВНЗ «Запорізький інститут економіки та інформаційних технологій».

Магістерська дипломна робота складається зі вступу, трьох розділів та висновків на 94 сторінки машинописного тексту. Робота містить 7 таблиць, 60 рисунків, 3 додатки та перелік посилань з 27 найменувань

# РОЗДІЛ 1
# НЕЙРОМЕРЕЖЕВЕ МОДЕЛЮВАННЯ: ОСНОВНІ НАПРЯМКИ

## 1.1. Вплив нейронних мереж на бізнес та людське життя

Машинне навчання, як і нейронні мережі вже давно стали частиною повсякденного життя. Кожна людина у своїй повсякденній діяльності використовує різноманітні додатки в архітектуру яких вбудовані різного роду і складності моделі машинного навчання. Складно виділити якусь сферу людського життя що не зазнала впливу машинного навчання.

Дійсно, нейронні мережі, які є ключовим компонентом штучного інтелекту (artificial intelligence), суттєво вплинули на різні аспекти людського життя, починаючи від медицини, фінансів, освіти, виробництва продукції і закінчуючи сферою розваг.

Якщо розглядати сферу бізнесу, то вплив нейронних мереж можна охарактеризувати наступними словами: ефективність, продуктивність і інноваційність. Наприклад, бізнес, задля підвищення конкурентноспрожності, може використовувати сучасні технології для розробки нових продуктів та послуг, які відповідають потребам клієнтів, персоналізоване обслуговування клієнтів, автоматизацію повсякденних завдань .

В таблиці 1 деталізовано сфери, які зазнали трансформаційні зміни завдяки нейронним мережам.

Таблиця 1

Сфери на які вплинув розвиток нейронних мереж

| Сфера | Трансформаційні зміни |
|---|---|
| Автоматизація та ефективність:<br><br>Промисловість і виробництво | Нейронні мережі забезпечують автоматизацію в промисловості завдяки використанню робототехніки та інтелектуальних систем, підвищуючи ефективність і скорочуючи кількість людської праці при виконанні повторюваних завдань |
| Бізнес-процеси | у бізнесі нейронні мережі застосовуються для оптимізації різних процесів, від обслуговування клієнтів до управління ланцюгом постачання, що призводить до підвищення продуктивності |
| Фінанси:<br><br>Виявлення шахрайства. | Нейронні мережі відіграють вирішальну роль у фінансових установах для виявлення шахрайських дій шляхом аналізу шаблонів і аномалій у транзакціях, підвищення безпеки |
| Алгоритмічна торгівля | На фондовому ринку нейронні мережі використовуються для алгоритмічної торгівлі, прогнозування на основі ринкових даних і оптимізації інвестиційних стратегій |
| Охорона здоров'я:<br><br>Діагностика, лікування та | Нейронні мережі використовуються в медичній візуалізації для більш точної діагностики, допомагаючи виявляти такі захворювання, як рак, на ранніх стадіях. Вони також роблять внесок у персоналізовану медицину, аналізуючи дані пацієнтів для адаптації планів лікування |
| Транспорт:<br><br>Автономні транспортні засоби. | Нейронні мережі є фундаментальною технологією для безпілотних автомобілів, які допомагають їм сприймати навколишнє середовище, приймати рішення та безпечно керувати транспортними засобами |
| Спілкування та обробка мови:<br><br>Обробка природної мови (NLP): | нейронні мережі забезпечують переклад мови, розпізнавання голосу та чат-ботів, покращуючи спілкування між людьми, які розмовляють різними мовами, і полегшують взаємодію між людиною та комп’ютером. |
| Розваги:<br><br>Рекомендаційні системи щодо контенту | Нейронні мережі стоять за системами рекомендацій на таких платформах, як Netflix, Spotify, Amazon, надаючи персоналізовані пропозиції щодо вмісту на основі вподобань користувачів. |

Продовження таблиці 1

| | |
|---|---|
| Ігри | в ігровій індустрії нейронні мережі використовуються для створення більш реалістичних персонажів і середовища, а також для покращення ігрового досвіду за допомогою адаптивного ШІ. |
| Освіта:<br><br>Адаптивне навчання | Нейронні мережі сприяють персоналізованому навчанню, адаптуючи навчальний контент до потреб і стилів навчання окремих учнів. |
| Наукові дослідження:<br><br>Економіка, фізика, біологія та інші. | У різних галузях науки нейронні мережі використовуються для аналізу даних, розпізнавання образів і моделювання, допомагаючи дослідникам у вирішенні складних проблем. |
| Безпека та аутентифікація | Нейронні мережі підвищують безпеку за допомогою систем розпізнавання обличчя та голосової аутентифікації, додаючи додатковий рівень захисту |

Таким чином, в сучасному житті нейронні мережі стали невід'ємною частиною технологічного прогресу, граючи ключову роль в різних галузях людського життя. Розроблені системи та технологічні рішення демонструють можливості та по суті трансформують сприйняття та взаємодію людини зі світом. Однак, незважаючи на останні досягнення, впровадження нейронних мереж порушує питання етики, приватності та соціальної справедливості.

## 1.2. Напрямки нейромережевого моделювання

Як вже зазначалось, нейронні мережі достатньо потужний інструмент для вирішення широкого кола задач. В першу чергу до таких задач можна віднести класичні задачі машинного навчання, а саме регресія та класифікація. Однак, вділяють і більш специфічні задачі, а саме задачі детекції, генерації та ін.

Слід зазначити, що нейронні мережі показують дуже обнадійливі результати саме при роботі з так званими складними структурами даних до яких можна віднести: зображення, тексти, часові ряди. Відповідно до зазначених складних структур даних виділяють наступні задачі: машинний зір, обробка природної мови та прогнозування.

Машинний зір (англ. Computer Vision) — це галузь computer science, яка займається розробкою та застосуванням алгоритмів та систем для обробки та аналізу зображень. Системи, які працюють на основі машинного зору можуть використовуватися для виявлення, розпізнавання та класифікації об'єктів у зображеннях [2].

Машинний зір має широкий спектр застосувань, таких як [15]:

- автоматизований контроль якості;
- автоматизоване розпізнавання облич;
- ветеринарія;
- медицина;
- наука;
- безпека;
- розваги.

Машинний зір може використовуватися для виконання таких завдань, як:

- Виявлення об'єктів у зображенні
- Розпізнавання об'єктів у зображенні
- Класифікація об'єктів у зображенні
- Вимірювання розмірів об'єктів у зображенні
- Виявлення руху в зображенні
- Створення цифрових моделей об'єктів у зображенні

Машинний зір є складною галуззю, яка потребує знань з математики, статистики, теорії ймовірності, інформатики, програмування та ін.

Ось кілька прикладів машинного зору в реальному світі:

- автомобілі, які управляються автопілотом використовують машинний зір для виявлення інших автомобілів, пішоходів і перешкод;

– системи розпізнавання облич використовуються для безпеки та контролю доступу;

– системи візуалізації та аналізу медичного медичної інформації використовують машинний зір для виявлення захворювань;

– системи контролю якості використовують машинний зір для перевірки продукції на наявність дефектів.

Повноцінні системи машинного зору зазвичай складаються з наступних компонентів [2]:

– камера - це пристрій, який фіксує зображення;

– процесор - це пристрій, який обробляє зображення, причому це може бути як центральний процесор, так і графічний процесор, який є більш ефективним для роботи з тензорами;

– програмне забезпечення, яке забезпечую поєднання апаратних компонент для виконання поставлених завдань.

Таблиця 2

Етапи процесу проведення дослідження за допомоги машинного зору

| Етап | Назва етапу | Опис процесу |
|---|---|---|
| I | Збір даних | процес отримання зображень з камери |
| II | Обробка зображень | процес підготовки зображень до подальшого аналізу |
| III | Аналіз зображень | процес виявлення, розпізнавання та класифікації об'єктів у зображеннях |
| IV | Відтворення результатів | процес надання результатів аналізу користувачеві |

Деякі з найпоширеніших алгоритмів машинного зору включають:

– алгоритми детекції об'єктів - це алгоритми, які виявляють об'єкти на зображенні;

– алгоритми розпізнавання об'єктів - це алгоритми, які ідентифікують об'єкти на зображенні.

– алгоритми класифікації об'єктів - це алгоритми, які класифікують об'єкти на зображенні за певними категоріями.

– алгоритми вимірювання розмірів об'єктів - це алгоритми, які вимірюють розміри об'єктів на зображенні.

– алгоритми виявлення руху - це алгоритми, які виявляють рух об'єктів на зображенні.

– алгоритми створення цифрових моделей об'єктів - це алгоритми, які відтворюють цифрові моделі об'єктів у зображенні.

Декілька прикладів того, як машинний зір використовується в реальному світі:

– автомобілі з автопілотом, використовують машинний зір для виявлення інших автомобілів, пішоходів і перешкод.

– системи розпізнавання облич використовуються для безпеки та контролю доступу.

– системи медичного візуалізації використовують машинний зір для виявлення захворювань.

– системи контролю якості використовують машинний зір для перевірки продукції на наявність дефектів.

– системи розпізнавання товарів використовують машинний зір для сортування товарів на складах, сигналізують про відсутність товарів на полицях.

– системи догляду за здоров'ям тварин використовують машинний зір для виявлення захворювань у тварин.

– системи точного землеробства використовують машинний зір для оцінки стану полів.

Технології машинного зору має колосальний потенціал, його активно будуть застосовувати у військовій і цивільній сфері, для створення безпілотних апаратів, що надасть цьому напрямку додатковий поштовх на розвиток.

Обробка природної мови (NLP) — це галузь computer science, яка займається розробкою та застосуванням алгоритмів та систем для розуміння та генерації

природної мови. Нейронні мережі (NN) — це один із найпопулярніших інструментів вирішення проблем NLP [12].

Нейронні мережі можуть бути використані для виконання широкого спектру завдань NLP, таких як:

- розпізнавання мови — це процес розпізнавання слів та речень у тексті;
- переклад мов — це процес перекладу тексту з однієї мови на іншу;
- синтетизація мови (генерування тексту заданою мовою) — це процес створення тексту, який відповідає заданим критеріям;
- відповіді на запитання — це процес надання відповідей на запитання у вигляді тексту;
- аналіз тексту — це процес виявлення тем та тенденцій у текстах.

Штучні нейронні мережі мають ряд переваг перед традиційними методами NLP. Вони можуть навчатися на великих обсягах даних, що дозволяє їм досягати високої точності. Вони також можуть бути адаптовані до нових завдань, що робить їх більш гнучкими, ніж традиційні методи.

Однак нейронні мережі також мають деякі недоліки. Нейромережеві моделі можуть бути складними та дорогими для навчання та розуміння, а також чутливими до шуму в даних, що може призвести до зниження точності.

Незважаючи на ці недоліки, нейронні мережі є потужним інструментом для NLP, які вже використовуються в багатьох комерційних продуктах і послугах, і їх використання, швидше за все, буде зростати в майбутньому.

Зазначимо деякі приклади, використання нейронних мереж для вирішення задач NLP [12]:

- Google Translate використовує нейронні мережі для перекладу текстів з однієї мови на іншу;
- Alexa використовує нейронні мережі для розуміння та відповіді на запитання;
- Siri використовує нейронні мережі для розуміння та відповіді на запитання;

– Grammarly використовує нейронні мережі для перевірки граматики та стилю;

– SpamAssassin використовує нейронні мережі для виявлення спаму.

Нейронні мережі мають потенціал змінити багато аспектів обробки природної мови, можуть зробити NLP більш точним, ефективним і доступним, яскравим прикладом таких тенденцій є ChatGPT. Не виключено, що можуть бути відкрити нові можливості для застосування NLP у різних сферах.

Наступною задачею є прогнозування за допомогою нейронних мереж — це процес використання нейронних мереж для створення прогнозів про майбутні події [16]. У якості таких подій можуть виступати:

– ціни на акції;

– погода;

– прогнозування споживання газу домогосподарствами;

– попит на будь які товари та послуги;

– ризики катастроф;

– поведінка споживачів.

Нейронні мережі мають ряд переваг перед традиційними методами прогнозування. Вони можуть навчатися на великих обсягах даних та виявляти скриті тенденції, що дозволяє їм досягати високого рівня точності. Дані методи також можуть бути адаптовані до нових даних, що робить їх більш гнучкими, ніж традиційні методи.

Слід зазначити, що нейронні мережі також мають деякі недоліки. Серед них: складність навчання та проблеми з інтерпритованістю, підвищена чутливість до шуму в даних, що може призвести до зниження точності даного класу моделей.

Незважаючи на ці недоліки, нейронні мережі є потужним інструментом для прогнозування, які використовуються в багатьох комерційних продуктах і послугах, не виключено, що їх застосування для вирішення схожих проблем буде зростати в майбутньому.

Зазначимо кілька прикладів того, як нейронні мережі використовуються для прогнозування різні компанії та служби [17]:

– фінансові компанії використовують нейронні мережі для прогнозування цін на акції;

– метеорологічні служби використовують нейронні мережі для покращення результатів пов’язаних з прогнозом погодних умов;

– маркетингові компанії використовують нейронні мережі для прогнозування попиту на товари та послуги;

– страхові компанії використовують нейронні мережі для прогнозування ризиків катастроф;

– компанії, які використовують нейронні мережі для прогнозування поведінки споживачів.

Для вирішення задач прогнозування використовують наступні типи нейронних мереж. Деякі з найпоширеніших типів включають:

– перцептрони — це прості нейронні мережі, які можуть бути використані для прогнозування лінійних відносин;

– рекурентні нейронні мережі — це нейронні мережі, які можуть бути використані для прогнозування нелінійних відносин;

– глубокі нейронні мережі — це нейронні мережі, які мають багато шарів, блоки LSTM та GRU, що дозволяє їм навчатися на складніших наборах даних.

Вибір нейронної мережі для прогнозування залежить від кількох факторів, включаючи:

– тип даних, які будуть використані для прогнозування;

– складність прогнозованої залежності;

– доступність даних для навчання.

Перед тим, як використовувати нейронну мережу для прогнозування, дані потрібно підготувати. Підготовка даних включає в себе ряд таких кроків, як:

– вибір правильних змінних для прогнозування;

– очищення даних від шуму;

– нормалізація даних.

Після того, як дані були підготовлені, нейронну мережу потрібно навчити на заданому наборі даних.

Після навчання нейронну мережу можна використовувати для прогнозування нових значень на основі нових вхідних даних.

При цьому оцінка точності прогнозу, можна визначити порівнявши його з фактичними значеннями (метрика MAE). Цей процес називається оцінкою точності.

Прогнозування за допомогою нейронних мереж є складним процессом, але воно дає вражаючи результати, а точність розроблених моделей постійно покращується.

Таким чином, нейромережеве моделювання охоплює широкий спектр напрямків та завдань, вони можуть бути використані для розв'язання різних проблем. Розвиток та дослідження в цій області здійснюються дуже активно, щорічно видять нові версії популярних архітектур і з’являються принципово нові підходи, відкриваючи нові можливості для застосування нейромереж у різних сферах життя.

## 1.3. Вимоги до апаратного та програмного забезпечення для розробки нейронних мереж

Для розробки нейронних мереж потрібно певне апаратне і програмне забезпечення, особливо якщо йдеться про тренування моделей «з нуля» на великих обсягах даних. У таблиці 3 наведені, загальні рекомендації щодо апаратного забезпечення для розробки нейронних мереж.

Таблиця 3

Загальні рекомендації щодо апаратного забезпечення для розробки нейронних мереж

| Тип апаратного забезпечення | Рекомендації |
|---|---|
| Центральний процесор (CPU) | Багато моделей машинного навчання у тому числі нейронних мереж можна тренувати на стандартних багатоядерних CPU. Для цього можуть підійти процесори сімейства Intel Core i7, Intel Core i9 або професійні процесори Intel Xeon (так звані серверні процесори). |

| | |
|---|---|
| | Однак, якщо говорити про тренування сучасних моделей з мільйонами і навіть більше параметрів, то використання для цих цілей центрального процесора є не дуже вдалою ідеєю. Звісно, що це можливо, але буде необхідно дуже багато часу. |

Продовження таблиці 3

| | |
|---|---|
| Графічний процесор (GPU): Мінімальні вимоги: | Для прискорення тренування нейронних мереж важливо мати графічний процесор (GPU). Мінімальні вимоги це NVIDIA GeForce GTX 1060 або аналогічні GPU від AMD можуть підійти для стандартних завдань. Слід зазначити, що важливом також є об’єм пам’яті відео карти і чи він більший, тим краще.<br>Для більш високої продуктивності та прискорення навчання глибоких нейронних мереж доцільно використовувати більш потужні GPU, такі як NVIDIA GeForce RTX 30 серії або професійні GPU NVIDIA Tesla або Quadro.<br>Використання GPU дає змогу прискорити навчання моделей у 4 рази і це мінімальний показник прискорення. |
| Тензорні процесори (TPU) | У разі використання інфраструктури хмарних сервісів для розробки та навчання моделей можна також розглянути можливість використання тензорних процесорів (наприклад, Google TPU), які спеціально розроблені для завдань машинного навчання.<br>Слід зазначити, що тензорні процесори доступні на безоплатній основі у блокнотах Google Colab та Kaggle. Нажаль, є обмеження по часу використання |
| Об'єм оперативної пам'яті (RAM): | Мінімальні вимоги це 16 ГБ оперативної пам'яті, але тут такі самі рекомендації як для GPU, чим більше тим краще. Рекомендований об’єм 32 ГБ |
| Сховище даних | Рекомендовано використовувати SSD з достатнім обсягом для встановлення програмного забезпечення та зберігання даних. |
| Інфраструктура хмарних сервісів | Для більш складних завдань та тренування великих моделей може бути доцільно використовувати інфраструктуру хмарних сервісів (наприклад, AWS, Azure, Google Cloud), де доступні потужні обчислювальні ресурси та гнучкість масштабування. |

Для розробки нейронних мереж використовується різноманітне програмне забезпечення, включаючи бібліотеки, фреймворки та інструменти. Можна виділити кілька популярних програмних рішень для розробки нейронних мереж.

У якості бібліотек та фреймворків для глибокого навчання використовують TensorFlow, який підтримує широкий спектр додатків, а також може взаємодіяти з великими обчислювальними кластерами. PyTorch, який має зручний та гнучкий

інтерфейс, достатньо популярний у співтоваристві дослідників та фахівців з глибокого навчання.

Також можна виділити Keras - це високорівнева бібліотека для побудови нейронних мереж, яка може використовуватися поверх TensorFlow або Theano (наразі не такий популярний). Переваги та недоліки вище перелічених бібліотек ще будуть розкриті пізніше.

MXNet, достатньо ефективний та гнучкий фреймворк, що підтримує різні рівні абстракції. Наділь дещо втратив частку популярності, але все ж таки залишається потужним інструментом, хоча і програє у популярності TensorFlow та PyTorch.

Доволі зручно використовувати інструменти візуалізації та налагодження. До такиї інструментів можна віднести: TensorBoard (для TensorFlow), який є інструментом візуалізації для відстеження процесу навчання та аналізу архітектури моделі; Visdom (для PyTorch) інтерактивна візуалізація моніторингу навчання PyTorch моделей, хоча є і реалізація TensorBoard для PyTorch. Бібліотеки Matplotlib та Seaborn широко використовуються для візуалізації даних та результатів.

Слід також виділити інтегровані середовища розробки (IDE), а саме Jupyter Notebook, це інтерактивне середовище розробки для створення та обміну документами, що містять код, візуалізацію та текст, а також його хмарні «аналоги» Google Colab, Kaggle Notebook. Саме ці дві IDE будуть використані при реалізації моделей, а в подальшому для розробки мікросервісу буде використаний PyCharm, який є популярним інтегрованим середовищем розробки для написання, налагодження та тестування коду.

В одному з розділів, буде також використовуватись такий спеціалізований інструмент як NLTK (Natural Language Toolkit) для демонстрації її можливостей в задачах обробки природної мови.

Вибір апаратного забезпечення також залежить від конкретних вимог завдання та доступності ресурсів. Важливо також враховувати, що деякі завдання, такі як тренування глибоких моделей з великою кількістю параметрів можуть зажадати значних ресурсів.

Що стосується вибору конкретних інструментів, то все залежить від конкретного завдання, вподобань розробника та рівня досвіду. Також слід враховувати елементи корпоративної культури та етики, багато компаній забороняють використовувати хмарні сервіси з метою захисту конфіденційних даних та комерційної таємниці.

## 1.4. Висновки до розділу

В сучасному житті нейронні мережі стали невід'ємною частиною технологічного прогресу, граючи ключову роль в різних галузях людського життя. Розроблені системи та технологічні рішення демонструють можливості та по суті трансформують сприйняття та взаємодію людини зі світом. Однак, незважаючи на останні досягнення, впровадження нейронних мереж порушує питання етики, приватності та соціальної справедливості.

Нейромережеве моделювання охоплює широкий спектр напрямків та завдань, вони можуть бути використані для розв'язання різних проблем. Розвиток та дослідження в цій області здійснюються дуже активно, щорічно виходять нові версії популярних архітектур і з’являються принципово нові підходи, відкриваючи нові можливості для застосування нейромереж у різних сферах життя.

Вибір апаратного забезпечення та конкретних інструментів напряму залежить від конкретних вимог завдання, доступності ресурсів, вподобань розробника та рівня досвіду. Важливо також враховувати, що деякі завдання, такі як тренування глибоких моделей з великою кількістю параметрів можуть зажадати значних ресурсів. Важливим також є вираховування елементів корпоративної культури та етики, багато компаній забороняють використовувати хмарні сервіси з метою захисту конфіденційних даних та комерційної таємниці.

# РОЗДІЛ 2

# ВИКОРИСТАННЯ МОВИ ПРОГРАМУВАННЯ PYTHON ДЛЯ СТВОРЕННЯ НЕЙРОННИХ МЕРЕЖ: ІНСТРУМЕНТИ, БІБЛІОТЕКИ

## 2.1. Основні бібліотеки для розробки нейронних мереж

Найбільш популярними бібліотеками для розробки нейронних мереж на мові програмування Python є TensorFlow від Google, Keras, як деяка надбудова для полегшення роботи з TensorFlow і Pytorch від Facebook. Необхідно представити більш детальну інформацію про ці бібліотеки.

TensorFlow — це відкрита бібліотека для машинного навчання, розроблена Google AI. Вона використовується для створення і навчання нейронних мереж для широкого спектру завдань, таких як розпізнавання образів, обробка природної мови та прогнозування [19].

Основні особливості використання бібліотеки TensorFlow [18]:

- підтримка різних типів нейронних мереж. TensorFlow підтримує широкий спектр типів нейронних мереж, включаючи згорткові нейронні мережі, рекурентні нейронні мережі та глибокі нейронні мережі;
- підтримка різних типів даних. TensorFlow може працювати з різними типами даних, включаючи числові дані, текстові дані та зображення;
- підтримка різних платформ. TensorFlow можна використовувати на різних платформах, включаючи Linux, macOS, Windows, Android і iOS.

Підтримка різних типів даних

TensorFlow може працювати з різними типами даних, включаючи:

- числові дані — це дані, які можна представити у вигляді чисел, наприклад, ваги і висота людини;

– текстові дані — це дані, які можна представити у вигляді тексту, наприклад, статті та книги;

– зображення — це дані, які можна представити у вигляді зображень, наприклад, фотографії і картини.

Підтримка різних платформ

TensorFlow можна використовувати на різних платформах, включаючи:

– Linux — операційна система, яка використовується на серверах і суперкомп'ютерах, є опенсорсним продуктом, широко застосується комерційними та некомерційними організаціями, є безкоштовною;

– macOS — це операційна система, яка використовується на персональних комп'ютерах розробником яких є компанія Apple;

– Windows — це операційна система, яка використовується на персональних комп'ютерах;

– Android — це операційна система, яка використовується на смартфонах і планшетах;

– iOS — це операційна система, яка використовується на iPhone і iPad, розробником якої також є компанія Apple.

Бібліотека TensorFlow також має ряд інших особливостей, які роблять його потужним інструментом для машинного навчання. До них відносяться [19]:

– підтримка розподіленого навчання. TensorFlow можна використовувати для навчання нейронних мереж на розподілених системах, що дозволяє підвищити продуктивність.

– підтримка GPU. TensorFlow можна використовувати з GPU, що може значно прискорити навчання нейронних мереж.

– підтримка інструментів для розробки. TensorFlow включає в себе ряд інструментів для розробки, таких як графічний інтерфейс і консоль.

TensorFlow — це потужна бібліотека для машинного навчання, яка може використовуватися для створення і навчання нейронних мереж для вирішення широкого спектру завдань. Вона підтримує широкий спектр типів нейронних

мереж, типів даних і платформ, що робить її універсальним інструментом для машинного навчання.

Keras - це відкрита бібліотека для машинного навчання, написана на Python. Вона використовується для створення і навчання нейронних мереж. Keras є високорівневою бібліотекою, яка побудована на основі TensorFlow [18]. Це означає, що Keras забезпечує простий і зручний інтерфейс для використання TensorFlow.

Keras має ряд переваг, включаючи:

– Простота використання. Keras має простий і зручний інтерфейс, який робить його легким у використанні для початківців.

– Модульність. Keras є модульною бібліотекою, що дозволяє користувачам легко створювати і налаштовувати свої власні нейронні мережі.

– Ефективність. Keras використовує TensorFlow, який є високоефективною бібліотекою машинного навчання.

Keras використовується для широкого спектру завдань машинного навчання, включаючи:

– Класифікація. Keras можна використовувати для класифікації даних, наприклад, для визначення того, чи є зображення собакою чи котом.

– Регресія. Keras можна використовувати для прогнозування даних, наприклад, для прогнозування цін на акції або погоди.

– Обробка природної мови. Keras можна використовувати для обробки природної мови, наприклад, для перекладу мов або створення чат-ботів.

Приклад побудови прогнозної моделі рекурентної нейронної мережі на Keras, яка призначена для прогнозування послідовних типів даних (sequential data types) наведено на рисунку 2.1.

Наведемо деякі з основних функцій бібліотеки Keras [18]:

– підтримка різних типів нейронних мереж, включаючи згорткові нейронні мережі, рекурентні нейронні мережі та змішані мережі;

– підтримка різних типів даних, а саме графічні образи, текст і аудіо;

– підтримка різних оптимізаторів, які використовуються для навчання нейронних мереж (наприклад, Adam);

– підтримка різних метрик або loss function, які використовуються для оцінки точності нейронних мереж.

```
model = tf.keras.models.Sequential([
        tf.keras.layers.LSTM(100, input_shape=(window_size,1), return_sequences=True),
        tf.keras.layers.LSTM(50, return_sequences=True),
        tf.keras.layers.LSTM(10),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(16, activation='relu'),
        tf.keras.layers.Dense(1)
])

model.compile(loss='mse',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['mape'])

lstm_model = model.fit(train,epochs=100)
```

Рис. 2.1. Приклад прогнозної моделі на основі рекурентної нейронної мережі (RNN) з LSTM модулями, яка розроблена за допомоги Keras.

Наведемо деякі з основних функцій бібліотеки Keras [19]:

– підтримка різних типів нейронних мереж, включаючи згорткові нейронні мережі, рекурентні нейронні мережі та змішані мережі;

– підтримка різних типів даних, а саме графічні образи, текст і аудіо;

– підтримка різних оптимізаторів, які використовуються для навчання нейронних мереж (наприклад, Adam);

– підтримка різних метрик або loss function, які використовуються для оцінки точності нейронних мереж.

Keras - це потужний інструмент для машинного навчання. Він простий у використанні і може бути використаний для широкого спектру завдань.

PyTorch - це відкрита бібліотека для машинного навчання, написана на Python. Вона використовується для створення і навчання нейронних мереж. PyTorch є високорівневою бібліотекою, яка дозволяє користувачам легко створювати і налаштовувати свої власні нейронні мережі [15].

PyTorch має ряд переваг, включаючи:

– динамічне графування. PyTorch використовує динамічне графування, що означає, що граф обчислень будується по мірі виконання коду. Це робить PyTorch більш гнучким і потужним, ніж бібліотеки, які використовують статичне графування;

– автоматичне диференціювання. PyTorch використовує автоматичне диференціювання для обчислення градієнтів функції втрат. Це дозволяє користувачам легко навчати нейронні мережі за допомогою алгоритмів градієнтного спуску;

– швидкість і ефективність. PyTorch використовує графічні процесори (GPU) для прискорення обчислень. Це робить PyTorch ідеальним для завдань машинного навчання, які вимагають великої кількості обчислень.

PyTorch використовується для широкого спектру завдань машинного навчання, включаючи [21]:

– класифікація. PyTorch можна використовувати для класифікації даних, наприклад, для визначення того, чи є зображення певним об’єктом чи ні.

– регресія. PyTorch можна використовувати для прогнозування даних, наприклад, для прогнозування цін на нерухомість або прогнозу погоди.

– обробка природної мови. PyTorch можна використовувати для обробки природної мови, наприклад, для перекладу з інших мов або створення чат-ботів.

PyTorch є потужним інструментом для машинного навчання. Він має ряд переваг, які роблять його ідеальним для широкого спектру завдань.

Приклад архітектури простої загорткової нейронної мережі (CNN) моделі побудованої за допомоги PyTorch наведено на рисунку 2.2.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.convo = nn.Conv2d(3, 16, 5) #224
        self.pool = nn.MaxPool2d(2, 2) #220
        self.conv2 = nn.Conv2d(16, 32, 5) #110
        self.dropout = nn.Dropout(0.4) #106
        self.fc1 = nn.Linear(32*53*53, 256*2)
        self.fc2 = nn.Linear(256*2, 210)
        self.fc3 = nn.Linear(210, num_classes)
        self.softmax = nn.LogSoftmax(dim=1)


    def forward(self, x):
        x = self.pool(F.relu(self.convo(x))) #before convo: 224 - after convo: 220
        x = self.pool(F.relu(self.conv2(x))) #110 - 106 - 53
        x = self.dropout(x)
        x = x.view(-1, 32 * 53 * 53)
        x = F.relu(self.fc1(x))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.softmax(self.fc3(x))
        return x

model = Net()
```

Рис. 2.2. Приклад архітектури простої CNN моделі побудованої за допомоги PyTorch

Також можна виділити такі переваги використання PyTorch:

– простота використання. PyTorch має простий і зручний інтерфейс, який робить його легким у використанні для початківців;

– модульність. PyTorch є модульною бібліотекою, що дозволяє користувачам легко створювати і налаштовувати свої власні нейронні мережі;

– ефективність. PyTorch використовує GPU для прискорення обчислень, що робить його ідеальним для завдань машинного навчання, які вимагають великої кількості обчислень;

– підтримка різних типів нейронних мереж. PyTorch підтримує різні типи нейронних мереж, включаючи згорткові нейронні мережі, рекурентні нейронні мережі та змішані мережі;

– підтримка різних типів даних. PyTorch підтримує різні типи даних, включаючи образи, текст і аудіо;

– підтримка різних оптимізаторів. PyTorch підтримує різні оптимізатори, які використовуються для навчання нейронних мереж;

– підтримка різних метрик. PyTorch підтримує різні метрики, які використовуються для оцінки точності нейронних мереж.

Так серед переваг бібліотеки по створенню нейронних мереж можна виділити:

– PyTorch є відкритою бібліотекою, що означає, що її можна безкоштовно використовувати та поширювати. Це робить її доступною для широкого кола користувачів, включаючи студентів, науковців та розробників.

– PyTorch має велику і активну спільноту, яка надає підтримку та ресурси. Це може бути корисним для початківців, які хочуть почати роботу з PyTorch.

– PyTorch постійно розвивається і покращується. Це означає, що користувачі завжди можуть отримувати доступ до найновіших функцій і поліпшень.

Існує безліч прикладів використання бібліотеки PyTorch в реальних проектах [21]:

– Google використовує PyTorch для розробки своїх продуктів, таких як Google Vision і Google Translate.

– Facebook використовує PyTorch для розробки своїх продуктів, таких як Facebook AI і Instagram.

– OpenAI використовує PyTorch для розробки своїх продуктів, таких як GPT-3.5 і Dactyl.

PyTorch є потужним і універсальним інструментом для машинного навчання. Він використовується в широкому спектрі завдань і є популярним вибором серед експертів і початківців.

Ось кілька додаткових переваг використання PyTorch:

– PyTorch забезпечує високий рівень контролю над процесом навчання нейронних мереж. Це дозволяє користувачам експериментувати з різними параметрами і архітектурами для отримання найкращої точності.

– PyTorch є гнучким і масштабованим. Він може бути використаний для вирішення завдань будь-якого масштабу, від невеликих навчальних наборів даних до великих наборів даних, що містять мільйони зразків.

– PyTorch є ефективним. Він використовує графічні процесори (GPU) для прискорення обчислень, що робить його ідеальним для завдань машинного навчання, які вимагають великої кількості обчислень.

В цілому, PyTorch є потужним і універсальним інструментом для машинного навчання. Він має ряд переваг, які роблять його ідеальним для вирішення широкого спектру завдань, причому він має достатньо низький поріг входу.

## 2.2 Інструменти реалізації нейронних мереж

Flask - це мікрофреймворк для веб-розробки на Python. Він був створений програмістом Мішелем Піньоле в 2010 році. Flask є популярним вибором для розробки веб-додатків Python, оскільки він простий у використанні, але все ж потужний і гнучкий.

Таблиця 4

Основні характеристики фреймворку Flask

| Характеристика | Опис характеристики |
|---|---|
| Мікрофреймворк | Flask не включає в себе багато функцій з коробки, але дозволяє легко додавати їх за допомогою сторонніх бібліотек. Це робить його простим у використанні, але все ж потужним і гнучким |
| Простота використання | Flask має простий і зрозумілий інтерфейс, що робить його легким у використанні для початківців |

Продовження таблиці 4

| | |
|---|---|
| Гнучкість | Flask є гнучким фреймворком, що дозволяє розробникам легко налаштовувати його під свої потреби |
| Масштабованість | Flask може бути використаний для розробки веб-додатків будь-якого масштабу, від невеликих сайтів до великих веб-сервісів |
| Швидкість і ефективність | Flask є швидким і ефективним фреймворком, оскільки він не включає в себе багато функцій з коробки |
| Спільнота | Flask має велику і активну спільноту, яка надає підтримку та ресурси користувачам |

Фреймворк Flask працює на основі моделі WSGI (Web Server Gateway Interface). WSGI - це стандартний спосіб взаємодії веб-серверів з веб-фреймворками.

Flask складається з наступних основних компонентів:

– основний файл. Основний файл - це файл Python, який містить код для запуску Flask;

– контролери. Контролери відповідають за обробку HTTP-запиту і повернення відповіді;

– шаблони. Шаблони використовуються для генерації HTML-коду для відображення веб-сторінок;

– бази даних. Flask підтримує роботу з різними типами баз даних, такими як MySQL, PostgreSQL і SQLite.

Для початку роботи з Flask потрібно його інсталювати. Для цього можна використовувати менеджер пакетів pip:

pip install flask

Після встановлення Flask можна створити новий проект. Для цього можна використовувати наступний код:

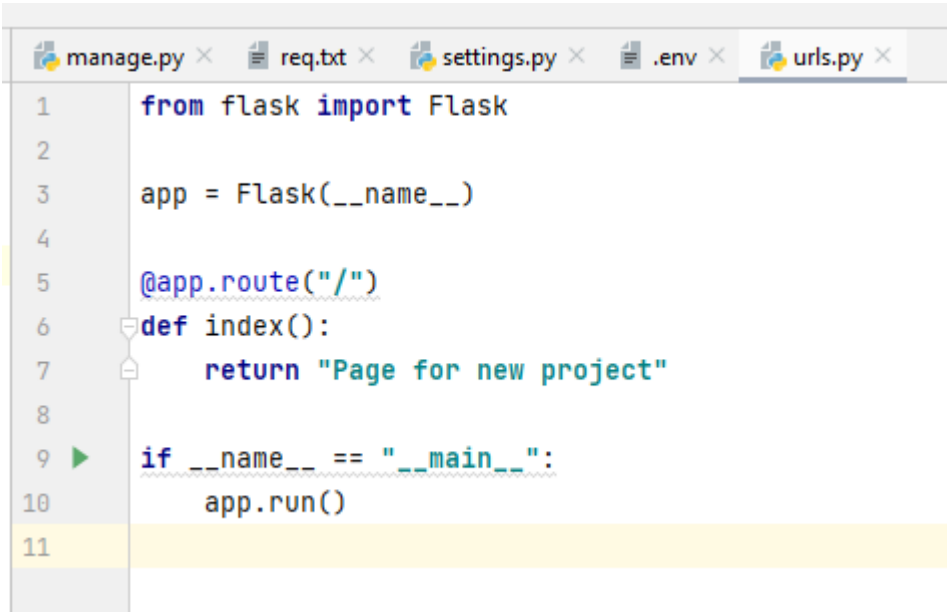


Рис. 2.3. Створення нового проєкту на Flask

Наведений на рисунку 2.3 код створює простий веб-сайт з однією сторінкою, яка відображає текст "Page for new project".

Розробка веб-додатків на Flask

Flask можна використовувати для розробки широкого спектру веб-додатків, включаючи:

– прості веб-сайти. Flask можна використовувати для розробки простих веб-сайтів, таких як особисті веб-сторінки або блоги;

– веб-додатки. Flask можна використовувати для розробки веб-додатків, таких як інтернет-магазини, системи управління контентом (CMS) та соціальні мережі;

– веб-сервіси. Flask можна використовувати для розробки веб-сервісів, таких як API та мікросервіси.

На рисунку 2.4 наведено приклад роботи з мікрофреймворком Flask.

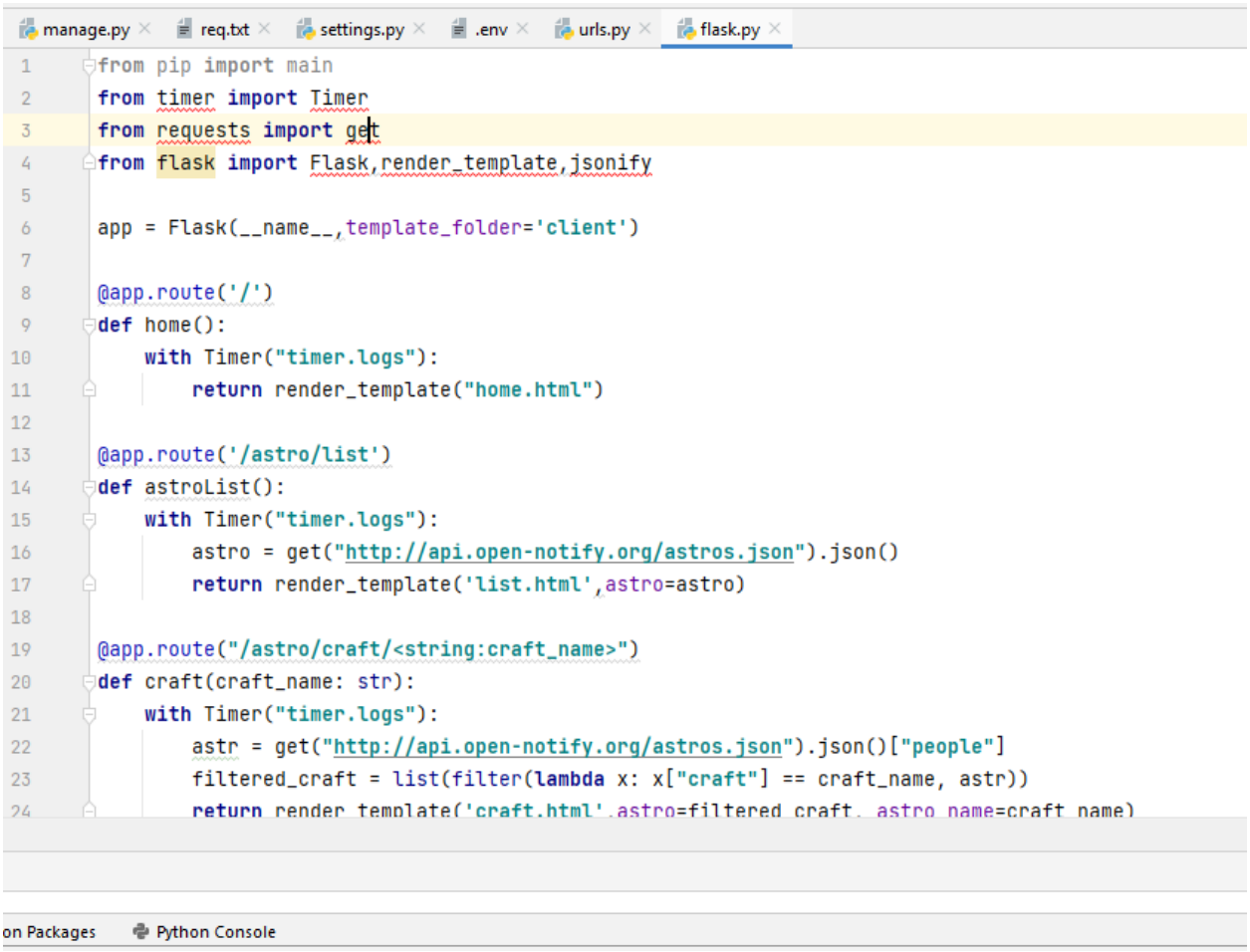


Рис. 2.4. Приклад розробки мікросервісу на Flask

Для розробки веб-додатків на Flask можна використовувати сторонні бібліотеки, які надають додаткові функції, такі як підтримка баз даних, кешування, безпеки та багато іншого.

Відомі веб-додатки, розроблені на Flask

- Reddit
- Pinterest
- Netflix
- LinkedIn
- Stripe

Таким чином, Flask - це потужний і універсальний фреймворк для веб-розробки на Python. Він є хорошим вибором для розробників будь-якого рівня досвіду, які шукають простий у використанні, але гнучкий і масштабований фреймворк.

Django - це веб-фреймворк для Python, який був створений в 2005 році. Він є популярним вибором для розробки веб-додатків Python, оскільки він пропонує широкий спектр функцій і можливостей.

Основні характеристики Django

- Модульність. Django є модульним фреймворком, що дозволяє розробникам легко додавати або видаляти функції;
- Гнучкість. Django є гнучким фреймворком, що дозволяє розробникам легко адаптувати його під свої потреби;
- Масштабованість. Django може бути використаний для розробки веб-додатків будь-якого масштабу, від невеликих сайтів до великих веб-сервісів;
- Безпека. Django має вбудовані функції безпеки, які допомагають захистити веб-додатки від атак;
- Спільнота. Django має велику і активну спільноту, яка надає підтримку та ресурси користувачам.

Як працює Django

Django працює на основі моделі MVC (Model-View-Controller). MVC - це архітектура веб-додатків, яка розділяє додаток на три основних компоненти:

- Модель: Модель відповідає за зберігання даних.
- Вид: Вид відповідає за відображення даних на веб-сторінках.
- Контролер: Контролер відповідає за обробку запитів користувачів і повернення відповідей.

Django має ряд додаткових функцій і можливостей, які можуть бути корисними для розробників веб-додатків. До них відносяться:

- Шаблонизатор Django - це потужний інструмент для генерації HTML-коду для відображення веб-сторінок.
- Система управління пакетами Django - це інструмент, який дозволяє розробникам легко встановлювати і керувати сторонніми бібліотеками.
- Адміністративна панель Django - це інструмент, який дозволяє розробникам створювати і керувати веб-додатками без необхідності писати код.

– Модулі Django - це набір готових рішень для поширених завдань веб-розробки.

Початок роботи з Django

Для початку роботи з Django потрібно встановити його. Для цього можна використовувати менеджер пакетів Pip

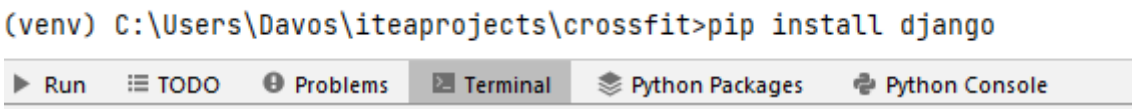


Рис. 2.5. Інсталяція фреймворку Django

Після встановлення Django можна створити новий проект. Для цього можна використовувати наступну команду:

django-admin startproject myproject

Ця команда створить новий проект Django з назвою "myproject".

Розробка веб-додатків на Django

Django можна використовувати для розробки широкого спектру веб-додатків, включаючи:

– Прості веб-сайти. Django можна використовувати для розробки простих веб-сайтів, таких як особисті веб-сторінки або блоги.

– Веб-додатки. Django можна використовувати для розробки веб-додатків, таких як інтернет-магазини, системи управління контентом (CMS) та соціальні мережі.

– Веб-сервіси. Django можна використовувати для розробки веб-сервісів, таких як API та мікросервіси.

На рис. 2.6. Зображено структуру проєкту на Django

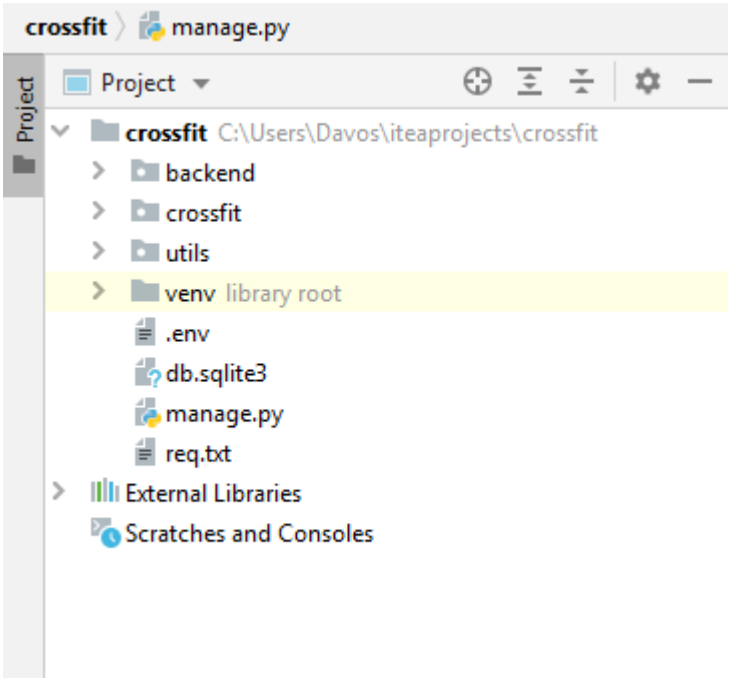


Рис. 2.6. Структура проєкту на Django

Для розробки веб-додатків на Django можна використовувати сторонні бібліотеки, які надають додаткові функції, такі як підтримка баз даних, кешування, безпеки та багато іншого.

Відомі веб-додатки, розроблені на Django

- Instagram
- Pinterest
- Spotify
- Mozilla Firefox
- Disqus

Таким чином, Django - це потужний і універсальний фреймворк для веб-розробки на Python. Він є хорошим вибором для розробників будь-якого рівня досвіду, які шукають фреймворк з широким спектром функцій і можливостей.

Django є популярним вибором для розробки веб-додатків Python. Він пропонує широкий спектр функцій і можливостей, що робить його хорошим вибором для розробників будь-якого рівня досвіду.

## 2.3. Особливості та відмінності загорткових та рекурентних нейронних мереж

Як вже зазначалось, нейронні мережі є потужним, сучасним інструментом, який можна використовувати для вирішення широкого кола завдань і які використовують переважно для аналізу складних структур даних. Сьогодення характеризується швидким розвитком технологій і тому нейронні мережі, та їх можливості лише розширюватимуться.

Нейронні мережі використовуються в сучасному світі для вирішення широкого кола завдань, включаючи:

– прогнозування, для побудови прогнозів майбутніх подій, таких як ціни на акції або попит на товари та послуги;

– розпізнавання образів, для розпізнавання таких як об’єктів як люди, предмети різної природи та ін.;

– обробка природної мови, а саме допомога при перекладі, розпізнавання мови або генерація тексті;

– автоматизація, нейронні мережі активно використовують для автоматизації виконання завдань, управління різного роду процесами або прийняття рішень.

Згорткові нейронні мережі (CNN або Convolutional Neural Network) є одними з найефективніших методів розпізнавання образів. Подібні архітектури нейронних мереж також зарекомендували себе в класифікації зображень. Вони мають ряд особливостей, які роблять їх особливо ефективними при роботі з зображеннями (див. таблицю 5).

Таблиця 5

Особливості загорткових нейронних мереж

| Особливість CNN | Опис особливостей CNN |
|---|---|
| Згортки | Згортки дозволяють CNN навчатися локальним особливостям зображень (образів). Це є принципово важливим, оскільки більшість зображень містять локальні особливості, які є унікальними для певного класу об'єктів (це дає чуттєву перевагу при класифікація або розпізнаванні образів). |

Продовження таблиці 5

| | |
|---|---|
| Фільтри | Згорткові нейронні мережі використовують фільтри, що дозволяє ефективно виявляти локальні особливості зображень. Фільтри зазвичай навчають на навчальному наборі даних, що дозволяє виявляли особливості, які є характерними для певного класу об'єктів. |
| Масштабна інваріантність | CNN є масштабно нечутливими, що означає, що вони можуть бути ефективними для розпізнавання образів різного розміру. Це важливо, оскільки зображення (образи) в реальному світі можуть бути різних розмірів. |

Як зазначалось раніше, CNN мають ряд певних переваг перед іншими методами розпізнавань образів. До переваг загорткових мереж можна віднести:

– точність, CNN часто показують більш високу точність розпізнавання образів порівняно з іншими моделями;

– масштабованість, згорткові нейронні мережі можна використовувати для роботи з великими наборами даних;

– універсальність згорткових нейронних мереж дозволяє використовувати їх для розпізнавання зображень різних класів.

Можна виділити декілька прикладів, які демонструють як CNN використовуються для розпізнавання образів:

– розпізнавання облич: CNN використовується для розпізнавання облич у системах безпеки за допомоги відео та фотокамер;

– розпізнавання об'єктів: CNN дозволяють ефективно розпізнавати об'єкти на зображеннях, у якості таких об’єктів можуть виступати машини, люди, тварини і т.п.;

– розпізнавання рукописного тексту: CNN використовується для розпізнавання рукописного тексту у документах (одним з найбільш популярних набір даних з рукописними цифрами є MNIST).

Щодо особливостей застосування CNN для розпізнавання образів, то можна виділити наступні:

– використання згорткових шарів. Згорткові шари дозволяють CNN навчатися локальним особливостям зображення, тобто краще розуміти деталі, що безпосередньо впливає на якість моделі;

– використання пулінгу. Пулінг дозволяє CNN зменшити розмір вихідних даних згорткових шарів, зберігаючи при цьому важливі особливості;

– використання повнозв'язних шарів. Повнозв'язні шари дозволяють CNN навчатися глобальним закономірностям у зображеннях.

Приклад простої CNN моделі наведено на рисунку 2.7.

```python
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=4):
        super(SimpleCNN, self).__init__()

        # Згорткові шари
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        # повнозв'язні шари
        self.fc1 = nn.Linear(32 * 8 * 8, 256)
        self.relu3 = nn.ReLU()
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = self.relu2(x)
        x = self.pool2(x)
```

Рис. 2.7. Приклад простої CNN моделі

Слід зазначити, що застосування CNN для розпізнавання образів потребує ретельного підбору архітектури нейронної мережі та параметрів навчання. Однак, при правильному підході, CNN можуть забезпечити високу точність розпізнавання зображень.

Що стосується рекурентних нейронних мереж (RNN), то це клас нейронних мереж, спроектований для роботи з послідовними даними та врахування залежностей у часі. Вони мають здатність "запам'ятовувати" попередні входи і

використовувати цю інформацію для обробки наступних входів. Це робить RNN придатними для різних завдань, таких як обробка природної мови (Natural Language Processing, NLP), часові ряди, генерація текстів та багато іншого.

Однак, у класичних RNN є проблема градієнта, що зникає, коли градієнти стають дуже маленькими, що ускладнює навчання моделі на довгих послідовностях. Для вирішення цієї проблеми було розроблено складніші архітектури, такі як довгострокові рекурентні мережі (Long Short-Term Memory, LSTM) та gated recurrent units (GRU), які більш ефективно обробляють залежності в часі.

Приклад простої RNN моделі з LSTM шаром, двома лінійними шарами та функцією активації ReLU наведено на рисунку 2.8. Дана модель більше підійде для прогнозування часових рядів.

```python
# Визначення простої RNN моделі в PyTorch
# Модель має LSTM шар, два лінійних шара та функцію активації ReLU
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=2):
        super(SimpleRNN, self).__init__()
        self.rnn = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc1 = nn.Linear(hidden_size, 64)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(64, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        out = self.fc1(out[:, -1, :])
        out = self.relu(out)
        out = self.fc2(out)
        return out

# Параметри моделі
input_size = 1
hidden_size = 64
output_size = 1
num_layers = 8

# Створення моделі
model = SimpleRNN(input_size, hidden_size, output_size, num_layers)
```

Рис. 2.8. Приклад простої RNN моделі

Незважаючи на відмінність та певні особливості двох типів нейронних мереж, слід зазначити, що це не означає що їх елементи не можна комбінувати.

Прикладом такої архітектури може бути комбінація згорткової нейронної мережі для отримання ознак з кадрів відео, бо відео рядок це по суті послідовність впорядкованих зображень, а потім рекурентної нейронної мережі для аналізу динаміки змін у часі.

Таким чином, згорткові та рекурентні нейронні мережі є достатньо ефективними, масштабованими та універсальними, що робить їх ідеальним вибором для багатьох завдань, які пов’язані з аналізом складних структур даних. У такій комбінованій архітектурі згорткові шари можуть бути використані для отримання просторових ознак з даних, а рекурентні шари - для роботи з послідовними залежностями даних.

## 2.4. Висновки до розділу

Існують багато різноманітних інструментів, які можна використовувати для розробки нейронних мереж, самими популярними з них є TensorFlow та PyTorch. В цілому, PyTorch є потужним і універсальним інструментом для машинного навчання. Він має ряд переваг, які роблять його ідеальним для вирішення широкого спектру завдань, причому він має достатньо низький поріг входу.

Існує достатня кількість потужних і універсальний фреймворків для веб-розробки на Python. Вони є хорошим вибором для розробників будь-якого рівня досвіду, які шукають фреймворк з широким спектром функцій і можливостей.

Що стосується найбільш ефективних типів нейронних мереж, то тут можна виділити згорткові та рекурентні нейронні мережі, які є достатньо ефективними, масштабованими та універсальними, що робить їх ідеальним вибором для багатьох завдань, які пов’язані з аналізом складних структур даних. У такій комбінованій

архітектурі згорткові шари можуть бути використані для отримання просторових ознак з даних, а рекурентні шари - для роботи з послідовними залежностями даних.

# РОЗДІЛ 3
# ІНСТРУМЕНТИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ НЕЙРОННИХ МЕРЕЖ

## 3.1. Реалізація задач класифікації текстів класичними алгоритмами машинного навчання

Незважаючи на те, що кожен день накопичується велика кількість інформації, іноді доводиться працювати в умовах, коли цієї інформації обмаль і тоді використання нейронних мереж стає неможливим. Причиною тому є те, що вона банально не встигне якісно навчитись. У цій ситуації на допомогу приходять класичні, перевірені часом методи та моделі.

Для прикладу можна розглянути як вирішувались завдання пов’язані з обробкою реальної мови до застосування і домінування нейронних мереж.

Якщо, розглядати проблему класифікації коментарів (наприклад, токсичні або ні), то перед початком роботи, треба спочатку підготувати дані, у даному випадку текстові дані мають свої особливості.

Підготовка текстових даних до моделювання складається з кількох етапів:

– токенізація тексту, тобто його розбиття на більш менші одиниці. Зазвичай таке розбиття відбувається по пробілах та знаках пунктуації;

– видалення знаків пунктуації, наприклад «.», «;» і т.п.;

– видалення так званих стоп-слів (stopwords), що є процесом видалення слів, які широко використовуються і несуть дуже мало корисної інформації;

– стеммінг (stemming), тобто стеммінг, можна охарактеризувати як процес метою якого є знаходження основи слова, якщо не вдаватись в подробиці, то видалення закінчення у заданому вихідному слові.

Наведені кроки є дуже важливими на етапі підготовки текстових даних до моделювання. Частково, ці процеси автоматизовані, єдиною проблемою може буде тільки підтримка рідної мови (особливо це стосується дуже рідких мов).

Розглянемо процес підготовки даних по зазначеній вище схемі на простому прикладі з одного речення, яке є випадково взятим з Інтернету реальним коментарем.

```
[32] sentence_example = '''
     Це покоління не зможе зрозуміти, що ми відчували в той час коли вперше побачили цей фільм.
     Для багатьох цей фільм був за межею можливого'''
     sentence_example
```

```
' \nЦе покоління не зможе зрозуміти, що ми відчували в той час коли вперше побачили цей фільм. \nДля багатьох
```

Рис. 3.1. Приклад випадкового текстового коментаря

Наступний крок проведення токенізації (tokenization). На рисунку 3.2 наведений приклад токенізації випадкового текстового речення за допомоги бібліотеки NLTK.

```
[36] from nltk import word_tokenize
     tokens = word_tokenize(sentence)
     print(tokens)
```

```
['Це', 'покоління', 'не', 'зможе', 'зрозуміти', ',', 'що', 'ми', 'відчували', 'в', 'той', 'час',
```

Рис. 3.2. Приклад токенізації випадкового текстового речення за допомоги бібліотеки NLTK.

В результаті дійсно було отримано list, елементами якого є структурні одиниці речення, а саме слова, знаки пунктуації, частки і т.п.

Наступним етапом є видалення з отриманого списку (tokens) знаків пунктуації.

```
tokens_without_punctuation = [i for i in tokens if i not in string.punctuation]
print(tokens_without_punctuation)
```

```
['Це', 'покоління', 'не', 'зможе', 'зрозуміти', 'що', 'ми', 'відчували', 'в', 'той', 'час', 'коли',
```

Рис. 3.3. Видалення зі списку отриманих токенів знаків пунктуації

Після видалення знаків пунктуації, слід перейти до видалення стоп-слів. Для видалення стоп-слів необов’язково використовувати бібліотеку NLTK, треба тільки знайти у відкритому доступі список stopwords для української мови. На рисунку 3.4 наведена частина stopwords для української мови.

```
ukrainian_stopwords = ['а', 'аби', 'абиде', 'абиким', 'абикого', 'абиколи', 'абикому
'абичию', 'абичия', 'абичиє', 'абичиєму', 'абичиєю', 'абичиєї', 'абичиї', 'абичиїй',
'абияка', 'абияке', 'абиякий', 'абияким', 'абиякими', 'абияких', 'абиякого', 'абияко
'або', 'абощо', 'авжеж', 'авось', 'ага', 'ад', 'адже', 'аж', 'ажень', 'аз', 'ай', 'а
'аніким', 'анікого', 'анікогісінько', 'аніколи', 'анікому', 'аніскільки', 'аніхто',
'аніяке', 'аніякий', 'аніяким', 'аніякими', 'аніяких', 'аніякого', 'аніякому', 'анія
'аніякісенька', 'аніякісеньке', 'аніякісенький', 'аніякісеньким', 'аніякісенькими',
'аніякісенької', 'аніякісеньку', 'аніякісенькі', 'аніякісенькій', 'аніякісенькім', '
'аніякісінькими', 'аніякісіньких', 'аніякісінького', 'аніякісінькому', 'аніякісінько
'аніякісінькім', 'ат', 'ато', 'атож', 'ау', 'ах', 'ач', 'ачей', 'аякже', 'б', 'ба',
'бац', 'баш', 'бе', 'беж', 'без', 'безперервно', 'бел', 'бер', 'би', 'бир', 'бич', '
'був', 'буває', 'буде', 'будем', 'будемо', 'будете', 'будеш', 'буду', 'будуть', 'буд
'бух', 'буц', 'буцім', 'буцімто', 'бі', 'біб', 'більш', 'більше', 'біля', 'в', 'в бі
процесі', 'в результаті', 'в ролі', 'в силу', 'в сторону', 'в супроводі', 'в ході',
```

Рис. 3.4. Список stopwords для української мови [10]

Застосуємо наведений список стоп-слів до списку токенів без знаків пунктуації. Результат наведено на рисунку 3.5.

```
tokens_without_stopwords_and_punctuation = [i for i in tokens_without_punctuation if i.lower() not in ukrainian_stopwords]
print(tokens_without_stopwords_and_punctuation)
```

```
['покоління', 'зможе', 'зрозуміти', 'відчували', 'вперше', 'побачили', 'фільм', 'фільм', 'межею', 'можливого']
```

Рис. 3.5. Список токенів без знаків пунктуації і стоп-слів

Залишилось застосувати до отриманого списку токенів без знаків пунктуації і стоп-слів стеммінг (stemming). Результат застосування стеммінгу наведено на рисунку 3.6.

```
stemming_tokens = [snowball.stem(i) for i in tokens_without_stopwords_and_punctuation]
print(stemming_tokens)
```

```
['поколін', 'змож', 'зрозуміт', 'відчува', 'вперш', 'побач', 'фільм', 'фільм', 'меж', 'можлив']
```

Рис. 3.6. Застосування стеммінгу до списока токенів без знаків пунктуації і стоп-слів

Для застосування отриманих токенів з метою побудови класифікації їх треба перетворити у числові значення. Для цього можна скористатись можливостями бібліотеки sklearn, які містить необхідний інструментарій, а саме TfidfVectorizer.

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Ініціалізуємо TfidfVectorizer
vectorizer = TfidfVectorizer()
# Тренуємо векторайзер та трансформуємо токени
X = vectorizer.fit_transform(stemming_tokens)
print(X.shape)
print(X.toarray())
vectorizer.get_feature_names_out()
```

```
(10, 9)
[[0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0.]]
array(['вперш', 'відчува', 'змож', 'зрозуміт', 'меж', 'можлив', 'побач',
       'поколін', 'фільм'], dtype=object)
```

Рис. 3.7. Перетворення токенів у числові вектори з використанням модуля TfidfVectorizer з бібліотеки sklearn

Зрозуміло, що кожне з повідомлень буде перетворено у вектори для подальшої їх обробки класифікаційною моделлю. Єдиною проблемою, яку треба вирішити до передачі даних моделі це те, що класифікаційні моделі відносяться до задач навчання з вчителем (supervised learning). Навчання з вчителем потребує розмічених даних, тобто кожен коментар повинен бути поміченим:

$$\begin{cases} 1, for\ toxic\ comments \\ 0, for\ non-toxic\ comments \end{cases}$$

Виходом з цієї ситуації може бути:

– розмітка коментарів вручну, що є дуже дорогою процедурою;

– пошук готового дата сету з розміченими коментарями, що є більш ефективних варіантом, але все залежить від мови коментарів.

Необхідно підкреслити, що подобного роду підготовка текстових даних до моделювання є важливим етапом. Однак, слід зазначити, що сучасні нейронні мережі вже мають вбудований токенайзер у тому числі і для української мови, знайти подібні рішення можна на сайті Hugging Face [9], як і предтрейнові моделі. У сучасних нейронних мережах замість TfidfVectorizer використовують більш просунуті техніки, а саме ембедінги (embedding), тобто перетворення елемента мови (наприклад, слова, речення і т.п.) у числовий вектор.

Після перетворення тексту у числові вектори, можна побудувати будь яку модель машинного навчання. Зупинимось на класичній моделі, а саме дереві рішень, зрозуміло що можна обрати і інші альтернативи, наприклад, логістичну регресію, метод опорних векторів (Support vector machine), або більш просунуті такі як Random Forest XGBoost. На рисунку 3.8 наведено приклад ініціалізації моделі дерева рішень з заданими параметрами.

```
from sklearn.tree import DecisionTreeClassifier

tree_clf = DecisionTreeClassifier(
    max_depth=10,
    min_samples_leaf=21,
    max_features=0.9,
    criterion="gini",
    random_state=1)

dt = tree_clf.fit(train_X, train_y)
y_pred = dt.predict_proba(test_X)
print("AUC: " + str(roc_auc_score(y_score=y_pred[:,1], y_true=test_y)))
```

Рис. 3.8. Ініціалізація моделі та навчання моделі DecisionTreeClassifier з бібліотеки sklearn

Дерева рішень схильні до перенавчання, тому має сенс це врахувати. В запропонованій моделі стоять обмеження на глибину дерева, яка дорівнює 10, та мінімальній кількості об’єктів в кожній гільці, яке дорівнює 21.

Зрозуміло, що це можуть бути неоптимальні значення, тому треба окремо попіклуватись про підбір гіперпараметрів. Це дозволить підвищити ефективність запропонованої моделі і отримати кращі результати по якійсь метрики якості, наприклад, ROC-AUC, розрахунок якої наведено на рисунку 3.8.

Для того щоб навити модель, а потім порахувати метрики якості і зрозуміти наскільки побудована модель ефективна, необхідно поділити дані на 2 групи трейнові дані, на яких будемо тренувати модель і тестові, на яких буде перевірятись якість побудованої моделі (див. рис. 3.9).

```
y = df['toxic'].values

X = df['comment'].values
```

```
from sklearn.model_selection import train_test_split

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.2, random_state=42)
```

Рис. 3.9. Поділ даних на трейнові та тестові в пропорції 80 на 20

Перед початком поділу даних, треба обрати таргетну (target або Y) зміну і фактори (features, X). Для наших даних Y – мітка класу, X – коментарі. Після цього вже можна поділити dataset на тренувальний та тестовий в пропорції 80 на 20, іноді використовують і 70 на 30.

Окремої уваги заслуговую вибір метрики якості, це може бути як простий accuracy, так і precision, recall, ROC-AUC і т.п.

Для даного прикладу ROC-AUC дорівнював 0.86, що говорить про непогану якість побудованої моделі, в ідеалі коли цей показник дорівнює одиниці.

Таким чином, було продемонстровано, що не тільки нейронні мережі показують непогані результати при роботі з текстовими даними. Класичні методи обробки даних та моделі машинного навчання все також дають дуже гарні результати. Особливо перевага таких методів проявляється коли даних недостатньо багато і нейронна мережа попросту не встигає якісно навчитесь, у такому випадку на допомогу приходять провірені класичні методики.

## 3.2. Вирішення задач обробки природної мови за допомоги нейронних мереж

Іншим способом вирішення задач пов’язаних з обробкою природної мови є застосування потужностей нейронних мереж. Існує багато різних завдань які пов’язані з NLP. Розглянемо один з популярних способів вирішення задача NLP, а саме застосування моделей трансформерів.

Трансформери є дуже популярними для роботи з ними використовується вже згаданий раніше ресурс [9]. На цьому ресурсі доступно велика кількість предтрейнових моделей, тобто моделей які вже є навченими і їх просто треба донавчити під вирішення конкретної задачі і конкретні дані.

Розглянемо як буде виглядати вирішення сформованої у попередньому розділі задачі. А саме задачі класифікації текстів по емоціональній забарвленості (Sentiment analysis). В термінах трансформерів це буде виглядати наступним чином див. рисунок 3.10.

Sentiment analysis (аналіз настрою) – це галузь обробки природної мови (NLP), яка вивчається науковцями та використовується у сфері комп'ютерної лінгвістики для визначення емоційного тону чи настрою, вираженого в тексті. Зазвичай ця задача вирішується за допомогою комп'ютерних алгоритмів, моделей машинного навчання та нейронних мереж.

Згадана галузь має велике значення в бізнесі, маркетингу та інших галузях, де важливо розуміти емоційний тон або настрій, що виражений в текстових даних.

```
# Імпорт необхідних бібліотек
from transformers import pipeline
# Створення пайплайну
nlp_sa = pipeline("sentiment-analysis")
```


Рис. 3.10. Імпорт трансформерів та створення pipeline

Спочатку імпортуємо з модуля transformers бібліотеку pipeline, далі створюємо pipeline і вказуємо для якою задачі ми будемо його застосовувати, в нашому випадку це “sentiment-analysis”.

Для оцінки якості застосованих моделей скористаємось двома виразами і перевіримо яку емоційну мітку вона надасть. Перший, це позитивний вираз – “Many people think you have the best work”, другий негативний “Only stupid people think you have the best work”, який також може заплутати модель, бо є дві умовні емоційні окраски “stupid” та “best”. Модель повинна повернути мітку класу:

– positive or negative;

– ймовірність, тобто наскільки модель впевнена що коментар або вираз належить до цього класу.

Результат застосування трансформерів для вирішення задачі сентімент аналізу наведено на рисунку 3.11.

```
result_pos = nlp_sa('Many people think you have the best work')#[0]
result_pos
```

```
[{'label': 'POSITIVE', 'score': 0.9998273253440857}]
```

```
result_neg = nlp_sa('Only stupid people think you have the best work')
result_neg
```

```
[{'label': 'NEGATIVE', 'score': 0.7057128548622131}]
```

Рис. 3.11. Результат застосування трансформерів для вирішення задачі sentiment analysis на двох прикладах

Модель повернула у першому випадку, що корпус належить до позитивного класу з ймовірністю 99,98%, у другому випадку до негативного класу з ймовірністю 70,57%. Оформлений код наведено на рисунку 3.12.

В області обробки природної мови (NLP), термін "корпус" вказує на колекцію текстових даних, яка зазвичай використовується для навчання, валідації або тестування моделей машинного навчання, а також для проведення досліджень у галузі обробки текстової інформації. Корпус може включати в себе тексти з різних джерел та мов, і його розмір може варіюватися від невеликого набору текстів до величезних обсягів інформації.

```
from transformers import pipeline
nlp_sent_anal = pipeline("sentiment-analysis")

result_1 = nlp_sent_anal('Many people think you have the best work')[0]
print(f"label: {result_1['label']}, with score: {round(result_1['score'], 4)}")
result_2 = nlp_sent_anal("Only stupid people think you have the best work")[0]
print(f"label: {result_2['label']}, with score: {round(result_2['score'], 4)}")
```

```
label: POSITIVE, with score: 0.9998
label: NEGATIVE, with score: 0.7057
```

Рис. 3.12. Код для sentiment analysis з label and score

На рисунку 3.13 розглянуто вирішення задачі sentiment analysis за допомоги pipeline, можна скористатись і іншим способом. А саме використати іншу модель, у тому числі вже натреновану і застосувати її до вирішення конкретної задачі. На рисунках наведено реалізація такого підходу і отриманий результат.

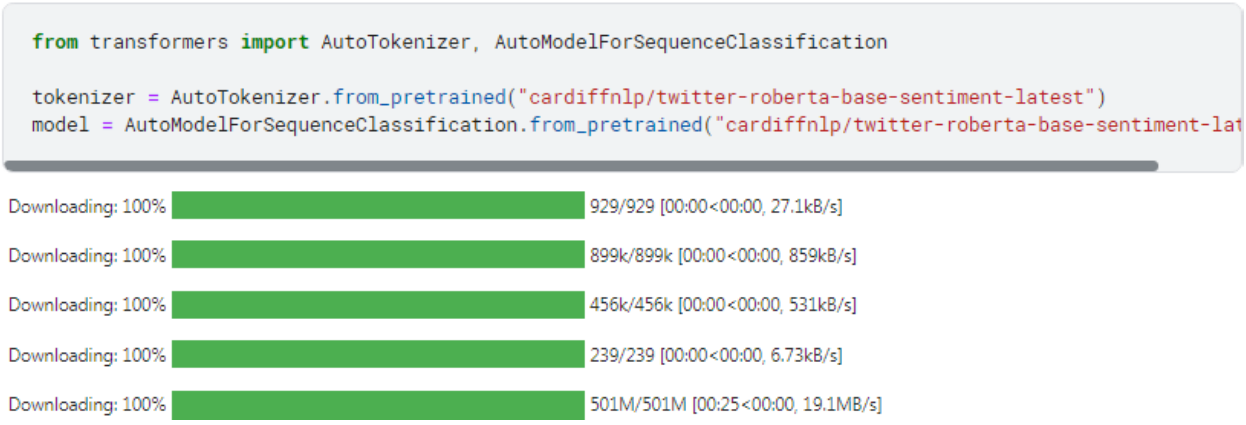


Рис. 3.13. Ініціалізація токенайзера і моделі

У даному випадку, можна імпортувати AutoTokenizer і AutoModelForSequenceClassification, а можна вказувати конкретну модель BERTSequenceClassification і конкретний токенайзер BERTokenizer, де BERT це достатньо популярна гугловська модель для NLP.

Tokenizer – це бібліотека для ефективної обробки та токенізації тексту. Вона пропонує інтерфейси для роботи з різними токенізаторами, такими як Byte Pair Encoding (BPE), SentencePiece та інші.

Можна використовувати будь які pre-trained моделі, тобто моделі які вже були натреновані на певних даних і які можна використовувати для вирішення різноманітних задач, у якості ресурсу таких моделей ідеально підходить Hugging Face [9]. Таким чином, було обрана модель ROBERTA навчена на

Hugging Face - це компанія та спільнота, яка створює та підтримує велику кількість ресурсів для розробки та застосування моделей машинного навчання, зокрема, моделей для обробки природної мови (NLP). Компанія визначається як "AI research organization" та відома своєю відкритістю та активним внеском в спільноту.

Після імпорту бібліотек і ініціалізації токенайзера та моделі, можна їх використовувати для вирішення задачі визначення тональності коментаря або будь якого тексту.

[7]:
```
from scipy.special import softmax

sequence = "Many people think you have the best work"

tokenizer = tokenizer(sequence, return_tensors="pt")

out = model(**tokenizer)
```

```
scores = out[0][0].detach().numpy()
scores = softmax(scores)
print(f"""
        neg:    {round(scores[0]*1,3)},
        neutral:{round(scores[1]*1, 3)},
        pos:    {round(scores[2]*1, 3)}""")
```

```
neg:    0.016,
neutral:0.11,
pos:    0.874
```

Рис. 3.14. Реалізація задачі по встановленню тональності тексту з результатами

Архітектура трансформерів дала потужний засіб для різноманітних задач в області обробки природної мови (NLP) та інших областях. Основні особливості трансформерів будуть розглянуті далію.

Механізм уваги (Attention Mechanism). Одна з головних інновацій трансформерів - це використання механізму уваги, який дозволяє моделі фокусуватися на різних частинах входу під час обчислення вихідного представлення. Це дозволяє моделі ефективно обробляти послідовності будь-якої довжини. На рисунку 3.15 представлено output токенайзера, який потім передається моделі.

```
[9]: tokenizer

[9]: {'input_ids': tensor([[    0, 10787,    82,   206,    47,    33,     5,   275,   173,     2]]), 'attention_mask': tensor([[1,
     1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

Рис. 3.15. Output токенайзера, який потім передається моделі

Треба звернути увагу, що окрім перекодованих елементів корпусу, токенайзер також повертає attention mask, по суті, ці елементи вказують моделі на що треба звернути увагу, якщо якесь зі значень в attention mask дорівнює нулю, це означає, що цей елемент не важливий.

Кодування позицій (Positional Encoding). Трансформери не мають вбудованого поняття про порядок послідовності. Тому, щоб врахувати позиційну інформацію, використовується кодування позицій для надання моделі інформації про розташування слів у послідовності.

Модель Encoder та Decoder. Треба зазначити, що трансформер складається з двох основних компонентів: Encoder (кодер) та Decoder (декодер). Encoder відповідає за обробку вхідної послідовності, а Decoder - за генерацію вихідної послідовності.

Нормалізація на шарі (Layer Normalization). Також треба відмітити, що нормалізація на кожному шарі допомагає підтримувати стабільність навчання та прискорює збіжність.

Це не всі елементи трансформерів, але основні.

На рисунку 3.16 наведено приклад реалізації задачі сентимент аналізу для декількох корпусів.

```
]:
from scipy.special import softmax

sequence_0 = "Many people think you have the best work"
sequence_1 = "Only stupid people think you have the best work"

sentences = [sequence_0, sequence_1]

for i in sentences:
    tokenizers = tokenizer(i, return_tensors ='pt')
    output = model(**tokenizers)

    scores = output[0][0].detach().numpy()
    scores = softmax(scores)

    print(f"""{i}:
        neg:    {round(scores[0]*1,3)},
        neutral:{round(scores[1]*1, 3)},
        pos:    {round(scores[2]*1, 3)}""")
```

```
Many people think you have the best work:
        neg:    0.016,
        neutral:0.11,
        pos:    0.874
Only stupid people think you have the best work:
        neg:    0.738,
        neutral:0.201,
        pos:    0.062
```

Рис. 3.16. Реалізації задачі сентимент аналізу для двох корпусів

Також треба проаналізувати архітектуру нейронної мережі BERT. На рисунку 3.17 показана частина архітектури моделі BERT.

```
model
```

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
```

Рис. 3.17. Архітектура моделі BERT

Треба звернути увагу, що в моделі першим йде блок ембедінгів (embeddings), моделі трансформери роблять векторі представлення для трьох компонентів текстів, а саме word_embeddings для слів в тексті, position embeddings для позиції (розташування) слів у тексті, token type embeddings для вказання окремих класів. По суті цей блок вказує на схожість слів.

Далі, йдуть блоки енкодерів (encoder), в цій моделі їх аж 12 (починаються з нуля, див. рисунок 3.18). В різних архітектурах нейронних мереж їх може бути різна кількість.

```
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (11): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
```

Рис. 3.18. Останній блок енкодерів в моделі BERT

Задача цих блоків це кодування тексту в якість useful features, тобто в ознаки, залежить від тих закономірностей які будуть знайденні в ембедінгах.

І останній елемент є дуже важливим, його структура представлена на рисунку 3.19.

```
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

Рис. 3.19. Bert pooler

Так модель повертає 768 прихованих ознак, до яких застосовується функція активація гіперболічного тангенсу, невеличкий дропаут і звичайний лінійний шар, або по суті лінійна або логістична регресія (для задач класифікації).

Звісно, мовні моделі можна використовувати для вирішення і іншого роду завдань. Наприклад, задачі відповіді на питання (Extractive Question Answering).

Extractive Question Answering (QA) - це завдання, в рамках якого система повинна виділити частину тексту або декілька частин тексту (відповіді) із вихідного документа, які найкращим чином відповідають на питання, що ставить користувач. Отже, відповіді генеруються шляхом вибору підходящих фрагментів тексту, а не генерацією тексту з нуля.

Таблиця 6

Основні етапи Extractive QA

| Назва етапу | Опис етапу |
|---|---|
| Постановка питання | Користувач ставить питання до системи. Це може бути будь-яке питання, на яке можна знайти відповідь у вихідному тексті |
| Аналіз тексту | Вихідний текст, в якому система буде шукати відповіді, піддається обробці. Це може включати токенізацію, лематизацію, векторизацію та інші методи обробки тексту |
| Виділення кандидатів на відповіді | Система визначає можливі місця, де може міститися відповідь, наприклад, речення або фрази, які мають високий ймовірний зв'язок з питанням. |
| Визначення важливості фрагментів | Важливість кандидатів на відповіді оцінюється, можливо, за допомогою методів машинного навчання або статистичних методів. Мета - вибрати найбільш інформативні частини тексту, які найкраще відповідають на питання |
| Виділення відповідей | Система витягає або виділяє фрагменти тексту, які були визначені як найкращі кандидати на відповіді |

Продовження таблиці 6

| | |
|---|---|
| Представлення відповіді користувачу | Знайдені фрагменти тексту відображаються або надсилаються користувачеві як відповідь на його питання. |

Перед тим як почати задавати відповіді і отримувати питання треба завантажити pipeline з модуля transformers. А також, необхідно передати контекст, тобто корпус для нашої моделі, він необхідний для того щоб модель на ньому донавчилась і давала відповіді, зазначаючи ймовірність впевненості.

```
from transformers import pipeline
nlp_QA = pipeline("question-answering")
context = r"""Ukraine is the second largest country in Europe.
Ukraine's capital is Kyiv. Ukraine shares borders with Moldova, Romania, Hungary, Slovakia, Poland, Belar
The Black Sea and the Sea of Azov lie to the south. Almost all of Ukraine is flat.
The grassland that covers the central and southern parts of the country is called the steppe.
In northern Ukraine are the Pripet Marshes, one of the largest wetlands in Europe.
The Carpathian Mountains rise in the west. The Crimean Mountains cross the Crimean Peninsula, a piece of
Ukraine's longest river is the Dnieper.
"""
```

Рис. 3.20. Завантаження бібліотек та передача контексту

Текст був узят з сайту [8]

Далі, необхідно передати моделі nlp_QA питання та зазначити контекст, тобто передати деякий текст (див. рисунок 3.21)

```
result = nlp_QA(question="Capital of Ukraine?", context=context)
print(f"Answer: '{result['answer']}', score: {round(result['score'], 4)}, start: {result['start']}, end:
result = nlp_QA(question="What river is in Ukraine?", context=context)
print(f"Answer: '{result['answer']}', score: {round(result['score'], 4)}, start: {result['start']}, end:
result = nlp_QA(question="How many people live in Ukraine?", context=context)
print(f"Answer: '{result['answer']}', score: {round(result['score'], 4)}, start: {result['start']}, end:
```

```
Answer: 'Kyiv', score: 0.7513, start: 71, end: 75
Answer: 'Dnieper', score: 0.6479, start: 616, end: 623
Answer: 'Ukraine is the second largest country in Europe', score: 0.0182, start: 0, end: 47
```

Рис. 3.21. Передача в модель питання, контексту та отримання відповіді

Можна зазначити, що модель згенерувала вірну відповідь на всі 3 запитання, причому при відповіді на частину питання, вона було доволі впевнена. Відповіді на перше та друге питання модель надала вірно з впевненістю в 75 та 65 відсотків

відповідно. На третє питання модель не змога надати вірну відповідь, бо інформація яка необхідна для відповіді на нього не зазначалась в переданому контексті, модель надала відповідь, але зазначила, що вона вкрай невпевнена, тобто ймовірність правильної відповіді дорівнює 1,8%.

Таким чином, можна стверджувати, що модель відпрацювала дуже гарно, відповіді були надані, ймовірності проставлені.

QA задачу також можна вирішити і без pipelin'у, способом, яки був розглянутий раніше. Спочатку треба імпортувати претрейновий токенайзер та обрати модель, передати контекст і запитання.

```
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
import torch
tokenizer = AutoTokenizer.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-sc
model = AutoModelForQuestionAnswering.from_pretrained("bert-large-uncased-whole-word-masking-
text = r"""
  Ukraine is the second largest country in Europe.
Ukraine's capital is Kyiv. Ukraine shares borders with Moldova, Romania, Hungary, Slovakia, P
The Black Sea and the Sea of Azov lie to the south. Almost all of Ukraine is flat.
The grassland that covers the central and southern parts of the country is called the steppe.
In northern Ukraine are the Pripet Marshes, one of the largest wetlands in Europe.
The Carpathian Mountains rise in the west. The Crimean Mountains cross the Crimean Peninsula,
Ukraine's longest river is the Dnieper.
"""
questions = [
    "Capital of Ukraine?",
    "What river is in Ukraine?",
    "Where are the largest wetlands in Europe?",
]
```

Рис. 3.22. Імпорт pre-trained токенайзера та моделі, передача контексту і запитань

Наступним кроком треба передати моделі запитання та отримати згенеровані відповіді на них.

```
for question in questions:
    inputs = tokenizer(question, text, add_special_tokens=True, return_tensors="pt")
    input_ids = inputs["input_ids"].tolist()[0]

    outputs = model(**inputs)
    answer_start_scores = outputs.start_logits
    answer_end_scores = outputs.end_logits

    answer_start = torch.argmax(
        answer_start_scores
    )
    answer_end = torch.argmax(answer_end_scores) + 1

    answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(input_ids[ans

    print(f"Question: {question}")
    print(f"Answer: {answer}")
```

```
Question: Capital of Ukraine?
Answer: kyiv
Question: What river is in Ukraine?
Answer: dnieper
Question: Where are the largest wetlands in Europe?
Answer: pripet marshes
```

Рис. 3.23. Передача моделі запитань та отримання відповіді на них

Це ще не всі задачі які можуть вирішувати мовні моделі. Однією з важливих задач є Masked Language Modeling.

Masked Language Modeling (MLM) - це метод навчання моделей обробки природної мови, таких як трансформери, за допомогою прихованих (замаскованих) токенів у тексті. Цей підхід дозволяє моделям вивчати контекст та залежності між словами в рамках задачі заповнення пропусків.

Основі етапи роботи подібного роду моделей наведено в таблиці 7.

Таблиця 7

Основні етапи MLM

| Назва етапу MLM | Опис етапу MLM |
|---|---|
| Маскування токенів | Для навчання моделі трансформера, певний відсоток токенів у вхідному тексті випадковим чином маскується (замасковується). Це може бути досягнуто шляхом заміни токенів на спеціальний токен-заповнювач |

Продовження таблиці 7

| | |
|---|---|
| Навчання на замаскованих токенах | Модель намагається відновити оригінальні токени на маскованих позначках. У процесі навчання модель отримує інформацію про контекст та взаємозв'язки між словами |
| Обчислення втрат | Після пропускання маскованих токенів через модель і отримання ймовірностей для різних токенів, обчислюються втрати (помилки) між передбаченими ймовірностями та реальними токенами |
| Оптимізація | Використовуючи методи оптимізації, такі як зворотне поширення помилок (backpropagation) та градієнтний спуск, модель поновлює свої параметри для покращення відповідності передбачених токенів реальним |

Приклад реалізації зазначеної задачі. Імпорт бібліотек і ініціалізація pipeline.

```
from transformers import pipeline
nlp_MLM = pipeline("fill-mask")
```

```
Downloading: 100% 480/480 [00:00<00:00, 15.0kB/s]
Downloading: 100% 331M/331M [00:06<00:00, 51.5MB/s]
Downloading: 100% 899k/899k [00:00<00:00, 2.51MB/s]
Downloading: 100% 456k/456k [00:00<00:00, 3.14MB/s]
Downloading: 100% 1.36M/1.36M [00:00<00:00, 2.25MB/s]
```

Рис. 3.24. Імпорт бібліотек і ініціалізація pipeline MLM

Передамо моделі ім’я надуманої людини і проаналізуємо отриману відповідь.

```
from pprint import pprint
pprint(nlp_MLM(f"Frank Maslov is {nlp.tokenizer.mask_token} of company."))
```

```
[{'score': 0.22713761031627655,
  'sequence': 'Frank Maslov is president of company.',
  'token': 394,
  'token_str': ' president'},
 {'score': 0.17186599969863892,
  'sequence': 'Frank Maslov is CEO of company.',
  'token': 1324,
  'token_str': ' CEO'},
 {'score': 0.16113628447055817,
  'sequence': 'Frank Maslov is head of company.',
  'token': 471,
  'token_str': ' head'},
 {'score': 0.1276240050792694,
  'sequence': 'Frank Maslov is chairman of company.',
  'token': 2243,
  'token_str': ' chairman'},
 {'score': 0.1232222393155098,
  'sequence': 'Frank Maslov is VP of company.',
  'token': 13913,
  'token_str': ' VP'}]
```

Рис. 3.25. Відповідь моделі на поставлений запит

Таким чином, модель генерує найбільш релевантні відповіді на встановлений запит і надає певну оцінку різним варіантам. У даному випадку найбільш ймовірно, що Френк Маслов президент компанії.

Masked Language Modeling широко використовується для попереднього навчання (pretraining) моделей, які потім можуть бути використані для різних завдань обробки природної мови, таких як машинний переклад, класифікація текстів, генерація тексту та інші. Такий підхід дозволяє моделям ефективно вивчати мовленнєві ознаки та граматичні структури.

Наведемо ще один приклад (див. рисунок 3.26)

```
from pprint import pprint
pprint(nlp_MLM(f"ZIEIT is {nlp.tokenizer.mask_token} university in the world."))
```

```
[{'score': 0.2396152764558792,
  'sequence': 'ZIEIT is oldest university in the world.',
  'token': 7763,
  'token_str': ' oldest'},
 {'score': 0.22916902601718903,
  'sequence': 'ZIEIT is largest university in the world.',
  'token': 1154,
  'token_str': ' largest'},
 {'score': 0.1546093225479126,
  'sequence': 'ZIEIT is best university in the world.',
  'token': 275,
  'token_str': ' best'},
 {'score': 0.09742164611816406,
  'sequence': 'ZIEIT is richest university in the world.',
  'token': 16386,
  'token_str': ' richest'},
 {'score': 0.036054227501153946,
  'sequence': 'ZIEIT is no university in the world.',
  'token': 117,
  'token_str': ' no'}]
```

Рис. 3.26. Відповідь Masked Language моделі на поставлений запит

В даному випадку модель завіряє, майже з рівною впевненістю, що ZIEIT або на найстаріший університет, або самий великий у світі.

Щоб зрозуміти чому модель видає такі відповіді, візьмемо ще одну модель і передамо той самий запит і подивимось на топ 10 варіантів.

```
from transformers import AutoModelForMaskedLM, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-cased")
model = AutoModelForMaskedLM.from_pretrained("distilbert-base-cased")

sequence = f"ZIEIT is {tokenizer.mask_token} university in the world."

input = tokenizer.encode(sequence, return_tensors="pt")
mask_token_index = torch.where(input == tokenizer.mask_token_id)[1]
token_logits = model(input).logits
mask_token_logits = token_logits[0, mask_token_index, :]
percentages = torch.topk(torch.softmax(mask_token_logits, 1), 10, dim=1).values.detach().nump
top_10_tokens = torch.topk(mask_token_logits, 10, dim=1).indices[0].tolist()
```

Рис. 3.27. Імпорт токенайзера та моделі для вирішення задачі Masked Language Modeling

Було імпортовано distilbert це дещо зменшена та полегшена модель BERT, за рахунок чого вона працює швидше, хоча може і давати трохи гірші результати. Результат роботи моделі наведено на рисунку 3.28.

```
for token, percent in zip(top_10_tokens, percentages):
    print(f"{percent}% - \t", sequence.replace(tokenizer.mask_token, tokenizer.decode([token]
```

```
29.600000381469727% -    ZIEIT is largest university in the world.
13.0% -          ZIEIT is oldest university in the world.
5.099999904632568% -     ZIEIT is youngest university in the world.
5.0% -   ZIEIT is biggest university in the world.
4.599999904632568% -     ZIEIT is smallest university in the world.
2.299999952316284% -     ZIEIT is second university in the world.
2.0% -   ZIEIT is first university in the world.
2.0% -   ZIEIT is tallest university in the world.
1.899999976158142% -     ZIEIT is third university in the world.
1.600000023841858% -     ZIEIT is richest university in the world.
```

Рис. 3.28. Результат роботи Masked LM

Треба зауважити, що модель, як і попередня яка було створена за допомоги pipeline, або модель яка підбиралась автоматично, показала майже той самий результат. Отриманий результат напряму залежить не тільки від якості моделі і її складності, а і від того на яких даних були натреновані моделі, в даному випадку DistilBERT була натренована на даних з вікіпедії та bookcorpus [1].

Однак, моделі трансформери можуть бути використанні не тільки для вирішення задачі класифікації, їх також можна налаштувати для вирішення задачі регресії. Такого роду моделі дуже часто використовуються коли у якості факторів виступають текстові дані, але таргетна змінна є безперервною величиною. Прикладом такої задачі може бути оцінка читабельності текстів (Readability). Приклад реалізації такої моделі, яка розроблена автором і була використана в змаганні по Readability, яке проходило на платформі Kaggle [7] наведено на рисунку 3.29.

```
from transformers import AutoConfig
import torch.nn as nn

class RegressionModel(torch.nn.Module):

    def __init__(self):
        super(RegressionModel, self).__init__()
        config = AutoConfig.from_pretrained(path_model)
        config.update({"output_hidden_states":True,
                       "hidden_dropout_prob": 0.0,
                       "layer_norm_eps": 1e-7})

        self.bert = AutoModel.from_pretrained(path_model, config=config)

        self.attention = nn.Sequential(
            nn.Linear(1024, 512),
            nn.Tanh(),
            nn.Linear(1024, 1),
            nn.Softmax(dim=1)
        )
```

Рис. 3.29. Завантаження моделі BERT з деякими змінами в config і додавання додаткових шарів

```
    def forward(self, input_ids, attention_mask, label=None): #token_type_ids,
        outputs = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask
            #token_type_ids=token_type_ids,
        )

        sequence_output = self.regressor(outputs.last_hidden_state[:,0])
        #sequence_output = torch(self.regressor(self.dropout(outputs.last_hidden_state[:,-1,:]

        logits = (sequence_output)

        loss = None
        if label is not None:
            loss_fn = torch.nn.MSELoss()
            logits = logits.view(-1).to(label.dtype)
            loss = torch.sqrt(loss_fn(logits, label.view(-1)))

        output = (logits,) + outputs[1:]
        return ((loss,) + output) if loss is not None else output
```

Рис. 3.30.Модифікація регресійної моделі на основі BERT для вирішення задачі readability

Можна побачити, що у якості функції втрат застосовується звичайний RMSE (корінь з середньо квадратичної помилки). Запропонована модель без додаткових модифікацій показала score (RMSE) = 0.516.

Таким чином, нейронні мережі потужний інструмент для аналізу текстових данних і вирішення широкого кола задач. Хоча розглянуті ще не всі задачі, які можуть вирішувані ці потужні моделі.

## 3.3. Класифікація зображень і нейромережеві моделі прогнозування часових рядів

Класифікація зображень за допомогою нейронних мереж є однією з основних та популярних задач машинного навчання та глибокого навчання. Основний принцип полягає в тому, щоб навчити нейронну мережу визначати, до якого класу належить зображення. Нижче наведено загальний огляд процесу класифікації зображень з використанням нейронних мереж:

Збираються дані для тренування мережі. Це може включати зображення різних класів, наприклад, коти та собаки для задачі класифікації тварин. Дані поділяються на тренувальний та тестувальний набори. Зображення можуть бути зменшені та нормалізовані для спрощення обробки.

Вибирається архітектура нейронної мережі. Зазвичай для класифікації зображень використовують згорткові нейронні мережі (Convolutional Neural Networks, CNN), оскільки вони добре впораються з обробкою просторової ієрархії ознак у зображеннях.

Модель тренується на тренувальних даних за допомогою методів зворотного поширення помилок. У процесі тренування модель стає здатною визначати класи об'єктів на зображеннях.

Модель оцінюється на тестових даних, які раніше не бачила, для визначення її точності та ефективності. Зазвичай використовують метрики, такі як точність (accuracy), матриця помилок (confusion matrix) та інші.

В залежності від результатів тестування може бути проведена настройка гіперпараметрів моделі для поліпшення її продуктивності. Це включає в себе вибір оптимальних параметрів, оптимізацію алгоритму навчання тощо.

Після успішного тренування і тестування модель може бути розгорнута для використання у реальних умовах, де вона може класифікувати нові зображення.

Важливу роль в при роботі з зображеннями грає albumentations. Це бібліотека для аугментації зображень в області комп'ютерного зору та глибокого навчання. Аугментація є важливим етапом в підготовці даних для тренування моделей класифікації зображень, і вона полягає в створенні різних варіантів вихідних зображень для розширення набору тренувальних даних.

Albumentations в класифікації зображень дозволяє створювати різноманітні варіації зображень за допомогою різних операцій аугментації, таких як обертання, зміщення, зміна масштабу, зеркальне відображення та інші. Це робить модель більш стійкою до різних перепадів в даних та поліпшує її здатність генералізації.

Також, аугментація допомагає уникнути перенавчання, коли модель стає занадто специфічною для тренувальних даних. Завдяки використанню різних варіантів зображень, модель навчається розпізнавати об'єкти незалежно від їх положення, розміру та інших характеристик.

До переваг аугментації, можна віднести, що дозволяє використовувати менше даних для досягнення схожого рівня ефективності моделі, оскільки розширення тренувального набору за рахунок аугментації.

Зберігається контекстуальна інформація про зображення під час аугментації, що допомагає моделі розпізнавати об'єкти в різних умовах та оточеннях.

Сама бібліотека albumentations пропонує зручний інтерфейс для використання різноманітних аугментаційних операцій, що спрощує їх використання в процесі підготовки даних. На рисунках 3.31 представлено застосування аугментацій зображень і отриманий результат.

```
from albumentations.pytorch import ToTensor

def get_training_augmentation():

    augmentation_pipeline = A.Compose(
        [
            A.OneOf([
                A.Compose([
                    A.SmallestMaxSize(224),
                    A.RandomCrop(224, 224),
                ], p=1),
                A.Compose([
                    A.SmallestMaxSize(400),
                    A.RandomCrop(224, 224),
                ], p=1)
            ], p=1),

            A.OneOf(
                [
                    A.RandomGamma(),
```

Рис. 3.31. Застосування різних аугментацій для зображень

В данному випадку застосовано зміна розміру самого зображення, зміна розміру кадрування, може бути застосовано кадрування по краях та темних зонах, тобто випадковим чином може бути застосована обрізка зображення, також застосовано RandomGamma - це одна з аугментаційних операцій, яку можна використовувати для обробки та покращення зображень в машинному навчанні. Ця операція зазвичай використовується для зміни яскравості та контрастності зображення шляхом застосування гамма-корекції.

Однак, це не всі перетворення які були застосовані до зображень, продемонструємо ще декілька.

```
A.OneOf(
    [
        A.RandomGamma(),
        A.RandomBrightness(),
        A.RandomContrast(),
        A.Blur(blur_limit=10),
        A.GaussNoise()
    ],
    p = 0.4
),
A.OneOf(
    [
        A.Rotate(limit = 360),
        A.Flip(p = 0.5),
    ],
    p = 0.4
),
A.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
```

Рис. 3.32. Застосування аугментаційних операцій до зображент у Pytorch

У даному випадку було зроблено зміни зображень, які стосуються зміни контрасту, яскравості, додано аугментаційну операцію blur, яка використовується для розмиття (зменшення чіткості) зображень. Це може бути корисно для покращення роботи моделей, які повинні впоратися з менш чіткими або розмитими зображеннями, або для зменшення впливу шумів на тренування моделей.

Процес розмиття може бути виконаний різними методами, такими як гаусівське розмиття (застосування фільтра Гауса), медіанне розмиття (використання медіанного фільтра) або інші методи фільтрації.

Також застосовані різні повороти та фліпи, перевершення зображення по різним осям.

Результат подібних перетворень наведено на рисунку 3.33.

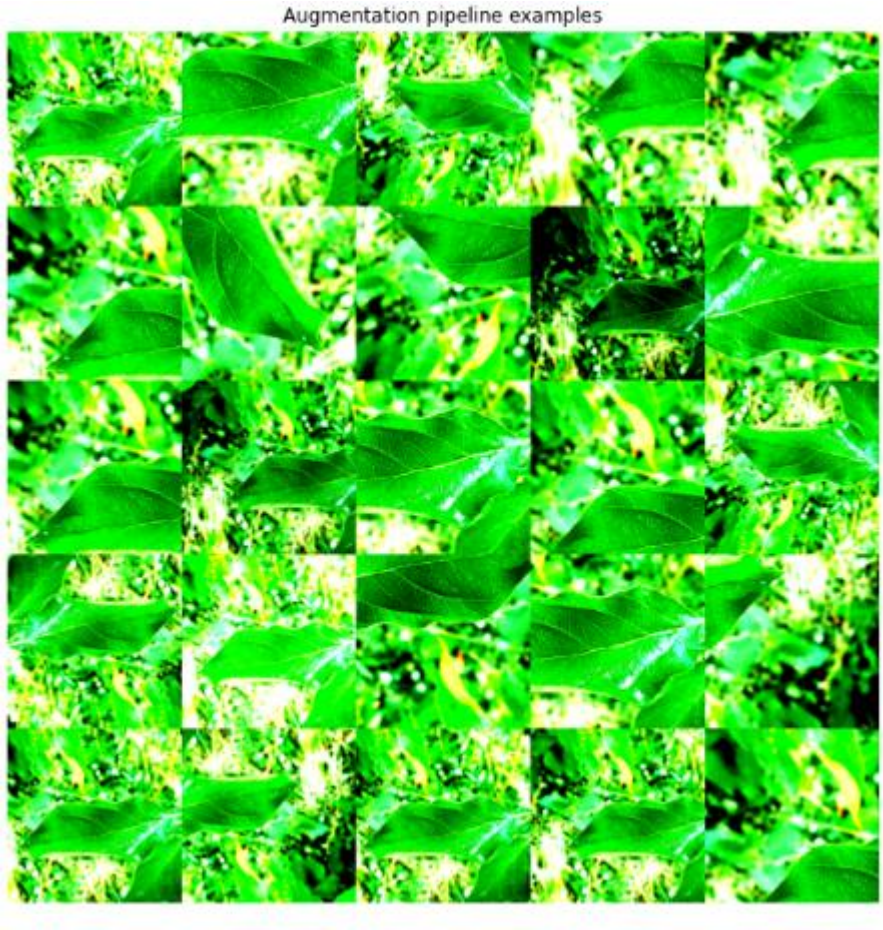


Рис. 3.33. Випадкове зображення після застосування аугментаційних операцій

Ці зображення були надані в рамках змагання з виявлення хвороб у рослин Plant Pathology 2021 - FGVC8 [6].

Застосування агументаційних операцій в рамках змагання дали змогу покращити метрику якості, зменшити ризик перенавчання та підвищити стабільність моделі.

Зменшення ризику перенавчання особливо важливе у змаганнях на Kaggle.

У якості моделі в данному змаганні була застосована предтренована модель EfficientNet.

EfficientNet - це сімейство нейронних мереж, які були розроблені для досягнення оптимального балансу між обчислювальною ефективністю та точністю в задачах класифікації зображень. Вони представляють собою модель глибокого

навчання, оптимізовану за розміром та обчислювальною складністю, що дозволяє ефективно використовувати ресурси обчислювальної техніки.

Модель EfficientNet здатна досягати високих результатів на завданнях класифікації зображень при значно менших обчислювальних та параметричних витратах порівняно з іншими архітектурами. Вона часто використовується у великих масштабних конкурсах та застосовується в різних галузях, включаючи комп'ютерне зору, медичне зображення та інші.

У змаганні саме використовувалась версія b-0, яка хоча і є більш «легкою», але вона достатньо швидка та ефективна. Перевірялись і більш «важкі» нейронні мережі, з більшої кількістю параметрів, але вони показали трохи гірший результат.

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                optimizer.step()
                scheduler.step()
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
```

Рис. 3.34. Функція тренування моделі

Всього було запропоновано використовувати 20 епох. Також в функції реалізовано збереження ваг для моделі яка покаже кращу якість. Якість моделі

оцінювалась по метрикам accuracy та f1-score. Результати наведено на рисунку 3.35.

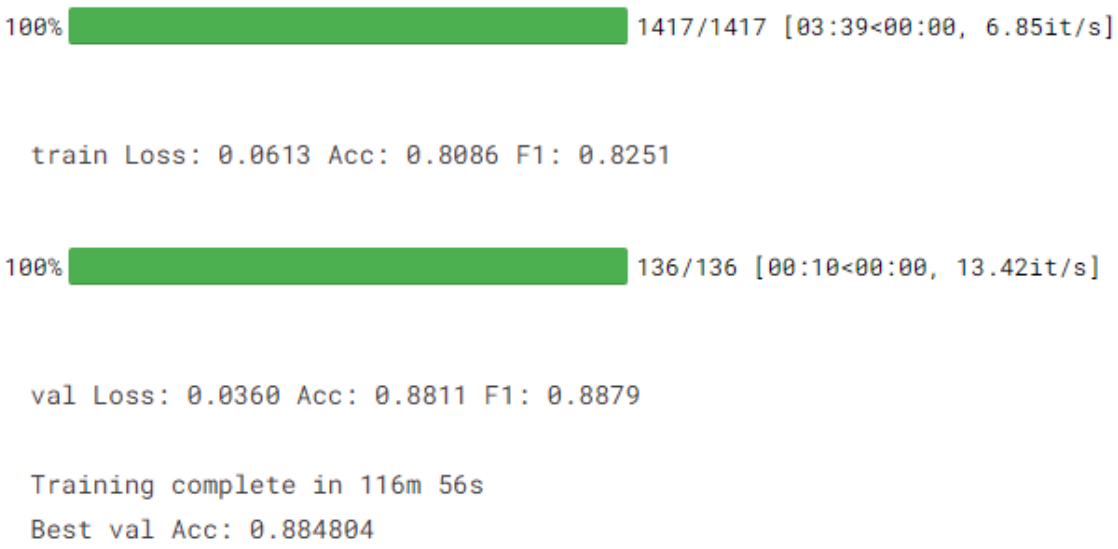


Рис. 3.35. Результати роботи моделі з урахуванням метрик якості

Всього тренування моделі зайняло 117 хвилин по метриці якості f1-score результат склав 0.8879.

Зрозуміло, що є і інші популярні архітектури, які часто використовуються для класифікації зображень, включають AlexNet, VGG (Visual Geometry Group), ResNet (Residual Networks), Inception, EfficientNet та інші. У багатьох випадках також використовуються аугментаційні техніки для підвищення кількості та різноманітності тренувальних даних.

Нейронні мережі - це потужний інструмент, який може використовуватися для прогнозування часових рядів. Вони можуть навчитися виявляти складні закономірності в даних, які можуть бути використані для прогнозування майбутніх значень.

Прогнозування часових рядів - це процес оцінки майбутніх значень часового ряду на основі його минулих значень. Це корисна задача в багатьох сферах, таких як бізнес, економіка, фінанси та наука.

Існує багато різних типів нейронних мереж, які можна використовувати для прогнозування часових рядів. Деякі з найпоширеніших типів включають:

– рекурентні нейронні мережі (RNN): RNNs добре підходять для прогнозування часових рядів, які є динамічними, тобто змінюються з часом.

– глибокі нейронні мережі (DNN): DNNs можуть бути більш точними, ніж RNNs, але вони також потребують більше даних для навчання.

– деконволюційні нейронні мережі (CNN): CNNs добре підходять для прогнозування часових рядів, які мають візуальну складову, наприклад, зображення або відео.

Приклад вирішення однієї з задача пов’язаних з прогнозуванням часових рядів з застосування нейронних мереж наведено на рисунку 3.36.

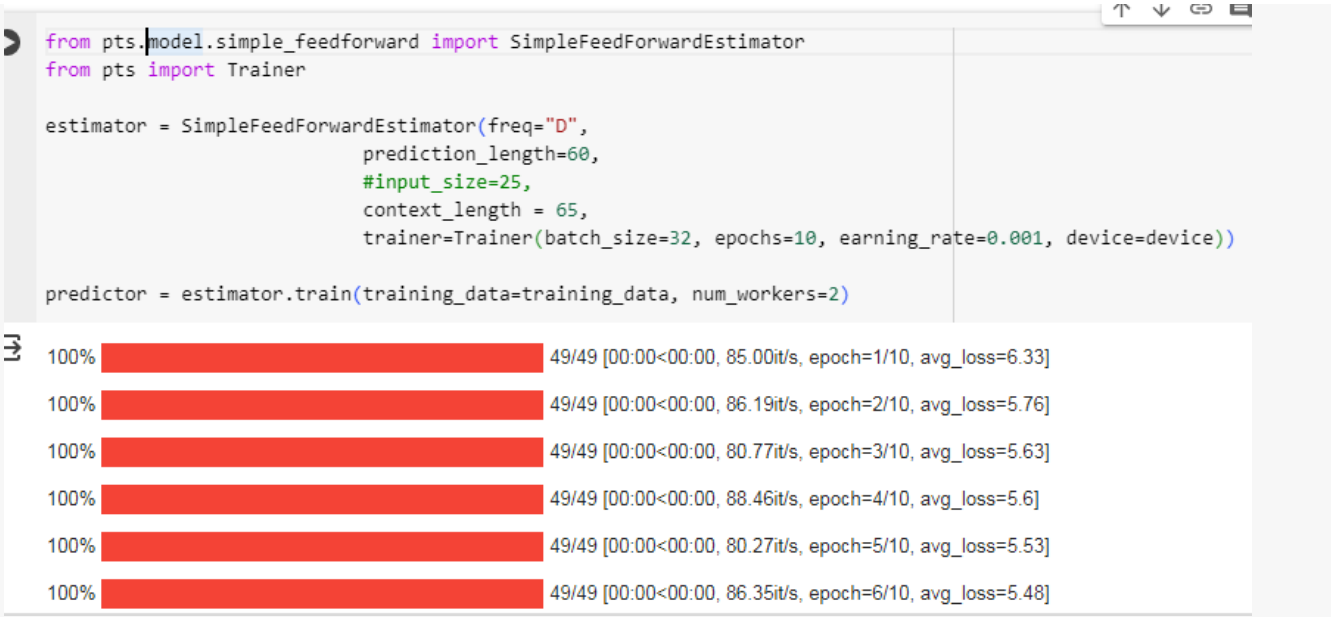


Рис. 3.36. Ініціалізація моделі SimpleFeedForwardEstimator

SimpleFeedForwardEstimator ініціалізована і навчена на щоденних даних за 3 роки, всього для навчання було обрано 10 епох. Прогноз робився на 60 днів і був отриманий наступний результат.

```
from gluonts.dataset.util import to_pandas

for test_entry, forecast in zip(test_data, predictor.predict(test_data)):
    to_pandas(test_entry)[-60:].plot(linewidth=2)
    forecast.plot(color='g', prediction_intervals=[50.0, 90.0, 95.0])
plt.show()#grid(which='both')
```

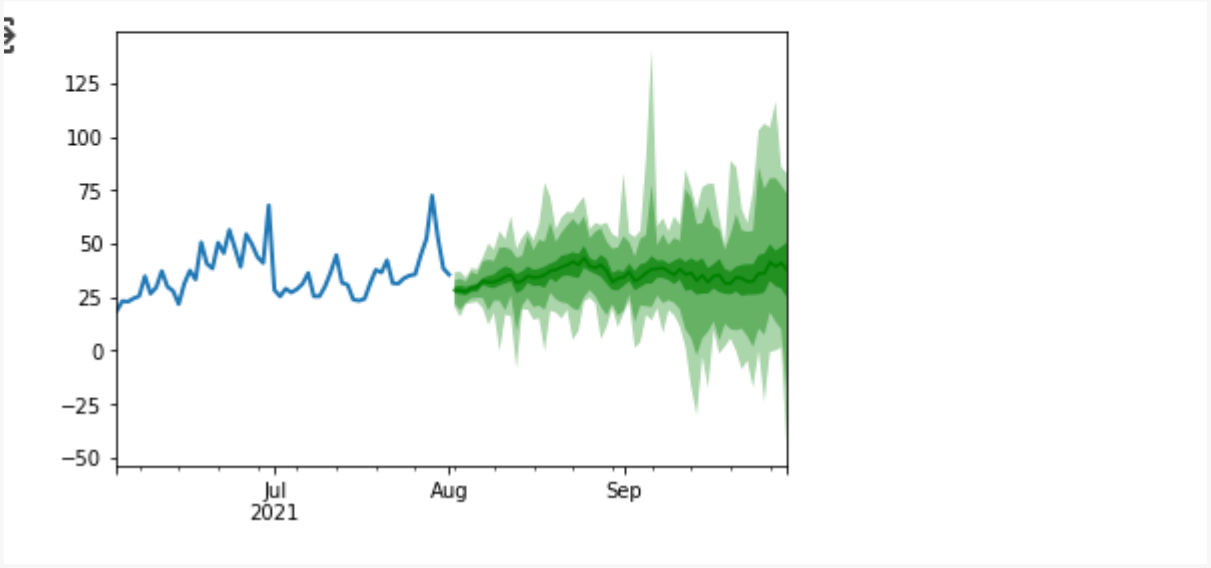


Рис. 3.37. Результат прогнозу представлений на графіку

Приведено не тільки результат прогнозу, а також і довірчі інтервали для цього прогнозу на рівнях в 50, 90 та 95%. Було розраховані наступні метрики якості.

```
mape(df["2021-08-01":"2021-09-29"], preds), mae(df["2021-08-01":"2021-09-29"], preds), rmse(df["2021-08-01":"2021-09-29"]
(30.669483889701993, 8.144096591822308, 9.81228426925222)
```

Рис. 3.38. Метрики якості побудованої моделі

Таким чином, модель при прогнозуванні на 60 днів вперед показала відсоткову помилку по метриці MAPE в 30,7%, по абсолютній метриці модель в середньому помилялась на 8.14 одиниць. На графіку отриманий прогноз в порівнянні з реальними даними буде виглядати наступним чином.

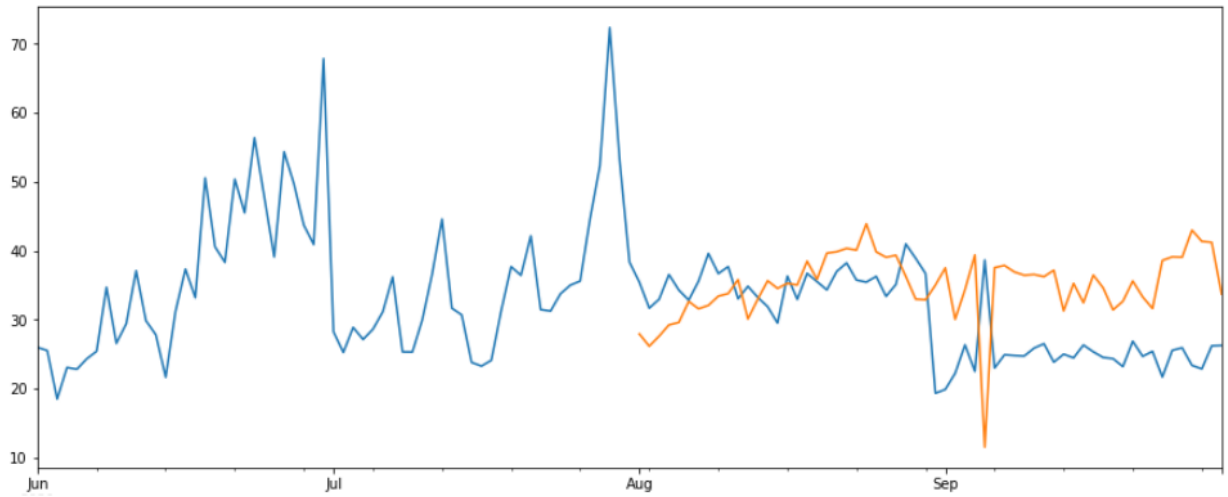


Рис. 3.39. Прогноз vs реальні дані.

Зрозуміло, що можна використовувати і більш складні моделі. Наприклад, напівкастомну LSTM модель.

```
class LSTM(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM, self).__init__()

        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)

        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        c_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        # Propagate input through LSTM
```

Рис. 3.40. Реалізація LSTM моделі

Модель була натренована з застосуванням 200 епох, результат представлено на рисунку 3.41.

```
plt.plot(dataY_plot)
plt.plot(data_predict)
plt.suptitle('Time-Series Prediction')
plt.show()
```

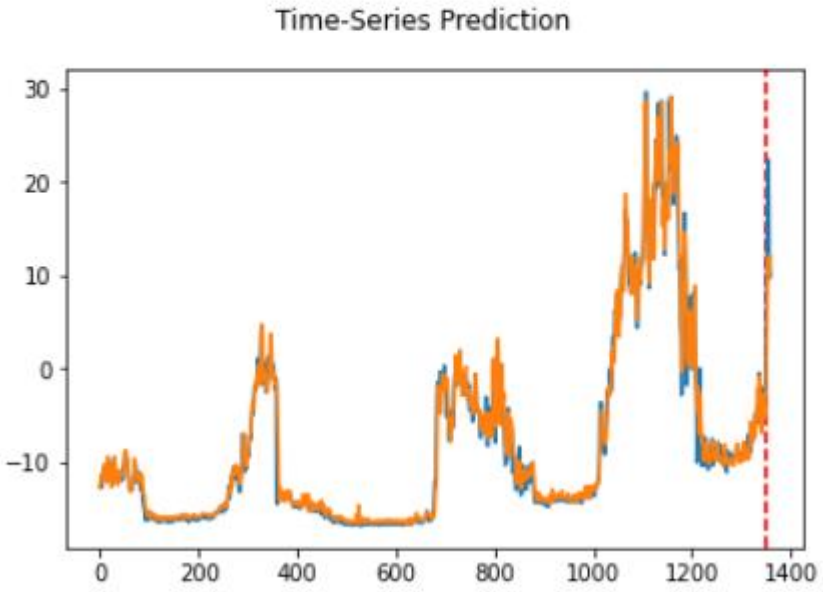


Рис. 3.41. Результат моделювання за допомоги LSTM моделі на 10 періодів вперед

В результаті були отримані наступні метрики якості.

```
MAPELoss(testY[seq_length:], lstm(testX)[seq_length:] )
```

```
tensor(10.4931, grad_fn=<MulBackward0>)
```

На тестовій виборці була отримана помилка всього в 10,49%, слід підкреслити, що прогноз будувався на 10 періодів вперед. Результат отриманого прогнозу можна вважати задовільним.

Класифікація зображень і нейромережеві моделі прогнозування часових рядів - це активні області досліджень. Дослідники працюють над розробкою нових методів, які можуть підвищити точність цих завдань.

Таким чином, наведені моделі які базуються як на створені власних архітектур, так і використання вже розроблених і предтренованих показають достатньо непогані результати. Однак, слід відмітити, що розробка сучасних і ефективних архітектур нейронних мереж під силу здебільше великим компаніям, які мають певні ресурси, як для швидкого навчання моделей і перевірки гіпотез, так і відповідний кадровий потенціал.

## 3.4. Розробка мікросервісу аналізу тональності тексту

Для розробки мікросервіса доцільно використовувати зручний та ефективний фрейморк, одним за таких фреймворків є Flask. Flask називають також мікрофреймворком, необтяжливий, дуже простий і має змогу взаємодіяти з різними додатковими інструментами, наприклад, jinja, різноманітними базами даних та інше. Про цей фреймворк вже було згадано раніше, наведені його переваги та недоліки.

Для роботи з Flask доцільно використовувати IDE або так звані інтегроване середовище розробки (Integrated Development Environment). Перед безпосереднім застосуванням Flask, його треба встановити та імпортувати. Команда додавання Flask до проєкту має наступний вигляд:

```
(venv) C:\Users\Davos\projects\sent_analisys>pip install flask
```

Слід підкреслити, що для проєкту доцільно створювати так зване віртуальне середовище (venv), воно дозволяє встановлювати бібліотеки безпосередньо для даного проєкту. Важливим атрибутом в розробці є створення файлу під назвою requirements.txt, в цей файл заносяться всі бібліотеки, які необхідні для роботоспроможності проєкту. Приклад структури такого файлу наведено на рисунку 3.42.

main.py | req.txt | style.css | home.html

```
asgiref==3.5.2
backports.zoneinfo==0.2.1
Django==4.0.6
sqlparse==0.4.2
tzdata==2022.1
Flask==3.0.0
fsspec==2023.12.2
gitdb==4.0.11
GitPython==3.1.40
huggingface-hub==0.20.1
idna==3.6
importlib-metadata==6.8.0
importlib-resources==6.1.1
itsdangerous==2.1.2
Jinja2==3.1.2
joblib==1.3.2
```

Рис. 3.42. Приклад структури файлу requirements.txt

Окрім Flasky необхідно встановити і спеціальні бібліотеки для проведення аналізу тональності тексту. В даному проєкті буде використано дві такі бібліотека, які вже були загадані раніше, а саме nltk та transformers. Їх теж треба встановити за допомоги команди pip install <назва бібліотеки>, єдине, для бібліотеки transformers має сенс застосовувати деяке уточнення.

```
(venv) C:\Users\Davos\projects\sent_analisys>pip install transformers[torch]
```

Рис. 3.43. Приклад додавання у проєкт бібліотеки transformers

Доцільно додавати не всю бібліотеку, вона достатньо важка по пам’яті, а саме torch версію. Слід відмітити, що до цього, при демонстрації можливостей нейронних мереж та простих архітектур використовувався саме pytorch.

Після установки необхідних бібліотек, можна приступати до безпосередньої розробки мікросервісу.

Спочатку розглянемо наскільки мікросервіс буде справлятись з поставленою задачею, а саме встановлення тональності текстових даних на базі бібліотеки nltk.

В nltk використовується економічна Rule-based модель, для встановлення тональності текстів використовується спеціальний словник з урахуванням тональності або почуттів. Для імпорту всіх необхідних бібліотек та спеціального словника достатньо чотирьох строчок коду.

main.py | req.txt | style.css | home.html

```
from flask import Flask, render_template, request
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon')
```

Рис. 3.44. Імпорт основних бібліотек необхідних для проєкту

Також, за допомоги декоратору треба прописати всі роути (route). В даному випадку, так як це мікросервіс, призначення якого виконання конкретної задачі, вистачить і одного такого декоратору і окремою функції, в якій буде реалізовано сценарій роботи мікросервісу.

main.py | req.txt | style.css | home.html

```
@app.route('/', methods=["GET", "POST"])
def main():
    if request.method == "POST":
        inp = request.form.get('inp')
        sid = SentimentIntensityAnalyzer()
        score = sid.polarity_scores(inp)
        if score["neg"] != 0:
            return render_template('home.html', message=f"Negative with score {score['neg']}")
        else:
            return render_template('home.html', message=f"Positive with score {score['pos']}")
    return render_template('home.html')
```

main() › if request.method == "POST" › if score["neg"] != 0

Рис. 3.45. Код з сценарієм роботи мікросервісу.

Основна задача, яку буде вирішувати розроблений мікросервіс, це встановлення тональності тексту, тобто, якщо спростити, то на вхід моделі необхідно передати якийсь коментар або текст, а модель повинна видати відповідь,

чи коментар є позитивним або негативним, також реалізовано вивід score, тобто наскільки модель впевнена у своєму прогнозі.

Таким чином, у змінну inp (input) передається деякий рядок. Змінна sid відповідає за ініціалізацію моделі, в яку безпосередню передається отриманий рядок. В змінну score записується відповідь моделі і далі формується повідомлення для користувача.

Для оформлення самої сторінки за допомоги якої користувач буде комунікувати з мікрсервісом створен окремий HTML файл.

main.py | req.txt | style.css | home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
    <title>Sentiment Analysis</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='styles/style.css')}}"
</head>
<div class="container">
    <div class="container-lg">
        <h1>SENTIMENT ANALYSIS</h1>
        <form method="POST" class="form">
            <input type="text" name="inp" placeholder="Enter Your Feeling" id="inp">
            <input type="submit" name="submit" id="btn" class="btn btn-primary">

        </form>
        <h1 class="output">{{message}}</h1>
    </div>
</div>
<body>
```

html › head › link

Рис. 3.46. HTML файл для створення домашньої сторінки

Для остаточного оформлення домашньої сторінки, також задані параметри стилю. Для цього були створені директорії static та style.

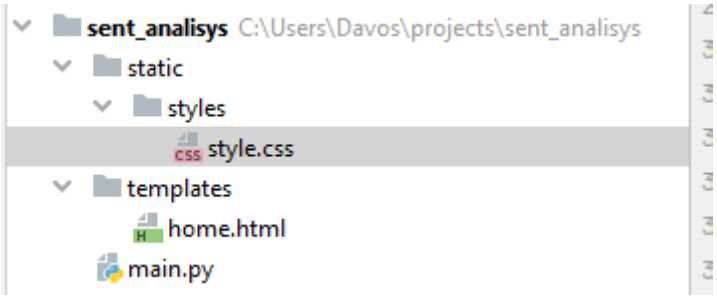

Рис. 3.47. Структура проєкту

В директорії style було створено css файл з зазначенням всіх необхідних параметрів.

main.py | req.txt | style.css | home.html

*.css files are supported by IntelliJ IDEA Ultimate

```
body {
font: 100% Arial, Helvetica, sans-serif;
line-height: 1.5;
position: relative;
background-color: rgb(3, 65, 65);;
font-weight:normal;
font-style:normal;

width: 1280px;
margin: 20 auto;
padding: 20px;
}

.container-lg {
width: 80%;
height: 550px;
color: rgb(20,20,20);
background-color: rgb(247, 247, 247);
border-radius: 20px;
text-align: center;
}

.heading {
text-align: center;
```

Рис. 3.48. Структура файлу style.css

Наведемо приклад роботи розробленого мікросервісу. У якості тестового рядка використаємо просте позитивне «I like it».

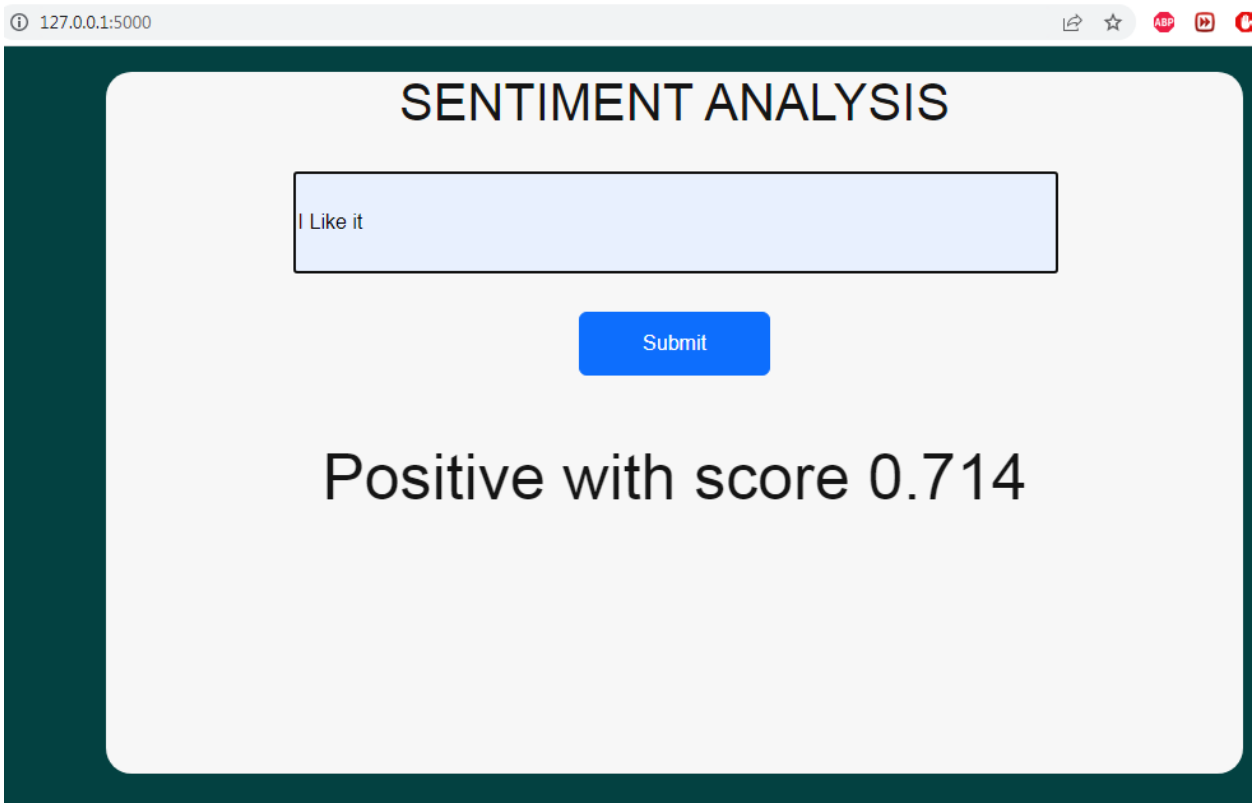


Рис. 3.49. Приклад роботи мікросервісу для позитивного рядка

Модель впевнена на 71%, що переданий рядок є позитивним.

Спробуємо перевірити модель на рядку з негативною тональністю.

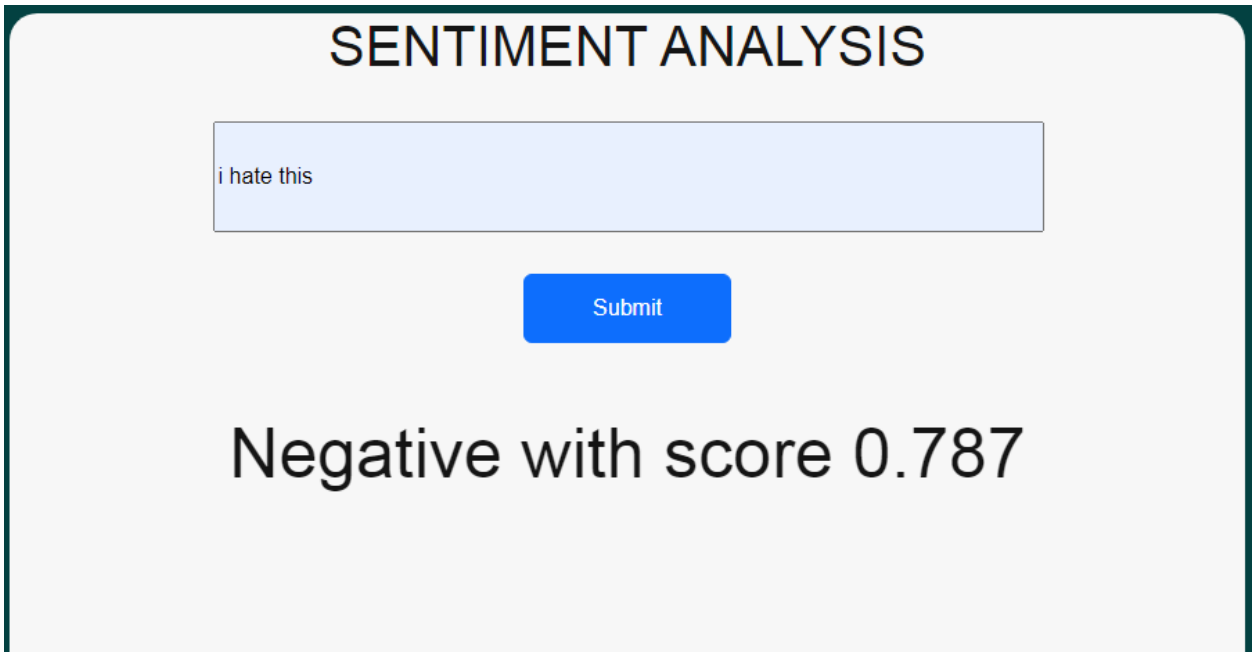


Рис. 3.50. Приклад функціонування моделі на прикладі негативного рядка

Таким чином, модель впевнена на 79%, що переданий рядок є негативним. Але є і деякі проблеми у цієї базової моделі, а саме, передамо наступний рядок «I like this place».

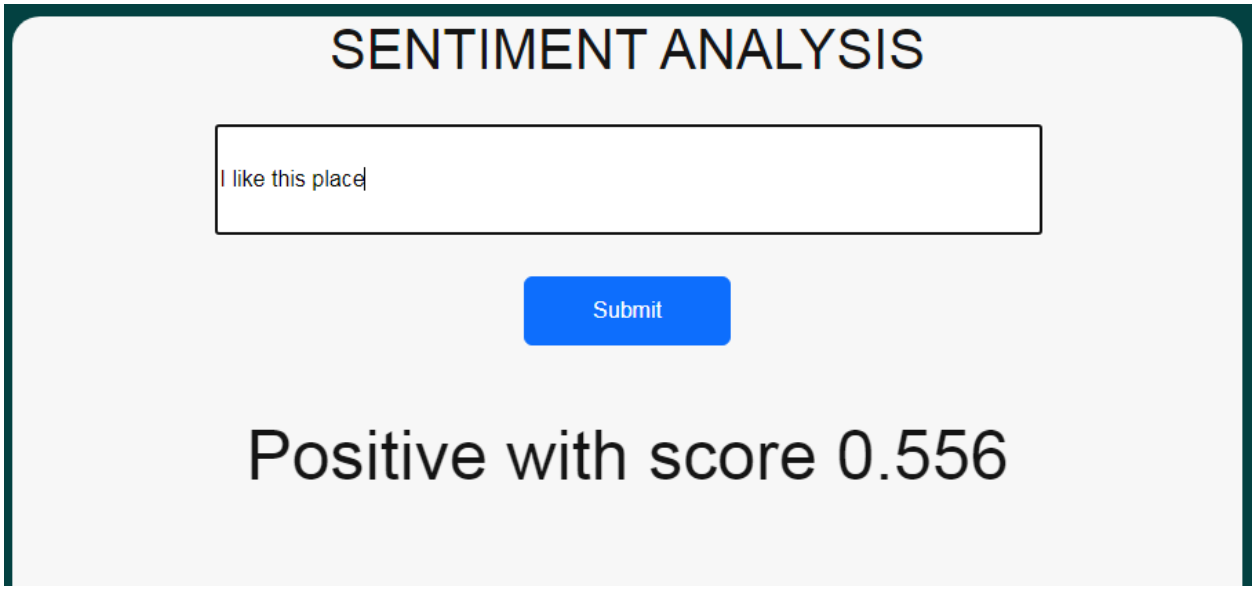


Рис. 3.51. Оцінка рядку моделлю тональності рядку «I like this place»

Таким чином, незважаючи на те, що рядок «I like this place» має позитивний окрас , модель впевнення в цьому всього на 56%. Тому доцільно замінити модель і використовувати нейронну мережу.

В перший раз доведеться завантажити модель і токенайзер.

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b (https://huggingface.co/distilbert-base
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100%|██████████| 629/629 [00:00<00:00, 20.2kB/s]
model.safetensors: 100%|██████████| 268M/268M [03:12<00:00, 1.39MB/s]
tokenizer_config.json: 100%|██████████| 48.0/48.0 [00:00<00:00, 3.08kB/s]
vocab.txt: 100%|██████████| 232k/232k [00:00<00:00, 928kB/s]
```

У якості моделі було взято облегшену модель BERT (distilbert). При цьому треба було зробити мінімальні зміни в коді (див. рисунок).

```python
from transformers import pipeline
app = Flask(__name__)

@app.route('/', methods=["GET", "POST"])
def main():
    if request.method == "POST":
        inp = request.form.get('inp')
        nlp_sa = pipeline("sentiment-analysis")
        score = nlp_sa(inp)
        return render_template('home.html', message=f"{score[0]['label']} with score {round(score[0]
    return render_template('home.html')
```

Отримаємо оцінку тональності рядка «I like this place» нейронною мережею.

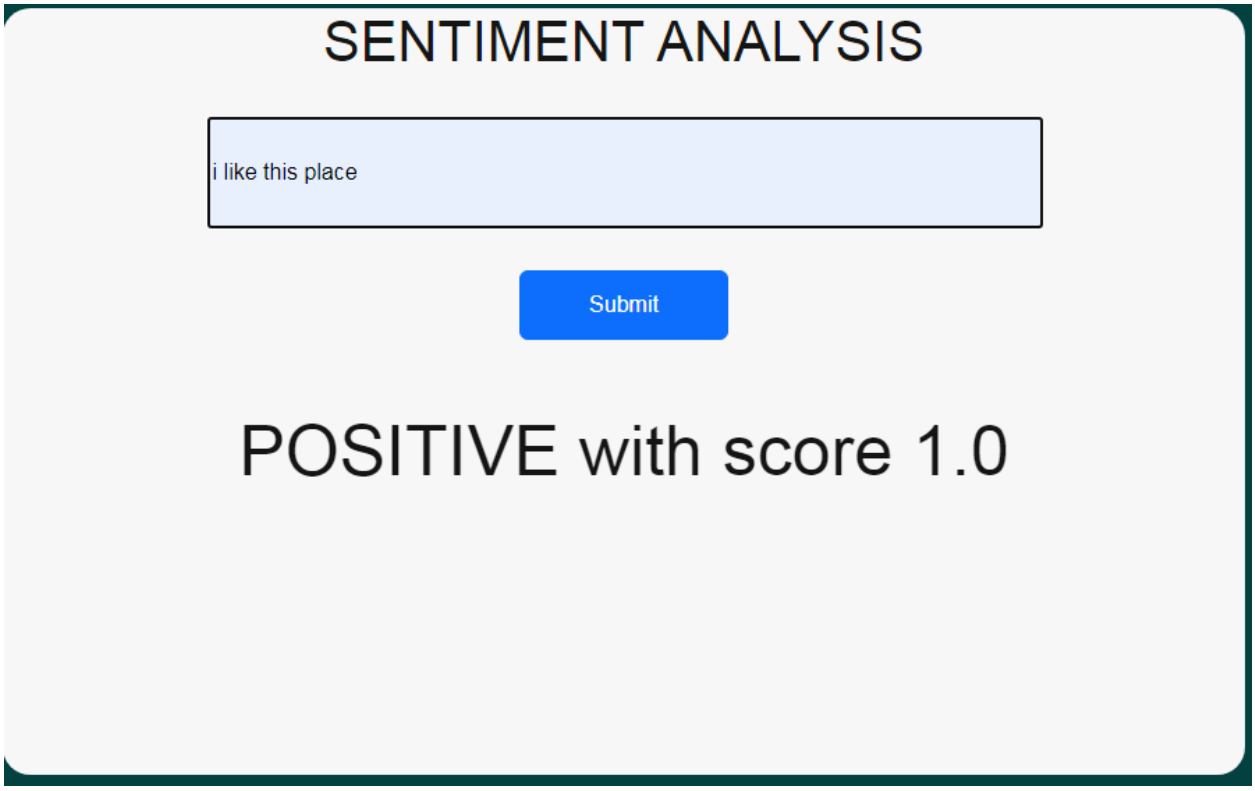


Рис. 3.52. Результат оцінки тональності рядка нейронною мережею

Таким чином, модель BERT майже на 100% впевнена, що даний рядок є позитивним і хоча зазначено, що score дорівнює 100%, це не зовсім так, реальний score дійсно дуже близький до 1, а саме 0.99985.

## 3.5. Висновки до розділу

Не тільки нейронні мережі показують непогані результати при роботі з текстовими даними. Класичні методи обробки даних та моделі машинного навчання все також дають дуже гарні результати. Особливо перевага таких методів проявляється коли даних недостатньо багато і нейронна мережа попросту не встигає якісно навчитесь, у такому випадку на допомогу приходять провірені класичні методики.

Класифікація зображень і нейромережеві моделі прогнозування часових рядів - це активні області досліджень. Дослідники працюють над розробкою нових методів, які можуть підвищити точність цих завдань.

Аналіз різних підходів в оцінці тональності текстів, дав змогу виявити, що для розробки міркосерсу найбільш ефективним є використання сучасних нейронних мереж, а саме моделі BERT різної модифікації, однак тут треба обирати між точністю та швидкістю.

# ВИСНОВКИ

Об’єктом дослідження є процес розробки програмних засобів реалізації нейромережевого моделювання.

Предметом дослідження є методи і практичні підходи до розробки програмних засобів реалізації нейромережевого моделювання.

В першому розділі проаналізовано предметну область. Досліджено вплив нейронних мереж на бізнес та людське життя. Розглянуті напрямки нейромережевого моделювання. Проведено аналіз до апаратного та програмного забезпечення для розробки нейронних мереж.

У другому розділі розглянуті основні бібліотеки для для розробки нейронних мереж. Проаналізовано інструменти реалізації нейронних мереж та особливості та відмінності згорткових та рекурентних нейронних мереж.

В третьому розділі реалізовано вирішення задачі класифікації тексту класичними алгоритмами машинного навчання. Продемонстровано способи вирішення задач обробки природної мови за допомоги нейронних мереж. Виконано реалізацію вирішення задач класифікації зображень і нейромережевих моделей прогнозування часових рядів. Розроблено мікросервіс аналізу тональності тексту.

Результатом роботи є розроблений мікросервіс аналізу тональності тексту,який може бути використаний спеціалістами різного профілю, такими як аналітики, маркетологи, служба підтримки і т.п.

Мікросервіси мають низку переваг, які роблять їх привабливим вибором для оцінки тональності тексту. Ці переваги можуть допомогти зробити оцінку тональності тексту більш адаптивною, масштабованою та стійкою до помилок.

Розроблений мікросервіс є самостійним і автономним, оскільки він не залежить від інших мікросервісів. Він може бути розміщений на будь-якому сервері та взаємодіяти з іншими мікросервісами через HTTP-запити і відповіді.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дистильована (облегшена) версія базової моделі BERT [Електронний ресурс] – Режим доступу: https://huggingface.co/distilbert-base-cased − Дата доступу: 21.09.23

2. Забезпечення безпеки малих сільськогосподарських підприємств з використанням технологій машинного зору / Н.О. Іванченко, Д.М. Квашук , О.С. Подскребко , О.М. Густера - Středoevropský věstník pro vědu a výzkum. – 2020. –№ 2 (63). – С. 51-56.

3. Інструменти аналізу ефективності взаємодії користувача з продуктом / Н.О. Іванченко, О.С. Подскребко. Proceedings of the IV International Scientific and Theoretical Conference, November 3, 2023. Bern, Swiss Confederation: International Center of Scientific Research, P.21-24.

4. Концептуальний підхід бізнесу до Data Science / Н.О. Іванченко, О.С. Подскребко Proceedings of the III International Scientific and Theoretical Conference, August 19, 2022. Tel Aviv, State of Israel: 2022. P. 134-135

5. Матвійчук А.В. Штучний інтелект в економіці: нейронні мережі, нечітка логіка: монографія / А. В. Матвійчук. – К.: КНЕУ, 2011. – 439 с.

6. Офіційна сторінка змагання з Plant Pathology [Електронний ресурс] – Режим доступу: https://www.kaggle.com/competitions/plant-pathology-2021-fgvc8 − Дата доступу: 30.09.23

7. Офіційна сторінка змагання з оцінки читабельності текстів (Readability) [Електронний ресурс] – Режим доступу: https://www.kaggle.com/competitions/commonlitreadabilityprize − Дата доступу: 02.09.23

8. Сайт з текстами на англійській мові [Електронний ресурс] – Режим доступу: https://kids.britannica.com/kids/article/Ukraine/345809 − Дата доступу: 01.10.23

9. Сайт компанії Hugging Face [Електронний ресурс] – Режим доступу: https://huggingface.co/ − Дата доступу: 01.10.23

10. Список stopwords для української мови. Режим доступу: https://github.com/skupriienko/Ukrainian-Stopwords/blob/master/stopwords_ua_list.txt − Дата доступу: 11.09.23

11. Тимощук П.В. Штучні нейронні мережі: навч. посібник/П. В.Тимощук. — Львів: Видавництво Львівської політехніки, 2011. — 444 с.

12. Artificial Intelligence: A Modern Approach / Stuart Russell, Peter Norvig. - Pearson India Education Services Private Limited, 2022. – 1292 p.

13. Business Intelligence, Analytics, and Data Science: A Managerial Perspective / Ramesh Sharda, Dursun Delen, Efraim Turban. - Pearson; 4th edition, 2017. – 512 p.

14. Deep Learning (Adaptive Computation and Machine Learning series) / Ian Goodfellow, Yoshua Bengio, Aaron Courville. - The MIT Press, 2016. – 800 p.

15. Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD / Jeremy Howard, Sylvain Gugger. - O'Reilly Media; 1st edition, 2020. – 621 p.

16. Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms / Nithin Buduma, Nikhil Buduma, Joe Papa. - O'Reilly Media; 2nd edition, 2022. – 387 p.

17. Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies / Kelleher John D., Macnamee Brian, D`Arcy Aoife. - The MIT Press; 2nd edition, 2020. – 856 p.

18. Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow / Magnus Ekman. - Addison-Wesley Professional; 1st edition, 2021. – 752 p.

19. Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2 / Sebastian Raschka and Vahid Mirjalili. - Packt Publishing; 3rd edition, 2019. - 772 p.

20. Machine Learning for Business Analytics: Concepts, Techniques and Applications in RapidMiner 1st Edition / Galit Shmueli, Peter C. Bruce, Amit V. Deokar, Nitin R. Patel. - Wiley; 1st edition, 2023. – 736 p.

21. Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python / Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili. - Packt Publishing, 2022. - 774 p.

22. Machine Learning: The Ultimate Guide to Machine Learning, Neural Networks and Deep Learning for Beginners Who Want to Understand Applications, Artificial Intelligence, Data Mining, Big Data and More / Herbert Jones. - Bravex Publications, 2020. – 182 p.

23. Neural Network Projects with Python: The ultimate guide to using Python to explore the true power of neural networks through six projects / James Loy. - Packt Publishing, 2019. – 308 p.

24. Neural Networks for Beginners: An Easy-to-Follow Introduction to Artificial Intelligence and Deep Learning / Brian Murray. - Springer; 3rd ed., 2023. – 476 p.

25. Python Data Science Handbook: Essential Tools for Working with Data / Jake VanderPlas. - O'Reilly Media; 2nd edition, 2023. – 588 p.

26. Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter / Wes McKinney. - O'Reilly Media; 3rd edition, 2022. – 579 p.

27. The Handbook of Data Science and AI: Generate Value from Data with Machine Learning and Data Analytics / Stefan Papp, Wolfgang Weidinger, Katherine Munro, Bernhard Ortner and oth. - Hanser Publications, 2022. - 576 p.

# ДОДАТОК А

## Фрагмент коду створення мікросервісу

```
from flask import Flask, render_template, request
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# import nltk
# nltk.download('vader_lexicon')

#Імпорт необхідних бібліотек
from transformers import pipeline
# Створення пайплайну
nlp_sa = pipeline("sentiment-analysis")
result_pos = nlp_sa('I like this place')#[0]
print(result_pos)

app = Flask(__name__)

@app.route('/', methods=["GET", "POST"])
def main():
    if request.method == "POST":
        inp = request.form.get('inp')
        nlp_sa = pipeline("sentiment-analysis")
        score = nlp_sa(inp)
        return render_template('home.html', message=f"{score[0]['label']} with score
{round(score[0]['score'],2)}")
    return render_template('home.html')
```

# ДОДАТОК Б

## Фрагмент коду створення мікросервісу (реалізація template)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
crossorigin="anonymous">
    <title>Sentiment Analysis</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='styles/style.css')}}"
</head>
<div class="container">
    <div class="container-lg">
        <h1>SENTIMENT ANALYSIS</h1>
        <form method="POST" class="form">
            <input type="text" name="inp" placeholder="Enter Your Feeling" id="inp">
            <input type="submit" name="submit" id="btn" class="btn btn-primary">

        </form>
        <h1 class="output">{{message}}</h1>
    </div>
</div>
<body>

</body>
</html>
```

# ДОДАТОК В

## Фрагмент коду вирішення NLP задач

```
# Імпорт необхідних бібліотек
from transformers import pipeline
nlp_sent_anal = pipeline("sentiment-analysis")

result_1 = nlp_sent_anal('Many people think you have the best work')[0]
print(f"label: {result_1['label']}, with score: {round(result_1['score'], 4)}")
result_2 = nlp_sent_anal("Only stupid people think you have the best work")[0]
print(f"label: {result_2['label']}, with score: {round(result_2['score'], 4)}")

from transformers import AutoTokenizer, AutoModelForSequenceClassification

tokenizer = AutoTokenizer.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment-latest")
model = AutoModelForSequenceClassification.from_pretrained("cardiffnlp/twitter-roberta-base-
sentiment-latest")

from scipy.special import softmax

sequence_0 = "Many people think you have the best work"
sequence_1 = "Only stupid people think you have the best work"

sentences = [sequence_0, sequence_1]

for i in sentences:
    tokenizers = tokenizer(i, return_tensors ='pt')
    output = model(**tokenizers)

    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
```

```
print(f"""{i}:
    neg:    {round(scores[0]*1,3)},
    neutral:{round(scores[1]*1, 3)},
    pos:    {round(scores[2]*1, 3)}""")
```