Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Аналіз неструктурованих текстів, записаних природньою мовою

Текстова частина до курсової роботи

за спеціальністю „Прикладна математика” 113

Керівник курсової роботи

к.т.н., доц. Жежерун О. П

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2023 р.

Виконав студент Поздняков А.О.

(прізвище та ініціали)

“____” _____ 2023 р.

м. Київ – 2023 рік

Календарний план виконання курсової роботи

Тема: Аналіз неструктурованих текстів, записаних природньою мовою

Календарний план виконання роботи:

№ п\п	Назва етапу дипломного проекту(роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень 2021р.	
2.	Робота з літературою за темою обробки природньої мови	Листопад - березень 2022р.	
3.	Аналіз методів NLP та рекурентних нейронних мереж	Березень - квітень 2022р.	
4.	Підготовка датасету та створення рекурентної нейронної мережі	Квітень - травень 2022р.	
5.	Перегляд змісту роботи керівником		
6.	Створення презентації		
7.	Захист курсової роботи		

Студент Поздняков А.О. _____

Керівник Жежерун О.П. _____

“_____”_____ 2023 р.

Зміст

Вступ

В сучасному світі інформаційні технології розвиваються неймовірними темпами. Це спричиняє щосекундну генерацію величезних обсягів інформації та даних. Крім того, за останні роки ці обсяги зросли в декілька разів, а більшість таких даних є неструктурованими, такими як звичайний текст.

В таких умовах перед фахівцями постає новий виклик - аналіз неструктурованих текстів з метою виявлення корисних знань або паттернів. Для цього використовують цілу галузь Computer Science - обробку природньої мови (Natural Language Processing), яка направлена на роботу з текстом та враховує його особливості.

Ця робота має на меті вивчення та застосування математичних методів аналізу неструктурованих текстів, записаних природньою мовою.

Розділ 1. Обробка природньої мови

1.1 Вступ до обробки природньої мови

Як вже було згадано у вступі, кожного дня генерується величезна кількість неструктурованого тексту, який не має конкретної організації або формату і ускладнює обробку та аналіз таких текстів. Зокрема, ця неструктурованість спричиняє низку проблем, оскільки стає неможливим використовувати загальні методи аналізу та алгоритми. Основним інструментом в такому випадку стає обробка природньої мови (Natural Language Processing), яка поєднує комп'ютерну лінгвістику, статистику та штучний інтелект. Обробка природньої мови використовується для розуміння та генерація мовлення, машинного перекладу, аналізу тональності тексту тощо.

1.2 Основні задачі в обробці природньої мови

Через важливу функцію мови у житті людини спектр задач у галузі обробки природньої мови є дуже обширним. Зокрема можна виділити декілька основних напрямків, які включають в себе:

- розпізнавання та класифікацію слів;
- синтаксичний аналіз (визначення синтаксичної структури речень та фраз, залежностей між словами);
- семантичний аналіз (визначення семантичного значення слів та фраз, семантичних відношень між ними);
- класифікація тексту (визначення категорії до якої належить текст);
- машинний переклад (переклад тексту з однієї мови на іншу);
- генерація тексту (створення тексту подібного до того, що створює людина, за визначеними правилами та контекстом).

1.3 Базові методи та інструменти в обробці природньої мови

Обробка природньої мови включає в себе доволі широке різноманіття інструментів. Розглянемо декілька базових інструментів, які зустрічаються у більшості випадків.

1.3.1 Стоп-слова

Найпоширенішим інструментом обробки природньої мови можна назвати видалення стоп-слів. Стоп-словами вважаються слова, які не несуть за собою велике змістове навантаження і повторюються надто часто. В основному це займенники, прийменники тощо.

```
from nltk.corpus import stopwords
print(stopwords.words('english')[:20])
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'yourself', 'yourselves', 'he', 'him', 'his']
```

Рисунок 1.1 Словник англійських стоп-слів бібліотеки nltk на Python

Для прикладу використаємо речення “This sentence is an example to show, which words are detected as stop-words” та знайдемо стоп-слова:

```
Початкове речення: This sentence is an example to show, which words are detected as stop-words
Знайдені стоп-слова: ['this', 'is', 'an', 'to', 'which', 'are', 'as']
```

Рисунок 1.2 Приклад знайдених стоп-слів у реченні

1.3.2 Токенізація

На практиці, для того, щоб визначити стоп-слова речення спочатку треба токенізувати речення. Токенізація - це один з видів обробки тексту,

який полягає в його розбитті на токени (окремі компоненти, зазвичай слова). Токенізація застосовується для того, щоб перевести текст в більш зрозумілий для комп’ютера формат, широко розширюючи можливість застосування інших операцій на окремі слова.

```
Початкове речення: This sentence is an example to show, which words are detected as stop-words
Токенізоване речення: ['this', 'sentence', 'is', 'an', 'example', 'to', 'show', ',', 'which', 'words', 'are', 'detected', 'as', 'stop-words']
```

Рисунок 1.3 Приклад токенізації речення

Як можна побачити на рисунку, токенізація розділяє слова на основі наявності знаку пробілу між ними. Крім того, також відокремлюються розділові знаки, але зазвичай їх просто видаляють, оскільки вони, як і стоп слова, майже не вливають на зміст.

1.3.3 Стемінг та лематизація

Наступними інструментами, що використовуються після токенізації є стемінг та лематизація. Загалом, їх ідея полягає в одному і тому ж - звести слово до більш простої загальної форми. Але відмінність полягає в тому, що стемінг просто вирізає зайві частини слова, іноді лишаючи навіть неіснуючі слова, а лематизація зводить все до леми - канонічної форми слова.

Існує декілька варіантів алгоритмів стемінгу, але найпопулярнішим з них є алгоритм Портера. Він зводить слово до загальної форми шляхом послідовного видалення закінчень. І, хоча він здатен працювати з будь якою мовою та має кращі результати, ніж інші алгоритми, іноді він видаляє занадто багато символів і на виході залишаються неіснуючі слова, наприклад як стеми “thi” та “sentenc” на рисунку, що походять від слів “thi” та “sentence” відповідно.

```
Початкове речення: This sentence is an example to show, which words are detected as stop-words
Знайдені стеми: ['thi', 'sentenc', 'is', 'an', 'exampl', 'to', 'show', ',', 'which', 'word', 'are', 'detect', 'as', 'stop-word']
```

Рисунок 1.4 Приклад використання стемінгу

Лематизація ж є точнішою за стемінг, але задля її роботи необхідні сформовані мовні словники, через що вона реалізована не для всіх мов та займає більше часу. Тому інколи замість неї обирають стемінг, аби знайти компроміс між часом та точністю. Для прикладу застосуємо лематизацію та отримаємо наступні леми для слів нашого речення “This sentence is an example to show, which words are detected as stop-words”:

```
Початкове речення: This sentence is an example to show, which words are detected as stop-words
Знайдені леми: ['this', 'sentence', 'is', 'an', 'example', 'to', 'show', ',', 'which', 'word', 'are', 'detected', 'a', 'stop-words']
```

Рисунок 1.5 Приклад лематизації речення

Розділ 2. Векторний простір

2.1 Векторний простір

Векторний простір - це математична модель, за допомогою якої можна представити об'єкти у вигляді векторів у *n*-вимірному просторі. В цій моделі кожен вектор відповідає окремому об'єкту, а його координати є відображенням певних його характеристик. У випадку аналізу текстів документи може бути представленим як вектор у векторному просторі, де кожна координата представляє важливість певного слова.

Таке представлення тексту дає фундаментальну можливість застосовувати математичні методи для обробки і аналізу текстів. Це значно розширює варіативність застосовуваних методів. Крім того, векторне представлення тексту дозволяє здійснювати операції задля визначення схожості документів, використовувати їх для кластеризації, класифікації тощо.

2.2 Мішок слів

Один з найпоширеніших способів побудови векторного простору для тексту - це мішок слів (bag of words). Суть цього підходу полягає у представленні тексту у виді вектору, кожна координата якого відповідає окремому слову, а значення координати відображає кількість входжень цього слова в документі.

Цей підхід є доволі простим у реалізації:

1. Будується словник унікальних слів, які зустрічаються у всьому наборі тексту.
2. Для кожного документа створюється свій окремий вектор, кожна координата якого відповідає слову зі словника, а значення цієї координати представляє кількість разів, скільки слово зустрічається у документі.

Приклад застосування bag of words наведено на рисунку:

Початкові речення:

```
["Cat loves fish",
 "Dog loves meat",
 "Cat loves cat"]
```

Матриця Bag of words:

cat	loves	fish	meat
1	1	1	0
0	1	0	1
2	1	0	0

Рисунок 2.1 Приклад застосування методу bag of words для кількох речень

Однак, bag of words має декілька недоліків - він не враховує порядок слів у документах і семантичні залежності між словами. Крім того, до сильного збільшення необхідного обсягу пам’яті призводять великі тексті. В такому випадку словнику унікальних слів збільшується, що призводить до значної кількості нулів у кінцевій матриці, оскільки деякі слова з одного документу можуть зовсім не зустрічатися в іншому.

2.3 TF-IDF

TF-IDF - статистичний метод, який використовується для оцінки важливості слів (термів) у текстових документах приведених до векторного простору. Він обчислює значення, яке враховує частоту терміну в документі (Term Frequency) та інверсну частоту терміну в корпусі документів (Inverse Document Frequency).

TF показує, наскільки часто термін зустрічається у конкретному документі, а терміни з вищою частотою є більш важливими для документу. Term Frequency розраховується за наступною формулою:

$$TF = \frac{n_t}{\sum_{i=1}^{k} n_i},$$

де n_t - кількість окремих слів у документі, а $\sum_{i=1}^{k} n_i$ - загальна кількість слів у документі.

Inverse Document Frequency (IDF), виходячи з назви, відображає зворотнє - наскільки документ є рідкісним або навпаки загальним. Високе значення IDF означає більшу вагомість терміну, який рідко зустрічається у всьому корпусі документів, і менше значення для загальновживаних термінів. IDF обчислюється за формулою:

$$IDF = ln(\frac{n_c}{\sum_{j=1}^{m} n_j}),$$

де n_c - загальна кількість документів у корпусі, $\sum_{j=1}^{m} n_j$ - кількість документів, в яких є дане слово.

Формула для обчислення TF-IDF виглядає так:

$$TF\text{-}IDF = TF * IDF$$

Завдяки TF-IDF можна порівнювати документи між собою та шукати схожість та відмінності між ними.

2.4 N-грами

N-грами також є доволі важливим інструментом обробки тексту, представленого у векторному просторі. Зазвичай їх використовують задля аналізу послідовностей слів у тексті. Найпоширенішим варіантом N-грам є біграми та триграми, що складаються з двох та трьох суміжних слів відповідно.

Біграми допомагають враховувати контекст та взаємозв’язок слів у реченні. На прикладі речення “Cat loves fish” можна виділити дві біграми “Cat loves” та “loves fish”.

Триграми враховують більш глибокі зв’язки та контекст у тексті. На прикладі речення “Cat loves fish” буде сформована єдина триграма - “cat loves fish”, оскільки це єдина послідовна комбінація трьох слів у цьому реченні.

Загалом, іноді використовують N-грами вищого порядку. Хоча вони й можуть враховувати складніші зв’язки у тексті, часто їх використання є надлишковим.

Завдяки використанню N-грам можна значно поширити різномаїття тексту у векторному простору, який буде представлений не тільки поодинокими словами, а й змістовними комбінаціями з декількох.

2.5 Word2Vec

Техніка Word2Vec базується на ідеї, що слова, які зустрічаються в подібних контекстах, мають подібні семантичні значення. Word2Vec відображає слова у векторний простір, в якому подібні слова розташовані близько одне до одного. Мабуть найвідомішим прикладом застосування Word2Vec є:

"король" - "чоловік" + "жінка" = "королева"

Word2Vec використовує дві основні архітектури:

- Continuous Bag of Words (CBOW)
- Skip-gram

Обидві ці архітектури будуються на нейронних мережах, які треба навчити на корпусі текстових даних.

2.5.1 Continuous Bag of Words

Принцип роботи CBOW полягає в тому, щоб передбачити слово на основі оточуючого контексту. У CBOW модель отримує набір слів, які є контекстом, і намагається передбачити центральне слово. Наприклад, у випадку речення "Cat loves fish", модель може отримати контекст {“cat”, “fish”}, і буде намагатися передбачити слово “loves”.

Тренування CBOW моделі складається з побудови нейронної мережі з одним прихованим шаром. Кожне слово у тренувальному тексті зводиться до векторної форми, а сама мережа намагається знайти оптимальні ваги, щоб якнайточніше передбачити центральне слово, враховуючи контекст.

2.5.2 Skip-gram

Skip-gram працює абсолютно навпаки порівняно з CBOW. У Skip-gram моделі центральне слово використовується для того, щоб передбачити контекст. Таким чином в розглянутому раніше реченні “Cat loves fish” модель отримує центральне слово "cat" і намагається передбачити його контекст навколо цього слова. Тобто модель спробує передбачити контекст {“cat”, “fish”} навколо слова “cat”, яке вона отримає на вході.

Аналогічно до CBOW, тренування Skip-gram моделі включає побудову нейронної мережі з одним прихованим шаром.

2.5.3 Переваги Word2Vec

Такий потужний інструмент, як Word2Vec, звісно ж, має декілька серйозних переваг:

- можливість отримати семантичні вектори для слів, завдяки яким можна визначити семантичні відношення до інших слів у тексті;
- суттєве зменшення розмірності даних, що допомагає оптимізувати використання пам’яті.

Розділ 3. Сентимент-аналіз та класифікація текстів

3.1 Вступ до сентимент-аналізу та класифікації текстів.

Сентимент-аналіз, також відомий як аналіз настроїв або оцінка емоцій, є процесом визначення емоційного тону або ставлення до тексту, записаного природньою мовою. Він знаходить широке застосування в галузях, таких як соціальний медіа моніторинг, відгуки користувачів, аналіз громадської думки та багато інших.

Класифікація текстів є процес присвоєння тексту певної мітки. Ця мітка може бути майже будь-якою характеристикою тексту (наприклад, спам або не спам). Для отримання точного результату класифікації використовуються різноманітні методи, які аналізують структуру та зміст тексту для визначення категорії, до якої він належить.

Сентимент-аналіз і класифікація текстів можуть бути реалізованими з використанням різних методів машинного навчання, і одним з них є рекурентні нейронні мережі (RNN).

3.2 Рекурентні нейронні мережі (RNN)

Рекурентні нейронні мережі (RNN) одним з видів нейронних мереж, які використовуються для обробки послідовних даних, таких як текст, мовлення, часові ряди тощо. У рекурентних мережах зв'язок між нейронами йде не тільки за класичною схемою (від нижнього шару до верхнього), а й від нейрона до попереднього значення цього нейрона або інших нейронів того ж шару. Така особливість дозволяє відобразити залежність змінної від своїх значень у різні моменти часу t - нейрон навчається використовувати не тільки поточний вхід і те, що з ним зробили

попередні нейрони, а й те, що відбувалося з самим нейроном та іншими нейронами на попередніх входах.

3.2.1 Структура мережі RNN

Звичайні нейронні мережі навчаються шляхом визначення оптимальних значень параметрів. Спочатку навчання ваги мережі зазвичай ініціалізуються випадковими значеннями. Потім для кожного вхідного сигналу виконується пряме поширення сигналу через мережу, що дає вихідне значення. Після цього обчислюється функція втрат, яка визначає наскільки відповідь мережі була правильною. Зазвичай для мінімізації функції втрат використовується такий алгоритм оптимізації, як градієнтний спуск, за допомогою якого коригуються ваги нейронної мережі.

Основна відмінність мережі RNN від звичайної нейронної мережі полягає в тому, що ваги на кожному шарі однакові, а всі шари ділять одні й самі змінні між собою.

Розглянемо найпростішу архітектуру рекурентної мережі - SimpleRNN. Використаємо нейрони з сигмоїдом в якості функції активації і будемо вважати, що всі переходи, крім зазначеного сигмоїда, лінійні, а на виході ми вирішуємо задачу класифікації за допомогою функції активації h. Архітектуру такої мережі зображено на рисунку 3.2.1 (члени b і c маються на увазі в стрілочках):

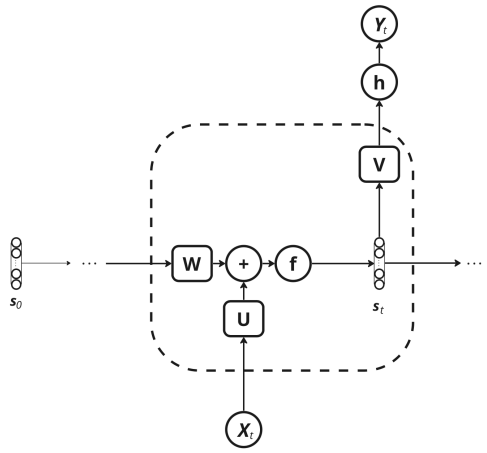


Рисунок 3.1 Архітектура комірки RNN

Нехай W - матриця вагів для переходу між прихованими станами, U - матриця вагів для входів, V - матриця вагів для виходів. Входи та приховані стани вважатимемо векторами. Крім того, вважатимемо, що на вхід завжди подається один вектор x. В результаті ми отримуємо наступну задачу мережі в момент часу t:

$$a_t = b + Ws_{t-1} + Ux_t$$

$$s_t = f(a_t)$$

$$o_t = c + Vs_t$$

$$y_t = h(o_t)$$

В цій задачі f - це нелінійність рекурентної мережі (наприклад, такі функції активації, як сигмоїда, тангенс тощо), h - функція, за допомогою якої виходить відповідь (наприклад, softmax у випадку класифікації). Блоком RNN називається частина від входів до обчислення o_t. Шар RNN - це блок, розгорнутий у часі на вхідну послідовність.

3.2.2 Навчання мережі RNN. Backpropagation through time

Backpropagation through time (BPTT) - модифікація стандартної версії алгоритму backpropagation, який використовується для навчання звичайних нейронних мереж прямого поширення. BPPT дозволяє мережі вчитися передбачати послідовності даних.

RNN отримує на вхід дані $x_1, x_2, ..., x_T$ та за допомогою математичних перетворень генерує послідовність вихідних даних $y_1, y_2, ..., y_T$. На кожному моменті часу t мережа обчислює вихід y_t та оновлює свій прихований стан s_t на основі поточного входу x_t та попереднього стану s_{t-1}.

Прихований стан RNN фактично є пам'яттю попередніх членів послідовності вхідних даних і оновлюється за наступною формулою:

$$s_t = f(Ws_{t-1} + Ux_t + b),$$

де W, як вже було зазначено, матриця вагів для переходу між прихованими станами, U - матриця вагів для входів, f - функція активації, а b - параметри мережі.

Першим етапом навчання RNN є обчислення прямого проходу мережі. Після того, як мережа обробить всю послідовність, за допомогою функції втрат (наприклад, середньоквадратична помилка) обчислюється помилка між передбаченням та даними, які надійшли на вході. Наступним кроком є обчислення зворотного проходу мережі за допомогою алгоритму зворотного поширення помилки. Зворотній прохід використовується для оновлення параметрів мережі, щоб зменшити значення функції втрати.

Для обчислення зворотного проходу в BPTT спочатку необхідно обчислити градієнти функції втрати по вихідних значеннях мережі $y_1, y_2, \ldots, y_T$. Нехай L - функція втрати на кожному кроці часу. Тоді градієнт функції втрати L по кожному вихідному значенню y_t можна обчислити наступним чином:

$$\frac{\partial L}{\partial y_t} = \frac{\partial L}{\partial \hat{y}_t},$$

де $\hat{y}_t$ - передбачене значення на кроці t, яке було отримане в результаті прямого проходу мережі.

Далі, для кожного кроку часу t, необхідно обчислити градієнти функції втрати по прихованому стану s_t. Це робиться за допомогою формули:

$$\frac{\partial L}{\partial s_t} = \sum_{k=t+1}^{T} \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial s_t},$$

де $\frac{\partial y_k}{\partial s_t}$ - градієнт виходу мережі y_k по прихованому стану s_t. Цей градієнт в свою чергу обчислюється за ланцюжком похідних, куди входять похідні по всім попереднім станам

$$\frac{\partial y_k}{\partial s_t} = \prod_{i=t+1}^{k-1} \frac{\partial s_{i+1}}{\partial s_i} \frac{\partial y_k}{\partial s_k},$$

де $\frac{\partial s_{i+1}}{\partial s_i}$ - градієнт прихованого стану s_{i+1} по прихованому стану s_i. Цей градієнт можна обчислити, використовуючи формулу похідної від функції активації:

$$\frac{\partial s_{i+1}}{\partial s_i} = f'(W) ,$$

де f' - саме похідна від функції активації.

Крім того, градієнт відповідності до ваг U і W можна обчислити за допомогою формул:

$$\frac{\partial L}{\partial U} = \sum_{t=1}^{T} \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial U} , \frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L}{\partial s_t} \frac{\partial s_t}{\partial W} ,$$

де $\frac{\partial y_t}{\partial U}$ - градієнт виходу мережі y_t по вагам U, а $\frac{\partial s_t}{\partial W}$ - градієнт прихованого стану s_t по вагам W. Ці градієнти можна обчислити, використовуючи ланцюг похідних:

$$\frac{\partial y_t}{\partial U} = x_t , \quad \frac{\partial s_t}{\partial W} = s_{t-1}$$

Після обчислення градієнтів для всіх вихідних значень та внутрішніх станів мережі за допомогою алгоритму градієнтного спуску оновлюються параметри мережі. Алгоритм градієнтного спуску полягає в оновленні параметрів в напрямку антиградієнту функції втрати з деяким кроком навчання (learning rate) α:

$$U_{t+1} = U_t - \alpha \frac{\partial L}{\partial U} , \quad W_{t+1} = W_t - \alpha \frac{\partial L}{\partial W} ,$$

де t - номер ітерації градієнтного спуску.

Алгоритм BPTT є ефективним та потужним інструментом для навчання рекурентних нейронних мереж, але його використання потребує уваги до ряду проблем, що можуть виникнути в процесі навчання, таких як:

- проблема зникнення або вибуху градієнту;
- проблема вибору довжини відрізка часу для зворотного поширення помилки - якщо використовувати дуже довгі відрізки часу для зворотного поширення помилки, то може виникнути проблема довгої затримки зворотного зв'язку та зниження швидкості навчання. А використання дуже коротких відрізків часу можуть призвести до того, що мережа не зможе бачити залежності у даних.
- BPTT вимагає обчислення градієнтів для кожного часового кроку та зберігання станів мережі для кожного часового кроку, що значно впливає на обчислювальну складність.

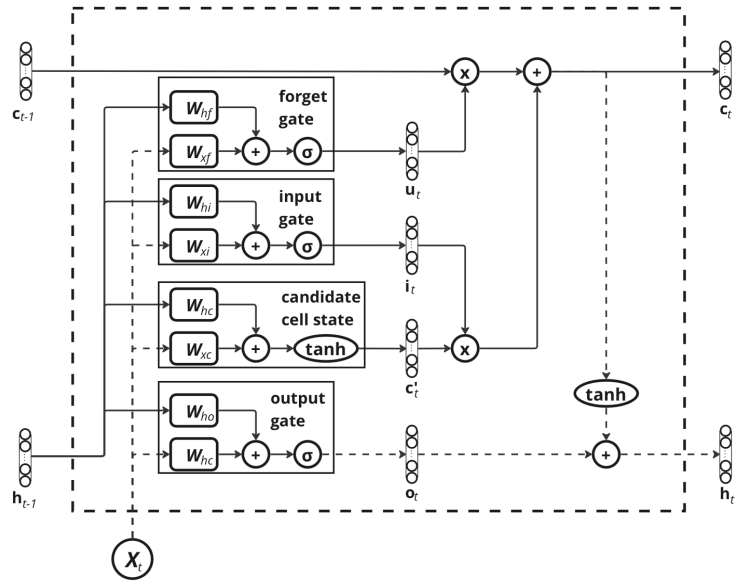
Але, враховуючи особливості рекурентних мереж, одна й та сама вага з матриці W бере участь у цьому процесі $T - 1$ раз, за кількістю переходів. Тобто рекурентна мережа — це дуже багаторівнева звичайна мережа, в якій ті ж самі ваги використовуються знову на кожному рівні. Такий підхід з використанням загальних ваг має наступні переваги:

- для їх зберігання достатньо однієї матриці;
- вони дозволяють уникати деяких проблем глибоких мереж: градієнти не зникають відразу ж.

3.3 Long Short-Term Memory (LSTM)

LSTM є покращеною структурою RNN, яка була розроблена для подолання проблеми втрати інформації на великих відстанях у часі. LSTM має здатність ефективно зберігати та використовувати інформацію залежно від контексту, що робить його особливо корисним для завдань сентимент-аналізу.

Основна ідея за LSTM полягає у використанні спеціальних рекурентних складових, які дозволяють контролювати потік інформації у мережі. Кожна складова має три основні види вузлів, які називають гейтами (gates): вхідний (input gate), вихідний (output gate) та забуваючий (forget gate). Входи контролюють, яка інформація має бути збережена, виходи регулюють, яка інформація передається наступним шарам мережі, а видалення пам'яті визначає, яка інформація повинна бути забута.



Архітектура комірки LSTM

Формально кажучи, якщо позначити через x_t вхідний вектор під час t, тоді h_t - вектор прихованого стану під час t, W_i - матриці ваг, що

застосовуються до входу, W_h — матриці ваг у рекурентних з'єднаннях, b — вектори вільних членів, ми отримаємо наступне формальне визначення того, як працює LSTM: на вході x_t, маючи прихований стан попереднього кроку h_{t-1} і власне стан комірки c_{t-1}, ми послідовно обчислюємо:

$$c'_t = tanh(Wx_c x_t + Wh_c h_{t-1} + b_{c'})$$

$$i_t = \sigma(Wx_i x_t + Wh_i h_{t-1} + b_i)$$

$$f_t = \sigma(Wx_f x_t + Wh_f h_{t-1} + b_f)$$

$$o_t = \sigma(Wx_o x_t + Wh_o h_{t-1} + b_o)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t$$

$$h_t = o_t \odot tanh(c_t)$$

Де c'_t відповідає candidate cell state на ілюстрації блоку LSTM (рисунок 3.2) , відповідно c_t - cell state, i_t - input gate, f_t - *forget gate,* o_t - output gate, h_t - block output.

Варто зазначити наявність комірки пам’яті (cell), яка присутня в кожному блоці LSTM. В її обчислені беруть участь компоненти ct-1 (вектор ваг комірки пам’яті на попередньому кроці t-1) та c’t (“кандидат” на вектор комірки пам’яті на поточному кроці). Перед тим, як c’t займе місце ct-1, вони обидва проходять через два гейти - вхідний (it) та забуваючий (ft), як у зазначеній вище формулі:

$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t$$

Таким чином нове значення “пам’яті” буде являти собою лінійну комбінацію старою “пам’яті” з коефіцієнтами забуваючого гейту (ft) і

кандидата “пам’яті” з коефіцієнтами вхідного гейту (it). Таким чином, чим меншими будуть коефіцієнти ft або it, тим меншим буде вплив значень пам’яті з попереднього кроку t-1 або поточного кроку t, і навпаки.

Завдяки розглянутим вище компонентам LSTM може ефективно моделювати довготривалі залежності в послідовностях. Він може зберігати в пам'яті інформацію з віддалених попередніх станів і використовувати її для обробки поточного вхідного сигналу. Це дозволяє LSTM враховувати контекст та залежності між різними частинами тексту при виконанні сентимент-аналізу.

3.3.1 Використання LSTM

Якщо перед нами стоїть задача классифікації тексту або сентимент-аналізу, ми можемо використовувати архітектуру LSTM для побудови моделі. Спочатку до тексту необхідно застосувати токенізацію, після цього один з методів векторизації та за необхідністю виконати інші операцій обробки тексту. Потім вектори представлення слів можуть бути використані як вхідні дані для LSTM задля вирішення задачі.

Розділ 4. Застосування аналізу неструктурованих текстів

У попередніх розділах ми розглянули та визначили, як працюють деякі з популярних методів та алгоритмів обробки природньої мови. Тепер визначимо, які проблеми можуть вирішувати ці методи для бізнесу та наукової сфери.

4.1 Аналіз соціальних мереж

Враховуючи кількість інформації, яку соціальні мережі генерують щосекундно, можна з впевненістю сказати, що ця галузь більше за інших потребує застосування методів обробки природньої мови та є дуже ефективним інструментом як для бізнесу, так і для дослідників.

Крім того, спектр задач з аналізу тексту в цій сфері є чи не найбільшим. З цих задач можна виділити:

- блокування неприпустимих дописів та коментарів;
- аналіз текстового контенту задля кластеризації користувачів за їх інтересами та подальшої рекомендації схожого контенту або контактів;
- семантичний аналіз дописів та коментарів з метою проведення соціологічних досліджень.

4.2 Аналіз відгуків та досвіду користувачів.

У компаніях, які можуть отримувати зворотній зв’язок від своїх користувачів аналіз тексту, записаного природньою мовою може напряму використовуватися з метою збільшення продажів або прибутку. Наприклад, за допомогою методів NLP можна визначати, які слова в негативних або

позитивних відгуках зустрічаються частіше, ніж інші, що допоможе звернути увагу на переваги або недоліки продукту. Для цього можна використовувати алгоритми виділення ключових слів, які описують відгуки користувачів.

4.3 Аналіз текстів з медичної сфери

Не так давно обробка природньої мови знайшла своє місце і у медичній сфері. Алгоритми здатні створити медичні рекомендації та визначити діагноз для користувачів виходячи з опису їх недуга. Звісно, ця сфера є ще доволі молодою та не варто абсолютно серйозно ставитися до порад таких алгоритмів, але вони все ж можуть стати гарними асистентами, як для початківців, так і для досвідчених лікарів.

4.4 Аналіз новин

Аналіз новин може використовуватися з метою розуміння громадських думок з того чи іншого питання. Хоча такий аналіз є суто соціологічним, на мою думку його застосування є безмежним. Наведемо декілька конкретних прикладів застосування аналізу новин:

- розуміння політичних настроїв населення під час підготовки до виборів;
- формування предметної бази для соціологічних досліджень.

Крім того, існує і прикладне застосування обробки тексту в цій галузі. За допомогою методів NLP можна виділяти ключові слова або точно відносити новину до певної тематики.

Розділ 5: Класифікація тексту за допомогою LSTM

В межах цієї роботи було реалізовано нейронну мережу на Python з використанням LSTM, яка здатна класифікувати текст. Об'єктами класифікації стали повідомлення, серед яких треба визначити спамові. Кожен такий об'єкт має мітку spam або ham в залежності від того, чи є він спамом або ні. Всього у датасеті 5572 повідомлення, яких буде достатньо, аби гарно натренувати модель. Завантажимо та обробимо ці дані.

```
# Завантажимо та виконаємо попередню обробку датасету
spam_data = pd.read_csv('spam.csv', encoding = "ISO-8859-1")[['v2', 'v1']]
spam_data.rename(columns={'v1':'category', 'v2':'message'}, inplace=True)
spam_data
```

	message	category
0	Go until jurong point, crazy.. Availabl...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA C...	spam
3	U dun say so early hor... U c already t...	ham
4	Nah I don't think he goes to usf, he li...	ham
5	FreeMsg Hey there darling it's been 3 w...	spam
6	Even my brother is not like to speak wi...	ham
7	As per your request 'Melle Melle (Oru M...	ham

Рисунок 5.1 Завантаження початкового датасету

Подивимося на процентне відношення спаму до нормальних повідомлень:

```
# Подивимося на розподіл spam та ham повідомлень
spam_data['category'].value_counts() / spam_data.shape[0] * 100
```

	category
ham	86.593683
spam	13.406317

Рисунок 5.2 Процентне відношення класів у датасеті

Приведемо категоріальну змінну “category”, яка є індикатором того, чи є повідомлення спамом, до числового виду. Спам перетворимо на 1, звичайні повідомлення на 0:

```
# Зведемо значення категорії (spam або ham) до категоріального формату, замінив spam на 1, а ham на 0
spam_data['category'].replace('spam', 1, inplace=True)
spam_data['category'].replace('ham', 0, inplace=True)
spam_data
```

	message	category
0	Go until jurong point, crazy.. Availabl...	0
1	Ok lar... Joking wif u oni...	0
2	Free entry in 2 a wkly comp to win FA C...	1
3	U dun say so early hor... U c already t...	0
4	Nah I don't think he goes to usf, he li...	0
5	FreeMsg Hey there darling it's been 3 w...	1
6	Even my brother is not like to speak wi...	0

Рисунок 5.3 Приведення категоріальної змінної “category” до числового виду

Крім того, приведемо всі повідомлення до нижнього регістру, аби під час подальшої трансформації даних слова з великими літерами не відокремлювались від таких самих, записаних у нижньому регістрі.

```
#Зведемо слова в повідомленнях до нижнього регістру
spam_data['message'] = spam_data['message'].str.lower()
spam_data
```

	message	category
0	go until jurong point, crazy.. availabl...	0
1	ok lar... joking wif u oni...	0
2	free entry in 2 a wkly comp to win fa c...	1
3	u dun say so early hor... u c already t...	0
4	nah i don't think he goes to usf, he li...	0
5	freemsg hey there darling it's been 3 w...	1
6	even my brother is not like to speak wi...	0

Рисунок 5.4 Приведення категоріальної змінної “category” до числового виду

Для подальшої обробки даних токенізуємо ці повідомлення, щоб мати змогу застосовувати методи NLP. Токенізацію виконуємо функцією word_tokenize бібліотеки nltk:

```
#Застосуємо токенізацію
spam_data['message'] = spam_data['message'].apply(lambda x: word_tokenize(x))
spam_data
```

	message	category
0	[go, until, jurong, point, ,, crazy, ...	0
1	[ok, lar, ..., joking, wif, u, oni, ...]	0
2	[free, entry, in, 2, a, wkly, comp, to,...	1
3	[u, dun, say, so, early, hor, ..., u, c...	0
4	[nah, i, do, n't, think, he, goes, to, ...	0
5	[freemsg, hey, there, darling, it, 's, ...	1
6	[even, my, brother, is, not, like, to, ...	0
7	[as, per, your, request, 'melle, melle,...	0

Рисунок 5.5 Токенізація повідомлень

Після застосування токенізації маємо змогу видалити стоп-слова. Для цього будемо перевіряти, чи є кожне окреме слово у словнику англійських стоп-слів бібліотеки nltk. Якщо слова у словнику немає - додаємо його до новоствореного повідомлення.

```
# Видалимо стоп-слова
def delete_stop_words(message):
    new_message = []

    for word in message:
        if word not in stopwords.words('english'):
            new_message.append(word)

    return new_message

spam_data['message'] = spam_data['message'].apply(delete_stop_words)
spam_data
```

	message	category
0	[go, jurong, point, ,, crazy, .., avail...	0
1	[ok, lar, ..., joking, wif, u, oni, ...]	0
2	[free, entry, 2, wkly, comp, win, fa, c...	1
3	[u, dun, say, early, hor, ..., u, c, al...	0
4	[nah, n't, think, go, usf, ,, life, aro...	0
5	[freemsg, hey, darling, 's, 3, week, 's...	1
6	[even, brother, like, speak, ., treat, ...	0
7	[per, request, 'melle, melle, (, oru, m...	0

Рисунок 5.6 Видалення стоп слів з повідомлень

Далі використаємо лематизацію, для того щоб привести різні форми слів до уніфікованої леми. Для цього використаємо функцію lemmatize класу WordNetLemmatizer бібліотеки nltk:

```
# Застосуємо лематизацію
lemmatizer = WordNetLemmatizer()

def lemmatize_message(message):
    new_message = []

    for word in message:
        new_message.append(lemmatizer.lemmatize(word))

    return new_message

spam_data['message'] = spam_data['message'].apply(lemmatize_message)
spam_data
```

	message	category
0	[go, jurong, point, ,, crazy, .., avail...	0
1	[ok, lar, ..., joking, wif, u, oni, ...]	0
2	[free, entry, 2, wkly, comp, win, fa, c...	1
3	[u, dun, say, early, hor, ..., u, c, al...	0
4	[nah, n't, think, go, usf, ,, life, aro...	0
5	[freemsg, hey, darling, 's, 3, week, 's...	1
6	[even, brother, like, speak, ., treat, ...	0
7	[per, request, 'melle, melle, (, oru, m...	0

Рисунок 5.7 Застосування лематизації до токенізованих повідомлень

Також визначимо довжину найбільшого повідомлення, це знадобиться нам під час для подальшої обробки та налаштування початкових параметрів нейронної мережі.

```
# Знайдемо значення максимальної кількості слів серед наших повідомлень, це знадобиться нам пізніше
max_len = max(spam_data['message'].apply(lambda x: len(x)))
```

Рисунок 5.8 Видалення стоп слів з повідомлень

Для навчання та тестування нейронної мережі розіб’ємо датасет на навчальну та тестову вибірки у відношенні 3:1 відповідно. Для цього використаємо функцію train_test_split бібліотеки scikit_learn:

```
#Розділимо наш датасет на тестові та тренувальні дані
X_train,X_test,Y_train,Y_test = train_test_split(spam_data['message'], spam_data['category'],test_size=0.25)
```

Рисунок 5.9 Розбиття датасету на на навчальну та тестову вибірки

Далі нам необхідно ще раз токенізувати повідомлення, але на цей раз за допомогою класу Tokenizer бібліотеки keras. Це необхідно для того, щоб привести дані до векторного простору та подальшого їх застосування під час навчання та тестування нейронної мережі. Обмежимо словник слів до 500 та використаємо допоміжні функції text_to_sequences та pad_sequences, щоб отримати наші дані у матричному числовому вигляді:

```
keras_tokenizer = Tokenizer(num_words=500)
keras_tokenizer.fit_on_texts(X_train)
#Створимо токенізовану числову послідовність для тренувального набору даних
X_train_sequences = keras_tokenizer.texts_to_sequences(X_train)
X_train_tokenized = sequence.pad_sequences(X_train_sequences,maxlen=max_len)
#Створимо токенізовану числову послідовність для тестового набору даних
X_test_sequences = keras_tokenizer.texts_to_sequences(X_test)
X_test_tokenized = sequence.pad_sequences(X_test_sequences,maxlen=max_len)
```

Рисунок 5.10 Розбиття датасету на на навчальну та тестову вибірки

Тепер, коли наші дані повністю підготовлені до процесу навчання нейронної мережі, створимо саму модель. Для цього використаємо клас Sequential бібліотеки Keras, який дозволить послідовно додавати шари до нашої мережі. Першим шаром є вхідний шар - Input з параметром shape=max_len. Це означає, що цей шар буде очікувати на вхід вектор довжиною max_len. Далі додаємо шар Embedding, який зменшує розмірність вхідного вектору, аби передати її до наступного шару. Далі

додаємо рекурентний шар LSTM, в якому буде 16 LSTM-комірок. За ним додаємо звичайний шар з 16 нейронами та функцією активації ReLU. Далі додаємо шар Dropout, який був створений задля боротьби з перенавчанням. Параметр 0.4 у цьому шарі означає, що з ймовірністю 0.4 кожен нейрон буде вимикатися під час кожного прямого проходу через нейронну мережу. І наостанок додаємо вихідний слой з 1 нейроном та функцією активації sigmoid, щоб отримати вихідне значення у проміжку [0;1].

```
# Створимо нейронну мережу за допомогою засобів бібліотеки Keras
model = Sequential()
#Вхідний шар, який буде отримувати одномірний масив з даними формату string
model.add(Input(shape=max_len))
model.add(Embedding(800, 20, input_length=max_len))
#Додаємо шар LSTM довжиною в 16
model.add(LSTM(16))
#Додаємо звичайний шар розмірністю 16 та з функцією активації ReLU
model.add(Dense(16, activation='relu'))
#Додаємо шар дропауту, як з ймовірністю 0.4 буде "вимикати" нейрони, аби уникнути перенавчання моделі
model.add(Dropout(.4))
#Додаємо шар, який буде повертати вихідне значення. Функцією активації обираємо sigmoid
model.add(Dense(1, activation='sigmoid'))
```

Рисунок 5.11 Розбиття датасету на на навчальну та тестову вибірки

Скомпілюємо модель та подивимося на її схему. Функцією втрат оберемо binary_crossentropy, а оптимізатор - RMSProp (Root Mean Squared Propagation). Метрикою, за якою будемо оцінювати якість моделі оберемо точність (accuracy).

```
model.summary()
model.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 203, 20)           16000
_________________________________________________________________
lstm (LSTM)                  (None, 16)                2368
_________________________________________________________________
dense (Dense)                (None, 16)                272
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
```

Рисунок 5.11 Параметри нейронної мережі

Приступимо до навчання моделі. Воно буде виконуватися 10 епох, обробляючи 128 прикладів за один раз. Для валідації виділимо 25% від цих прикладів.

```
model.fit(X_train_tokenized, Y_train, batch_size=128, epochs=10, validation_split=0.25)
```

```
Epoch 1/10
25/25 [==============================] - 4s 62ms/step - loss: 0.5647 - accuracy: 0.8430 - val_loss: 0.4018
- val_accuracy: 0.8584

Epoch 10/10
25/25 [==============================] - 1s 50ms/step - loss: 0.0903 - accuracy: 0.9786 - val_loss: 0.0811
- val_accuracy: 0.9809
```

Рисунок 5.12 Процес навчання нейронної мережі

Як можна побачити на рисунку 5.12, точність моделі після першої епохи приблизно 84%. Після останньої, десятої епохи, точність на навчальних даних піднялася майже 98%, а значення функції втрат значно знизилося. Перевіримо точність нейронної мережі на тестовому наборі даних:

```
test_evaluation = model.evaluate(X_test_tokenized, Y_test)
print(f"Значення функції втрат - {test_evaluation[0]}")
print(f"Точність моделі на тестовому датасеті - {test_evaluation[1]}")
```

```
44/44 [==============================] - 0s 10ms/step - loss: 0.0865 - accuracy: 0.9713
Значення функції втрат - 0.08653804659843445
Точність моделі на тестовому датасеті - 0.9712849855422974
```

Рисунок 5.13 Перевірка точності нейронної мережі на тестовому наборі даних

Виходячи з рисунку 5.13, наша модель демонструє 97% точність на наборі даних з 1393 повідомлень. Отже, наша нейронна мережа здатна з 97% ймовірністю визначати, чи є повідомлення спамом.

Висновок

Під час роботи було проаналізовано методи обробки природньої мови, таки як токенізація, стемінг, лемінг, bag of words та інші. Було розглянуто їх переваги та недоліки, сфери використання та особливості застосування. Крім того, було розглянуто алгоритми машинного навчання, а саме рекурентні нейронні мережі та їх модифікації.

В практичній частині роботи було використано зазначені методи аналізу природньої мови та створено рекурентну нейронну мережу з метою класифікації тексту, яка з 97% ймовірністю правильно відокремлює спамові повідомлення від звичайних.

Список використаних джерел

1. Ніколенко С. І., Кадурін А. А., Архангельська Є. О, *Глибоке навчання. Занурення в світ нейронних мереж*, 2018
2. Davy Cielen, Arno Meysman, Mohamed Ali, *Introducing Data Science: Big Data, Machine Learning, and more, using Python tools,* 2016
3. Hobson Lane, Cole Howard, Hannes Hapke, *Natural Language Processing in Action,* 2019
4. *Keras layers API*, https://keras.io/api/layers/
5. *NLTK Documentation*, https://www.nltk.org