УДК 004.912
DOI: 10.18523/2617-3808.2022.5.16-25

Глибовець А. М., Бікчентаєв М. О.

ПРОГРАМНА СИСТЕМА ПЕРЕВІРКИ НА ПЛАГІАТ УКРАЇНСЬКИХ ТЕКСТІВ

Метою цієї роботи є опис методології побудови програмної системи (застосунку) перевірки на плагіат наукових публікацій українською мовою з використанням двох моделей машинного навчання – Word2Vec і BERT. Ми розглядаємо виявлення зовнішнього плагіату в українських текстах, що передбачає порівняння вхідного документа з документами в колекції. Вбудовування слів використовували для порівняння документів, оскільки тексти зі схожим значенням або контекстом створюють подібні вбудовування слів. За допомогою Word2Vec і BERT ми перетворюємо кожен документ на ряд убудовувань слів. Розрахунок міри подібності для цих убудовувань допомагає визначити схожість документів. Інтерфейс програми розроблено з використанням бібліотеки React. Вебзастосунок використовує бібліотеку компонентів Material UI і базу даних MongoDB. Бекенд написано з використанням мов програмування Python і Flask.

Ключові слова: пошук плагіату, BERT, Word2Vec, машинне навчання.

Вступ

Нині з розвитком світової павутини люди мають доступ до величезного обсягу інформації, яку можна використовувати по-різному. Зокрема спрощується можливість вдаватися до плагіату – використання чужих ідей як власних. У багатьох сферах життя трапляються випадки плагіату, зокрема в наукових дослідженнях та освіті [8; 14]. Плагіат також може мати різні форми: від прямого копіювання-вставлення до перефразування та перебудови речень. Виявлення плагіату є досить складною проблемою. Багато методів, які побудовані на знаходженні найдовшої загальної підсеквенції або n-грами та спільних слів між документами [8; 10], можуть у багатьох випадках не працювати. Тому останнім часом при побудові програмних систем виявлення плагіату дедалі більше застосовують машинне навчання і семантичний аналіз [10].

Метою цієї роботи є опис методології побудови програмної системи (застосунку) перевірки на плагіат наукових публікацій українською мовою з використанням двох моделей машинного навчання – Word2Vec [9; 3; 4; 22] і BERT [12; 11].

Загальні підходи до визначення плагіату

Згідно з Кембриджським словником, *плагіат* – це процес або практика використання ідей або роботи іншої людини під своїм ім'ям.

Дослідники поділяють плагіат на кілька видів [14]:

a) *прямий* або *дослівний плагіат* (автор копіює текст слово в слово з джерела і вставляє його у свій твір);
b) *джерело плагіату* (автор посилається на ресурси, яких не існує, або не посилається на всі використовувані ресурси);
c) *перефразування плагіату* (автор змінює структуру речень оригінального тексту шляхом переставляння або видалення слів, заміни їх синонімами тощо);
d) *мозаїчний*, або *клаптиковий плагіат* (більш досконала варіація попереднього типу, автор переплітає оригінальний текст (ймовірно, перефразований) із безліччю різних джерел, зокрема його ідеї і перспективу);
e) *плагіат ідей* (цей різновид виявити найважче, автор використовує (не визнаючи) ідеї або висновки з інших творів як фундамент для своєї роботи).

Усі види плагіату зазвичай зводяться до зміни словникового запасу тексту або його синтаксичного чи семантичного змісту.

Зміни словникового запасу передбачають додавання, видалення або заміну слів. Ми можемо виявити зміни цього типу за допомогою таких методів, як найдовша загальна підсеквенція або n-грами. Чим більше спільних термінів мають документи, тим більше вони схожі [10].

© *Глибовець А. М., Бікчентаєв М. О.,* 2022

Синтетичні зміни – це зміни в структурі речення, наприклад переставлення слів і словосполучень або зміни граматики в реченні. Щоб виявити такі зміни, ми використовуємо синтаксичні одиниці тексту, наприклад POS-теги для пошуку подібних документів [10].

Семантичні зміни охоплюють зміни двох попередніх типів, а також перефразування тексту. Щоб виявити ці зміни, ми проводимо смисловий аналіз текстів. Наприклад, деякі методи використовують синоніми, антоніми, гіперніми та гіпоніми для виявлення змін і порівняння текстів [10].

Процес виявлення плагіату можна поділити на дві категорії – *зовнішнє* та *внутрішнє* виявлення [10].

Зовнішнє виявлення плагіату порівнює вхідний документ з іншими доступними документами. *Внутрішній плагіат* знаходить частини вхідного документа, які написав інший автор.

Виявлення плагіату є частиною оброблення природної мови (Natural Language Processing, NLP). Нині існує безліч рішень для виявлення лексичного або синтаксичного плагіату, основаних на методах NLP, зокрема концепції використання корпусів мов типу WordNet [10].

Останніми роками значний розвиток мали підходи з використанням глибинного машинного навчання. У них основну увагу приділяють виявленню високорівневих (абстрактних) особливостей даних. Крім того, мережі глибинного навчання не потребують складних інженерних або маркованих даних. Найчастіше тут розглядають мережу глибинного навчання *BERT* і двошарову нейронну мережу *Word2Vec*. Коротко охарактеризуємо їх.

Word2Vec приймає за вхід великий корпус слів і виробляє векторний простір у кілька сотень вимірів. Кожен вектор представляє окреме слово з корпусу. Слова, які мають схоже значення (або контекст), мають вектори, близькі один до одного. Вектори, створені Word2Vec, називають *убудовуванням слів* [9]. Реалізувати Word2Vec можливо двома способами: через *неперервний мішок слів* (*Continuous Bag of Words*, *CBOW*) або *Skip-gram*.

CBOW намагається передбачити цільове слово (наприклад, «ганок») із навколишнього контексту або, простіше кажучи, з навколишніх слів («собака сидить на ...»). *Skip-gram* намагається передбачити навколишній контекст із цільового слова [9].

Після того, як згенерували набір даних, переходять до навчання моделі. Цей процес дуже схожий для CBOW і Skip-gram.

Вхідний вектор – це векторне представлення одного слова. Його довжина дорівнює розміру словникового запасу. Він має нулі скрізь, крім індексу, який відповідає вхідному слову [9].

Прихований шар є стандартним повністю з'єднаним шаром, вага якого є словом embeddings. Вихідний шар дає нам імовірність того, що інші слова будуть сусідами вхідного слова.

Щоб отримати вбудовування слів, ми видаляємо вихідний шар після навчання моделі. Отже, ми отримаємо модель, яка вироблятиме вбудовування слів замість імовірностей появи слів [9].

Тепер розглянемо, як ми тренуємо модель передбачати сусідні слова. Передусім ми ініціюємо модельні ваги з випадковими числами. Потім ми вибираємо першу функцію з першої вибірки набору даних і віддаємо її моделі.

Останній крок навчання Word2Vec може бути обчислювально значним, особливо якщо розмір словникового запасу великий. Зокрема, для підвищення ефективності моделі ми можемо перемкнути завдання з прогнозування ймовірності сусідніх слів на прогнозування ймовірності того, що два слова будуть сусідами, де 1 зазначить, що слова є сусідами, а 0 – ні [3]. Це перетворить нашу модель із нейронної мережі на логістичну регресію, що дає змогу виконувати обчислення з набагато більшою швидкістю [3].

Щоб виконати цей перемикач, ми також повинні змінити структуру набору даних. Не буде вхідного і цільового слова, а буде слово введення, вихідне слово і ціль, яка міститиме значення 0 або 1 [3].

Також залишається ще одна проблема. Оскільки ціль завжди 1, ми можемо отримати модель, яка завжди виводитиме 1, досягаючи 100 % точності, але нічого не вивчаючи на цьому шляху. Для боротьби з цим використовують *негативні вибірки* в наборі даних – пари слів, які не є сусідами (тому метою для них є 0) [3].

Щоб заповнити вихідні слова, яких бракує, випадковим способом відбираємо слова зі словникового запасу. Імовірність виділення слова як негативної вибірки залежить від частоти (кількості слів) цього слова. Конкретно кожному слову надається вага, що дорівнює його частоті, піднесеній до степеня. Ймовірність для вибору цього слова полягатиме в тому, що вагу буде поділено на суму ваг усіх інших слів [9].

Ідею додавання негативної вибірки навіяно концепцією шумо-контрастної оцінки. Ми протиставляємо фактичні сигнали (пари слів, які є сусідами) шуму (негативній вибірці) [3].

На початку навчання відбувається ініціалізація двох матриць – *матриці вбудовування* і *контекстної матриці*, які зберігають убудовування наших слів зі словникового запасу і мають рівні розміри [5]. Кількість рядків відповідає кількості слів у лексиці, а кількість стовпців – потрібній довжині векторів вбудовування слів (300 сотень – загальне значення). Цей підхід був використаний Google у моделі Word2Vec, яку було навчено на наборі даних Google Новин [22].

Матриці ініціалізують випадковими значеннями і вибирають одну позитивну вибірку та пов'язані з нею негативні вибірки з набору даних. Потім кодують ці слова і множать вектор вхідного слова на матрицю вбудовування і вектори вихідних слів на контекстну матрицю. Потім беруть точковий добуток вхідного вбудовування з кожним із вихідних убудовувань. Отримане число вказуватиме на схожість вхідних і вихідних убудовувань [3].

Однак іноді Word2Vec може бути недостатньо точним [12], оскільки не враховує порядок слів, у якому вони з'являються. Це може привести до втрати деякої синтаксичної і семантичної інформації речення.

Наприклад [12], речення «Ти йдеш туди, щоб навчити не грати.» і «Ти йдеш туди грати, а не вчити.» матимуть аналогічне подання у векторному просторі, однак сенс їх інший. Також залежно від контексту слова можуть мати різне значення.

Отже, нам потрібна модель, яка збереже контекстну інформацію, що стосується слів у реченні. Однією з таких моделей є *BERT*.

Спершу ознайомимося з трансформерами, які широко застосовують у BERT. *Трансформер* – це компонент, який використовують у нейронних мережах для оброблення послідовних даних, як-от текст або дані часових рядів. Досить часто трансформери застосовують у сфері NLP [21]. Трансформер приймає вхідний текст у вигляді послідовності векторів і перетворює його на вектор під назвою *кодування*, а потім декодує його назад в іншу послідовність [21].

Трансформери також використовують *механізм уваги*, що допомагає моделі «запам'ятати» відносини між вхідними токенами. Це можна застосовувати, наприклад, у машинному перекладі речень. Механізм уваги дасть моделі змогу перекладати слова типу «it» словами правильного роду іспанською або французькою мовами, звертаючи увагу на всі сусідні слова в оригінальному реченні [21].

Кожен кодер складається з двох шарів: *самоуваги* і *подавання вперед*. Шар самоуваги – це місце, де механізм уваги використовують для створення кращих кодувань слів. Потім вихід шару самоуваги подається в нейронну мережу feed-forward [2], а вихід мережі feed-forward – на наступний кодер.

Декодер має два однакові шари й шар *уваги кодера-декодера* між ними [2]. Шари уваги кодерів і декодерів діють аналогічно, проте вони мають кілька невеликих, але істотних відмінностей [5]. Кожен шар кодера (декодера) має залишковий зв'язок навколо себе і за ним іде крок нормалізації шару.

Перш ніж ми передамо будь-який текст кодерам, потрібно його попередньо обробити. Ця процедура складається з трьох етапів [6]:

a) згенерувати вбудовування слів для кожного слова у вхідному тексті;
b) обчислити *кодування позицій* для кожного слова у вхідному тексті;
c) об'єднати ці кодування, підсумувавши їх.

Кодери-трансформери обробляють слова з вхідного тексту паралельно, при цьому кожне слово йде його окремим «шляхом». Тож інформація про розташування слів зазвичай втрачається. *Кодування позицій* використовують для запам'ятовування розташування слів у вхідному реченні.

Шар самоуваги працює з матрицями, де кількість стовпців відповідає довжині вбудовування слова, а кількість рядків – це гіперпараметр, який ми можемо встановити (зазвичай вона дорівнює довжині найдовшого речення).

Після того, як послідовність введення пройшла весь шлях через стек кодувальників, вихід верхнього кодера передається всім декодерам, які використовуватимуться в шарі уваги кодера-декодера. Цей шар дуже схожий на шар уваги кодера, але замість матриць використовується вихід стека кодера. Вихід стека декодера передається на *лінійний шар*, за яким іде *шар softmax*.

Лінійний шар проєктує вектор, створений стеком декодера, в набагато більший вектор, який називають *вектором логітів*. Вектор logits міститиме оцінки за кожне унікальне слово в словниковому запасі (яке походить від навчального набору даних).

Шар softmax перетворить ці оцінки на ймовірності (які є позитивними і складаються до одиниці). Слово, яке відповідає найбільшій ймовірності, потім буде виведено моделлю.

Перший спосіб використання BERT призначений для класифікації речень [1]. Навчання такого класифікатора майже не потребують змін у BERT. Нам потрібно тренувати тільки класифікатор.

Деякі випадки використання BERT передбачають *аналіз настроїв* (наприклад, дали відгук, скажіть, позитивний він чи негативний) та *перевірку фактів* (наприклад, дайте речення, скажіть, чи є це претензією, чи ні). BERT було побудовано на кількох ідеях, які охоплюють *трансформери*, *ELMo* та *OpenAI Transformer*.

ELMo – це модель, яку використовують для створення вбудовування слова з урахуванням контексту. Моделі з подібною метою, такі як Word2Vec або GloVe, генерують убудовування незалежно від контексту слова [1]. ELMo вводить поняття *контексно аналізованих вкладень слів*. Тому, перш ніж надати слову вбудовування, вона дивиться на все речення, де розташоване це слово.

Для цього ELMo використовує двонаправлений LSTM, навчений прогнозувати наступне слово в реченні (завдання, яке називають *моделюванням мови*). Двонаправлений LSTM складається з моделі мови вперед (містить інформацію про поточне слово та інші слова *до* нього) і моделі зворотної мови (яка містить інформацію про поточне слово та інші слова *після* нього) [1].

Щоб створити контекстуалізоване вбудовування слів для одного слова, вектори, створені кожним шаром LSTM, об'єднуються, множаться на вагу та підсумовуються разом.

Пізніше з'явилася краща альтернатива LSTM – трансформери. Як ми бачили в попередніх розділах, трансформери, з їхньою структурою кодер-декодер, є ідеальними для машинного перекладу. Наприклад, OpenAI Transformer намагається використовувати концепції трансформерів для створення чутливої мовної моделі для завдань NLP.

Навчивши модель, ми можемо використовувати її для інших завдань.

BERT спирається на ці концепції і пропонує модель на основі трансформерів, яка використовує як наступні, так і попередні слова для створення вбудовувань.

Використання Word2Vec і BERT для виявлення плагіату

У цій статті ми розглядаємо виявлення зовнішнього плагіату в українських текстах, що передбачає порівняння вхідного документа з документами в колекції. Ми використовуватимемо вбудовування слів для порівняння документів, оскільки тексти зі схожим значенням або контекстом створюють подібні вбудовування слів.

Використовуючи Word2Vec і BERT, ми перетворюємо кожен документ на ряд убудовувань слів. Розрахунок міри подібності для цих убудовувань допоможе нам визначити, схожі документи чи ні.

Одними з найпоширеніших способів обчислення векторної подібності є *евклідова відстань* і *косинусна подібність* [9].

Word2Vec і BERT можна навчити створювати вбудовування для текстів довільною мовою. Однак для навчання цих моделей потрібен досить великий корпус текстів, написаних цією мовою, і значна кількість обчислювальних ресурсів. Тому в нашому дослідженні ми використали попередньо навчені моделі Word2Vec і BERT.

Word2Vec для української мови забезпечується lang-uk, відкритою спільнотою фахівців у галузі комп'ютерного оброблення текстів [13], і доступний у трьох варіантах, кожен з яких тренується на окремому текстовому корпусі: художній літературі, новинах або «уберкорпусі» (Ubercorpus – це великий корпус текстів з українських періодичних видань обсягом 6 Гб). Розміри і кількість токенів у кожному корпусі наведені в табл. 1 [13].

Таблиця 1

Кількісна характеристика Ubercorpus

Ім'я	Кількість токенів	Розмір (стислий)
Фантастика	18 323 509	41 Мб
Новини	461 451 019	1,1 Гб
Уберкорпус	665 419 885	1,6 Гб

Для кожного текстового корпусу доступно кілька варіантів попереднього оброблення: токенізований; токенізовані та з малої літери; токенізований і лематизований; токенізовані, з малої літери й лематизовані.

Кожна модель створює 300-вимірний вектор убудовування. Ми використовували токенізовану, з малої літери та лематизовану версію Word2Vec, навчену на Ubercorpus.

BERT забезпечується фреймворком Sentence-Transformers, який містить різні моделі BERT для створення вбудовувань речень, тексту й зображень [17]. У цій роботі ми використовували «paraphrase-multilingual-mpnet-base-v2», який є багатомовною моделлю, навченою для створення вбудовування речень.

Застосовуючи раніше згадані реалізації моделей BERT і Word2Vec, ми розробили програму для виявлення плагіату в українських текстах. Інтерфейс завершеного вебзастосунку наведено на рис. 1.

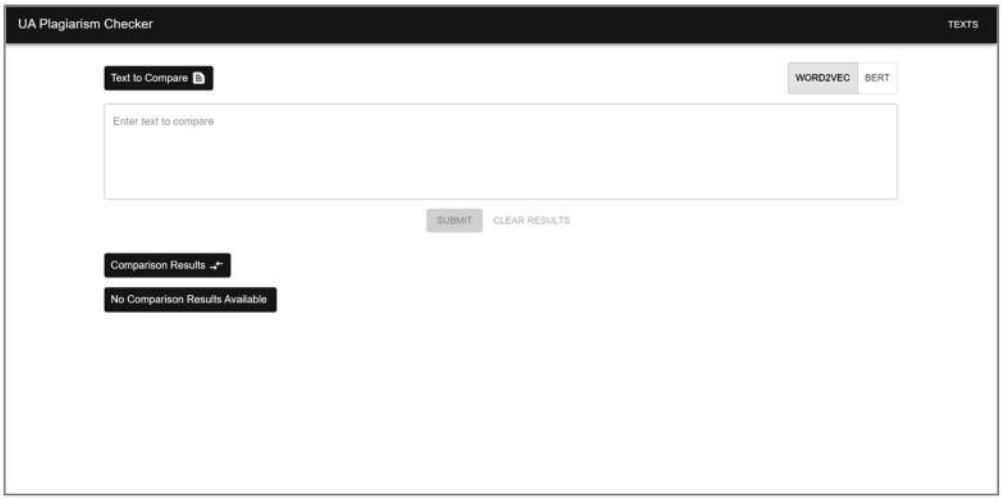

Рис. 1. Перевірка плагіату

Інтерфейс програми розроблено з використанням *бібліотеки* React, а JavaScript ми застосували для побудови користувальницьких інтерфейсів [16].

Вебзастосунок використовує бібліотеку компонентів *Material UI* й пропонує безліч готових компонентів для відображення входів, кнопок, текстових абзаців та інших обов'язкових елементів на вебсторінці [15].

На головній сторінці програми користувач може ввести текст, який буде порівнюватися з текстами з бази даних програми щодо схожості. Порівняння здійснюватиметься шляхом генерації вбудовування слів із текстів. Подібність вкладень обчислюється за допомогою або косинусної подібності, або евклідової відстані.

Користувачі можуть вибрати модель, яка використовуватиметься для створення вбудовування слів, за допомогою кнопок у верхньому правому куті вікна введення тексту.

Приклад порівняння тексту наведено на рис. 2.

Метрику порівняння можна змінити за допомогою двох кнопок поруч із міткою «Результати». Тексти бази даних програми можна змінити, натиснувши на кнопку «Тексти» у верхньому правому куті сторінки: відкриється нова сторінка з можливістю перегляду наявних текстів, видалення наявного тексту або додавання нового тексту в базу даних (рис. 3).

Розрахунок вставок слів і їх порівняння виконується на бекенді програми, написаному з використанням мови програмування Python і *Flask*, полегшеного фреймворку для розроблення API і вебдодатків в цілому [7].

Функція compare_texts відповідає за створення та порівняння вбудовування слів (рис. 4).

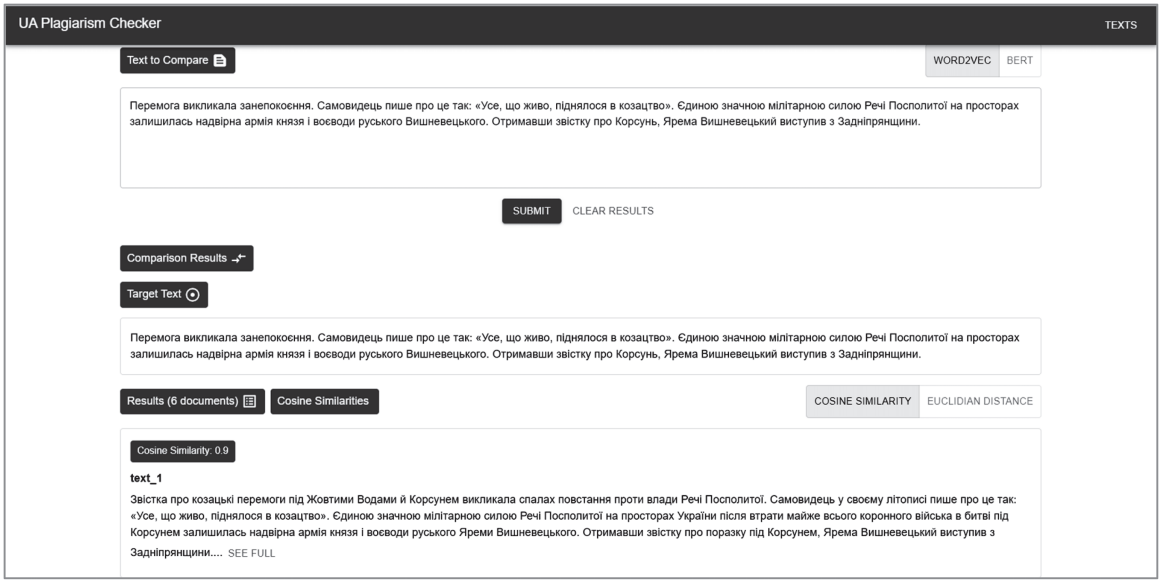

Рис. 2. Результати порівняння тексту

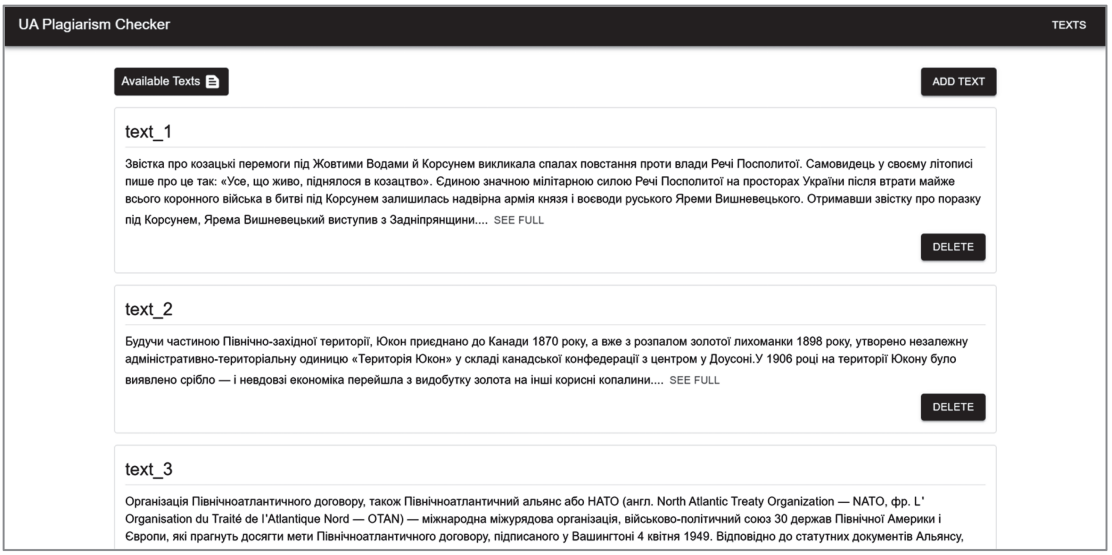

Рис 3. Список текстів

```
@app.route('/compare/', methods=["POST"])
def compare_texts():
    # Extract data from the request
    raw_data = request.get_json()
    query_text = str(raw_data['text'])
    model = str(raw_data['model'])

    # Extract all documents from the DB
    cursor = ua_texts_collection.find({}).sort("title")
    documents = [UaText(**doc) for doc in cursor]

    # List of texts
    texts = [query_text] + [doc.text for doc in documents]
    # List of slugs
    slugs = [query_text] + [doc.slug for doc in documents]

    document_embeddings = []

    if model == 'word2vec':
        processed_texts = process_texts(texts)

        tfidfvectoriser = get_tf_idf_model(processed_texts, my_tokenizer)
        tokenizer = get_tokenizer(processed_texts)

        document_embeddings = get_document_embeddings(
            tfidfvectoriser, tokenizer, w2v_model, processed_texts)
    elif model == 'bert':
        unnorm_processed_texts = process_texts(texts, False)

        document_embeddings = sbert_model.encode(unnorm_processed_texts)
    else:
        return custom_error({'message': 'Model not found'}, 404)

    scores = get_model_scores(document_embeddings, slugs)

    scores['cosine_similarities']['similar'] = list(
        map(lambda item: list(map(str, item)), scores['cosine_similarities']['similar']))
    scores['euclidian_distances']['similar'] = list(
        map(lambda item: list(map(str, item)), scores['euclidian_distances']['similar']))

    return jsonable_encoder(scores, exclude_none=True)
```

Рис. 4. Функція compare_texts

Ця функція отримує запит із текстом на порівняння з іншими текстами та бажану модель для розрахунку вбудовувань. Після вилучення цих даних із запиту ми витягуємо всі документи з бази даних *MongoDB* і створюємо список текстів і список *slugs*. У контексті цієї програми slug – це хеш, сформований шляхом застосування алгоритму SHA256 до текстового вмісту, об'єднаного з випадковим рядком, створеним *uuid* – бібліотекою Python для створення унікальних ідентифікаторів. Службові елементи безпечні за URL-адресами і використовуються для ідентифікації документів при виконанні запитів.

```
def get_document_embeddings(tfidfvectoriser, tokenizer, w2v_model, texts):
    tfidf_vectors = tfidfvectoriser.transform(texts)
    tfidf_vectors = tfidf_vectors.toarray()

    tokenized_documents = tokenizer.texts_to_sequences(texts)

    tokenized_paded_documents = tf.keras.utils.pad_sequences(
        tokenized_documents, padding='post')

    vocab_size = len(tokenizer.word_index) + 1
    embedding_matrix = np.zeros((vocab_size, 300))

    for word, i in tokenizer.word_index.items():
        if word in w2v_model:
            embedding_matrix[i] = w2v_model[word]

    document_word_embeddings = np.zeros(
        (tokenized_paded_documents.shape[0], tokenized_paded_documents.shape[1], 300))

    for i in range(tokenized_paded_documents.shape[0]):
        for j in range(tokenized_paded_documents.shape[1]):
            document_word_embeddings[i][j] = embedding_matrix[tokenized_paded_documents[i][j]]

    document_embeddings = np.zeros((tokenized_paded_documents.shape[0], 300))
    words = tfidfvectoriser.get_feature_names_out()

    for i in range(len(document_word_embeddings)):
        for j in range(len(words)):
            document_embeddings[i] += embedding_matrix[tokenizer.word_index[words[j]]
                                                       ] * tfidf_vectors[i][j]

    return document_embeddings
```

Рис. 5. Функція get_document_embeddings

Після ініціалізації списків тексти проходять попереднє оброблення. Для цього використовується функція process_texts, що передбачає поділ тексту на речення й речення на лексеми (слова), зниження маркерів, видалення розділових знаків, будь-яких спеціальних символів та стоп-слів – загальних слів, вилучення яких зазвичай не впливає на зміст тексту. У випадку з Word2Vec токени також зазнають *лематизації*, щоб звести флективну форму слова назад до свого кореня за допомогою морфологічного аналізу [20].

Після етапу попереднього оброблення генеруємо вбудовування тексту. Цей процес варіюється залежно від обраної користувачем моделі.

Оскільки Word2Vec може генерувати вбудовування лише для окремих слів, нам потрібен спосіб обчислити вбудовування для всього документа з вбудовування його слів. Одним із способів зробити може бути множення вбудовування кожного слова на його *вагу tf-idf* і потім підсумовування отриманих векторів [18].

tf-idf (частотно-обернена частота документа) присвоює вагу кожному слову в документі. Ця вага залежить від кількості входжень слова в поточному документі (скорочено – *tf*) і кількості документів, які містять дане слово (скорочено – *idf*) == [19]:

$$idf_t = \log\left(\frac{N}{df_t}\right),$$

де *N* у чисельнику – кількість документів у збірнику, а знаменник визначає кількість документів, які містять термін *t*.

Маючи *tf* і *idf*, ми можемо обчислити *tf-idf* терміна *t* із документа *d*:

$$tf - idf_{t,d} = tf_{t,d} \times idf_t.$$

Функція get_document_embeddings обчислює вбудовування документа для Word2Vec за допомогою описаного вище процесу (рис. 5).

Функція get_document_embeddings також використовує *токенізатор* для перетворення документа на послідовність цифр, надаючи кожному слову в документі унікальний ідентифікатор за допомогою функції texts_to_sequences.

На відміну від Word2Vec, BERT може генерувати вбудовування для всього документа, тому нам потрібно лише викликати функцію кодування.

Після того, як ми розрахували вбудовування, використовується get_model_score функція для порівняння вбудовувань із використанням косинусної подібності та евклідової відстані (рис. 6).

Функція get_model_score виконує попарне порівняння векторів вбудовування за допомогою cosine_similarity і euclidean_distance функцій, що надаються бібліотекою scikit-learn.

Результати порівняння використовують для сортування списку службових елементів документа щодо конкретного документа за допомогою функції most_similar. Відсортовані результати відправляються на фронтенд і відображаються для користувача.

```
def most_similar(text_id, slugs, similarity_matrix, matrix):
    similar = []
    target = slugs[text_id]

    if matrix == 'cosine':
        similar_ix = np.argsort(similarity_matrix[text_id])[::-1]
    elif matrix == 'euclidean':
        similar_ix = np.argsort(similarity_matrix[text_id])

    for ix in similar_ix:
        if ix == text_id:
            continue
        similar.append([slugs[ix], similarity_matrix[text_id][ix]])

    return {"target": str(target), "similar": list(similar)}


def compare_documents(document_embeddings):
    pairwise_similarities = cosine_similarity(document_embeddings)
    pairwise_differences = euclidean_distances(document_embeddings)
    return (pairwise_similarities, pairwise_differences)


def get_model_scores(document_embeddings, slugs):
    (pairwise_similarities, pairwise_differences) = compare_documents(
        document_embeddings)

    cosine_similarities = most_similar(
        0, slugs, pairwise_similarities, 'cosine')
    euclidian_distances = most_similar(
        0, slugs, pairwise_differences, 'euclidean')

    return {"cosine_similarities": cosine_similarities, "euclidian_distances": euclidian_distances}
```

Рис. 6. Функція get_model_score

Висновок

У цій статті ми описали методологію побудови програмної системи (застосунку) перевірки на плагіат наукових публікацій українською мовою з використанням двох моделей машинного навчання – Word2Vec і BERT. Ми також розглянули виявлення зовнішнього плагіату в українських текстах, що передбачає порівняння вхідного документа з документами в колекції. Вбудовування слів використовувалось для порівняння документів, оскільки тексти зі схожим значенням або контекстом створюють подібні вбудовування слів.

Застосовуючи Word2Vec і BERT, ми перетворюємо кожен документ на ряд убудовувань слів. Розрахунок міри подібності для цих убудовувань допомагає визначити, схожі документи чи ні.

Інтерфейс програми розроблено з використанням *бібліотеки* React, а JavaScript застосовано для побудови користувальницьких інтерфейсів. Вебзастосунок використовує бібліотеку компонентів *Material UI* і базу даних *MongoDB*. Бекенд написано з використанням мови програмування Python і *Flask*.

Ця програма дає користувачеві змогу вводити текст, який буде порівнюватися з текстами з бази даних програми, використовуючи косинусну подібність або евклідову відстань як метрику. Порівняння виконується за допомогою вбудованих слів, розрахованих за попередньо навченою моделлю BERT або Word2Vec. Користувач може вибрати модель і показник схожості за допомогою інтерфейсу користувача програми.

Додаток можна вдосконалити, щоб не лише виводити показник подібності, а й виділяти подібні речення в текстах. Також можна додати можливість зберігати результати порівняння.

Розроблений застосунок можна використовувати в системах перевірки плагіату для курсових, кваліфікаційних і дипломних робіт закладів вищої освіти України.

Список літератури

1. Alammar J. The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) [Electronic resource] / Jay Alammar. – Mode of access: https://jalammar.github.io/illustrated-bert/.
2. Alammar J. The Illustrated Transformer [Electronic resource] / Jay Alammar. – Mode of access: https://jalammar.github.io/illustrated-transformer/.
3. Alammar J. The Illustrated Word2vec [Electronic resource] / Jay Alammar. – 2019. – Mode of access: https://jalammar.github.io/illustrated-word2vec/.
4. Ali Z. A simple Word2vec tutorial [Electronic resource] / Zafar Ali. – Mode of access: https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1.
5. Doshi K. Transformers Explained Visually – Not Just How, but Why They Work So Well [Electronic resource] / Ketan Doshi. – 2021. – Mode of access: https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3.
6. Doshi K. Transformers Explained Visually (Part 2): How it works, step-by-step [Electronic resource] / Ketan Doshi. – 2021. – Mode of access: https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34.
7. Flask [Electronic resource]. – Mode of access: https://palletsprojects.com/p/flask/.
8. Gharavi E. A Deep Learning Approach to Persian Plagiarism Detection / E. Gharavi, K. Bijari, K. Zahirnia // FIRE (Working Notes). –2016. – Pp. 154–159.
9. Gilyadov J. Word2Vec Explained [Electronic resource] / Julian Gilyadov. – 2017. – Mode of access: https://israelg99.github.io/2017-03-23-Word2Vec-Explained/.

10. Hambi El Mostafa. A deep learning-based technique for plagiarism detection: a comparative study / Hambi El Mostafa, Faouzia Benabbou // International Journal of Artificial Intelligence. – 2020. – Vol. 9, No. 1. – Pp. 81–90.
11. Horev R. BERT Explained: State of the art language model for NLP [Electronic resource] / Rani Horev. – 2018. – Mode of access: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.
12. Kathrani K. All about Embeddings [Electronic resource] / Kashyap Kathrani. – 2022. – Mode of access: https://medium.com/@kashyapkathrani/all-about-embeddings-829c8ff0bf5b.
13. lang-uk: Models [Electronic resource]. – Mode of access: https://lang.org.ua/uk/models/.
14. Malnik J. 7 common types of plagiarism explained [Electronic resource] / Jessica Malnik. – 2022. – Mode of access: https://writer.com/blog/types-of-plagiarism/.
15. MUI [Electronic resource]. – Mode of access: https://mui.com/.
16. React [Electronic resource]. – Mode of access: https://reactjs.org/.
17. SentenceTransformers [Electronic resource]. – Mode of access: https://www.sbert.net/.
18. Varun. Calculating Document Similarities using BERT, word2vec, and other models [Electronic resource] / Varun. – 2020. – Mode of access: https://towardsdatascience.com/calculating-document-similarities-using-bert-and-other-models-b2c1a29c9630.
19. Varun. The quantitative value of text, tf-idf and more... [Electronic resource] / Varun. – 2020. – Mode of access: https://medium.com/analytics-vidhya/the-quantitative-value-of-text-tf-idf-and-more-e3c7883f1df3
20. What is the difference between stemming and lemmatization? [Electronic resource]. – Mode of access: https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/.
21. Wood T. Transformer Neural Network [Electronic resource] / Thomas Wood – 2020. – Mode of access: https://deepai.org/machine-learning-glossary-and-terms/transformer-neural-network.
22. word2vec [Electronic resource]. – Mode of access: https://code.google.com/archive/p/word2vec/.

References

Alammar, Jay. *The Illustrated Bert, Elmo, and Co. (How NLP Cracked Transfer Learning)*. Visualizing Machine Learning One Concept at a Time. https://jalammar.github.io/illustrated-bert/.

Alammar, Jay. The Illustrated Transformer. The Illustrated Transformer – Jay Alammar – Visualizing Machine Learning One Concept at a Time. https://jalammar.github.io/illustrated-transformer/.

Alammar, Jay. The Illustrated word2vec. Visualizing Machine Learning One Concept at a Time. https://jalammar.github.io/illustrated-word2vec/.

Ali, Zafar. (2019, Jan. 7). *A Simple word2vec Tutorial*. Medium, Medium. https://medium.com/@zafaralibagh6/a-simple-word-2vec-tutorial-61e64e38a6a1.

What Is the Difference between Stemming and Lemmatization? Bitext. https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/.

Doshi, Ketan. (2021, June 3). *Transformers Explained Visually (Part 2): How It Works, Step-by-Step*. Medium, Towards Data Science. https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34.

Doshi, Ketan. (2021, June 8). *Transformers Explained Visually - Not Just How, but Why They Work so Well*. Medium, Towards Data Science. https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3.

Flask. Pallets. https://palletsprojects.com/p/flask/.

Gharavi, E., Bijari, K., & Zahirnia, K. (2016). A Deep Learning Approach to Persian Plagiarism Detection. *FIRE (Working Notes)*, 154–159.

Gilyadov, Julian. *Word2Vec Explained. Hacker's Blog – Get a Beer and Join Me down This Geeky Rabbit Hole Adventure.* https://israelg99.github.io/2017-03-23-Word2Vec-Explained/.

Google Code Archive – Long-Term Storage for Google Code Project Hosting. Google, https://code.google.com/archive/p/word2vec/.

Hambi, El Mostafa, & Faouzia, Benabbou. (2020). A deep learning-based technique for plagiarism detection: a comparative study. *International Journal of Artificial Intelligence*, *9* (1), 81–90.

Horev, Rani. (2018, Nov. 17*). Bert Explained: State of the Art Language Model for NLP*. Medium, Towards Data Science. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

Kathrani, Kashyap. (2022, Apr. 8). *All about Embeddings*. Medium, Medium. https://medium.com/@kashyapkathrani/all-about-embeddings-829c8ff0bf5b.

LANG-UK. Lang. https://lang.org.ua/uk/models/.

Malnik, Jessica. (2022, Mar. 7). *7 Common Types of Plagiarism Explained*. Writer. https://writer.com/blog/types-of-plagiarism/.

React – a JavaScript Library for Building User Interfaces. A JavaScript Library for Building User Interfaces. https://reactjs.org/.

SentenceTransformers Documentation. https://www.sbert.net/.

The React Component Library You Always Wanted. MUI. https://mui.com/.

Varun. (2020, Dec. 3). *Calculating Document Similarities Using Bert and Other Models*. Medium. https://towardsdatascience.com/calculating-document-similarities-using-bert-and-other-models-b2c1a29c9630.

Varun. (2020, Aug. 19). *The Quantitative Value of Text, TF-IDF and More...* Medium, Analytics Vidhya. https://medium.com/analytics-vidhya/the-quantitative-value-of-text-tf-idf-and-more-e3c7883f1df3.

Wood, T. (2020, July 7). *Transformer Neural Network*. DeepAI. https://deepai.org/machine-learning-glossary-and-terms/transformer-neural-network.

A. Hlybovets, M. Bikchentaev

SOFTWARE SYSTEM OF CHECKING FOR PLAGIARISM OF UKRAINIAN TEXTS

The purpose of this work is to describe the methodology of building a software system (application) for plagiarism checking of scientific publications in the Ukrainian language using two machine learning models, Word2Vec and BERT. We consider the detection of external plagiarism in Ukrainian texts.

Plagiarism is usually defined as the passing off someone else's ideas as your own. As the Internet becomes more and more accessible every day, a huge amount of data becomes available to people. Nowadays, it is quite easy to find a suitable study and plagiarize it instead of developing one's own from scratch.

Plagiarism undermines the efforts of the researcher whose work has been plagiarized and gives the plagiarist the opportunity to over-praise himself; such a person can be detrimental when appointed to an important position.

Many fields of life are susceptible to plagiarism, including research and education. Plagiarism can also take many forms: from straight up copy-paste to paraphrasing and sentence restructuring. This makes plagiarism a rather complex problem, where methods, such as longest common subsequence or n-grams, based on finding shared words between documents, might not work. Therefore, we might consider applying deep learning to the problem of plagiarism detection.

In this article we discussed the concept of plagiarism and listed its types. Two machine learning models have been proposed for plagiarism detection: Word2Vec and BERT. We also provided an overview of both models and described how they could be used in the problem of plagiarism detection.

A web application for plagiarism detection in the Ukrainian language has been developed. This application features React, a JavaScript framework, on the frontend and Python on the backend. To store application data, MongoDB is used.

This application allows a user to input a text that will be compared with the texts from the application database using cosine similarity or Euclidean distance as metrics. Comparison is performed using word embeddings, calculated by pre-trained BERT or Word2Vec model. A user can choose the model and similarity metrics using the application's UI.

The application can be further improved to not only output similarity metric but also highlight the similar sentences in the texts.

Keywords: machine learning, BERT, Word2Vec, plagiarism detection.

Матеріал надійшов 04.08.2022

Creative Commons Attribution 4.0 International License (CC BY 4.0)