Харківський національний університет імені В.Н. Каразіна

Факультет математики і інформатики

Кафедра прикладної математики

Кваліфікаційна робота

рівень: магістр

на тему «Порівняння точності класифікацій твітів при застосуванні різних типів нейронних мереж»

Виконала: студентка групи МП62 II курсу,
(другий магістерський рівень),
спеціальності 113
“Прикладна математика”
освітньо-професійної програми
“Прикладна математика”
Каруник Н.Д.В.

Керівник: старший викладач
кафедри прикладної математики
Сузікова О.Г.

Рецензент: доктор техн. наук,
доц., професор кафедри
комп’ютерно-інтегрованих
технологій,
автоматизації та робототехніки
ХНУРЕ
Ромашов Ю.В.

Харків — 2024 рік

Анотації

Каруник Нестані Дея Володимирівна. Порівняння точності класифікацій твітів при застосуванні різних типів нейронних мереж.

Кваліфікаційна робота магістра на тему «Порівняння точності класифікацій твітів при застосуванні різних типів нейронних мереж».

Кваліфікаційна робота магістра присвячена завданню порівняння точності навчання класифікації двох мереж, побудованих на основі BERT та DistilBERT. Навчання та класифікація відбуваються на основі реальних твітів з групи соціальної мережі Facebook. Перед реалізацією навчання твіти передобробляються очищенням тексту, нормалізацією і лематизацією. Майже увесь процес реалізований у Google Colab за допомогою Python.

Ключові слова: нейронні мережі, класифікація тексту, BERT, передобробка тексту, твіти.

Karunyk Nestani Deia Volodymyrivna. Comparison of the accuracy of tweet classifications using different types of neural networks.

Qualification work of the master on the topic "Comparison of the accuracy of tweet classifications using different types of neural networks."

The qualification work of the master is devoted to the task of comparing the accuracy of learning the classification of two networks built on the basis of

BERT and DistilBERT. Training and classification are based on real tweets from the Facebook social network group. Before implementing the training, tweets are pre-processed by text cleaning, normalization and lematization. Almost the entire process is implemented in Google Colab using Python.

Keywords: neural networks, text classification, BERT, text pre-processing, tweets.

Зміст

Вступ **5**

1. Основні положення теорії нейронних мереж **6**

1.1. Базова теорія нейронних мереж 6

1.2. Трансформери 10

1.3. BERT та DistilBERT 13

2. Постановка задачі класифікації **16**

2.1. Постановка задачі. Збір вхідних параметрів для задачі класифікації 16

2.2. Передобробка текстів і класифікація 17

3. Реалізація задачі класифікації за ключовими словами за допомогою нейронних мереж **21**

3.1. Програмна реалізація поставленної задачі двома мережами . 21

3.2. Аналіз результатів класифікації, висновки для групи, порівняння роботи двох мереж 28

Висновки **30**

Список використаних джерел **32**

Вступ

Мною була розглянута задача класифікації твітів за ключовими словами за допомогою нейронних мереж.

Нейронні мережі нині набирають значних обертів в технічних галузях розвитку. Ідея реалізувати алгоритми, що дозволили б машинам виконувати та автоматизовувати задачі, що виконуються гнучкими та надпотужними природними нейронними мережами – мізками, заполонила голови розробників та інноваторів. Саме тому я звернулася до цих теоретичних засад. До того ж, реалізований алгоритм буде мати можливість адаптовуватися під суміжні задачі, що робить його у якійсь мірі доволі гнучким і корисним для узагальненої задачі класифікації подібних текстів.

Сам процес класифікації було вирішено відпрацювати двома різними, хоча схожими, нейромережами. Цей етап додає дослідженню експериментальний характер, щоб продемонструвати порівняння отриманих результатів та вибір більш вигідної мережі для виконання задачі класифікації.

Предметом кваліфікаційної роботи є налаштування викладеного вище багатоетапного процесу побудови нейронної мережі для класифікації твітів.

Об'єктом кваліфікаційної роботи є ключові слова твітів користувачів соціальних мереж.

Метою кваліфікаційної роботи є порівняння роботи двох нейронних мереж, побудованих на базі BERT і DistilBERT, у задачі навчання класифікації твітів за ключовими словами.

Розділ 1. Основні положення теорії нейронних мереж

Статті [1], [2] присвячені дослідженню базової теорії нейронних мереж. У цьому розділі буде наведено основні положення цих робіт, на основі яких буде побудоване усе дослідження.

1.1. Базова теорія нейронних мереж

Означення 1.1 (Нейронна мережа)**.** Нейронна мережа — це модель, яка імітує роботу та складні процеси людського мозку. Останній має неймовірно великі обчислювальні потужності для керуванням складними процесами прийняття рішень, розпізнавання образів, тощо. Нейронні мережі складаються з вузлів (нейронів) та зв'язків (синапсів), що пов'язують вузли між собою.

З 1940-их років галузь машинного навчання та розробки нейронних мереж пережила багато змін, починаючи від становлення першої найпростішої штучної моделі нейронів і синапсів, створення персептронів до домінування RNN та CNN з більш складними варіаціями комбінацій вузлів та зв'язків.

Нейронні мережі не мають вбудованого розуміння ознак вхідних даних, але їх компоненти допомагають наблизитися до цього природного процесу: нейрони, синапси, ваги, зміщення, функція активації, тощо.

Нейрони отримують на вхід дані, які перевіряються функціями активації та іншими функціями, наприклад входженням у певний пороговий інтервал.

Синапси регулюють передачу інформації за допомогою вагів і зміщення.

Навчання нейронної мережі проходить у три етапи: отримання на вхід певного об'єму данних і обчислення (input), видача оброблених данних (output), та ітеративні процеси, які можуть допомогти обчисленням стати більш точними в певних задачах.

Нейронна мережа може складатися із десятків тисяч нейронів, що можуть утворювати шари: вхідний і вихідний шари, а також декілька прихованих шарів, що складаються із нейронів та синапсів. Прохід даних цими шарами може набувати двох виглядів: прямого ходу та оберненого.

Прямий хід:

- Вхідний шар: шар, що складається з нейронів, які приймають на вхід певний об'єм даних (input).
- Ваги та синапси: у кожного синапса є своя вага, що визначає «міцність» з'єднання. У ході навчання цей параметр синапсів може змінюватися.
- Приховані шари: нейрони у цьому шарі приймають на вхід значення, обробляють їх шляхом множення на ваги та пропускання через функцію активації та віддають на наступні шари.
- Вихідний шар: усі процеси, викладені вище, можуть повторюватися. Коли ж процес завершено, ми отримуємо кінцевий результат у вигляді вихідного шару.

Обернений хід:

- Обчислення втрат: дані, отримані на вихідному шарі, порівнюються із реальними значеннями, і різниця між ними обчислюється за допомогою функції втрат. Наприклад, для задачі лінійної регресії функція втрат приймає вигляд

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widetilde{y}_i)^2$$

- Градієнтний спуск: шляхом зміни вагів на основі похідної функції втрат по відошенню до кожного значення мережа намагається зменшити її значення.

- Навчання: під час навчання процес, викладений вище, повторюється ітеративно, даючи можливість мережі «навчитися», адаптуватися до даних та виявити певні закономірності для подальшої роботи.

- Функції активації: базуючись на усіх вхідних даних (input) функція активації визначає, чи «активувати» певний нейрон. Наприклад, це може бути ReLU, сигмоїда, тощо.

Означення 1.2 (MSE)**.** MSE — середньоквадратична помилка.

Означення 1.3 (ReLU)**.** ReLU — одна з найпоширеніших нелінійних активаційних функцій, що використовується в нейронних мережах. Вона визначається так:

$$ReLU(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

Означення 1.4 (Сигмоїда)**.** Сигмоїда — це одна з найпоширеніших активаційних функцій в нейронних мережах, яка перетворює вхідне значення в діапазон між 0 та 1. Вона має форму S-подібної кривої, тому її часто називають функцією S-подібної активації.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Типи нейронних мереж:

- Нейронна мережа прямого розповсюдження — простий тип нейронної мережі, де дані рухаються синапсами тільки в напрямку від вхідного шару до вихідного.
- Багатошаровий персептрон — нейронна мережа з трьома або більше шарами, де є один вхідний шар, один вихідний, а між ними приховані шари; використовує функцію активації.
- CNN — це тип нейронної мережі, спеціально створений для обробки зображень. Вона застосовує згорткові шари для автоматичного вивчення складних ознак з вхідних зображень, що дозволяє ефективно виконувати завдання розпізнавання та класифікації зображень. CNN справили революцію в області комп'ютерного зору та мають важливе значення для таких задач, як виявлення об'єктів і аналіз зображень.
- RNN — це тип нейронної мережі, який розроблений для обробки послідовних даних. Цей тип мережі ідеально підходить для задач, де важливі контекстні залежності, таких як прогнозування часових рядів та обробка природної мови, оскільки вона використовує цикли зворотного зв'язку, що дають змогу зберігати інформацію в мережі.
- LTSM — це вид RNN, розроблений для вирішення проблеми згасаючого градієнта під час навчання. Він використовує пам'ятні осередки та контролюючі механізми, які дозволяють вибірково зчитувати, записувати і видаляти інформацію.

1.2. Трансформери

Тепер після ознайомлення із загальними теоретичними засадами нейронних мереж, розглянемо теорію трансформерів.

Нейронна мережа трансформер — відносно нова архітектура нейронної мережі, що була представлена 2017 року та спеціалізується на вирішенні задач типу послідовність-в-послідовність, наприклад обробки текстів і природної мови. Ми вже згадували вище тип нейронних мереж, який спеціалізується на обробці послідовних данних — RNN. І дійсно, з цього все почалося, тому зупинимося на RNN ще раз.

Навідміну від звичайних нейронних мереж, RNN була розроблена, щоб приймати на вхід (input) послідовність даних, де кожний структурний елемент має вплив на сусідні елементи, що встановлює певні зв'язки між ними. Наприклад, у реченні слова умовно впливають одне на одного та встановлюють смислові зв'язки. Наприклад, якщо в реченні із пропущенним першим словом: « — світило дуже яскраво, тому мені довелося вдягти окуляри», — ви без проблем зможете відновити втрачене слово, або надати обмежений перелік слів, які пасуватимуть за контекстом.

RNN гарно пасує для обробки даних у вигляді таких структур:

- Vector-Sequence: приймає на вхід вектори фіксованого розміру, а видає вектори будь-якого розуміру.
- Sequence-Vector: приймає на вхід вектори будь-якого розміру, а видає вектори фіксованого розуміру.
- Sequence-to-Sequence: приймає на вхід послідовність даних і видає також послідовність даних. Це найбільш розповсюджений варіант використання RNN.

На жаль, RNN мають свої недоліки. По-перше, вони доволі повільні, а

по-друге вони погано можуть впоратися із довгостроковими зв'язками, коли відстань між «важливими» структурними елементами в послідовності є занадто великою. Наприклад, у реченні «Яблука ростуть на —» мережа легко визначить, що пропущене слово, безперечно, «яблуні». А ось у реченні «Я народилася в Україні, прожила в ній 23 роки та дуже люблю її культуру, а також спокійно володію — мовою» мережі буде важко визначити пропущене слово, хоча для нас людей воно також здається очевидним.

Довгострокова короткочасна пам'ять (LTSM) — це особливий вид RNN, що може вивчати довгострокові зв'язки та вирішувати проблему згасаючого градієнту. Нейрони цієї мережі мають додаткове відгалудження, що дозволяє пропустити довгострокову обробку та зберігати пам'ять більш довгий проміжок часу, але знову ж таки, цей підхід не вирішує проблему повністю.

Одним зі способів покращення проблеми згасаючого градієнту є використання шляху «фокусування» моделі на важливих частинах вхідних параметрів. Наприклад, якщо перед вами стоїть задача знайти в супермаркеті макарони, ви можете вирішити це питання двома шляхами. Першим способом ви можете йти вздовж кожної полиці супермакету, аж поки не дійдете до макаронів. Другим способом ви можете знайти відділ бакалії, далі відділ макаронів і взяти потрібну пачку. Напевно, другий спосіб буде набагато швидшим. Люди зазвичай приймають подібні рішення на підсвідомому рівні.

Attention модель відрізняється від класичної моделі Sequence-to-Sequence. Тепер кодер передає декодеру усі приховані стани, навіть проміжні. Декодер також виконує додатковий крок наступним чином:

- Перевіряє отриманий прихований стан.
- Кожний такий стан отримує певну оцінку.

- Кожна оцінка множиться на відповідну їй оцінку softmax. Відповідно вплив важливих станів посилюється, а неважливих — послаблюється.

Приблизно так працює увага (attention).

Далі у 2017 році було представлено нову мережу, архітектура якої отримала назву трансформатора. По-перше, вхідні дані можуть передаватися паралельно, що дозволяє ефективно використовувати потужності GPU, а отже прискорювати навчання. Також використовується Attention шар, який вирішує проблему згасаючого градієнту.

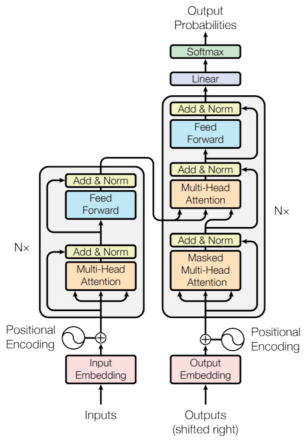


Рис. 1.1: Схема роботи трансформера

Блок кодера:

Перед тим, як дані поступають у блок кодера, вони проходять деяку передобробку. По-перше, всі дані перетворюються у вектори. По-друге, позиційні кодувальники створюють вектори, які допомагають зрозуміти контекст кожного слова (адже одні і ті самі слова можуть мати різне значення у різному контексті).

Далі створюється вектор уваги (attention), який визначає важливість кожного слова, що входить у блок тексту, відносно інших слів. Після цього отримані вектори проходять через обробку для наступних шарів. Саме на цьому етапі кожний вектор уваги проходить обробку паралельно з іншими, що і дозволяє прискорити процес у порівнянні з RNN.

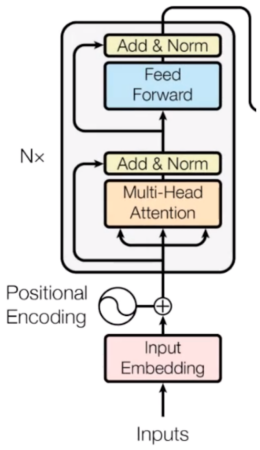


Рис. 1.2: Схема роботи кодера

Блок декодера:

Спочатку із векторами відбувається дещо схоже на те, що відбувалося на етапі кодера. Далі відбувається етап маскування, яких накладає маску (виключає) слово з речення та вимагає від декодера відновити його за контекстом. Так у декілька ітерацій модель і навчається.

Якщо передати вектори уваги в блок прямого зв'язку, він перетворить їх у форму, зручну для декодера або лінійного шару. Потім дані проходять через softmax, який перетворює їх у ймовірності, після чого обирається слово з найбільшою ймовірністю.

1.3. BERT та DistilBERT

BERT (Bidirectional Encoder Representations from Transformers) — це модель глибокого навчання, розроблена Google 2018 року та представлена 2019, яка призначена для обробки природної мови. В основі BERT лежить мережа-трансформер, проте BERT має певні відмінності. Сам він є автокодером і використовує двосторонню увагу (attention), тобто контекст слова в одиниці тексту визначається не тільки у напрямку зліва направо, а і у зворотньому напрямку. Це дозволяє більш точно визначати контекст слів

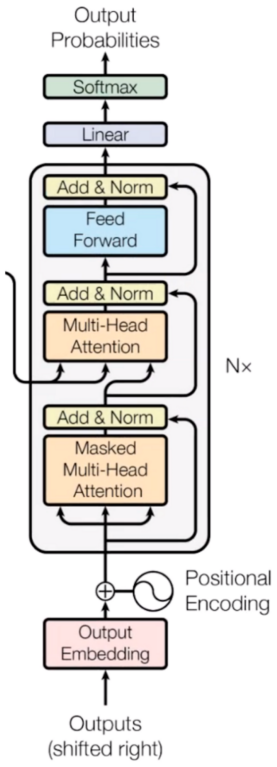


Рис. 1.3: Схема роботи кодера

і, відповідно, навчатися.

Перед подачею тексту до мережі текст проходить етап токенізації — розділення тексту на слова, які є в словнику BERT (близько 30-и тисяч слів), або на структурні елементи (наприклад, якщо певного слова немає в словнику, то воно може бути розділене на певні частини, які окремо в словнику присутні).

BERT навчається на двох задачах: генерації пропущеного замаскованого токену та передбачення наступного речення. Це дає змогу моделі глибоко тренуватися двонаправленому представленню мови, що дозволяє їй аналізувати природню мову, виявляти залежності та контексти на високому рівні. А також модель гарно працює на достатньо великих об'ємах даних завдяки розпаралелізації процесів.

Декілька варіантів навченої мережі доступні на публічних репозиторіях Google Research, а також у відкритому фреймворку TensorFlow. Для осо-

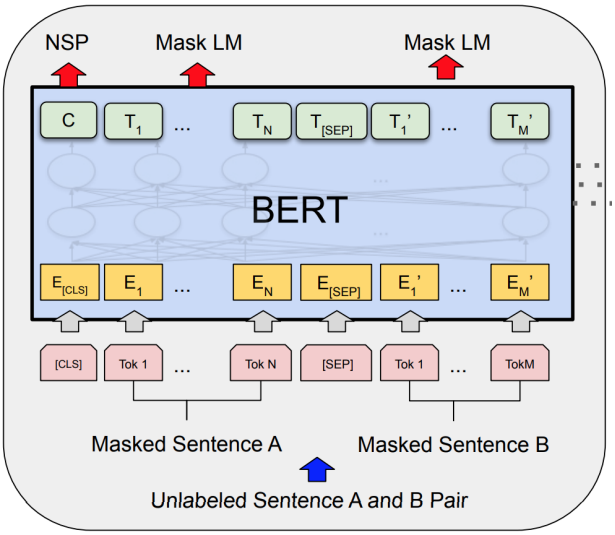


Рис. 1.4: Схема роботи BERT

бливостей розв'язання своєї задачі модель зазвичай дотреновується, як ми і зробили при розв'язанні нашої проблеми.

DistilBERT є зменшеною версією оригінальної моделі BERT. Якщо остання має близько 100-300 мільйонів параметрів, то більше компактна версія DistilBERT має лише 66 мільйонів, проте її обчислювальні потужності можуть зберігатися аж до 97 відсотків. Відповідно, ця модель працює швидше та економніше з точки зору ресурсів, хоча BERT може давати більше точні результати. У нашій же задачі незначні відхилення точності можуть і не вплинути на кінцевий результат. Але точну відповідь на це питання ми і дамо в ході дослідження.

Розділ 2. Постановка задачі класифікації

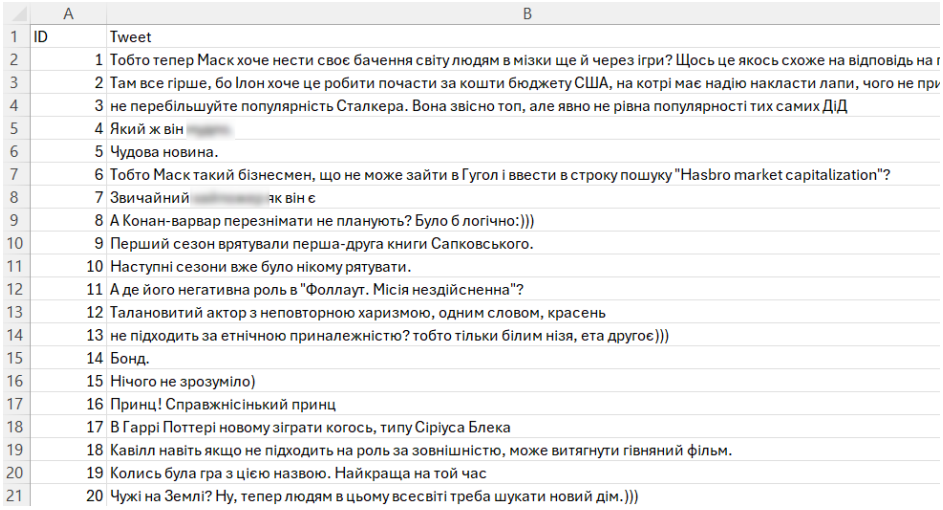
2.1. Постановка задачі. Збір вхідних параметрів для задачі класифікації

Задача для дослідження має декілька етапів. Загалом, перед нами стоїть завдання порівняння двох нейронних мереж у процесі навчання класифікації твітів за ключовими словами.

Спочатку для реалізації цієї задачі ми маємо обрати предмет дослідження твітів. Ним буде група у соціальній мережі Facebook від назвою «Український гік-простір» за посиланням: https://www.facebook.com/groups/uageekspace. Далі з цієї групи необхідно викачати твіти — коментарі під усіма її постами. Дані будуть сформовані у Microsoft Excel Worksheet (.xlsx) документ.

Збір вхідних параметрів

Документ складається з 460 унікальних твітів під постами групи https://www.facebook.com/groups/uageekspace (українська група, що об'єднує любителів книжок, ігор, манги, тощо...). Надалі вони будуть класифікуватися за ключовими словами.



	A	B
1	ID	Tweet
2	1	Тобто тепер Маск хоче нести своє бачення світу людям в мізки ще й через ігри? Щось це якось схоже на відповідь на г
3	2	Там все гірше, бо Ілон хоче це робити почасти за кошти бюджету США, на котрі має надію накласти лапи, чого не при
4	3	не перебільшуйте популярність Сталкера. Вона звісно топ, але явно не рівна популярності тих самих ДіД
5	4	Який ж він
6	5	Чудова новина.
7	6	Тобто Маск такий бізнесмен, що не може зайти в Гугол і ввести в строку пошуку "Hasbro market capitalization"?
8	7	Звичайний ак він є
9	8	А Конан-варвар перезнімати не планують? Було б логічно:)))
10	9	Перший сезон врятували перша-друга книги Сапковського.
11	10	Наступні сезони вже було нікому рятувати.
12	11	А де його негативна роль в "Фоллаут. Місія нездійснення"?
13	12	Талановитий актор з неповторною харизмою, одним словом, красень
14	13	не підходить за етнічною приналежністю? тобто тільки білим нізя, ета другоє)))
15	14	Бонд.
16	15	Нічого не зрозуміло)
17	16	Принц! Справжнісінький принц
18	17	В Гаррі Поттері новому зіграти когось, типу Сіріуса Блека
19	18	Кавілл навіть якщо не підходить на роль за зовнішністю, може витягнути гівняний фільм.
20	19	Колись була гра з цією назвою. Найкраща на той час
21	20	Чужі на Землі? Ну, тепер людям в цьому всесвіті треба шукати новий дім.)))

Рис. 2.1: Вигляд документу, сформованого з твітів

2.2. Передобробка текстів і класифікація

Перед тим, як почати будувати нейронну мережу для класифікації твітів за ключовими словами, ми покладемо інформацію про передобробку тексту, яка може покращити якість навчання моделі. Передобробку тексту буде виконано трьома етапами:

- Очищення тексту: з твітів видаляються HTML-теги, всі символи, окрім літер, та пробіли (зайві пробіли також видаляються).

Лістинг 2.1: Очищення тексту

```
def clean_text(texts):
    cleaned_texts = []
    for text in texts:
        text = re.sub(r'<.*?>', '', text)
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        text = re.sub(r'\s+', '␣', text).strip()
        cleaned_texts.append(text)
    return cleaned_texts
```

- Нормалізація тексту: текст твітів приводиться до нижнього регістру.

Лістинг 2.2: Нормалізація тексту

```
def normalize_text(texts):
    normalized_texts = []
    for text in texts:
        text = text.lower()
        normalized_texts.append(text)
    return normalized_texts
```

- Лематизація тексту: приведення слів до їх початкової форми (побачила -> побачити, краще -> гарно)

Лістинг 2.3: Лематизація тексту

```
nltk.download('punkt')
nltk.download('wordnet')
def lemmatize_text(texts):
    lemmatizer = WordNetLemmatizer()
    lemmatized_texts = []
    for text in texts:
        words = nltk.word_tokenize(text)
        lemmatized_words = [lemmatizer.lemmatize(word)
            for word in words]
        lemmatized_texts.append(' '.join(
            lemmatized_words))
    return lemmatized_texts
```

Також для подальшого тренування моделі, необхідно буде токенізувати фрази, які поступають до моделі.

Для цього ми виведемо графік, який покаже довжини фраз.

Лістинг 2.4: Довжини фраз

```
seq_len = [len(str(i).split()) for i in train_text]
pd.Series(seq_len).hist(bins = 50)
```

Для токенізації було обрано значення в 32 токени.

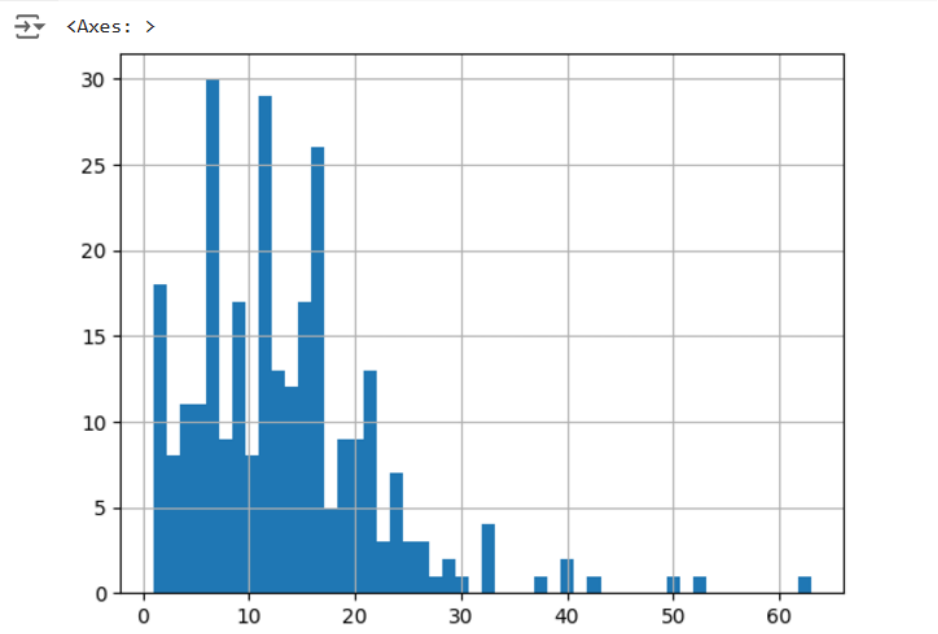


Рис. 2.2: Графік довжин твітів

Класифікація буде проводитися за наступними категоріями:

- Кіно: [’фільм’, ’кіно’, ’актор’, ’режисер’, ’сцена’, ’сюжет’, ’екшн’, ’драма’, ’комедія’, ’трейлер’, ’прем’єра’, ’кінозал’, ’перегляд’, ’кінематограф’, ’сцена’, ’звукові ефекти’, ’кінокритик’, ’серія’, ’серіал’]
- Книги: [’книга’, ’манга’, ’роман’, ’твір’, ’автор’, ’персонажі’, ’сюжет’, ’літературний’, ’бібліотека’, ’читати’, ’сторінки’, ’літературний жанр’, ’глава’, ’опис’, ’мальовка’]
- Інше: все, що не підійшло для класів вище.

Належність до кожного класу визначається за ключовими словами.

Лістинг 2.5: Лейблування твітів

```
labels = []
for tweet in tweets:
    if any(keyword in tweet for keyword in books_keywords):
        labels.append(0)
    elif any(keyword in tweet for keyword in film_keywords):
        labels.append(1)
```

```
    else:
        labels.append(2)
```

У наступному розділі перейдемо до процесу навчання.

Розділ 3. Реалізація задачі класифікації за ключовими словами за допомогою нейронних мереж

3.1. Програмна реалізація поставленної задачі двома мережами

Лістинг 3.1: Програмна реалізація

```
import re
import random
import numpy as np
import pandas as pd
import nltk
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from nltk.stem import WordNetLemmatizer

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

```
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

tqdm.pandas()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=5)

file_path = 'tweets.xlsx'
df = pd.read_excel(file_path)
tweets_exp= df.iloc[:, 2].astype(str).to_numpy()
tweets = np.tile(tweets_exp, 5)

#---
def clean_text(texts):
    cleaned_texts = []
    for text in texts:
        text = re.sub(r'<.*?>', '', text)
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        text = re.sub(r'\s+', '␣', text).strip()
        cleaned_texts.append(text)
    return cleaned_texts
tweets_v1 = clean_text(tweets)



#---
def normalize_text(texts):
    normalized_texts = []
```

```
    for text in texts:
        text = text.lower()
        normalized_texts.append(text)
    return normalized_texts
tweets_v2 = normalize_text(tweets_v1)


#---
nltk.download('punkt_tab')
nltk.download('punkt')
nltk.download('wordnet')
def lemmatize_text(texts):
    lemmatizer = WordNetLemmatizer()
    lemmatized_texts = []
    for text in texts:
        words = nltk.word_tokenize(text)
        lemmatized_words = [lemmatizer.lemmatize(word) for word
            in words]
        lemmatized_texts.append(' '.join(lemmatized_words))
    return lemmatized_texts
tweets_v3 = lemmatize_text(tweets_v2)


tweets = tweets_v3
#---

labels = []
for tweet in tweets:
    if any(keyword in tweet for keyword in books_keywords):
        labels.append(0)
    elif any(keyword in tweet for keyword in film_keywords):
        labels.append(1)
    else:
        labels.append(2)
```

```
train_text, train_temp = train_test_split(tweets, test_size=0.4,
    random_state=42)
val_text, test_text = train_test_split(train_temp, test_size
    =0.5, random_state=42)
train_labels, labels_temp = train_test_split(labels, test_size
    =0.4, random_state=42)
val_labels, test_labels = train_test_split(labels_temp,
    test_size=0.5, random_state=42)

tokens_train = tokenizer.batch_encode_plus(
    train_text,
    max_length=32,
    padding='max_length',
    truncation=True,
    return_tensors='pt'
)
tokens_val = tokenizer.batch_encode_plus(
    val_text,
    max_length=32,
    padding='max_length',
    truncation=True,
    return_tensors='pt'
)
tokens_test = tokenizer.batch_encode_plus(
    test_text,
    max_length=32,
    padding='max_length',
    truncation=True,
    return_tensors='pt'
)

train_seq = tokens_train['input_ids']
train_mask = tokens_train['attention_mask']
```

```
train_y = torch.tensor(train_labels)

val_seq = tokens_val['input_ids']
val_mask = tokens_val['attention_mask']
val_y = torch.tensor(val_labels)

test_seq = tokens_test['input_ids']
test_mask = tokens_test['attention_mask']
test_y = torch.tensor(test_labels)

batch_size = 8

train_data = TensorDataset(train_seq, train_mask, train_y)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler,
    batch_size=batch_size)

val_data = TensorDataset(val_seq, val_mask, val_y)
val_sampler = SequentialSampler(val_data)
val_dataloader = DataLoader(val_data, sampler=val_sampler,
   batch_size=batch_size)

class BERT_Arch(nn.Module):
    def __init__(self):
        super(BERT_Arch, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased
            ')
        self.dropout = nn.Dropout(0.1)
        self.fc1 = nn.Linear(768, 512)
        self.fc2 = nn.Linear(512, 3)

    def forward(self, sent_id, mask):
        outputs = self.bert(sent_id, attention_mask=mask)
        cls_hs = outputs.last_hidden_state[:, 0, :]
```

```
        x = self.fc1(cls_hs)
        x = self.dropout(x)
        x = self.fc2(x)
        return x


model = BERT_Arch().to(device)

optimizer = torch.optim.AdamW(model.parameters(), lr=1e-3)
class_counts = np.bincount(train_labels)
total_count = len(train_labels)
class_weights = [total_count / class_counts[i] if class_counts[i] != 0 else 0 for i in range(len(class_counts))]
class_weights = torch.tensor(class_weights).float().to(device)
cross_entropy = nn.CrossEntropyLoss(weight=class_weights)

def train():
    model.train()
    total_loss = 0
    total_preds = []

    for batch in tqdm(train_dataloader):
        batch = [b.to(device) for b in batch]
        sent_id, mask, labels = batch
        model.zero_grad()
        preds = model(sent_id, mask)
        loss = cross_entropy(preds, labels)
        total_loss += loss.item()
        loss.backward()
        optimizer.step()
        total_preds.append(preds.detach().cpu().numpy())

    avg_loss = total_loss / len(train_dataloader)
    total_preds = np.concatenate(total_preds, axis=0)
```

```
    return avg_loss, total_preds

def evaluate():
    model.eval()
    total_loss = 0
    total_preds = []

    for batch in tqdm(val_dataloader):
        batch = [b.to(device) for b in batch]
        sent_id, mask, labels = batch

        with torch.no_grad():
            preds = model(sent_id, mask)
            loss = cross_entropy(preds, labels)
            total_loss += loss.item()
            total_preds.append(preds.detach().cpu().numpy())

    avg_loss = total_loss / len(val_dataloader)
    total_preds = np.concatenate(total_preds, axis=0)
    return avg_loss, total_preds



epochs = 3
best_valid_loss = float('inf')

for epoch in range(epochs):
    print(f'\nEpoch {epoch + 1} / {epochs}')
    train_loss, _ = train()
    valid_loss, _ = evaluate()
    print(f'Training loss: {train_loss:.3f}, Validation loss: {
        valid_loss:.3f}')

model.eval()
with torch.no_grad():
```

```
    preds = model(test_seq.to(device), test_mask.to(device))
    test_preds = preds.argmax(dim=1).cpu().numpy()

test_df = pd.DataFrame({'tweet': test_text, 'target':
    test_labels, 'pred': test_preds})
print(classification_report(test_df['target'], test_df['pred']))
```

Для відпрацювання мережі DistilBERT ми лишаємо увесь код таким самим, тільки замість BertTokenizer та BertForSequenceClassification використовуємо DistilBertTokenizer та DistilBertForSequenceClassification відповідно.

3.2. Аналіз результатів класифікації, висновки для групи, порівняння роботи двох мереж

У результаті роботи двох мереж були отримані такі результати:

BERT:

	precision	recall	f1-score	support
0	0.95	0.92	0.94	132
1	0.93	0.95	0.94	74
2	0.92	0.90	0.91	254
accuracy			0.93	460
macro avg	0.93	0.92	0.93	460
weighted avg	0.93	0.93	0.93	460

DistilBERT:

	precision	recall	f1-score	support
0	0.92	0.94	0.93	132
1	0.91	0.89	0.90	74
2	0.94	0.93	0.93	254
accuracy			0.91	460
macro avg	0.92	0.92	0.92	460
weighted avg	0.93	0.93	0.93	460

Таким чином, за результатами відпрацювання двох мереж можемо сказати, що більш точне навчання має мережа, побудована на основі мережі BERT, бо вона є більше розширеною і точною версією. Проте з боку ресурсів, то DistilBERT є моделлю, що рахує в понад 2 рази швидше, тому якщо результати дослідження не мають бути надзвичайно точними, то у цілях економії часу та обчислювальних потужностей можна використовувати і більш компактну мережу DistilBERT, адже її показник точності є також задовільним.

Висновки

Кваліфікаційна робота магістра була присвячена розв'язанню задачі порівняння якості навчання двох нейронних мереж класифікації твітів користувачів у групі в соціальній мережі Facebook.

Розв'язання задачі відбувалося у декілька етапів:

- Вивчення базової теорії нейронних мереж, у тому числі Transformers, BERT та DistilBERT.
- Пошук групи в мережі Facebook, завантаження з цієї групи твітів у документ табличного формату.
- Реалізація двох нейронних мереж на основі BERT та DistilBERT.
- Навчання цих двох мереж на основі зібраних даних.
- Порівняння параметрів, що описують точність навчання, отриманих в результаті навчання.

Дослідження відбувалося за допомогою взаємодії з ресурсами Facebook (https://www.facebook.com/groups/uageekspace), Microsoft Excel та Google Colab, де відбувалася основна реалізація мовою Python.

У результаті мною були отримані такі результати:

- BERT: [0.93, 0.93, 0.93]
- DistilBERT: [0.91, 0.92, 0.93]

З отриманих вище результатів було зроблено висновок, що більш точною є модель на базі BERT, проте показники DistilBERT також є гарними, тому в залежності від задачі, якщо не вимагається мати максимальну точність, можна використати другий варіант мережі для економії часу обчислення в понад 2 рази.

Отже, в ході дослідження було повністю реалізовано поставлену задачу порівняння якості навчання двох нейронних мереж на процесі класифікації твітів користувачів у групах соціальних мереж.

Список використаних джерел

[1] https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/, 2024.

[2] https://builtin.com/artificial-intelligence/transformer-neural-network, 2024.

[3] Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019. P. 2-5.

[4] Lingqing Song, Ensuring Brand Safety by Using Contextual Text Features: A Study of Text Classification with BERT. 2022. P. 10-13.

[5] Vrishali Chakkarwar, Sharvari Tamane, Ankita Thombre, A Review on BERT and Its Implementation in Various NLP Tasks. 2023.

[6] https://medium.com/analytics-vidhya/paper-summary-bert-pre-training-of-deep-bidirectional-transformers-for-language-understanding-861456fed1f9, 2021.