

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з Основ програмування

(назва дисципліни)

на тему: "Пошукова семантична система на основі даних сайту
stackoverflow.com із використанням машинного навчання"

Студентів 1 курсу, групи ІП-14, 11

Коткова Тимура Максимовича

Сідака Кирила Ігоровича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник Головченко Максим Миколайович

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____

Національна оцінка _____

Члени комісії

(підпис)

(вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

(вчене звання, науковий ступінь, прізвище та ініціали)

Київ 2022

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрямок "ІПЗ"

Курс 1 Група ІП-11

Семестр 2

Курс 1 Група ІП-14

Семестр 2

ЗАВДАННЯ

на курсову роботу студентів

Коткова Тимура Максимовича,

Сідака Кирила Ігоровича

(прізвище, ім'я, по батькові)

1. Тема роботи Пошукова семантична система на основі даних сайту stackoverflow
із використанням машинного навчання

2. Строк здачі студентом закінченої роботи 12.06.2022

3. Вихідні дані до роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Дата видачі завдання 10.02.2022

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	10.02.2022	
2.	Підготовка ТЗ	02.05.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	03.05.2022	
4.	Розробка сценарію роботи програми	04.05.2022	
5.	Узгодження сценарію роботи програми з керівником	04.05.2022	
6.	Розробка (вибір) алгоритму рішення задачі	04.05.2022	
7.	Узгодження алгоритму з керівником	04.05.2022	
8.	Узгодження з керівником інтерфейсу користувача	05.05.2022	
9.	Розробка програмного забезпечення	06.05.2022	
10.	Налагодження розрахункової частини програми	06.05.2022	
11.	Розробка та налагодження інтерфейсної частини програми	07.05.2022	
12.	Узгодження з керівником набору тестів для контрольного прикладу	25.05.2022	
13.	Тестування програми	26.05.2022	
14.	Підготовка пояснювальної записки	05.06.2022	
15.	Здача курсової роботи на перевірку	12.06.2022	
16.	Захист курсової роботи	15.06.2022	

Студент _____
(підпис)

Керівник _____
(підпис)

Головченко Максим Миколайович
(прізвище, ім'я, по батькові)

" ____ " _____ 2022 р.

АНОТАЦІЯ

Пояснювальна записка до курсової: 109 сторінок, 9 рисунків, 8 таблиць, 9 посилань.

Об'єкт дослідження: пошукова семантична система на основі даних сайту stackoverflow.com із використанням машинного навчання.

Мета роботи: дослідження методів машинного навчання з використанням моделі Word2vec, GRU та мультиноміальної логістичної регресії, розробка програмного забезпечення для здійснення пошуку та візуалізації результатів за допомогою бота в телеграмі.

Опановано розробку програмного забезпечення з використанням ООП. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв'язання кожної з них.

Виконана програмна реалізація пошукової семантичної системи на основі даних сайту stackoverflow.com із використанням машинного навчання.

ЗМІСТ

ВСТУП.....	7
1 ПОСТАНОВКА ЗАДАЧІ.....	8
2 ТЕОРИТИЧНІ ВІДОМОСТІ.....	9
3 ОПИС АЛГОРИТМІВ.....	14
3.1. Алгоритм запуску бота	18
3.2. Алгоритм виводу знайдених результатів.....	18
3.3. Алгоритм кодування тегів у заданому датасеті.....	19
3.4. Алгоритм декодування тегів у заданому датасеті.....	20
3.5. Алгоритм збереження закодованих тегів.....	21
3.6. Алгоритм тренування моделі для передбачення категорії.....	21
3.7. Алгоритм розбиття заданого датасету на категорії	22
3.8. Алгоритм векторизації.....	23
3.9. Алгоритм підготовки даних	23
3.10. Алгоритм побудови та навчання моделі GRU.....	24
3.11. Алгоритм пошуку статей за заданим пошуковим запитом	24
4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	26
4.1. Діаграма класів програмного забезпечення.....	26
4.2. Опис методів частин програмного забезпечення	26
4.2.1. Користувацькі методи	26
4.2.2. Стандартні методи	28
5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	32
5.1. План тестування	32
5.2. Приклади тестування	32
6 ІНСТРУКЦІЯ КОРИСТУВАЧА.....	36

6.1. Робота з програмою.....	36
6.2. Формат вхідних та вихідних даних.....	40
6.3. Системні вимоги програмного забезпечення	40
ВИСНОВКИ.....	42
ПЕРЕЛІК ПОСИЛАНЬ	43
ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ	44
ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ	47
app.py	48
config.py	49
handle_replies.py.....	50
help.py	51
search.py	52
start.py	55
search_keyboard.py	56
state_storage.py	57
set_bot_commands.py	58
loader.py	59
encode_tags.py	60
save_encoded_tags.py	63
categories_separation.py	64
train_categories.py.....	67
concatenate_datasets.py.....	69
getting_data.py	70
clean_tags.py	72
create_datasets.py.....	75

data_from_site.py	76
gru_model.py	80
tag_predictor.py	83
vectorization.py	85
prepare_data.py	87
search_pipeline.py	90
html_to_text.py	98
normalize_content.py	101
normalize_functions.py	104
logger.py	108

ВСТУП

Дана робота присвячена розробці Пошукової семантичної системи на основі даних сайту stackoverflow.com із використанням машинного навчання. Задача програмного забезпечення полягає в текстовому відображенні всіх доступних статей за заданим пошуковим запитом в телеграм-боті, та всіх додаткових значень.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде знаходити задану кількість статей з сайту stackoverflow.com, найбільш відповідних до заданого запиту, наступними методами:

а) метод передбачення тегів із використанням моделі машинного навчання;

б) метод передбачення категорії по заданим тегам із використанням логістичної регресії;

Вхідними даними для даної роботи є запит, який заданий у вигляді текстового повідомлення боту в телеграмі, та кількість статей, які задані у вигляді текстового повідомлення боту в телеграмі та які треба отримати, що знаходяться в межах від 1 до 10.

Вихідними даними для даної роботи є список статей з сайту stackoverflow.com у вигляді текстового повідомлення в телеграмі від бота, а саме питання статті у вигляді гіперпосилання, оцінка схожості конкретної статті із заданим запитом (коефіцієнт подібності), теги, відповідним чином оброблений текст питання статті та час, витрачений на пошук статей. Програмне забезпечення повинно видавати статті за умови, що введена кількість статей є натуральним числом в межах від 1 до 10. Якщо це не так, то програма (бот) повинна видати відповідне повідомлення. Якщо не було знайдено жодної відповідної статті (з коефіцієнтом подібності більше 0,8) для заданого запиту, то програма (бот) повинна видати відповідне повідомлення.

2 ТЕОРИТИЧНІ ВІДОМОСТІ

Текстовий запит можна розбити на слова та представити кожне у числовому форматі, тобто у вигляді вектору дійсних чисел, та, просумувавши ці вектори, поділити отриманий вектор на кількість початкових векторів, отримавши один вектор, координати якого будуть середніми арифметичними відповідних координат початкових векторів, за допомогою такого методу як вкладання слів [1]. Сутність цього методу полягає в тому, що кожному слову ставиться у відповідність вектор дійсних чисел, кожне число відповідає певній характеристиці цього слова, наприклад: семантичному відношенню та різним семантичним подібностям. Для цієї задачі використовується алгоритм *word2vec* [2]. Алгоритм *word2vec* використовує нейромережеву модель для навчання пов'язаностей слів із великого корпусу тексту. Вектори для слів ретельно підбираються таким чином, щоб проста математична функція (косинусна подібність векторів, тобто косинус кута між двома векторами) вказувала на рівень семантичної подібності між словами, представленими цими векторами. Щоб виробляти розподілене представлення слів, *word2vec* може використовувати будь-яку з двох архітектур моделей: неперервну торбу слів (НТС, continuous bag-of-words, CBOW) та неперервний пропуск-грам (continuous skip-gram). В даному випадку використовується саме неперервна торба слів. В архітектурі неперервної торби слів модель передбачує поточне слово з вікна слів навколишнього контексту. Порядок слів контексту не впливає на передбачення (припущення торби слів). На рисунку 2.1 зображена архітектура моделі CBOW.

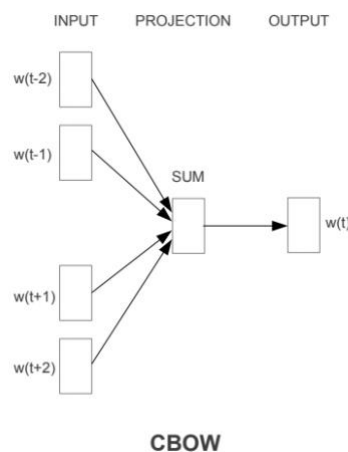


Рисунок 2.1 – модель CBOW

Архітектура моделі CBOW намагається передбачити потрібне слово за словами, які знаходяться біля нього в контексті. Ця модель приймає на вхід розподілені представлення слів у контексті (наприклад, вектор one-hot-representation, де координата з номером слова у словнику буде дорівнювати 1, а інші – 0), щоб передбачити потрібне слово. Таким чином, на вході моделі буде певна конкретна кількість векторів розмірності $1 \times V$ [4], де V – це кількість слів у словнику, який використовується для навчання моделі. У прихованому шарі нейронної мережі кожен вектор множиться на відповідну матрицю розмірності $V \times E$, де E – це гіперпараметр, що відповідає розмірності результуючих векторів, тобто на виході утворюється певна кількість (дорівнює кількості вхідних векторів) векторів розмірності $1 \times E$, що перетворюються в один вектор шляхом обчислення середнього значення для відповідних координат кожного вектору. Цей результуючий вектор передається вже в шар softmax (нормованої експоненційної функції) [5] :

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^E e^{z_k}} \text{ для } j = 1, \dots, E$$

Після обчислення цієї функції активації для даного вектору результатом є вектор розмірності $1 \times E$, кожна координата якого є певною характеристикою. Цей вектор буде використовуватись для обчислення косинусу подібності між цим вектором запиту та кожним вектором питання статті із множини відібраних статей.

Для передбачення найбільш відповідних тегів для даного запиту використовуються вентильні рекурентні вузли (GRU - Gated recurrent units) [6]. Повний рекурентний вузол працює наступним чином. На вхід подаються значення вектору входу x_t та значення виходу (при $t = 0$, вектор виходу $h_0 = 0$). По ним обчислюється претендент на нове значення виходу — вектор вузла скидання (*reset gate vector*) r_t , який обчислюється як функція активації (зазвичай сигмоїд) від матричного виразу по параметрам . Незалежно, подібним чином, обчислюється вектор вузла уточнення (*update gate vector*) z_t . Цей вектор містить значення, які визначають, чи варто залишити значення зі старого вектору, чи

взяти нове значення. Фактично, це набір «вентилів» (*gate*), які «пропускають» або старе, або нове значення. Далі обчислюється вектор виходу h_t , в якому з ймовірністю z_t береться старе значення з вектору h_{t-1} , або з ймовірністю $1 - z_t$ обчислюється нове значення. Формули для обчислень наступні:

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h), \text{ де } \circ - \text{добуток}$$

Адамара (поелементний добуток матриць)

Змінні:

x_t – вектор входу;

h_t – вектор виходу;

z_t – вектор вузла уточнення;

r_t – вектор вузла скидання;

W, U та b – матриці та вектор параметрів.

Функції активації:

σ_g – сигмоїдна функція [7]:

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

σ_h – гіперболічний тангенс [8]:

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

На рисунку 2.2 зображений повний рекурентний вузол.

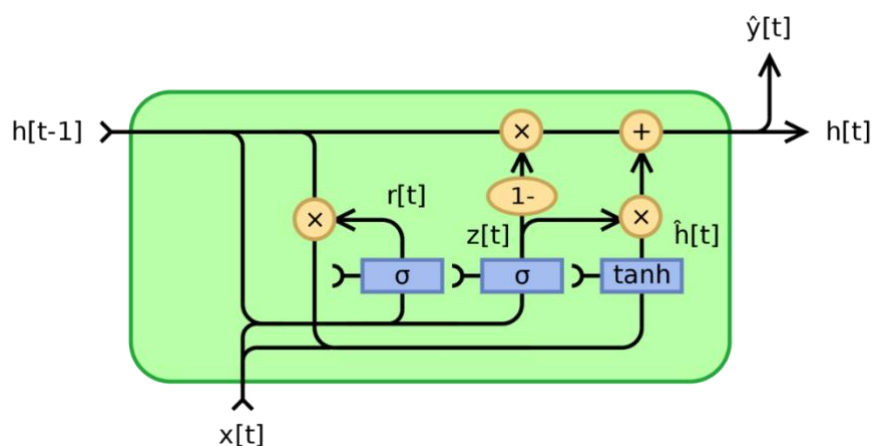


Рисунок 2.2 – повний рекурентний вузол

Для передбачення категорії заданого запиту по отриманим тегам використовується модель мультиноміальної логістичної регресії [9]. У статистиці мультиноміальна логістична регресія — це метод класифікації, який узагальнює логістичну регресію на багатокласові проблеми, тобто з більш ніж двома можливими дискретними результатами. Мультиноміальна логістична регресія використовується, коли відповідна залежна змінна є номінальною (еквівалентно категоричною, що означає, що вона потрапляє в будь-яку з набору категорій, які не можуть бути впорядковані будь-яким значущим чином) і для якої існує більше двох категорій. Для нашої задачі номінальні значення це теги, які містить та чи інша стаття. Як приклад: одна стаття може містити теги `python`, `django`, `web`, а інша може містити теги `c#`, `.net`, `string`. Ми не можемо заздалегідь знати, які значення тегів буде містити вхідний у модель елемент.

Результатом передбачення даної моделі є вектор з K елементів, де K — кількість категорій, причому кожний елемент (координата) — це ймовірність того, що заданим даним буде відповідати певна конкретна категорія, а сума всіх елементів даного вектору буде дорівнювати одиниці. Таким чином, для n -ої категорії з K категорій формула обчислення ймовірності цієї категорії для i -го елемента наступна:

$$\Pr(Y_i = n) = \frac{e^{\beta_n \cdot x_i}}{1 + \sum_{k=1}^{K-1} e^{\beta_k \cdot x_i}}$$

Обчислення оптимальних коефіцієнтів у даній моделі зводиться до використання певної ітеративної процедури, які є алгоритмом, що працює на основі градієнтного спуску. На вхід же в нашому випадку модель приймає вектор `one-hot-representation` розмірності 2000, де координата даного вектору з індексом даного тегу у словнику тегів, де зберігається 2000 найпопулярніших тегів, буде дорівнювати 1, а інші — 0. В якості порогу ймовірності обрано 0,95, тобто, якщо для передбачених тегів максимальна ймовірність серед ймовірностей кожної з категорій буде як мінімум 0,95, то для пошуку статей для заданого запиту буде використовуватись датасет, де усі статті відповідають цій категорії (попередньо ця ж модель мультиноміальної логістичної регресії була використана для

розбиття великого датасету на менші датасети по категоріям). Якщо ж ця ймовірність буде менша 0,95, то будуть використовуватись 3 датасети (які відповідають трьом найбільш ймовірним категоріям).

3 ОПИС АЛГОРИТМІВ

Перелік всіх основних змінних та їхнє призначення наведено в таблиці

Таблиця 3.1 – Основні змінні та їхні призначення

Змінна	Призначення
text	Пошуковий запит, введений користувачем, який буде зберігатися у стані користувача
MAX_LIMIT	Максимальна можлива кількість статей (обмеження зверху на кількість статей)
num	Кількість статей, що введена користувачем
user_data	Словник, що містить дані, які зберігаються в поточному стані користувача (пошуковий запит користувача з ключем “search_text”)
search_text	Пошуковий запит, введений користувачем, який отримується зі словника user_data за ключем “search_text”
reply_text	Текст зі списком статей та часом пошуку, тобто текст повідомлення бота
t_0	Поточний час (в секундах) перед запуском процедури пошуку статей
articles	Список знайдених статей (список словників)

Продовження таблиці 3.1

Змінна	Призначення
df	Таблиця у форматі датафрейму, що містить наступні колонки: заголовок статті, її теги та категорію
tags_dict	Словник, що в якості ключів містить індекси тегів у відповідній колонці df, а в якості значень самі теги
tags_freq	Словник, що в якості ключів містить теги, а в якості значень кількість кожного тегу у відповідній колонці df
keys	Список із 2000 найчастіших тегів
values	Відповідні індекси тегів зі списку keys
df_tags	Таблиця у форматі датафрейму, що містить наступні колонки: тег та код, де в колонці тегів містяться елементи списку keys, а в колонці кодів – елементи списку values (відповідний індекс для кожного тегу)
tag_keys	Копія змінної df_tags
new_enc_tags	Двовірний масив, де кожний підмасив містить індекси тегів відповідної статті
temp_ind	Тимчасовий список, який містить індекси тегів поточної статті

Продовження таблиці 3.1

Змінна	Призначення
new_dec_tags	Двовірний масив, де кожний підмасив містить теги відповідної статті
temp_tags	Тимчасовий список, який містить теги поточної статті
encoded_df	Таблиця у форматі датафрейму, що містить наступні колонки: заголовок статті, індекси її тегів та категорію
df_tags_keys	Таблиця у форматі датафрейму, що містить наступні колонки: тег та код, де в колонці тегів містяться елементи списку keys, а в колонці кодів – елементи списку values (відповідний індекс для кожного тегу)
decoded_df	Таблиця у форматі датафрейму (створена на основі encoded_df), що містить наступні колонки: заголовок статті, її теги та категорію
y_bin	Матриця, яка представляє one-hot encoding для тегів кожної статті
X_train_tags	Вибірка 80% з y_bin
X_test_tags	Вибірка 20% з y_bin
y_train_tags	Масив типу pandas.Series, що містить категорії статті для відповідних рядків з X_train_tags

Продовження таблиці 3.1

Змінна	Призначення
y_test_tags	Масив типу pandas.Series, що містить категорії статті для відповідних рядків з X_test_tags
logreg_tags	Модель логістичної регресії для передбачення категорії статті по її тегах
y_pred	Масив передбачених категорій для кожного елемента X_test_tags
main_data	Датасет у вигляді таблиці (датафрейму), що містить статті із сайту stackoverflow з колонками, такими як: заголовок статті, текст питання, теги та інші
df_sample	Вибірка з поточних 10000 рядків з main_data
w2v_model	Модель word2vec, яка перетворює слово у векторну форму
W2V_SIZE	Розмірність вектору моделі word2vec
W2V_WINDOW	Розмір вікна для моделі word2vec, тобто кількість сусідніх слів
W2V_EPOCH	Кількість ітерацій для навчання моделі word2vec
W2V_MIN_COUNT	Обмеження на мінімальну частоту слів, тобто слова з меншою чистотою будуть ігноруватися

Продовження таблиці 3.1

Змінна	Призначення
<code>vocab_size</code>	Розмір словника для моделі <code>word2vec</code> , тобто кількість слів
<code>tags</code>	Множина передбачених тегів для заданого пошукового запиту
<code>all_title_embeddings</code>	Матриця, що містить векторні представлення для кожного заголовку статті
<code>search_res</code>	Список знайдених статей (список словників)

3.1. Алгоритм запуску бота

Функція запуску

1. ПОЧАТОК
2. Використати функцію `set_default_commands` для `dispatcher`.
3. Викинути *An unexpected error occurred* при помилках.
4. КІНЕЦЬ

Головна функція

1. ПОЧАТОК
2. Виконуємо метод `start_polling` об'єкту `executor`.
3. КІНЕЦЬ

3.2. Алгоритм виводу знайдених результатів

Функція для пошуку потрібної статті

1. ПОЧАТОК
2. Змінній `text` присвоюємо значення `text` з об'єкту `message`.
3. Перевіряємо чи значення змінної `text` в десятковому записі та чи належить значення проміжку від 0 до `MAX_LIMIT`.
 - 3.1. Якщо виконується умова, виводимо повідомлення *Searching...*
 - 3.2. Присвоюємо змінній `num` значення `text`, яке переведено в `int`.
 - 3.3. В змінну `user_data` записуємо результат виконання методу `get_data` об'єкта `state`.

- 3.4. Виконуємо метод *finish* об'єкта *state*.
 - 3.5. В змінну *search_text* записуємо значення масиву *user_data*
 - 3.6. В змінну *t_0* записуємо значення часу.
 - 3.7. В змінну *articles* записуємо результат виконання функції *search_results*.
 - 3.8. Якщо знайшлися такі статті
 - 3.8.1. В змінну *reply_text* записуємо значення *Articles:\n—————
—————\n*.
 - 3.8.2. Проходимося по всіх статтях
 - 3.8.2.1. Виводимо для кожної назву, рейтинг схожості, теги, тіло статті.
 - 3.8.2.2. Виводимо, за скільки часу був завершений пошук.
 - 3.9. Якщо не знайшлися, то виводимо відповідне повідомлення.
 - 4. Якщо не відповідає умові, то викидаємо помилку.
 - 5. Виконуємо метод *reply* об'єкту *message*.
 - 5.1. В разі помилки виконуємо метод *answer* об'єкту *message*.
 - 6. КІНЕЦЬ
- 3.3. Алгоритм кодування тегів у заданому датасеті
- Клас *Encoder*
- 1. ПОЧАТОК
 - 2. Метод *__init__*
 - 2.1. Створюємо конструктор класа.
 - 3. Функція *keys_from_tags*
 - 3.1. Копіюємо значення *df*.
 - 3.2. Розділяємо теги *df* знаком *|*.
 - 3.3. Створюємо порожній об'єкт *tags_dict*.
 - 3.4. Створюємо порожній об'єкт *tags_freq*.
 - 3.5. Присвоюємо *ind* 0.
 - 3.6. Проходимося циклом по тегам по *i*
 - 3.6.1. Проходимося циклом по тегам по *j*
 - 3.6.1.1. Якщо *tags[i][j]* не в списку ключів
 - 3.6.1.1.1. Присвоюємо *tags_dict[i][j]* *ind*.
 - 3.6.1.1.2. Присвоюємо *tags_freq[i][j]* 0.
 - 3.6.1.1.3. Додаємо до *ind* 1.
 - 3.6.1.2. Якщо ні, додаємо до *tags_freq[i][j]* 1.
 - 3.7. Створюємо в *tags_freq* словник з айтемів.
 - 3.8. Створюємо в *keys* список ключів.
 - 3.9. Створюємо в *values* значення.

- 3.10. Присвоюємо змінній `df_tags` результат виконання методу `DataFrame`.
 - 3.11. Присвоюємо змінній `df_tags['tag']` значення ключів.
 - 3.12. Присвоюємо змінній `df_tags['code']` значення значень.
 - 3.13. Повертаємо `df_tags`.
 4. Метод `encode_tags`
 - 4.1.Копіюємо значення `df`.
 - 4.2.Створюємо новий масив `new_enc_tags`.
 - 4.3.Записуємо ключі.
 - 4.4.Проходимося циклом по рядкам айтемів
 - 4.4.1. Створюємо новий масив `temp_ind`.
 - 4.4.2. Проходимося циклом по тегам, розділеним |
 - 4.4.2.1. Присвоюємо змінній `code` значення відповідного значення.
 - 4.4.2.2. Доповнюємо масив `temp_ind` значенням `code`.
 - 4.4.3. Доповнюємо масив `new_enc_tags` значенням `t`.
 - 4.5.Записуємо теги в `df.tags`.
 - 4.6.Повертаємо `df`.
 5. КІНЕЦЬ
- 3.4. Алгоритм декодування тегів у заданому датасеті
- Клас *Decoder*
1. ПОЧАТОК
 2. Створюємо конструктор класу.
 3. Метою `decode_tags`
 - 3.1.Копіюємо значення `df`.
 - 3.2.Створюємо новий масив `new_dec_tags`.
 - 3.3.Записуємо ключі.
 - 3.4.Проходимося циклом по рядкам айтемів
 - 3.4.1. Створюємо новий масив `temp_tags`.
 - 3.4.2. Проходимося циклом по тегам, розділеним |
 - 3.4.2.1. Присвоюємо змінній `code` значення відповідного значення.
 - 3.4.2.2. Доповнюємо масив `temp_tags` значенням `tag`.
 - 3.4.3. Доповнюємо масив `new_dec_tags` значенням `temp_tags`.
 - 3.5.Записуємо теги в `df.tags`.
 - 3.6.Повертаємо `df`.
 4. КІНЕЦЬ

3.5. Алгоритм збереження закодованих тегів

1. ПОЧАТОК
2. Записуємо в *env* результат виконання функції *Env*.
3. Викликаємо метод *read_env* об'єкта *env*.
4. Записуємо в *logger* результат виконання функції *get_logger*.
5. Записуємо в *DATA_PATH* результат виконання функції *env.str* з параметром *DATA_PATH*.
6. Записуємо в змінну *dataframe* результат виконання функції *read_csv* бібліотеки *pandas* з параметрами *DATA_PATH + 'categories_data.csv'*, *engine='pyarrow'*.
7. Записуємо в змінну *encoder* результат виконання функції *Encoder* параметрами *dataframe*, 2000.
8. Записуємо в *encoded_df* теги.
9. Перетворюємо *encoded_df* в формат *csv*.
10. Виводимо повідомлення про успішне збереження.
11. Записуємо в змінну *df_tags_keys* значення ключів з тегів.
12. Перетворюємо *df_tags_keys* в формат *csv*.
13. Виводимо повідомлення про успішне збереження.
14. Записуємо в змінну *encoder* результат виконання функції *Decoder* параметрами *encoded_df, df_tags_keys*.
15. Записуємо в *decoded_df* декодовані теги.
16. Перетворюємо *decoded_df* в формат *csv*.
17. Виводимо повідомлення про успішне збереження.
18. КІНЕЦЬ

3.6. Алгоритм тренування моделі для передбачення категорії

1. ПОЧАТОК
2. Записуємо в *env* результат виконання функції *Env*.
3. Викликаємо метод *read_env* об'єкта *env*.
4. Записуємо в *logger* результат виконання функції *get_logger*.
5. Фільтруємо попередження з *ignore*.
6. Записуємо в *DATA_PATH* результат виконання функції *env.str* з параметром *DATA_PATH*.
7. Записуємо в *MODELS* результат виконання функції *env.str* з параметром *MODELS*.
8. Об'являємо функції для розділення тегів.
 - 8.1. Якщо строка не пуста
 - 8.1.1. Проходимося по рядку та розділяємо теги знаком «|».

9. Записуємо у *encoded_df* результат виконання методу *read_csv* бібліотеки *pandas* з параметрами *DATA_PATH* + '*enc_dataset.csv*', *engine='pyarrow'* та виконуємо після цього метод *sample* з параметром *frac=1*.
 10. Записуємо у *encoded_df.tags* результат виконання методу *apply* об'єкта *encoded_df.tags*.
 11. Виконуємо метод *dropna* об'єкта *encoded_df* з параметрами *inplace=True* та *axis=0*.
 12. Присвоюємо *tags* значення з *encoded_df.tags*.
 13. Записуємо в *multilabel_binarizer* результат виконання функції *MultiLabelBinarizer*.
 14. Записуємо в *y_bin* результат виконання методу *fit_transform* об'єкта *multilabel_binarizer* з параметрами *encoded_df.tags*.
 15. Викликаємо метод *dump* бібліотеки *pickle* з параметрами *multilabel_binarizer* та *open(MODELS + 'mlb.pkl', 'wb')*.
 16. Записуємо відповідні теги в змінні *X_train_tags*, *X_test_tags*, *y_train_tags*, *y_test_tags*.
 17. Записуємо в *logreg_tags* результат виконання функції *LogisticRegression* з параметрами *n_jobs=1* та *C=1e5*.
 18. Записуємо в *logreg_tags* результат виконання методу *fit* цього ж об'єкта з параметрами *X_train_tags*, *y_train_tags*.
 19. В *y_pred* записуємо результат виконання методу *predict* об'єкта *logreg_tags* з параметром *X_test_tags*.
 20. Викликаємо метод *dump* бібліотеки *pickle* з параметрами *logreg_tags* та *open(MODELS + 'model_tags.pkl', 'wb')*
 21. Виводимо точність.
 22. Виводимо звіт з класифікації.
 23. КІНЕЦЬ
- 3.7. Алгоритм розбиття заданого датасету на категорії
1. ПОЧАТОК
 2. Проходимося циклом по в межах цілої частини від ділення довжини *main_data* діленої на 10000 + 1.
 - 2.1. Записуємо в *df_sample* масив ключ: значення.
 - 2.2. Записуємо в *df_dict* результат виконання функції *get_df* з параметром *df*.
 - 2.3. Проходимося по значеннях ключів та *df* у *df_dict.items*.
 - 2.4. Якщо лічильник = 0

2.4.1. Виконуємо функцію *save_dataset* з параметрами *key*, *df_sample*, *True*, *'w'*.

2.5.Інакше

2.5.1. Виконуємо функцію *save_dataset* з параметрами *key*, *df_sample*, *False*, *'w'*.

2.6.Виводимо скільки було оброблено айтемів.

3. КІНЕЦЬ

3.8. Алгоритм векторизації

1. ПОЧАТОК

2. Записуємо в *logger* результат виконання функції *get_logger* з параметром без обробки помилок.

3. Записуємо в *W2V_SIZE* значення 300.

4. Записуємо в *W2V_WINDOW* значення 7.

5. Записуємо в *W2V_EPOCH* значення 32.

6. Записуємо в *W2V_MIN_COUNT* значення 10.

7. Записуємо в змінну *documents* результат деструктуризації масиву *preprocessed_data.post_corpus*.

8. Записуємо в змінну *w2v_model* результат виконання методу *Word2Vec* об'єкту *gensim.models.word2vec* з параметрами *vector_size=W2V_SIZE*, *window=W2V_WINDOW*, *min_count=W2V_MIN_COUNT*, *workers=8*.

9. Виконуємо метод *build_vocab* у об'єкта *w2v_model* з параметром *documents*.

10. Записуємо в змінну *words* список ключів об'єкта *w2v_model.wv.key_to_index*.

11. Записуємо в змінну *vocab_size* довжину списку ключів *words*.

12. Друкуємо розмір словнику.

13. Тренуємо векторизацію слів.

14. КІНЕЦЬ

3.9. Алгоритм підготовки даних

1. ПОЧАТОК

2. Записуємо в змінну *tag_encoder* результат виконання функції *MultiLabelBinarizer*.

3. Записуємо в *tags_encoded* результат виконання методу *fit_transform* об'єкту *tag_encoder* з параметром *final_tag_data*.

4. В змінну *data* записуємо результат виконання методу *DataFrame* бібліотеки *pandas*. Параметр – стовбці з значенням *corpus_code_combined*.

5. Записуємо в масив `data_preprocessed_data.post_corpus`.
6. Завантажуємо модель W2V.
7. Розділяємо тренувальні і тестові дані.
8. Виводимо розмір тренувальних і тестових даних.
9. Перетворюємо тестові і тренувальні дані в доповнені послідовності.
10. Створюємо вбудовану матрицю.
11. Проходимось циклом по айтемах.
12. Якщо слово є в моделі, доповнюємо матрицю.
13. Виводимо розмір матриці.
14. КІНЕЦЬ

3.10. Алгоритм побудови та навчання моделі GRU

1. ПОЧАТОК
2. Створюємо Sequential модель.
3. Додаємо до моделі Embedding прошарок.
4. Додаємо до моделі GRU прошарок.
5. Додаємо до моделі Dense прошарок.
6. Виконуємо дропаут.
7. Робимо батч-нормалізацію.
8. Додаємо до моделі Dense прошарок.
9. Компілюємо модель.
10. Робимо сумарну статистику.
11. Тренуємо модель.
12. Починаємо підбір моделі.
13. Закінчуємо підбір моделі.
14. Зберігаємо ваги.
15. Переводимо модель в JSON формат.
16. КІНЕЦЬ

3.11. Алгоритм пошуку статей за заданим пошуковим запитом

1. ПОЧАТОК
2. Робимо передобробку тексту.
3. Створюємо пошуковий вектор.
4. Отримуємо передбачені теги.
5. Створюємо масив `search_res`.
6. Створюємо масив `all_title_embeddings`.
7. Сортуюмо значення по кількості тегів.
8. Видаляємо дублікати.
9. Перевиставляємо індекси.

10. Обчислюємо tfidf.
11. Обчислюємо tfidf пошукового рядку.
12. Отримуємо вбудовану назву з моделі W2V.
13. Обчислюємо косинусну подібність.
14. Проходимося циклом по iteritems.
15. Створюємо новий об'єкт temp.
16. Доповнюємо результати пошуку об'єктом temp.
17. Якщо значення подібності менше 0.8 нічого не повертаємо.
18. Повертаємо результат пошуку.
19. КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Діаграма класів програмного забезпечення

Діаграма класів розробленого програмного забезпечення наведена на рисунку 4.1.

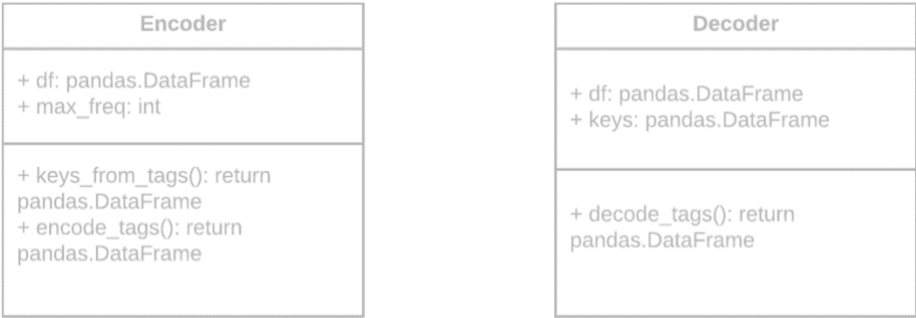


Рисунок 4.1 – Діаграма класів

4.2. Опис методів частин програмного забезпечення

4.2.1. Користувацькі методи

У таблиці 4.1 наведено користувацькі методи, використані при розробці програмного забезпечення.

Таблиця 4.1 – Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	-	on_startup	Запуск бота	Dispatcher – передається подальший метод	Повертає результат виконання внутрішнього методу або помилку

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
2	-	Input_limit	Обмеження для поля вводу та виведення списку статей	message: Message, state: FSMContext	Повертає статті
3	Handle_replies	Reply_to_message	Відповідь на повідомлення	Message, text	Повертає відповідь
4	Encoder	keys_from_tags	Збирає ключі з тегів	Конструктор	Теги
5	Encoder	encode_tags	Кодує теги	Конструктор	Теги
6	Decoder	decode_tags	Декодує теги	Конструктор	Теги
7	-	split_tags	Розділяє теги	Рядок	Розділені теги
8	-	predict_category	Передбачає категорію	Список тегів	Список результатів
9	-	get_df	Отримання df	data	Словник df
10	-	save_dataset	Зберігає дані	key_cat, dataframe, header=False, mode='a'	Збереження
11	-	get_logger	Використовує ся для логеру	Немає вхідних параметрів	Немає вихідних параметрів

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
12	-	exception	Використовує ся для виведення повідомлення	Повідомлен ня про помилку	Повертає повідомлен ня про помилку

4.2.2. Стандартні методи

У таблиці 4.2 наведено стандартні методи, використані при розробці програмного забезпечення.

Таблиця 4.2 – Стандартні методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	Bot_tg	set_default_commands	Використовує ся для налаштування дефолтних команд бота	Dispatcher	Повертає повідомлен ня про завершене налаштува ння
2	executor	Start_polling	Використовує ся для створення loop	dp, on_startup= on_startup, skip_updates=True	Сигнал завершення
3	str	Isdecimal	Переверяє чи число десятькове	-	Повертає True, якщо десятькове

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
4	Int	Int	Переводить число в десяткову форму	Число	Повертає число в десятковій формі
5	Message	Reply	Виводить повідомлення	Повідомлен ня	Вивід повідомлен ня
6	FSMContext	Get_data	Отримує дані	-	Дані
7	FSMContext	Finish	Закінчує процес	-	Закінчення процесу
8	Time	time	Отримання часу	-	час
9	Search_engi ne	Search_results	Пошук результатів	search_text, num	результат
10	-	Range	Межі пошуку	Діапазон	Числа
11	fmt	text	Текст на сторінці	Потрібний текст	текст
12	fmt	hlink	Створення посилання	Текст посилання	посилання
13	fmt	hbold	Напівжирний текст	Текст	Напівжирн ий текст
14	pandas	copy	Зробити копію	Змінна для копії	Копію
15	str	split	Розділити	Строка	Масив
16	str	dict	Словник	Строка	Словник
17	str	list	Список	Строка	Список

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
18	pandas	items	Взяти айтем	-	Масив айтемів
19	Pandas	DataFrame	Структура даних	-	Структуру даних
20	Pandas	iterrows	Ітерує по рядкам	-	-
21	Array	append	Додавання до масиву	Дані для масиву	Новий масив
22	str	join	Сполучення в строку	Масив	Строку
23	Env	read_env	Прочитати env формат	-	-
24	Env	read_csv	Прочитати csv файл	-	-
25	Env	to_csv	Перевести файл в csv	Файл	Новий файл
26	logger	info	Отримати інформацію	Текст для виведення	Текст
27	warnings	filterwarnings	Фільтрація помилок	Параметр для помилок	-
28	Pandas	dropna	для видалення рядків та стовпців зі значеннями NULL	-	-

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
29	Pandas	apply	для застосування функцій	-	-
30	Pandas	sample	Повернення випадкової вибірки елементів	-	випадкової вибірки елементів
31	Sklearn	fit_transform	підігнати модель до даних, а потім перетворити дані відповідно до підібраної моделі	Модель	Перетворен а модель
32	Object	dump	записує об'єкт Python у файл у форматі JSON	Об'єкт	Файл JSON
33	Sklearn	train_test_split	Ділить вибірку на тренувальну та на тестову	Вибірка	Дві вибірки
34	Sklearn	fit	підігнати модель до даних	Модель	Модель
35	Pandas	predict	Зробити передбачення	Дані	Передбачен а модель

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. План тестування

Складемо план тестування програмного забезпечення, за допомогою якого протестуємо весь основний функціонал та реакцію на виключні ситуації

а) Тестування правильності введених значень у телеграм-боті

1) Тестування введення пошукових запитів, які не відповідають темі сайту `stackoverflow` (такі, що не належать до програмування)

2) Тестування введення кількості статей

б) Тестування правильності передбачення моделі GRU тегів на основі тексту запиту

1) Тестування процентного відношення правильно класифікованих запитів до неправильно класифікованих.

в) Тестування правильності передбачення моделі мультиноміальної логістичної регресії категорії на основі тегів

1) Тестування процентного відношення правильно класифікованих категорій до неправильно класифікованих.

5.2. Приклади тестування

Проведемо тестування програмного забезпечення згідно з розробленим планом, фіксуючи мету, початковий стан програми, вхідні дані, схему проведення, очікуваний результат і стан програми після проведення випробувань кожного тесту в окрему таблицю (таблиці 5.1-5.4).

Таблиця 5.1 – Тестування введення значень, які не відповідають темі сайту `stackoverflow`

Мета тесту	Перевірити, чи введений користувачем запит відповідає тематиці програмування
Початковий стан програми	Відкрите діалогове вікно в телеграм-боті
Вхідні дані	Mattermost

Продовження таблиці 5.1

Схема проведення тесту	Проконтролювати, чи буде бот видавати статті за заданим запитом
Очікуваний результат	Повідомлення про відсутність відповідних статей для даного запиту
Стан програми після проведення випробувань	Видано повідомлення: «No corresponding articles were found for such request: "Mattermost"»

Таблиця 5.2 – Тестування введення кількості статей

Мета тесту	Перевірити, чи введений користувачем текст є натуральним числом від 1 до 10 включно
Початковий стан програми	Відкрите діалогове вікно в телеграм-боті
Вхідні дані	69
Схема проведення тесту	Проконтролювати, чи буде бот видавати повідомлення про введення нового правильного значення (натуральне число від 1 до 10)
Очікуваний результат	Повідомлення про некоректність введених даних та пропозиція ввести ще раз
Стан програми після проведення випробувань	Видано повідомлення: «Incorrect input. Only non-negative integers are allowed which are ≤ 10 . Try again:»

Таблиця 5.3 – Тестування процентного відношення правильно класифікованих запитів до неправильно класифікованих

Мета тесту	Перевірити, наскільки правильно модель GRU за метрикою ассигасу (точність) передбачує теги для заданих запитів
Початковий стан програми	Відкрита консоль в інтерактивному середовищі розробки (IDE)
Вхідні дані	Тестова вибірка оброблених значень (числових представлень текстових запитів) та відповідних тегів для них
Схема проведення тесту	Проконтролювати виконання програми для тестування і побачити результат її виконання
Очікуваний результат	Повідомлення про виконання програми і результат метрики ассигасу
Стан програми після проведення випробувань	Видано результат тестування за метрикою ассигасу, який становить 92.13%

Таблиця 5.4 – Тестування процентного відношення правильно класифікованих категорій до неправильно класифікованих

Мета тесту	Перевірити, наскільки правильно модель LogisticRegression за метрикою ассигасу (точність) передбачує категорії для заданих тегів
------------	--

Продовження таблиці 5.4

Початковий стан програми	Відкрита консоль в інтерактивному середовищі розробки (IDE)
Вхідні дані	Тестова вибірка оброблених значень (числових представлень тегів) та відповідних категорій для них
Схема проведення тесту	Проконтролювати виконання програми для тестування і побачити результат її виконання
Очікуваний результат	Повідомлення про виконання програми і результат метрики accuracy
Стан програми після проведення випробувань	Видано результат тестування за метрикою accuracy, який становить 91.92%

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1. Робота з програмою

Після запуску виконавчого файлу з розширенням *.exe запускається телеграм бот. Коли користувач переходить в особистий чат з ботом, відкривається вікно з привітанням і коротким описом функціоналу бота (рисунок 6.1), якщо, звісно, користувач до цього не писав боту.

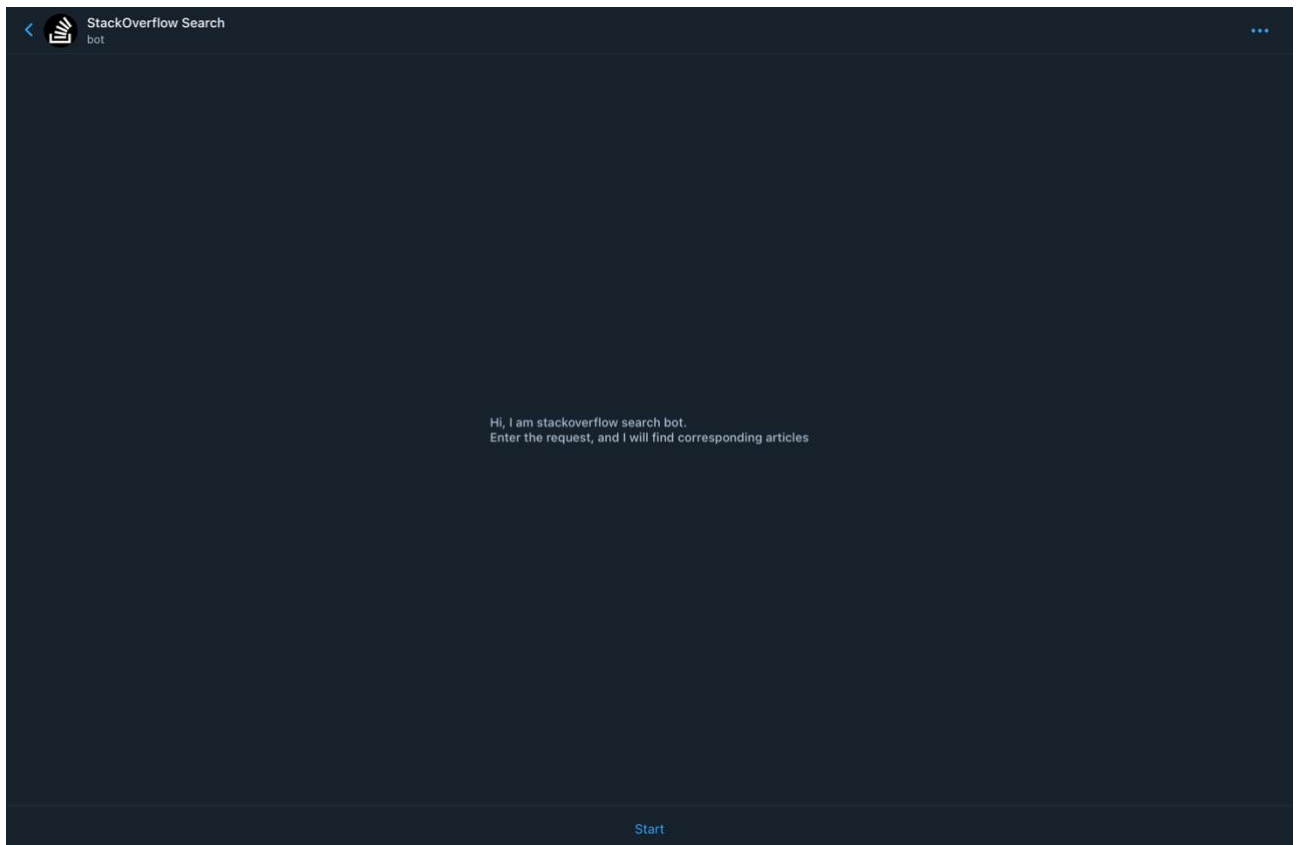


Рисунок 6.1 – Головне вікно бота

Після натиснення кнопки Start (що є аналогічним до написання боту команди /start, якщо користувач вже користувався ботом до цього) бот відправляє текстове повідомлення у вигляді привітання з коротким описом його основних можливостей та знизу з'являється меню з двох кнопок: Help та Search 🔍 (рисунок 6.2).

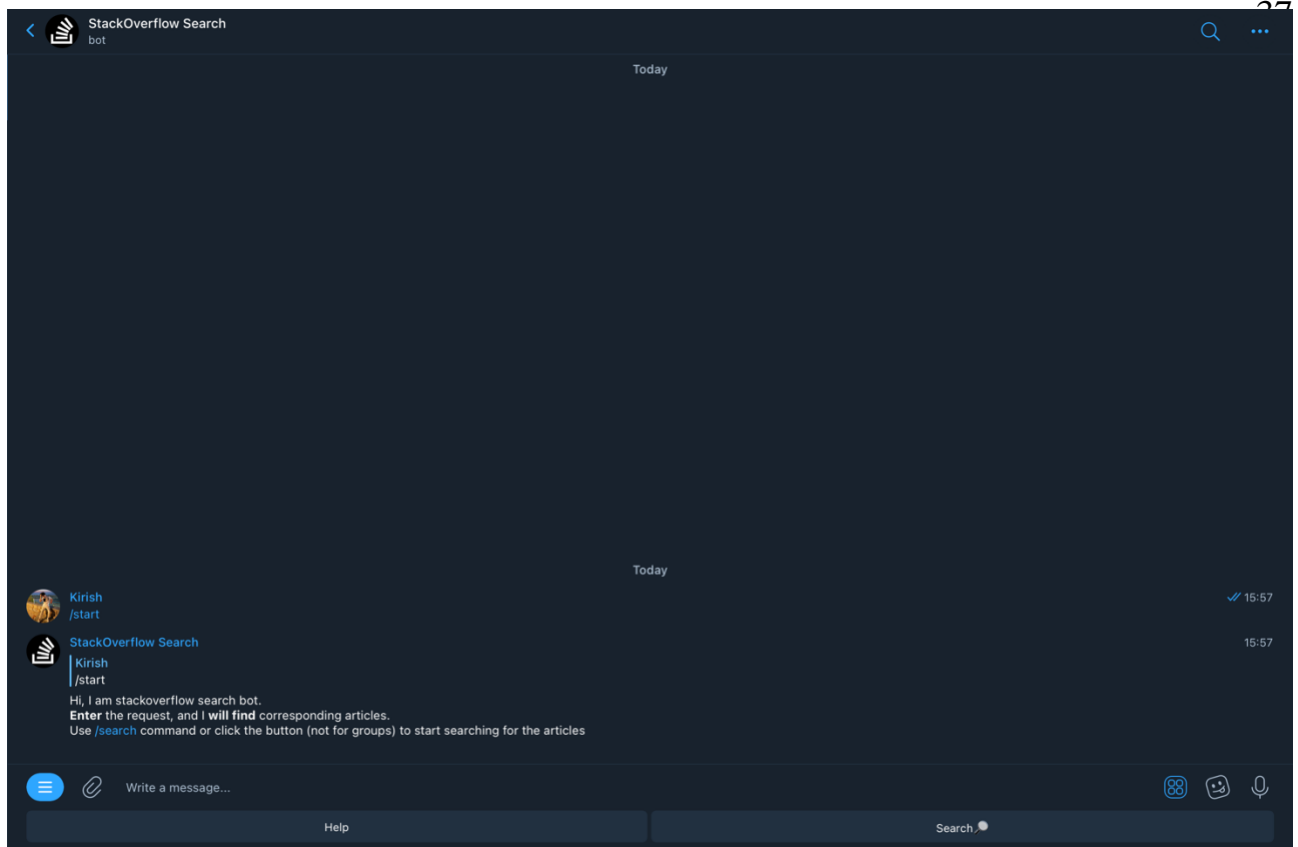


Рисунок 6.2 – Виклик команди /start бота

При натисненні на кнопку Search 🔍 або ж при виклику команди /search бот надсилає текстове повідомлення, в якому пропонується ввести пошуковий запит користувачу (рисунок 6.3).

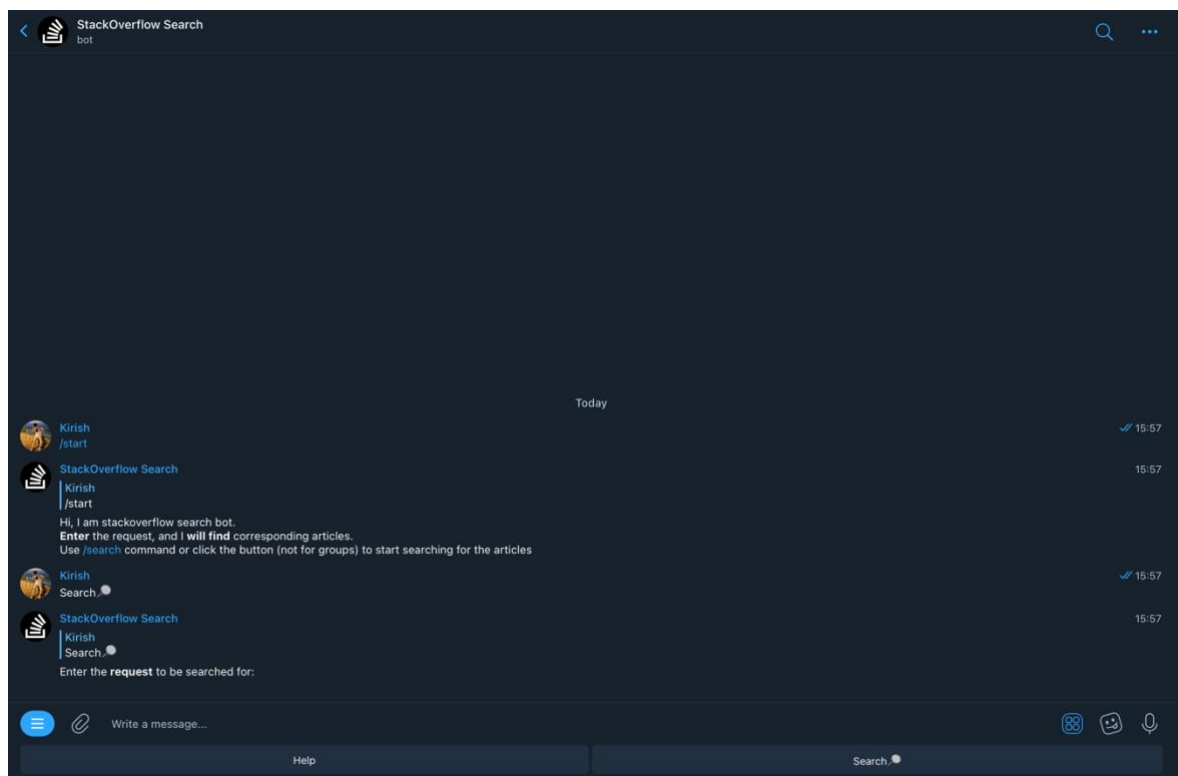


Рисунок 6.3 – Повідомлення про введення пошукового запиту

Далі для введення пошукового запиту необхідно ввести цей запит у вигляді текстового повідомлення боту. Після введення запиту бот надсилає текстове повідомлення про введення кількості статей (від 1 до 10) (рисунок 6.4).

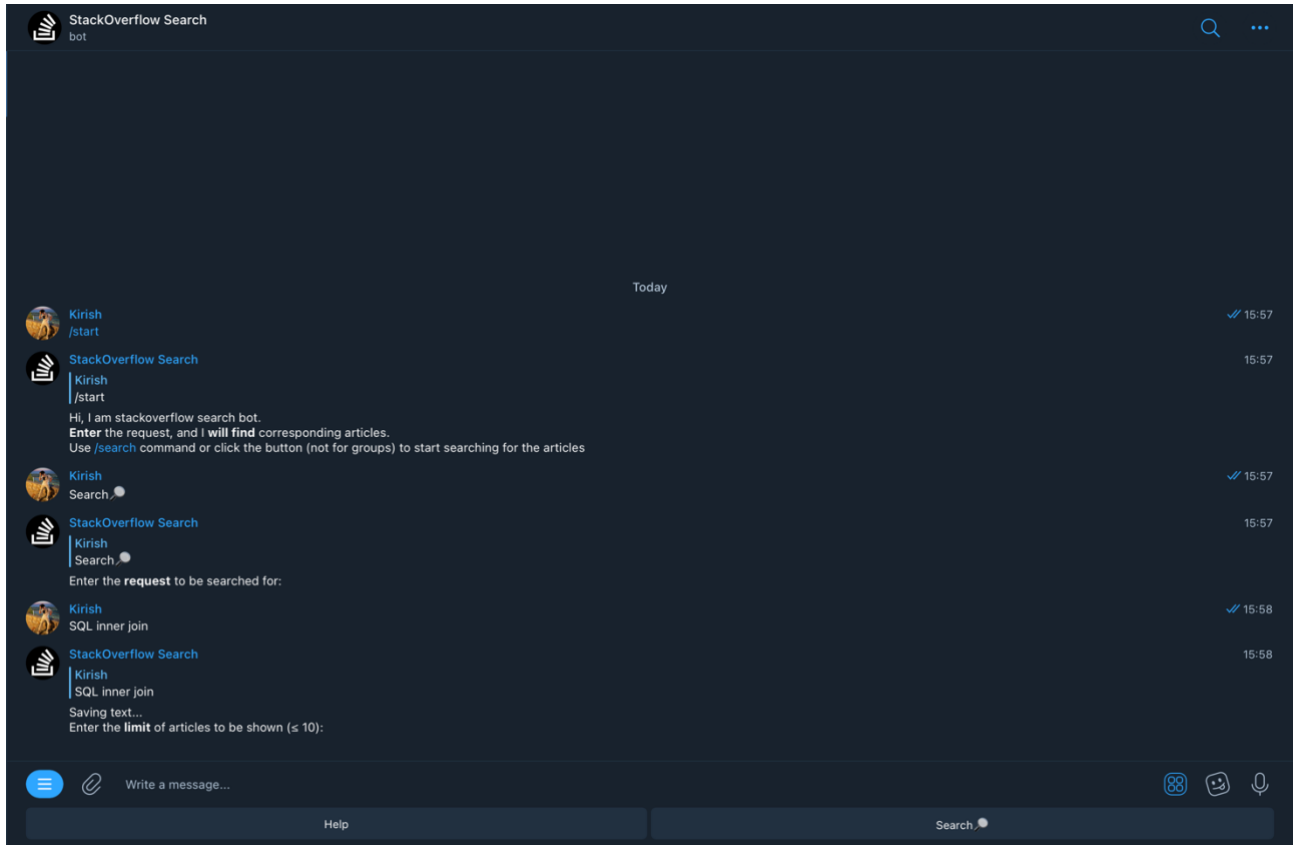


Рисунок 6.4 – Повідомлення про введення кількості статей

Потім для введення кількості статей необхідно ввести число у вигляді текстового повідомлення боту. Після введення кількості статей бот надсилає текстове повідомлення зі списком, що містить задану кількість статей (питання статті у вигляді гіперпосилання, косинусу подібності, тегів, та обробленого тексту питання статті) та часом, за який було здійснено пошук (рисунок 6.5).

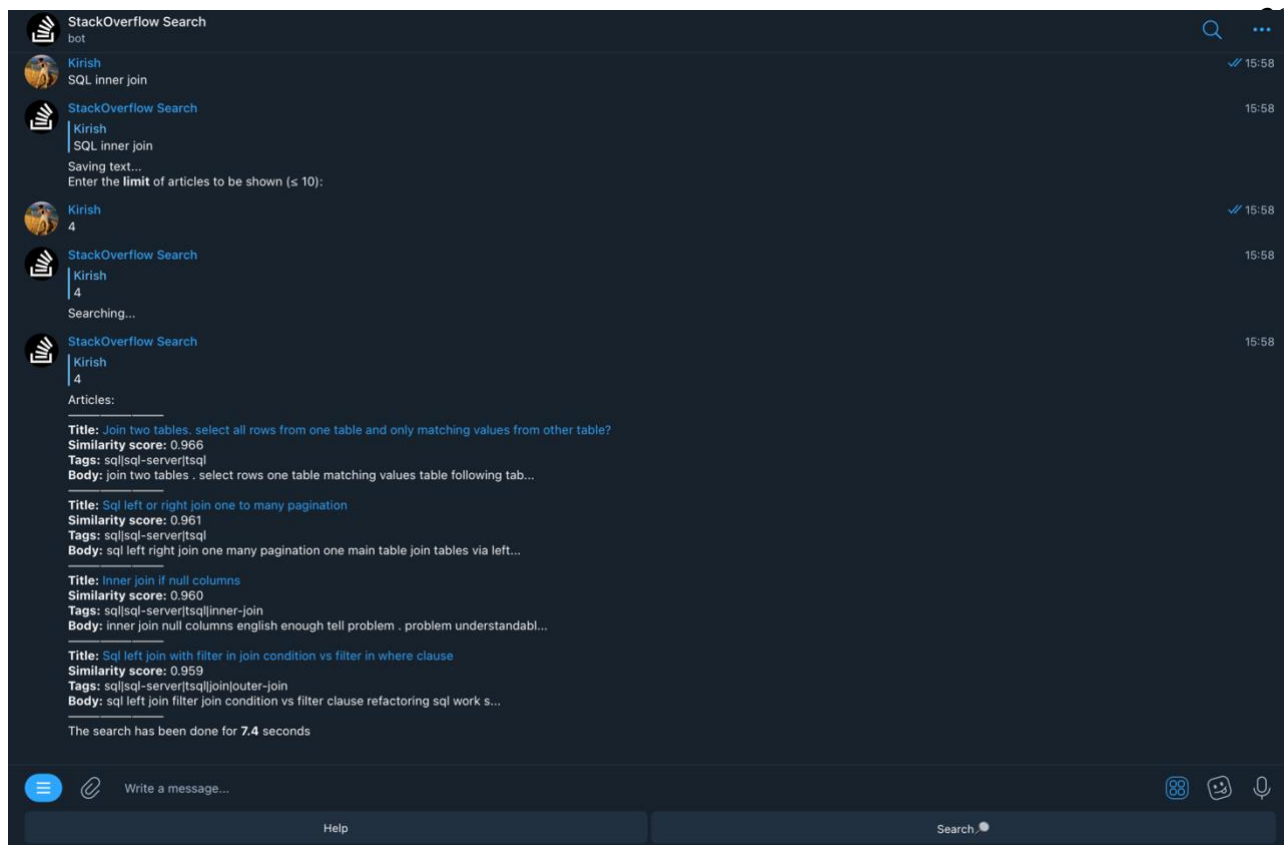


Рисунок 6.5 – Повідомлення зі знайденими статтями

При виклику команди /help або натисненні кнопки Help, бот надсилає текстові повідомлення з описом його команд (рисунок 6.6).

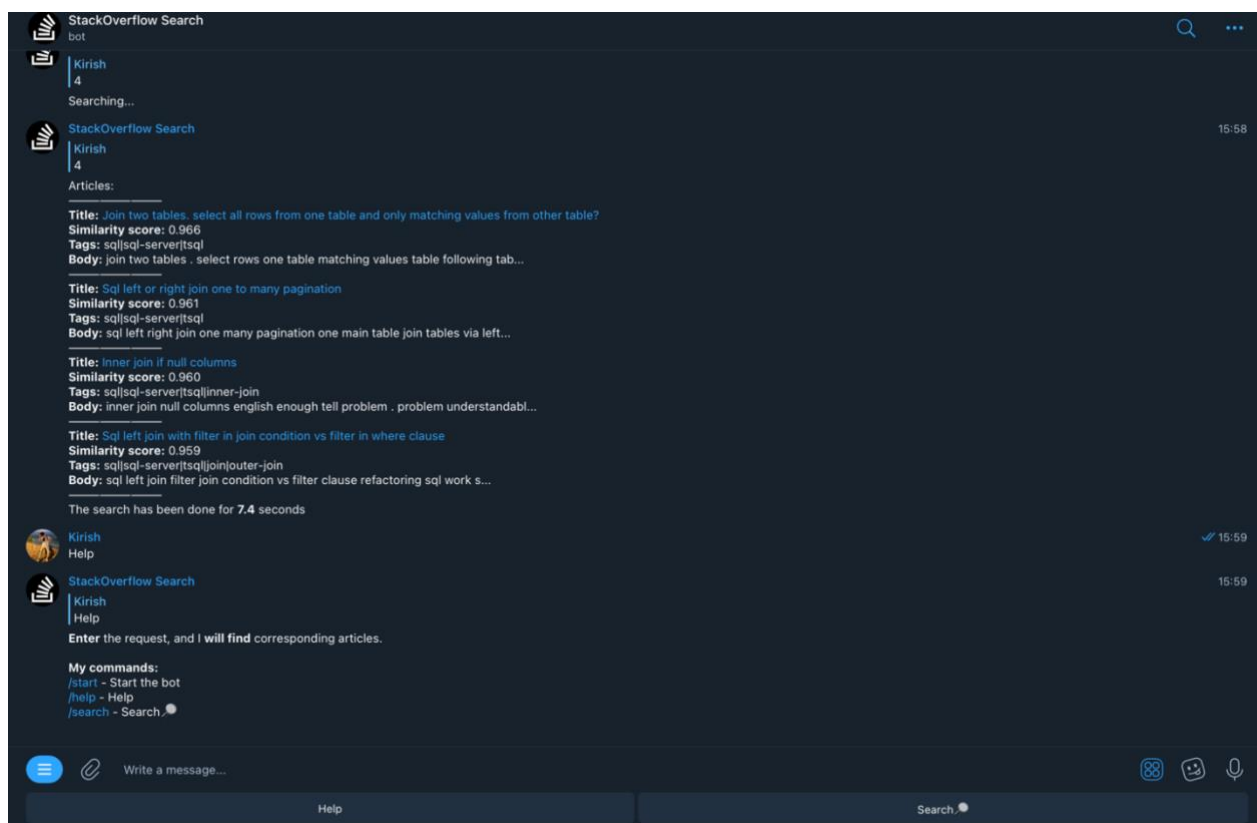


Рисунок 6.6 – Повідомлення з описом команд бота

6.2. Формат вхідних та вихідних даних

Вхідними даними є пошуковий запит у вигляді текстового повідомлення та кількість статей у вигляді текстового повідомлення. Вихідними даними є список статей, що містить задану кількість статей (питання статті у вигляді гіперпосилання, косинусу подібності, тегів, та обробленого тексту питання статті) та часом, за який було здійснено пошук у вигляді текстового повідомлення або ж відповідне повідомлення для одного з двох випадків: якщо введена кількість статей не є натуральним числом від 1 до 10 включно або не було знайдено відповідних статей для заданого пошукового запиту (коефіцієнт подібності, тобто косинус подібності, менше 0,8).

6.3. Системні вимоги програмного забезпечення

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

	Мінімальні	Рекомендовані
Операцій на система	Windows 10/Windows11/MacOS10.11+/IOS15+/ Android11+ (з останніми оновленнями)	Windows 10/Windows 11/MacOs12.3.1/IOS15.5/A ndroid12 (з останніми оновленнями)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
Оператив на пам'ять	4 GB RAM	8 GB RAM
Відеоадап тер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	
Дисплей	800x600	1024x768 або краще

Продовження таблиці 6.1

	Мінімальні	Рекомендовані
Прилади введення	Клавіатура, комп'ютерна миша	
Додаткове програмне забезпечення	Telegram Android 5.5+, Telegram Desktop 1.5.11+, Telegram IOS 5.2+, Telegram MacOS 4.9+	

ВИСНОВКИ

Отже, ми дослідили принцип роботи та архітектури моделей word2vec, GRU та мультиноміальної логістичної регресії й створили пошукову семантичну систему на основі даних сайту stackoverflow.com з використанням цих моделей машинного навчання, протестували її та реалізували у вигляді телеграм бота.

ПЕРЕЛІК ПОСИЛАНЬ

1. Вкладання слів : веб-сайт. URL:
https://uk.wikipedia.org/wiki/Вкладання_слів (дата звернення: 02.06.2022).
2. Word2vec : веб-сайт. URL: <https://uk.wikipedia.org/wiki/Word2vec> (дата
звернення: 02.06.2022).
3. Word2Vec Explained : веб-сайт. URL:
<https://towardsdatascience.com/word2vec-explained-49c52b4ccb71> (дата
звернення: 02.06.2022).
4. NLP 101: Word2Vec — Skip-gram and CBOW : веб-сайт. URL:
<https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314> (дата звернення: 02.06.2022).
5. Softmax : веб-сайт. URL: <https://uk.wikipedia.org/wiki/Softmax> (дата
звернення 02.06.2022).
6. Вентильний рекурентний вузол : веб-сайт. URL:
https://uk.wikipedia.org/wiki/Вентильний_рекурентний_вузол (дата
звернення: 02.06.2022).
7. Сигмоїда : веб-сайт. URL: <https://uk.wikipedia.org/wiki/Сигмоїда> (дата
звернення: 02.06.2022).
8. Гіперболічні функції : веб-сайт. URL:
<https://studfile.net/preview/2303121/page:3/> (дата звернення: 03.06.2022).
9. Multinomial logistic regression : веб-сайт. URL:
https://en.wikipedia.org/wiki/Multinomial_logistic_regression (дата
звернення: 03.06.2022).

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

Керівник: Головченко Максим Миколайович

«11» квітня 2022 р.

Виконавець:

Студенти: *Котков Тимур Максимович,*

Сідак Кирил Ігорович

«11» квітня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи на тему:

"Пошукова семантична система на основі даних сайту stackoverflow.com із використанням машинного навчання"

з дисципліни: «Основи програмування»

Київ 2022

1. *Мета:* Метою курсової роботи є розробка семантичної пошукової системи з використанням машинного навчання
2. *Дата початку роботи:* «11» квітня 2022 р.
3. *Дата закінчення роботи:* «12» червня 2022 р.
4. *Вимоги до програмного забезпечення.*

1) Функціональні вимоги:

- Можливість ввести пошуковий запит у вигляді текстового повідомлення (у чаті з телеграм ботом);
- Можливість ввести обмеження на кількість статей (результатів пошуку), у вигляді текстового повідомлення (у чаті з телеграм ботом);
- Можливість побачити оцінку рівня схожості кожної статті із заданим пошуковим запитом (у чаті з телеграм ботом);
- Можливість перейти за посиланням на статтю, що нас цікавить, з сайту stackoverflow.com (у чаті з телеграм ботом);
- Фільтрація вводу інформації (при введенні кількості статей боту неможливо ввести буквенний вираз, ненатуральне число, або число, яке не належить відрізьку [1, 10]).

2) Нефункціональні вимоги:

- Версії Telegram Android 5.5+, Telegram Desktop 1.5.11+, Telegram IOS 5.2+, Telegram MacOS 4.9+;
- Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.

5. *Стадії та етапи розробки:*

- 1) Об'єктно-орієнтований аналіз предметної області задачі (до __.__.2022 р.)
- 2) Об'єктно-орієнтоване проектування архітектури програмної системи (до __.__.2022 р.)
- 3) Розробка програмного забезпечення (до __.__.2022р.)
- 4) Тестування розробленої програми (до __.__.2022р.)
- 5) Розробка пояснювальної записки (до __.__.2022 р.).
- 6) Захист курсової роботи (до __.__.2022 р.).

6. *Порядок контролю та приймання.* Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

*Тексти програмного коду «Пошукова семантична система на основі даних сайту
stackoverflow.com із використанням машинного навчання»*

(Найменування програми(документа))

Електронний носій

(Вид носія даних)

62 арк, 34.8 Кб

(Обсяг програми (документа), арк., Кб)

студента групи ПП-14 I курсу

Коткова Т.М.,

студента групи ПП-11 I курсу

Сідака К.І.


```
app.py
from aiogram import executor
from bot_tg.loader import dp
import bot_tg.handlers
from bot_tg.utils.set_bot_commands import set_default_commands
from logger import get_logger

logger = get_logger()

async def on_startup(dispatcher):
    try:
        await set_default_commands(dispatcher)
    except Exception:
        logger.exception('An unexpected error occurred')

if __name__ == '__main__':
    executor.start_polling(dp, on_startup=on_startup, skip_updates=True)
```


```
config.py
from environs import Env
env = Env()
env.read_env()
BOT_TOKEN = env.str("BOT_TOKEN")
HEROKU_APP_NAME = env.str("HEROKU_APP_NAME")
ADMINS = env.list("ADMINS")
MAX_LIMIT = env.int("MAX_LIMIT")
```

```
handle_replies.py
from aiogram.utils.exceptions import BadRequest
from aiogram.types import Message

async def reply_to_message(message: Message, text, reply_markup=None):
    try:
        if reply_markup:
            await message.reply(text, reply_markup=reply_markup,
disable_web_page_preview=True)
        else:
            await message.reply(text, disable_web_page_preview=True)
    except BadRequest:
        if reply_markup:
            await message.answer(text, reply_markup=reply_markup,
disable_web_page_preview=True)
        else:
            await message.answer(text, disable_web_page_preview=True)
```

help.py

```
from aiogram.types import Message, ChatType
from .handle_replies import reply_to_message
from bot_tg.loader import dp
from aiogram.dispatcher.filters import Text
import aiogram.utils.markdown as fmt
```

```
@dp.message_handler(Text(equals="Help"), chat_type=ChatType.PRIVATE)
@dp.message_handler(commands='help')
async def bot_help(message: Message):
    text = fmt.text(
        fmt.text(f"{fmt.hbold('Enter')} the request, and I {fmt.hbold('will find')}
corresponding articles.\n"),
        fmt.text(f"{fmt.hbold('My commands:')}"),
        fmt.text(f"/start - Start the bot"),
        fmt.text(f"/help - Help"),
        fmt.text(f"/search - Search "),
        sep='\n'
    )
    await reply_to_message(message, text)
```

```

search.py
from aiogram.dispatcher import FSMContext
from bot_tg.loader import dp
from .handle_replies import reply_to_message
from search_engine.prediction_model.search_pipeline import search_results
from bot_tg.states.state_storage import States
from aiogram.types import Message, ChatType
from bot_tg.data.config import MAX_LIMIT
from aiogram.utils.markdown import hlink
import aiogram.utils.markdown as fmt
from aiogram.dispatcher.filters import Text
import time

```

```

@dp.message_handler(Text(equals='Search 🔍'), chat_type=ChatType.PRIVATE)
@dp.message_handler(commands="search")
async def enter_search_mode(message: Message):
    await States.input_text.set()
    await message.reply(fmt.text(f"Enter the {fmt.hbold('request')} to be searched for:"))

```

```

@dp.message_handler(state=States.input_text)
async def input_request(message: Message, state: FSMContext):
    await state.update_data(search_text=message.text)
    await States.input_limit.set()
    await message.reply(fmt.text(f"Saving text..."
                                   f"\nEnter the {fmt.hbold('limit')} of articles to be shown (\u2264
{MAX_LIMIT}):"))

```

```

@dp.message_handler(state=States.input_limit)
async def input_limit(message: Message, state: FSMContext):
    text = message.text
    if text.isdecimal() and 0 < int(text) <= MAX_LIMIT:
        await message.reply("Searching...")
        num = int(text)
        user_data = await state.get_data()
        await state.finish()
        search_text = user_data['search_text']
        t_0 = time.time()
        articles = search_results(search_text, num)
        if articles:
            reply_text = "Articles:\n—————\n"
            for i in range(num):
                reply_text += fmt.text(
                    fmt.text(f'{fmt.hbold("Title:")} {hlink(articles[i]["title"].capitalize(),
articles[i]["url"])}'),
                    fmt.text(f'{fmt.hbold("Similarity score:")}
{articles[i]["similarity_score"]}')},
                    fmt.text(f'{fmt.hbold("Tags:")} {articles[i]["tags"]}')},
                    fmt.text(f'{fmt.hbold("Body:")} {articles[i]["body"][:75]}...\n—————
—————\n'),
                    sep='\n'
                )
            reply_text += fmt.text(f"The search has been done for
{fmt.hbold(round(time.time() - t_0, 1))} seconds\n\n")
        else:
            reply_text = fmt.text(f"No corresponding articles were found for such request:
"{fmt.hbold(search_text)}")

```

```
else:
    reply_text = fmt.text(f"{fmt.hbold('Incorrect input')}. "
                          f"Only non-negative integers are allowed which are \u2264"
                          f"{MAX_LIMIT}."
                          f"\nTry again:")
    await reply_to_message(message, reply_text)
```

```

start.py
from .handle_replies import reply_to_message
from bot_tg.loader import dp
from aiogram.types import Message, ChatType
from logger import get_logger
from bot_tg.keyboards import main_keyboard
import aiogram.utils.markdown as fmt

logger = get_logger()

@dp.message_handler(commands='start')
async def start(message: Message):
    text = fmt.text(f"Hi, I am stackoverflow search bot.\n"
                    f"{fmt.hbold('Enter')} the request, and I {fmt.hbold('will find')} "
                    f"corresponding articles."
                    f"\nUse /search command or click the button (not for groups) to start"
                    f"searching for the articles")
    if message.chat.type in [ChatType.SUPERGROUP, ChatType.GROUP]:
        await reply_to_message(message, text)
    else:
        await reply_to_message(message, text, main_keyboard)

```



```
search_keyboard.py
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

main_keyboard = ReplyKeyboardMarkup([
    [
        KeyboardButton("Help"),
        KeyboardButton("Search 🔍")
    ]
], resize_keyboard=True)
```

state_storage.py

```
from aiogram.dispatcher.filters.state import StatesGroup, State
```

```
class States(StatesGroup):
```

```
    input_text = State()
```

```
    input_limit = State()
```

```
set_bot_commands.py  
from aiogram.types import BotCommand  
  
async def set_default_commands(dp):  
    await dp.bot.set_my_commands(  
        [  
            BotCommand("start", "Start the bot"),  
            BotCommand("help", "Help"),  
            BotCommand("search", "Search 🔍")  
        ]  
    )
```

loader.py

```
from aiogram import Bot, Dispatcher
```

```
from aiogram.types import ParseMode
```

```
from aiogram.contrib.fsm_storage.memory import MemoryStorage
```

```
from bot_tg.data import config
```

```
bot = Bot(token=config.BOT_TOKEN, parse_mode=ParseMode.HTML)
```

```
dp = Dispatcher(bot, storage=MemoryStorage())
```

```
encode_tags.py
```

```
import pandas as pd
```

```
class Encoder:
```

```
    def __init__(self, df, max_freq: int):
```

```
        self.df = df
```

```
        self.max_freq = max_freq
```

```
    def keys_from_tags(self):
```

```
        df = self.df.copy(deep=True)
```

```
        df.tags = [i.split('|') for i in df['tags'].to_list()]
```

```
        tags_dict = { }
```

```
        tags_freq = { }
```

```
        ind = 0
```

```
        for i in range(len(df.tags)):

```

```
            for j in range(len(df.tags[i])):

```

```
                if not df.tags[i][j] in list(tags_dict.keys()):

```

```
                    tags_dict[df.tags[i][j]] = ind

```

```
                    tags_freq[df.tags[i][j]] = 0

```

```
                    ind += 1

```

```
            else:

```

```
                tags_freq[df.tags[i][j]] += 1

```

```
tags_freq = dict(sorted(tags_freq.items(), key=lambda x: x[1], reverse=True))
```

```
keys = list(tags_freq.keys())[:self.max_freq]
```

```
values = [tags_dict[i] for i in keys]
```

```
df_tags = pd.DataFrame()
```

```
df_tags['tag'] = keys
```

```
df_tags['code'] = values
return df_tags
```

```
def encode_tags(self):
    df = self.df.copy(deep=True)
    new_enc_tags = []
    tag_keys = self.keys_from_tags()
    for i in df.iterrows():
        temp_ind = []
        for j in i[1].tags.split('|'):
            try:
                code = tag_keys.loc[tag_keys['tag'] == j].code.values[0]
                temp_ind.append(code)
            except:
                continue
        new_enc_tags.append(temp_ind)
    df.tags = ['|'.join([str(j) for j in i]) for i in new_enc_tags]

    return df
```

```
class Decoder:
    def __init__(self, df, keys):
        self.df = df
        self.keys = keys

    def decode_tags(self):
        df = self.df.copy(deep=True)
        new_dec_tags = []
        for i in df.iterrows():
```

```
temp_tags = []
for j in i[1].tags.split('|'):
    try:
        tag = self.keys.loc[self.keys['code'] == int(j)].tag.values[0]
        temp_tags.append(tag)
    except:
        continue
new_dec_tags.append(temp_tags)
df.tags = ['|'.join([str(j) for j in i]) for i in new_dec_tags]

return df
```

```
save_encoded_tags.py
from encode_tags import Encoder, Decoder
import pandas as pd
from environs import Env
from logger import get_logger

env = Env()
env.read_env()
logger = get_logger(handle_errors=False)

DATA_PATH = env.str("DATA_PATH")

dataframe = pd.read_csv(DATA_PATH + 'categories_data.csv', engine='pyarrow')

encoder = Encoder(dataframe, 2000)

encoded_df = encoder.encode_tags()
encoded_df.to_csv(DATA_PATH + 'enc_dataset.csv', index=False)
logger.info('File enc_dataset.csv was saved')

df_tags_keys = encoder.keys_from_tags()
df_tags_keys.to_csv(DATA_PATH + 'tags_keys.csv', index=False)
logger.info('File tags_keys.csv was saved')

decoder = Decoder(encoded_df, df_tags_keys)
decoded_df = decoder.decode_tags()
decoded_df.to_csv(DATA_PATH + 'dec_dataset.csv', index=False)
logger.info('File dec_dataset.csv was saved')
```



```
categories_separation.py
```

```
import pickle
```

```
import pandas as pd
```

```
from environs import Env
```

```
env = Env()
```

```
env.read_env()
```

```
DATA_PATH = env.str("DATA_PATH")
```

```
FINAL_DATA = env.str("FINAL_DATA")
```

```
MODELS = env.str("MODELS")
```

```
df_keys = pd.read_csv(DATA_PATH + 'tags_keys.csv', engine='pyarrow')
```

```
main_data = pd.read_parquet(DATA_PATH + 'final_data.gz', engine='pyarrow')
```

```
def encode_tags(list_of_tags: list, keys: pd.DataFrame):
```

```
    new_l = []
```

```
    for k in list_of_tags:
```

```
        if k in keys.tag.values:
```

```
            new_l.append(keys[keys.tag == k].code.values[0])
```

```
    return new_l
```

```
model = pickle.load(open(MODELS + 'model_tags.pkl', 'rb'))
```

```
mlb = pickle.load(open(MODELS + 'mlb.pkl', 'rb'))
```

```
def predict_category(list_of_tags):
```

```
    t = encode_tags(list_of_tags, df_keys)
```

```

tags = mlb.transform([t, []])
full_res = model.predict_proba(tags)[0]
res = model.predict(tags)[0]
return list(full_res), res

```

```

def get_df(data):
    dict_of_df = {}
    for i in data.iterrows():
        spl_tags = i[1].tags.split('|')
        full, category = predict_category(spl_tags)
        if max(full) > 0.95:
            try:
                dict_of_df[category] = pd.concat([dict_of_df[category], i[1].to_frame().T])
            except:
                dict_of_df[category] = i[1].to_frame().T

        else:
            for j in [model.classes_[full.index(c)] for c in sorted(full)[-3:]]:
                try:
                    dict_of_df[j] = pd.concat([dict_of_df[j], i[1].to_frame().T])
                except:
                    dict_of_df[j] = i[1].to_frame().T

    return dict_of_df

```

```

def save_dataset(key_cat, dataframe, header=False, mode='a'):
    if "html" in key_cat:
        key_cat = 'html_css'

```

```
dataframe.to_csv(f'{DATA_PATH}dbc/{key_cat}.csv', index=False, mode=mode,
header=header)
```

```
if __name__ == '__main__':
    for i in range((len(main_data) // 10000) + 1):
        df_sample = main_data[i * 10000: (i + 1) * 10000]
        df_dict = get_df(df_sample)
        for key, df_sample in df_dict.items():
            if i == 0:
                save_dataset(key, df_sample, True, 'w')
            else:
                save_dataset(key, df_sample, False, 'a')
        print(f'From {i * 10000} to {(i + 1) * 10000} was processed!')
```

```

train_categories.py
import pickle
import warnings
import pandas as pd
from environs import Env
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.preprocessing import MultiLabelBinarizer
from logger import get_logger

env = Env()
env.read_env()
logger = get_logger()

warnings.filterwarnings('ignore')

DATA_PATH = env.str("DATA_PATH")
MODELS = env.str("MODELS")

def split_tags(string):
    if string:
        return [int(i) for i in string.split('|')]

encoded_df = pd.read_csv(DATA_PATH + 'enc_dataset.csv',
engine='pyarrow').sample(frac=1)
encoded_df.tags = encoded_df.tags.apply(split_tags)

```

```

encoded_df.dropna(inplace=True, axis=0)
tags = encoded_df.tags

multilabel_binarizer = MultiLabelBinarizer()
y_bin = multilabel_binarizer.fit_transform(encoded_df.tags)
pickle.dump(multilabel_binarizer, open(MODELS + 'mlb.pkl', 'wb'))

X_train_tags, X_test_tags, y_train_tags, y_test_tags = train_test_split(y_bin,
encoded_df['category'], test_size=0.2,
                                random_state=0)

# region tags model
logreg_tags = LogisticRegression(n_jobs=1, C=1e5)
logreg_tags = logreg_tags.fit(X_train_tags, y_train_tags)
y_pred = logreg_tags.predict(X_test_tags)
pickle.dump(logreg_tags, open(MODELS + 'model_tags.pkl', 'wb'))
print('accuracy %s' % accuracy_score(y_pred, y_test_tags))

print(classification_report(y_test_tags, y_pred))
# endregion

```

```
concatenate_datasets.py
import pandas as pd
from environs import Env

env = Env()
env.read_env()
DATA_PATH = env.str("DATA_PATH")

preprocessed_data1 = pd.read_parquet(f'{DATA_PATH}out1.gzip')
preprocessed_data2 = pd.read_parquet(f'{DATA_PATH}out2.gzip')
preprocessed_data3 = pd.read_parquet(f'{DATA_PATH}out3.gzip')
preprocessed_data4 = pd.read_parquet(f'{DATA_PATH}out4.gzip')
preprocessed_data5 = pd.read_parquet(f'{DATA_PATH}out5.gzip')

main_data1 = pd.concat([preprocessed_data1, preprocessed_data2])
main_data2 = pd.concat([preprocessed_data3, preprocessed_data4,
preprocessed_data5])

main_data1.to_parquet(f'{DATA_PATH}raw_data1.gzip', compression='gzip',
index=False)
main_data2.to_parquet(f'{DATA_PATH}raw_data2.gzip', compression='gzip',
index=False)
```

```

getting_data.py
from google.cloud import bigquery
import os
from environs import Env
import gdown
from logger import get_logger

logger = get_logger()

env = Env()
env.read_env()
URL = env.str("URL")
OUT_FILE = env.str("OUT_FILE")
DATE_SIZE = env.str("DATE_SIZE")

gdown.download(URL, OUT_FILE, quiet=False)
out_file = os.path.abspath(OUT_FILE)

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = out_file
client = bigquery.Client()

def build_query(min_id, max_id):
    query = f"""
        SELECT
            q.id, q.title, q.body, q.tags, a.body as answers, a.score
        FROM
            `bigquery-public-data.stackoverflow.posts_questions` AS q
        INNER JOIN
            `bigquery-public-data.stackoverflow.posts_answers` AS a
    """

```

```

ON
    q.id = a.parent_id
WHERE
    q.id BETWEEN {int(min_id)} AND {int(max_id)}
    AND q.view_count > 250
    AND q.accepted_answer_id IS NOT NULL
"""

return query

# saving data in csv by 5m rows (with filter 5m rows ~= 850k rows)
for i in range(6):
    query = build_query(35e6 + 5e6 * i, 35e6 + 5e6 * (i + 1))
    try:
        dataframe = (
            client.query(query).result().to_dataframe()
        )
        dataframe.to_parquet(f"out{i + 1}.gzip", compression='gzip', index=False)
        logger.info(f"Data (5m items (from {35 + 5 * i}m to {35 + 5 * (i + 1)}m)) "
                    f"was successfully downloaded and converted to CSV-file")
    except Exception as ex:
        logger.exception("An error occurred while pulling data from the database")

```



```

clean_tags.py
import pandas as pd
from environs import Env
from logger import get_logger

env = Env()
env.read_env()

DATA_PATH = env.str("DATA_PATH")

logger = get_logger(handle_errors=False)

def code_from_key(keys_data, key):
    if key in keys_data.tag.values:
        return keys_data[keys_data.tag == key].code.values[0]

def delete_elem_from_tags(df, condition, tag_to_delete, mode='num'):
    ind = 0

    def delete_tag(index, cond, ttd):
        for i in df.tags.values:
            if not i == "":
                if mode == 'num':
                    i = [int(j) for j in i.split('|')]
                elif mode == 'str':
                    i = [str(j) for j in i.split('|')]
                if df.category[index] == condition:
                    if cond in i and ttd in i:

```

```

        l_buff = i[:,]
        del l_buff[l_buff.index(ttd)]
        df.at[index, "tags"] = '|'.join([str(j) for j in l_buff])
    index += 1

if mode == 'num':
    cond_key = code_from_key(df_keys, condition)
    ttd_key = code_from_key(df_keys, tag_to_delete)
    delete_tag(ind, cond_key, ttd_key)
elif mode == 'str':
    delete_tag(ind, condition, tag_to_delete)

df_dec = pd.read_csv(DATA_PATH + 'dec_dataset.csv', engine='pyarrow')
df_enc = pd.read_csv(DATA_PATH + 'enc_dataset.csv', engine='pyarrow')
df_keys = pd.read_csv(DATA_PATH + 'tags_keys.csv', engine='pyarrow')

delete_elem_from_tags(df_enc, 'asp.net', 'c#')
delete_elem_from_tags(df_enc, '.net', 'c#')
delete_elem_from_tags(df_enc, 'c#', 'asp.net')
delete_elem_from_tags(df_enc, 'c#', '.net')
delete_elem_from_tags(df_enc, 'c++', 'c')
delete_elem_from_tags(df_enc, 'c', 'c++')
df_enc.dropna(inplace=True, axis=0)
df_enc.to_csv(DATA_PATH + 'enc_dataset.csv', index=False)
logger.info("df_enc dataset was modified")
delete_elem_from_tags(df_dec, 'asp.net', 'c#', 'str')
delete_elem_from_tags(df_dec, 'c#', 'asp.net', 'str')
delete_elem_from_tags(df_dec, 'c#', '.net', 'str')
delete_elem_from_tags(df_dec, '.net', 'c#', 'str')

```

```
delete_elem_from_tags(df_dec, 'c++', 'c', 'str')
delete_elem_from_tags(df_dec, 'c', 'c++', 'str')
df_dec.dropna(inplace=True, axis=0)
df_dec.to_csv(DATA_PATH + 'dec_dataset.csv', index=False)
logger.info("df_dec dataset was modified")
```

```
create_datasets.py
from data_from_site import CategoryDataset

categories = ['c%23', 'c%2b%2b', 'javascript', 'python', 'java', 'php', 'android', 'api'
              'jquery', 'iOS', 'database', 'r', 'c', 'asp.net', 'ruby', '.net', 'django', 'angularjs',
              'reactjs',
              'regex', 'data-science', 'linux', 'spring', 'windows', 'git', 'macos', 'visual-studio',
              'scala',
              'perl', 'api', 'algorithm', 'excel', 'html/css']

c = CategoryDataset(categories)
c.create_and_save_dataset()
```

```

data_from_site.py
import requests
import pandas as pd
from bs4 import BeautifulSoup
from search_engine.processing_data.normalize_functions import preprocess_text
from stackapi import StackAPI
from environs import Env
from stackapi.stackapi import StackAPIError
from logger import get_logger

env = Env()
env.read_env()
logger = get_logger()

DATA_PATH = env.str("DATA_PATH")
SITE = StackAPI(name='stackoverflow')
categories_dict = {'c%23': 'c#', 'c%2b%2b': 'c++'}

class CategoryDataset:
    def __init__(self, categories):
        self.df = pd.DataFrame()
        self.df['article_index'] = None
        self.df['title'] = None
        self.df['tags'] = None
        self.df['category'] = None
        self.categories = categories

    def get_ids(self, search_text, c_type, search_size):
        search_query = '+'.join(search_text.split(' '))

```

```

id_list = []
categories_list = []
for i in range(1, int((search_size / 50)) + 1):
    if c_type == 'tag':
        result_html = requests.get(

f'https://stackoverflow.com/questions/tagged/{search_text}?tab=votes&page={i}&pa
gesize=50') \
        .content
    else:
        result_html = requests.get(

f'https://stackoverflow.com/search?page={i}&tab=Relevance&pagesize=50&q={sear
ch_query}').content
    soup = BeautifulSoup(result_html, "html.parser")
    ids = soup.find_all('div', class_='s-post-summary js-post-summary')
    category = categories_dict[search_text] if search_text in
categories_dict.keys() else search_text
    if ids:
        for row in ids:
            id_list.append(int(row.attrs['data-post-id']))
            categories_list.append(category)
    else:
        raise TimeoutError
    self.df = pd.concat([self.df, pd.DataFrame(pd.DataFrame.from_dict(
        {"article_index": id_list, "title": None, "tags": None, "category":
categories_list}))), ignore_index=True)
    return self.df.shape[0] - len(id_list)

def filter_values(self, search_size, start_index):

```

```

for i in range(int(search_size / 20)):
    try:
        qs = SITE.fetch('questions',
                        ids=self.df.article_index.values[start_index + i * 20: start_index
+ (i + 1) * 20])
    except StackAPIError as ex:
        if ex.message == 'no method found with this name':
            self.filter_values(search_size - 500, start_index)
        elif 'too many requests from this IP, more requests available' in ex.message:
            input("Change location in VPN (then enter ok): ")
            self.filter_values(search_size, start_index)
        else:
            logger.exception('An unexpected error occurred')

    for row in qs['items']:
        s = row['title']
        s = s.replace('&#39;', '"')
        self.df.loc[self.df['article_index'] == row['question_id'], 'tags'] =
|.join(row['tags'])
        self.df.loc[self.df['article_index'] == row['question_id'], 'title'] =
preprocess_text(s)

def create_and_save_dataset(self):
    i = 0
    while i < len(self.categories):
        c_type = 'tag' if ' ' not in self.categories[i] else 'query'
        size = 2500 if c_type == 'tag' else 500
        try:
            start_index = self.get_ids(self.categories[i], c_type, size)
            self.filter_values(size, start_index)

```

```

df = self.df.dropna()
df.drop(columns='article_index', inplace=True)
df.drop_duplicates(inplace=True)
if i == 0:
    df.to_csv(DATA_PATH + 'categories_data.csv', index=False, mode='a',
header=True)
else:
    df.to_csv(DATA_PATH + 'categories_data.csv', index=False, mode='a',
header=False)
    logger.info(f'Category {self.categories[i]} was successfully added to the
dataset')
    print(df.tail(10))
except TimeoutError:
    print('Please enter captcha for ', self.categories[i])
    input('Input "ok": ')
    continue
i += 1

```



```

gru_model.py
import spacy

from keras.metrics import BinaryAccuracy, Precision, Recall
from keras.models import Sequential
from keras.layers import Dense, Embedding, Dropout, GRU
from keras.layers import BatchNormalization
from environs import Env

import numpy as np
from logger import get_logger
import keras.backend as K

def f1_metric(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2 * (precision * recall) / (precision + recall + K.epsilon())
    return f1_val

logger = get_logger()

EN = spacy.load('en_core_web_sm')

env = Env()
env.read_env()
TRAIN_TEST_PATH = env.str("TRAIN_TEST_PATH")
VOCAB_SIZE = env.int("VOCAB_SIZE")

```

```
W2V_SIZE = 300
```

```
MAX_SEQUENCE_LENGTH = 300
```

```
EMBEDDING_DIM = 300
```

```
X_train_padded = np.load(TRAIN_TEST_PATH + 'x_train_padded.npy',
allow_pickle=True)
```

```
y_train = np.load(TRAIN_TEST_PATH + 'y_train.npy', allow_pickle=True)
```

```
X_test_padded = np.load(TRAIN_TEST_PATH + 'x_test_padded.npy',
allow_pickle=True)
```

```
y_test = np.load(TRAIN_TEST_PATH + 'y_test.npy', allow_pickle=True)
```

```
embedding_matrix = np.load(TRAIN_TEST_PATH + 'embedding_matrix.npy',
allow_pickle=True)
```

```
result_metrics = [
    BinaryAccuracy(name='accuracy'),
    Precision(name='precision'),
    Recall(name='recall'),
    f1_metric
]
```

```
model = Sequential()
```

```
model.add(
```

```
    Embedding(VOCAB_SIZE + 1, W2V_SIZE, weights=[embedding_matrix],
input_length=MAX_SEQUENCE_LENGTH, trainable=False))
```

```
model.add(GRU(300, activation='relu', kernel_initializer='he_normal'))
```

```
model.add(Dense(400, activation='relu', kernel_initializer="he_normal"))
```

```
model.add(Dropout(0.5))
```

```
model.add(BatchNormalization())
```

```
model.add(Dense(150, activation='relu'))
```

```
model.add(Dense(1000, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy',  
              metrics=result_metrics,  
              optimizer="adam")  
model.summary()
```

```
# Train Model
```

```
BATCH_SIZE = 1024  
logger.info("Start fitting model")  
history = model.fit(x=X_train_padded, y=y_train,  
                    batch_size=BATCH_SIZE,  
                    epochs=20,  
                    validation_split=0.1,  
                    verbose=2)
```

```
logger.info("End of fitting model")
```

```
model.save_weights("stack.h5")  
model_json = model.to_json()  
json_file = open("stack.json", "w")  
json_file.write(model_json)  
json_file.close()
```

```

tag_predictor.py
import pandas as pd
import spacy
import warnings
import pickle
from environs import Env

env = Env()
env.read_env()
FINAL_DATA = env.str("FINAL_DATA")
TRAIN_TEST_PATH = env.str("TRAIN_TEST_PATH")

EN = spacy.load('en_core_web_sm')
warnings.filterwarnings('ignore')

preprocessed_data = pd.read_csv(FINAL_DATA)

preprocessed_data.tags = preprocessed_data.tags.apply(lambda x: x.split('|')) #
Making the list of tags
tag_freq_dict = {}
for tags in preprocessed_data.tags:
    for tag in tags:
        if tag not in tag_freq_dict:
            tag_freq_dict[tag] = 0
        else:
            tag_freq_dict[tag] += 1

# Get most common tags
tags_to_use = 1000

```

```
tag_freq_dict_sorted = dict(sorted(tag_freq_dict.items(), key=lambda x: x[1],
reverse=True))
final_tags = list(tag_freq_dict_sorted.keys())[:tags_to_use]

# Change tag data to only for final_tags
final_tag_data = []
for tags in preprocessed_data.tags:
    temp = []
    for tag in tags:
        if tag in final_tags:
            temp.append(tag)
    final_tag_data.append(temp)

with open(TRAIN_TEST_PATH + "tokenizer.txt", "wb") as fp: # Pickling
    pickle.dump(final_tag_data, fp)
```



```
w2v_model.build_vocab(documents)
words = list(w2v_model.wv.key_to_index.keys())
vocab_size = len(words)
print("Vocab size", vocab_size)

# Train Word Embeddings
w2v_model.train(documents, total_examples=len(documents),
epochs=W2V_EPOCH)
w2v_model.save(MODELS_PATH + '/SO_word2vec_embeddings.bin')
logger.info("END embedding created")
```

```

prepare_data.py

from environs import Env
from logger import get_logger
import pandas as pd
from keras.preprocessing.text import Tokenizer
import pickle

logger = get_logger(handle_errors=False)

env = Env()
env.read_env()
TRAIN_TEST_PATH = env.str("TRAIN_TEST_PATH")
W2V_MODEL_PATH = env.str("W2V_MODEL_PATH")
FINAL_DATA = env.str("FINAL_DATA")

W2V_SIZE = 300
MAX_SEQUENCE_LENGTH = 300
EMBEDDING_DIM = 300

preprocessed_data = pd.read_csv(FINAL_DATA)

tokenizer = Tokenizer()
tokenizer.fit_on_texts(preprocessed_data.post_corpus)

with open(TRAIN_TEST_PATH + "tokenizer.txt", 'wb') as tokenizer_file:
    pickle.dump(tokenizer, tokenizer_file)

word_index = tokenizer.word_index
vocab_size = len(word_index)
print(f'Found {len(word_index)} unique tokens.')

```



```

if __name__ == '__main__':

    from sklearn.preprocessing import MultiLabelBinarizer
    import numpy as np
    from search_engine.prediction_model.tag_predictor.tag_predictor import
final_tag_data
    from sklearn.model_selection import train_test_split
    from gensim.models import Word2Vec
    from keras.preprocessing.sequence import pad_sequences

    tag_encoder = MultiLabelBinarizer()
    tags_encoded = tag_encoder.fit_transform(final_tag_data)

    data = pd.DataFrame(columns=['corpus_code_combined'])
    data["corpus_code_combined"] = preprocessed_data.post_corpus

    w2v_model = Word2Vec.load(W2V_MODEL_PATH)

    # Split into train and test set
    X_train, X_test, y_train, y_test =
train_test_split(np.array(data.corpus_code_combined),
                  tags_encoded, test_size=0.2, random_state=42)

    print("TRAIN size:", len(X_train))
    print("TEST size:", len(X_test))

    # Convert the data to padded sequences
    X_train_padded = tokenizer.texts_to_sequences(X_train)

```

```

X_train_padded = pad_sequences(X_train_padded,
maxlen=MAX_SEQUENCE_LENGTH)

print('Shape of data tensor:', X_train_padded.shape)

X_test_padded = tokenizer.texts_to_sequences(X_test)
X_test_padded = pad_sequences(X_test_padded,
maxlen=MAX_SEQUENCE_LENGTH)

print('Shape of data tensor:', X_test_padded.shape)

# Embedding matrix for the embedding layer
embedding_matrix = np.zeros((vocab_size + 1, W2V_SIZE))
for word, i in tokenizer.word_index.items():
    if word in w2v_model.wv:
        embedding_matrix[i] = w2v_model.wv[word]
print(embedding_matrix.shape)

np.savez_compressed(TRAIN_TEST_PATH + 'x_train.npz', X_train)
np.savez_compressed(TRAIN_TEST_PATH + 'x_test.npz', X_test)
np.savez_compressed(TRAIN_TEST_PATH + 'y_train.npz', y_train)
np.savez_compressed(TRAIN_TEST_PATH + 'y_test.npz', y_test)
np.savez_compressed(TRAIN_TEST_PATH + 'x_train_padded.npz',
X_train_padded)
np.savez_compressed(TRAIN_TEST_PATH + 'x_test_padded.npz',
X_test_padded)
np.savez_compressed(TRAIN_TEST_PATH + 'embedding_matrix.npz',
embedding_matrix)

logger.info("All datasets were saved")

```

```
search_pipeline.py
import pandas as pd
from keras.metrics import Precision, Recall
import keras.losses
import numpy as np
from search_engine.processing_data.normalize_functions import preprocess_text
import gensim
import tensorflow as tf
from keras.models import model_from_json
from sklearn.metrics.pairwise import cosine_similarity
from keras.preprocessing.sequence import pad_sequences
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
import pickle
import os
import keras.backend as K
import nltk
from logger import get_logger
from environs import Env

logger = get_logger()
env = Env()
env.read_env()
FINAL_DATA = env.str("FINAL_DATA")
MODELS = env.str("MODELS")
DATA_PATH = env.str("DATA_PATH")
CATEGORIES_DIRECTORY = DATA_PATH + 'dbc/'
MAX_SEQUENCE_LENGTH = 300
TRAIN_TEST_PATH = env.str("TRAIN_TEST_PATH")
nltk.download('stopwords')
```

```
df_keys = pd.read_csv(DATA_PATH + 'tags_keys.csv', engine='pyarrow')
title_embeddings = np.load(TRAIN_TEST_PATH + 'embedding_matrix.npz',
allow_pickle=True)
title_embeddings = title_embeddings.f.arr_0
```

```
# Import saved Word2vec Embeddings
```

```
w2v_model = gensim.models.word2vec.Word2Vec.load(MODELS +
'SO_word2vec_embeddings.bin')
```

```
model_tags = pickle.load(open(MODELS + 'model_tags.pkl', 'rb'))
```

```
mlb = pickle.load(open(MODELS + 'mlb.pkl', 'rb'))
```

```
dict_of_dfs = { }
```

```
for file in os.scandir(CATEGORIES_DIRECTORY):
```

```
    if file.is_file():
```

```
        df = pd.read_csv(file.path, engine='pyarrow')
```

```
        dict_of_dfs[file.name[:-4]] = df
```

```
def encode_tags(list_of_tags: list, keys: pd.DataFrame):
```

```
    new_l = []
```

```
    for k in list_of_tags:
```

```
        if k in keys.tag.values:
```

```
            new_l.append(keys[keys.tag == k].code.values[0])
```

```
    return new_l
```

```

def get_category_df(list_of_tags, num_of_tags):
    t = encode_tags(list_of_tags, df_keys)
    tags = mlb.transform([t, []])
    full_res = list(model_tags.predict_proba(tags)[0])
    if max(full_res) < 0.95:
        res = [model_tags.classes_[full_res.index(c)] for c in sorted(full_res)[-
num_of_tags:]]
        return pd.concat([dict_of_dfs[i] for i in res])
    else:
        res = model_tags.classes_[full_res.index(max(full_res))]
        return dict_of_dfs[res]

```

```

def f1_metric(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2 * (precision * recall) / (precision + recall + K.epsilon())
    return f1_val

```

Custom loss function to handle multilabel classification task (modified cross entropy)

```

def multitask_loss(y_true, y_pred):
    # Avoid divide by 0
    y_pred = K.clip(y_pred, K.epsilon(), 1 - K.epsilon()) # K.epsilon() = 1e-7
    # Multi-task loss

```

```

    return K.mean(K.sum(-y_true * K.log(y_pred) - (1 - y_true) * K.log(1 - y_pred),
axis=1))

```

```

def load_tag_encoder():

```

```

    with open(TRAIN_TEST_PATH + "final_tags.txt", "rb") as final_tag: #

```

Unpickling

```

        final_tag_data = pickle.load(final_tag)
        tag_encode = MultiLabelBinarizer()
        tags_encoded = tag_encode.fit_transform(final_tag_data)
        return tag_encode

```

```

def predict_tags(text, num_of_tags):

```

```

    # Tokenize text

```

```

    x_test = pad_sequences(tokenizer.texts_to_sequences([text]),
maxlen=MAX_SEQUENCE_LENGTH)

```

```

    # Predict

```

```

    prediction = model.predict([x_test])[0]

```

```

    pr_sort = np.sort(prediction)

```

```

    for i, value in enumerate(prediction):

```

```

        if value in pr_sort[-num_of_tags:]:

```

```

            prediction[i] = 1

```

```

        else:

```

```

            prediction[i] = 0

```

```

    tags = tag_encoder.inverse_transform(np.array([prediction]))

```

```

    return tags

```

```

# Load model and other relevant stuff

```

```

tag_encoder = load_tag_encoder()

with open(TRAIN_TEST_PATH + "tokenizer.txt", 'rb') as tokenizer_file:
    tokenizer = pickle.load(tokenizer_file)

keras.losses.multitask_loss = multitask_loss
graph = tf.compat.v1.get_default_graph()
with open(MODELS + 'stack.json', 'r') as json_file:
    loaded_model_json = json_file.read()
    model = model_from_json(loaded_model_json,
                           custom_objects={'f1': f1_metric, 'recall': Recall, 'precision':
Precision})
# load weights into new model
model.load_weights(MODELS + "stack.h5")

def question_to_vec(question, embeddings, dim=300):
    question_embedding = np.zeros(dim)
    valid_words = 0
    for word in question.split(' '):
        if word in embeddings.wv.index_to_key:
            valid_words += 1
            question_embedding +=
embeddings.syn1neg[embeddings.wv.key_to_index[word]]
    if valid_words > 0:
        return question_embedding / valid_words
    else:
        return question_embedding

```

```
def most_common(string, tags):
```

```
    tag_list = string.split('|')
```

```
    count = 0
```

```
    for i in tag_list:
```

```
        if i in tags:
```

```
            count += 1
```

```
    return count
```

```
def search_results(search_string, num_results):
```

```
    # preprocessing the input search string
```

```
    search_string = preprocess_text(search_string)
```

```
    search_vect = np.array([question_to_vec(search_string, w2v_model)])
```

```
    # Getting the predicted tags
```

```
    tags = list(predict_tags(search_string, 5))
```

```
    tags = [item for t in tags for item in t]
```

```
    preprocessed_data = get_category_df(tags, 3)
```

```
    tags = set(tags)
```

```
    search_res = []
```

```
    all_title_embeddings = []
```

```
    preprocessed_data['common_tags_num'] = preprocessed_data['tags'].apply(lambda
x: most_common(x, tags))
```

```
    preprocessed_data.sort_values(by=['common_tags_num'], inplace=True)
```

```
    preprocessed_data = preprocessed_data[-500:]
```

```
    preprocessed_data.drop_duplicates(inplace=True)
```

```
    preprocessed_data.reset_index(inplace=True, drop=True)
```



```

# calculating the tfidf
masked_vectorizer = TfidfVectorizer(ngram_range=(1, 2),
max_features=preprocessed_data.shape[0])
masked_vectorizer.fit_transform(preprocessed_data['post_corpus'].values)

# calculating the tfidf of the input string
input_query = [search_string]
search_string_tfidf = masked_vectorizer.transform(input_query)

# getting the title embedding from word to vec model
for title in preprocessed_data.post_corpus:
    title = preprocess_text(title)
    all_title_embeddings.append(question_to_vec(title, w2v_model))
all_title_embeddings = np.array(all_title_embeddings)

# calculating the cosine similarity
cosine_similarities = pd.Series(cosine_similarity(search_vect,
all_title_embeddings)[0])

for i, j in cosine_similarities.nlargest(int(num_results)).iteritems():
    output = preprocessed_data.iloc[i].post_corpus
    temp = {
        'title': str(preprocessed_data.original_title[i]),
        'url': str(preprocessed_data.question_url[i]),
        'similarity_score': str(j)[:5],
        'body': str(output),
        'tags': str(preprocessed_data.tags[i])
    }
    search_res.append(temp)
    if float(search_res[0]['similarity_score']) < 0.8:

```

```
    return  
    return search_res
```

```
html_to_text.py
import pandas as pd
from tqdm import tqdm
from bs4 import BeautifulSoup
from textblob import TextBlob

title_list = []
content_list = []
url_list = []
comment_list = []
sentiment_polarity_list = []
sentiment_subjectivity_list = []
vote_list = []
tag_list = []
corpus_list = []

def lxml_to_text(body):
    content = body
    soup = BeautifulSoup(content, 'lxml')
    if soup.code:
        soup.code.decompose() # Remove the code section
    tag_p = soup.p
    tag_pre = soup.pre
    text = ""
    if tag_p:
        text = text + tag_p.get_text()
    if tag_pre:
        text = text + tag_pre.get_text()
```

```
return text
```

```
def content_to_tokens(dataframe):
    for i, row in tqdm(dataframe.iterrows()):
        title_list.append(row.title) # Get question title
        tag_list.append(row.tags) # Get question tags

        # Questions
        text_body = lxml_to_text(row.body)

        content_list.append(
            str(row.title) + ' ' + str(text_body)) # Append title and question body data to
the updated question body

        url_list.append('https://stackoverflow.com/questions/' + str(row.id))

        # Answers
        text_answers = lxml_to_text(row.combined_answers)
        comment_list.append(text_answers)

        vote_list.append(row.combined_score) # Append votes

        corpus_list.append(
            content_list[-1] + ' ' + comment_list[-1]) # Combine the updated body and
answers to make the corpus

        sentiment = TextBlob(row.combined_answers).sentiment
        sentiment_polarity_list.append(sentiment.polarity)
        sentiment_subjectivity_list.append(sentiment.subjectivity)
```

```
content_token_df = pd.DataFrame({'original_title': title_list, 'post_corpus':  
corpus_list,  
                                'question_content': content_list, 'question_url': url_list,  
                                'tags': tag_list, 'overall_scores': vote_list,  
                                'answers_content': comment_list,  
                                'sentiment_polarity': sentiment_polarity_list,  
                                'sentiment_subjectivity': sentiment_subjectivity_list})  
  
return content_token_df
```

```

normalize_content.py
import spacy
import heapq
import pandas as pd
from search_engine.processing_data.normalize_functions import preprocess_text
from environs import Env
from search_engine.processing_data.html_to_text import content_to_tokens

env = Env()
env.read_env()
DATA_PATH = env.str("DATA_PATH")

EN = spacy.load('en_core_web_sm')

def save_parts(dataframe, nd):
    for i in range(5, int(dataframe.shape[0] / 100000) + 1):
        if i == int(dataframe.shape[0] / 100000):
            df = dataframe[int(i * 10e4):int(dataframe.shape[0])]
        else:
            df = dataframe[int(i * 10e4):int((i + 1) * 10e4)]

        content_token_df = normalize_content(df)
        content_token_df.to_parquet(f"{DATA_PATH}/df{nd}/main_data_{i}.gzip",
compression='gzip', index=False)

def normalize_content(dataframe):
    content_token_df = content_to_tokens(dataframe)

```

```

# Convert raw text data of tags into lists
content_token_df.tags = content_token_df.tags.apply(lambda x: x.split('|'))

# Make a dictionary to count the frequencies for all tags
tag_freq_dict = {}
for tags in content_token_df.tags:
    for tag in tags:
        if tag not in tag_freq_dict:
            tag_freq_dict[tag] = 0
        else:
            tag_freq_dict[tag] += 1

most_common_tags = heapq.nlargest(1250, tag_freq_dict, key=tag_freq_dict.get)

final_indices = []
for i, tags in enumerate(content_token_df.tags.values.tolist()):
    # The minimum length for common tags should be 2
    if len(set(tags).intersection(set(most_common_tags))) > 1:
        final_indices.append(i)

final_data = content_token_df.iloc[final_indices]

# Preprocess text for 'question_body', 'post_corpus' and a new column
'processed_title'

final_data.question_content = final_data.question_content.apply(lambda x:
preprocess_text(x))

final_data.post_corpus = final_data.post_corpus.apply(lambda x:
preprocess_text(x))

final_data['processed_title'] = final_data.original_title.apply(lambda x:
preprocess_text(x))

```

```

# Normalize numeric data for the scores
final_data['overall_scores'] = (final_data.overall_scores -
final_data.overall_scores.mean()) / (
    final_data.overall_scores.max() - final_data.overall_scores.min())

final_data.tags = final_data.tags.apply(lambda x: '|'.join(x)) # Combine the lists
back into text data
final_data = final_data.drop(['answers_content'], axis=1)

return final_data

df1 = pd.read_parquet(f'{DATA_PATH}/de_duplicated_data1.gzip',
engine="pyarrow")
save_parts(df1, 1)

```



```

normalize_functions.py
import spacy
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer

nltk.download('stopwords')
EN = spacy.load('en_core_web_sm')

def tokenize_text(text):
    # Apply tokenization using spacy to docstrings.
    tokens = EN.tokenizer(text)
    return [token.text.lower() for token in tokens if not token.is_space]

def to_lowercase(words):
    """Convert all characters to lowercase from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = word.lower()
        new_words.append(new_word)
    return new_words

def remove_punctuation(words):
    """Remove punctuation from list of tokenized words"""
    new_words = []
    for word in words:

```

```

new_word = re.sub(r'^\w\s', '', word)
if new_word != "":
    new_words.append(new_word)
return new_words

```

```

def remove_stopwords(words):
    """Remove stop words from list of tokenized words"""
    new_words = []
    for word in words:
        if word not in stopwords.words('english'):
            new_words.append(word)
    return new_words

```

```

def normalize(words):
    words = to_lowercase(words)
    words = RegexpTokenizer(r"[a-zA-Z0-9._+#]+").tokenize(' '.join(words))
    words = remove_stopwords(words)
    return words

```

```

def tokenize_code(text):
    # A very basic procedure for tokenizing code strings.
    return RegexpTokenizer(r"[a-zA-Z0-9._+#]+").tokenize(text)

```

```

def preprocess_text(text):
    return ' '.join(normalize(tokenize_text(text)))

```

```

preprocess_data.py
import pandas as pd
from environs import Env

env = Env()
env.read_env()
DATA_PATH = env.str("DATA_PATH")

df1 = pd.read_parquet(f'{DATA_PATH}/raw_data1.zip', engine="pyarrow")
df2 = pd.read_parquet(f'{DATA_PATH}/raw_data2.zip', engine="pyarrow")

# Check if dataframes contains NaN values
print(df1.isna().sum(), df2.isna().sum())

# Drop NaN values
df1 = df1.dropna(axis=0)
df2 = df2.dropna(axis=0)

# Check if dataframes contains duplicated values
print(df1.duplicated().any(), df2.duplicated().any())

# Combining duplicate titles with one answer to one title with many answers (also
combining their score)
de_duplicated_data1 = df1.groupby(['id', 'title', 'body', 'tags'], as_index=False) \
    .agg(combined_answers=('answers', lambda x: "\n".join(x)),
    combined_score=('score', 'sum'))
de_duplicated_data2 = df2.groupby(['id', 'title', 'body', 'tags'], as_index=False) \
    .agg(combined_answers=('answers', lambda x: "\n".join(x)),
    combined_score=('score', 'sum'))

```

```
de_duplicated_data1.to_parquet(f"{DATA_PATH}de_duplicated_data1.gzip",  
compression='gzip', index=False)  
de_duplicated_data2.to_parquet(f"{DATA_PATH}de_duplicated_data2.gzip",  
compression='gzip', index=False)
```

```
logger.py
```

```
import logging
```

```
from environs import Env
```

```
env = Env()
```

```
env.read_env()
```

```
ROOT = env.str("ROOT_PATH")
```

```
class LevelFilter(logging.Filter):
```

```
    def __init__(self, level):
```

```
        super().__init__()
```

```
        self.__level = level
```

```
    def filter(self, record: logging.LogRecord) -> bool:
```

```
        return record.levelno <= self.__level
```

```
def get_logger(handle_info=True, handle_errors=True) -> logging.Logger:
```

```
    logger = logging.getLogger(__name__)
```

```
    format_str = u'%(filename)s [LINE:%(lineno)d] #%(levelname)-8s [%(asctime)s]
```

```
%(message)s'
```

```
    formatter = logging.Formatter(format_str)
```

```
    logger.setLevel(level=logging.INFO)
```

```
    if handle_info:
```

```
        file_info_handler = logging.FileHandler(f'{ROOT}/info.log')
```

```
        file_info_handler.setLevel(logging.INFO)
```

```
        file_info_handler.setFormatter(formatter)
```

```
        file_info_handler.addFilter(LevelFilter(logging.INFO))
```

```
    logger.addHandler(file_info_handler)
```

```
if handle_errors:
    file_error_handler = logging.FileHandler(f'{ROOT}/errors.log')
    file_error_handler.setLevel(logging.ERROR)
    file_error_handler.setFormatter(formatter)
    logger.addHandler(file_error_handler)
logging.basicConfig(format=format_str, level=logging.INFO)
return logger
```