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-11, 14

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 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "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 Головченко Максим Миколайович

(підпис) (прізвище, ім’я, по батькові)

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 р.

**АНОТАЦІЯ**

Пояснювальна записка до курсової: 97 сторінок, 5 рисунків, 17 таблиць, 5 посилань.

Об’єкт дослідження: гра «Шахові фігури».

Мета роботи: дослідження методів обрахунку взять на шаховій дошці, розробка програмного забезпечення для обрахунку усіх можливих взять, доступних в позиції на шаховій дошці.

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 гри «Шахові фігури»

ЗМІСТ

[1 Постановка задачі 8](#_Toc105450337)

[2 Теоретичні відомості 9](#_Toc105450338)

[3 Опис алгоритмів 14](#_Toc105450339)

[3.1. Загальний алгоритм 14](#_Toc105450340)

[3.2. Алгоритм методу 1 14](#_Toc105450341)

[4 Опис програмного забезпечення 15](#_Toc105450342)

[4.1. Діаграма класів програмного забезпечення 15](#_Toc105450343)

[4.2. Опис методів частин програмного забезпечення 15](#_Toc105450344)

[4.2.1. Користувацькі методи 15](#_Toc105450345)

[4.2.2. Стандартні методи 15](#_Toc105450346)

[5 Тестування програмного забезпечення 16](#_Toc105450347)

[5.1 План тестування 16](#_Toc105450348)

[6 Інструкція користувача 20](#_Toc105450349)

[6.1 Робота з програмою 20](#_Toc105450350)

[6.2 Формат вхідних та вихідних даних 24](#_Toc105450351)

[6.3 Cистемні вимоги програмного забезпечення 24](#_Toc105450352)

[Висновки 26](#_Toc105450353)

[Перелік посилань 27](#_Toc105450354)

[ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ 28](#_Toc105450355)

[ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ 31](#_Toc105450356)

**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.

# 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.

# Теоретичні 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.



Рис. 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 [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 , де E – це гіперпараметр, що відповідає розмірності результуючих векторів, тобто на виході утворюється певна кількість (дорівнює кількості вхідних векторів) векторів розмірності , 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] :

Після обчислення цієї функції активації для даного вектору результатом є вектор розмірності , 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 та значення виходу (при , вектор виходу ). По ним обчислюється претендент на нове значення виходу — вектор вузла скидання (*reset gate vector*) , який обчислюється як функція активації (зазвичай сигмоїд) від матричного виразу по параметрам W, U та b. Незалежно, подібним чином, обчислюється вектор вузла уточнення (*update gate vector*) . 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 , в якому з ймовірністю береться старе значення з вектору , або з ймовірністю обчислюється нове значення. Формули для обчислень наступні:

, де – добуток Адамара (поелементий добуток матриць)

Змінні:

– вектор входу;

– вектор виходу;

– вектор вузла уточнення;

– вектор вузла скидання;

*–* матриці та вектор параметрів.

Функції активації:

– сигмоїдна функція [7]:

– гіперболічний тангенс [8]:

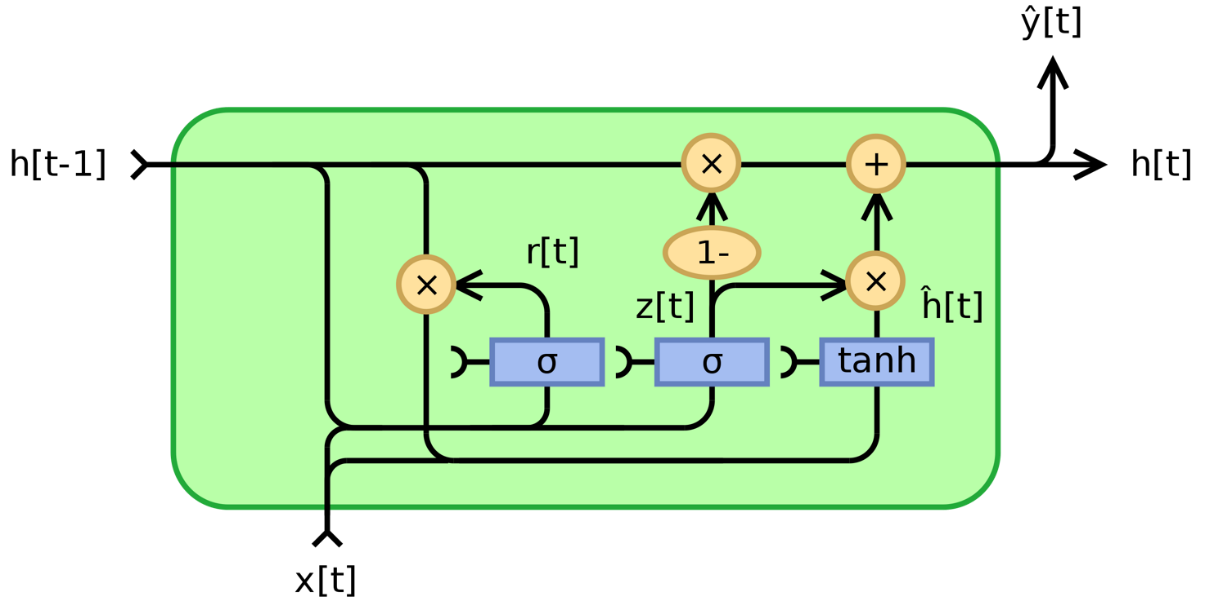
На рисунку 2.2 зображений повний рекурентний вузол.

Рис. 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 к категорій формула обчислення ймовірності цієї категорії для i-го елемента наступна:

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 категорії 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).

# 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

|  |  |
| --- | --- |
| 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 | Двомірний масив, де кожний підмасив містить індекси тегів відповідної статті |
| temp\_ind | Тимчасовий список, який містить індекси тегів поточної статті |

Продовження таблиці 3.1

|  |  |
| --- | --- |
| new\_dec\_tags | Двомірний 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

|  |  |
| --- | --- |
| 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 тегам |
| 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

|  |  |
| --- | --- |
| vocab\_size | Розмір словника для моделі word2vec, тобто кількість слів |
| tags | Множина передбачених тегів для заданого пошукового запиту |
| all\_title\_embeddings | Матриця, що містить векторні представлення для кожного заголовку статті |
| search\_res | Список знайдених статей (список словників) |

## 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

## 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.
   1. Якщо виконується умова, виводимо повідомлення *Searching...*
   2. Присвоюємо змінній *num* значення *text*, яке переведено в int.
   3. В змінну *user\_data* записуємо результат виконання методу *get\_data* об’єкта *state*.
   4. Виконуємо метод *finish* об’єкта *state.*
   5. В змінну *search\_text* записуємо значення масиву *user\_data*
   6. В змінну *t\_0* записуємо значення часу.
   7. В змінну *articles* записуємо результат виконання функції *search\_results.*
   8. Якщо знайшлися такі статті
      1. В змінну *reply\_text* записуємо значення *Articles:\n—————————\n.*
      2. Проходимося по всіх статтях
         1. Виводимо для кожної назву, рейтинг схожості, теги, тіло статті.
         2. Виводимо, за скільки часу був завершений пошук.
   9. Якщо не знайшлися, то виводимо відповідне повідомлення.
4. Якщо не відповідає умові, то викидаємо помилку.
5. Виконуємо метод *reply* об’єкту *message*.
   1. В разі помилки виконуємо метод *answer* об’єкту *message*.
6. КІНЕЦЬ

## Алгоритм кодування тегів у заданому датасеті

Клас *Encoder*

1. ПОЧАТОК
2. Метод \_\_init\_\_
   1. Створюємо конструктор класа.
3. Функція keys\_from\_tags
   1. Копіюємо значення df.
   2. Розділяємо теги df знаком |.
   3. Створюємо порожній об’єкт tags\_dict.
   4. Створюємо порожній об’єкт tags\_freq.
   5. Присвоюємо ind 0.
   6. Проходимося циклом по тегам по i
      1. Проходимося циклом по тегам по j
         1. Якщо tags[i][j] не в списку ключів
            1. Присвоюємо tags\_dict[i][j] ind.
            2. Присвоюємо tags\_freq[i][j] 0.
            3. Додаємо до ind 1.
         2. Якщо ні, додаємо до tags\_freq[i][j] 1.
   7. Створюємо в tags\_freq словник з айтемів.
   8. Створюємо в keys список ключів.
   9. Створюємо в values значення.
   10. Присвоюємо змінній df\_tags результат виконання методу DataFrame.
   11. Присвоємо змінній df\_tags[‘tag’] значення ключів.
   12. Присвоємо змінній df\_tags[‘code’] значення значень.
   13. Повертаємо df\_tags.
4. Метод encode\_tags
   1. Копіюємо значення df.
   2. Створюємо новий масив new\_enc\_tags.
   3. Записуємо ключі.
   4. Проходимось циклом по рядкам айтемів
      1. Створюємо новий масив temp\_ind.
      2. Проходимось циклом по тегам, розділеним |
         1. Присвоюємо змінній code значення відповідного значення.
         2. Доповнюємо масив temp\_ind значенням code.
      3. Доповнюємо масив new\_enc\_tags значенням t.
   5. Записуємо теги в df.tags.
   6. Повертаємо df.
5. КІНЕЦЬ

## Алгоритм декодування тегів у заданому датасеті

Клас *Decoder*

1. ПОЧАТОК
2. Створюємо конструктор класу.
3. Метож decode\_tags
   1. Копіюємо значення df.
   2. Створюємо новий масив new\_dec\_tags.
   3. Записуємо ключі.
   4. Проходимось циклом по рядкам айтемів
      1. Створюємо новий масив temp\_tags.
      2. Проходимось циклом по тегам, розділеним |
         1. Присвоюємо змінній code значення відповідного значення.
         2. Доповнюємо масив temp\_tags значенням tag.
      3. Доповнюємо масив new\_dec\_tags значенням temp\_tags.
   5. Записуємо теги в df.tags.
   6. Повертаємо df.
4. КІНЕЦЬ

## 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

## Алгоритм тренування моделі для передбачення категорії за тегами статті

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.
   1. Якщо строка не пуста
      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

## Алгоритм розбиття заданого датасету на категорії за допомогою моделі

1. ПОЧАТОК
2. Проходимося циклом по в межах цілої частини від ділення довжини *main\_data* діленої на 10000 + 1.
   1. Записуємо в *df\_sample* масив ключ: значення.
   2. Записуємо в *df\_dict* результат виконня функції *get\_df* з параметром *df.*
   3. Проходимося по значеннях ключів та *df* у *df\_dict.items.*
   4. Якщо лічильник = 0
      1. Виконуємо функцію *save\_dataset* з параметрами *key, df\_sample, True, 'w'*.
   5. Інакше
      1. Виконуємо функцію *save\_dataset* з параметрами *key, df\_sample, False, 'w'*.
   6. Виводимо скільки було оброблено айтемів.
3. КІНЕЦЬ

## 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

## 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

## 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-нормализацію.
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

## Алгоритм пошуку деякої кількості статей та інформації про низ 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. Сортуємо 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

# Опис програмного забезпечення

## Діаграма класів програмного забезпечення

Діаграма класів розробленого програмного забезпечення наведена на рисунку 4.1.

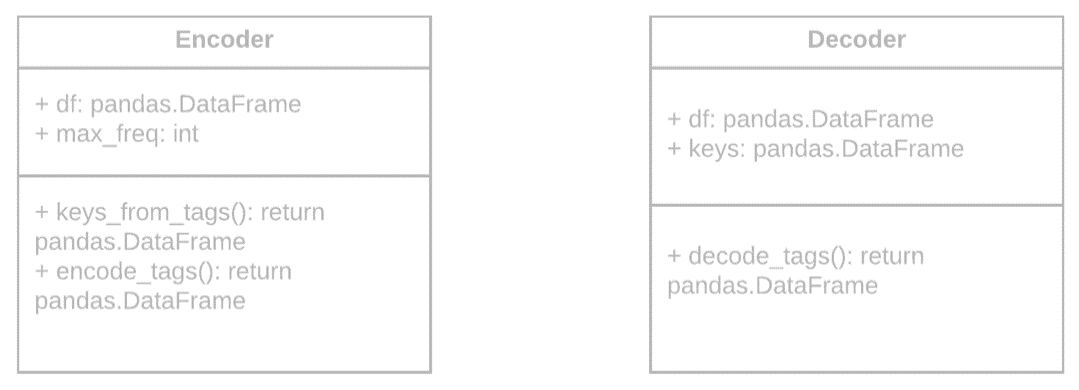


Рисунок 4.1 – Діаграма класів

## Опис методів частин програмного забезпечення

### Користувацькі методи

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

Таблиця 4.1 – Користувацькі методи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | Використовується для логеру | Немає вхідних параметрів | Немає вихідних параметрів |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12 | - | exception | Використовується для виведення повідомлення | Повідомлення про помилку | Повертає повідомлення про помилку |

### Стандартні методи

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

Таблиця 4.2 – Стандартні методи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 1 | Bot\_tg | set\_default\_commands | Використовується для налаштування дефолтних команд бота | Dispatcher | Повертає повідомлення про завершене налаштування |
| 2 | executor | Start\_polling | Виокристовуєтьсядля створення loop | dp, on\_startup=on\_startup, skip\_updates=True | Сигнал завершення |
| 3 | str | Isdecimal | Переверяє чи число десяткове | - | Повертає True, якщо десяткове |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | Int | Int | Переводить число в десяткову форму | Число | Повертає число в десятковій формі |
| 5 | Message | Reply | Виводить повідомлення | Повідомлення | Вивід повідомлення |
| 6 | FSMContext | Get\_data | Отримує дані | - | Дані |
| 7 | FSMContext | Finish | Закінчує процес | - | Закінчення процесу |
| 8 | Time | time | Отримання часу | - | час |
| 9 | Search\_engine | 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 | Напівжирний 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | Перетворена 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 | Передбачена модель |

# 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 accuracy (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 accuracy |
| Стан програми після проведення випробувань | Видано результат тестування за метрикою accuracy, 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 accuracy (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% |

# Інструкція користувача

### 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 кількості 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

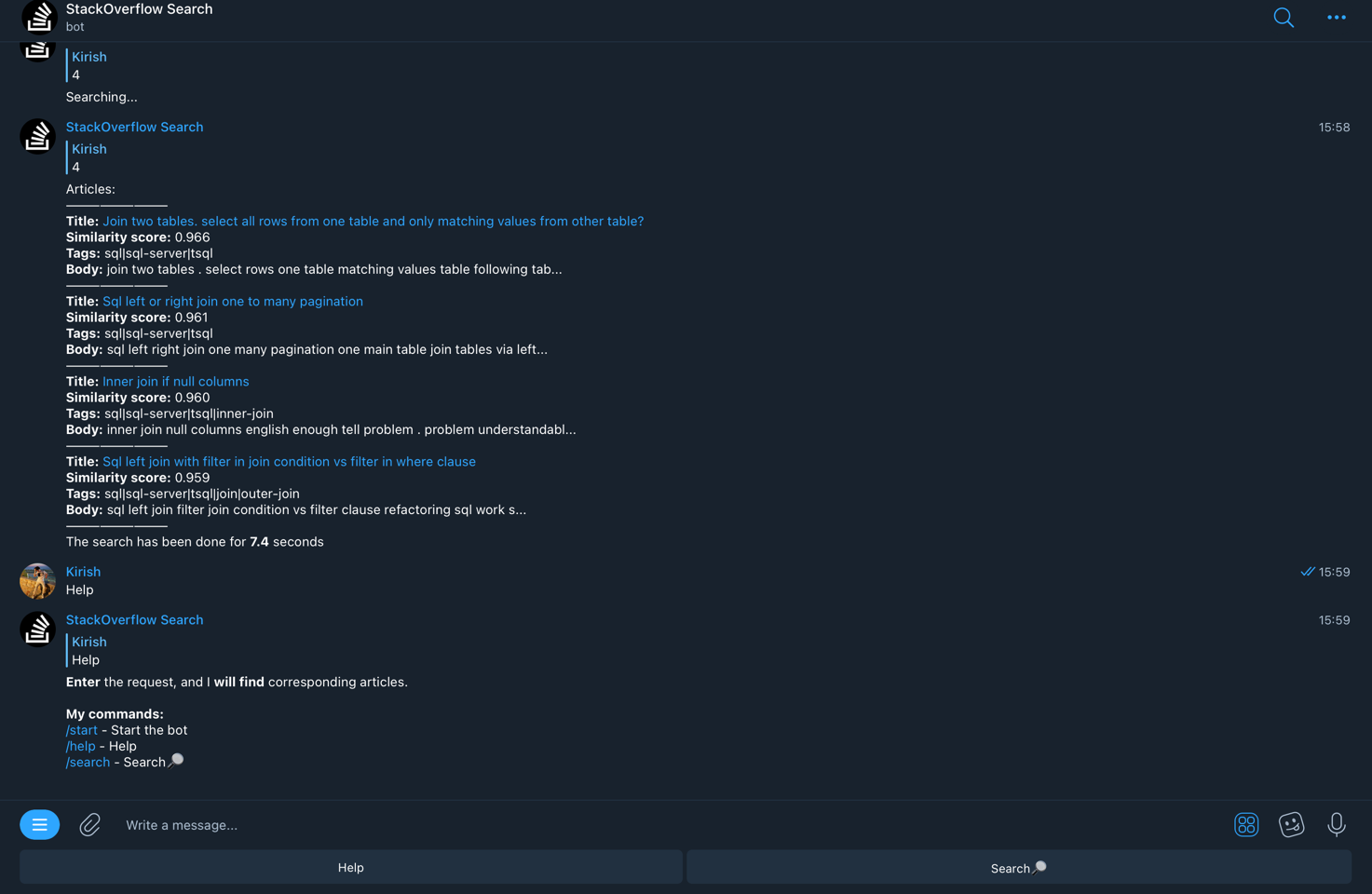
 При виклику команди /help або натисненні кнопки Help, бот надсилає текстове повідомлення з описом його команд (рисунок 6.6).

Рисунок 6.6 – Повідомлення з описом команд бота

### 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 кількості 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).

### Cистемні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows 10/Windows11/MacOS10.11+/IOS15+/Android11+ (з останніми обновленнями) | Windows 10/Windows 11/MacOs12.3.1/IOS15.5/Android12  (з останніми обновленнями) |
| Процесор | Intel® Pentium® ІІІ  1.0 GHz або  AMD Athlon™ 1.0 GHz | Intel® Pentium® D або AMD Athlon™ 64 X2 |
| Оперативна пам'ять | 4 GB RAM | 8 GB RAM |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 800х600 | 1024х768 або краще |

Продовження таблиці 6.3

|  |  |  |
| --- | --- | --- |
|  | 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 введені кількості статей боту неможливо ввести буквений вираз, ненатуральне число, або число, яке не належить відрізку [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

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 {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 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.gzip', 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, 'с++', 'с')

delete\_elem\_from\_tags(df\_enc, 'с', '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}&pagesize=50') \

.content

else:

result\_html = requests.get(

f'https://stackoverflow.com/search?page={i}&tab=Relevance&pagesize=50&q={search\_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)

vectorization.py

import gensim

from search\_engine.prediction\_model.tag\_predictor.tag\_predictor import preprocessed\_data

from environs import Env

import numpy as np

from pathlib import Path

from logger import get\_logger

env = Env()

env.read\_env()

MODELS\_PATH = env.str("MODELS")

W2V\_MODEL\_PATH = env.str("W2V\_MODEL\_PATH")

my\_file = Path(W2V\_MODEL\_PATH)

if not my\_file.is\_file():

logger = get\_logger(handle\_errors=False)

W2V\_SIZE = 300

W2V\_WINDOW = 7

W2V\_EPOCH = 32

W2V\_MIN\_COUNT = 10

documents = [\_text.split() for \_text in np.array(preprocessed\_data.post\_corpus)]

w2v\_model = gensim.models.word2vec.Word2Vec(vector\_size=W2V\_SIZE,

window=W2V\_WINDOW,

min\_count=W2V\_MIN\_COUNT,

workers=8)

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.gzip', engine="pyarrow")

df2 = pd.read\_parquet(f'{DATA\_PATH}/raw\_data2.gzip', 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