Search Engine

Damian Tworek

April 2022

1 Idea

Wyszukiwarka stron internetowych. Dostaje na wejściu stringa i zwraca na wyjściu k najbardziej zbliżonych linków do szukanej frazy.

2 Plan

1. Pobranie dużej liczby stron z angielskiej wikipedii.
2. Przefiltrowanie stron, tak żeby został sam tekst
3. Usunięcie cyfr. Użycie stemmera (sprowadzenie słowa do rdzenia (np. likely -> like))
4. Zbudowanie macierzy, której wierszami są dokumenty, a kolumnami słowa. Wartość macierz[dokument][slowo] będzie liczbą wystąpień słowa w dokumencie.
5. TFIDF na macierzy
   - Term frequency - liczba wystąpień słowa w dokumencie / całkowita liczba słów w dokumencie.
   - Inverse Document Frequency - log (liczba dokumnetów / liczba dokumentów, które zawierją słowo)
   - TFIDF (Term Frequency - Inverse Document Frequency) - Term Frequency * Inverse Document Frequency
6. Zapisanie macierzy
7. Napisanie funkcji query(str, k), która zwraca k najbardziej zbliżonych dokumentów na podstawie korelacji wektorów.
8. Stworzenie graficznego interfejsu użytkownika.

3 Web scraping

Używając biblioteki requests oraz BeautifulSoup.

Zaczynam szukać kilkuset podstron zaczynając w poniższych linkach:

- 'https://en.wikipedia.org/wiki/Mathematics',
- 'https://en.wikipedia.org/wiki/Biology',
- 'https://en.wikipedia.org/wiki/Physics',
- 'https://en.wikipedia.org/wiki/Car',
- 'https://en.wikipedia.org/wiki/History',
- 'https://en.wikipedia.org/wiki/Esports',
- 'https://en.wikipedia.org/wiki/Music',
- 'https://en.wikipedia.org/wiki/Sport',
- 'https://en.wikipedia.org/wiki/Dog',
- 'https://en.wikipedia.org/wiki/Geography',
- 'https://en.wikipedia.org/wiki/English$_W ikipedia'$

1. Sprawdzam, żeby każdy link do, którego wchodzę był z wikipedii angielskiej oraz nie należał do żadnej z następujących kategorii: Category, File, png, jpg, Template, Help.
2. Usuwam skrypty i elementy stylu z HTMLa.
3. Zapisuje do pliku o nazwie routa np. dla https://en.wikipedia.org/wiki/Car to będzie Car. Taki zabieg, żeby artykuły się nie powtarzały.

4 Tworzenie macierzy

Będę operować na macierzy rzadkiej, bo będzie to efektywniejsze, gdyż macierz gęsta w większości będzie pokryta zerami.

1. Wczytuję dokumenty do listy.
2. Przekazuje listę dokumentów do instancji klasy CountVectorizer oraz maksymalną liczbę kolumn, a także metodę tokenizacji. Klasa ta zwraca macierz rzadką, gdzie wartość macierzy w wierszu d a kolumnie s, oznacza liczbę wystąpień słowa s w dokumencie d.

   Metoda tokenizacji:

   (a) Stostuję regex, wykorzystując bibliotekę *re* - zostawiam tylko wyrażenia mające litery alfabetu angielskiego.

(b) Wykorzystuję *Porter Stemmer* z biblioteki *NLTK* do zostawienia tylko rdzenia słowa.

3. Używam na otrzymanej macierzy TfidfTransformer w celu eliminacji istotności długości dokumentu.

4. Normalizuje wiersze, żeby miały długość jeden.

5 Backend

Flask posłużył do utworzenia lokalnego serwera, który gdy dostanie zapytanie o link np. 192.16.8.0.1/student/QUERY, to zwróci najbardziej zbliżone linki do QUERY.

6 Backend

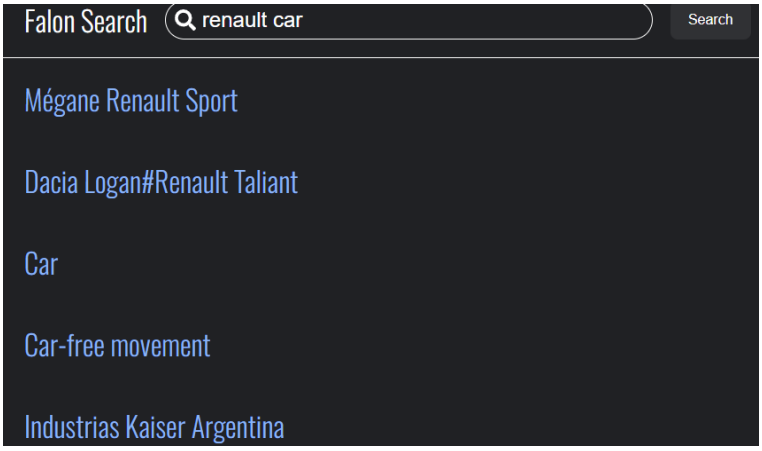
React odpowiada za interaktywny interface użytkownika oraz routing.

7 Eliminacja szumu w macierzy

W celu redukcji szumu w macierzy stosuję metodę SVD z biblioteki scipy dla macierzy rzadkich.

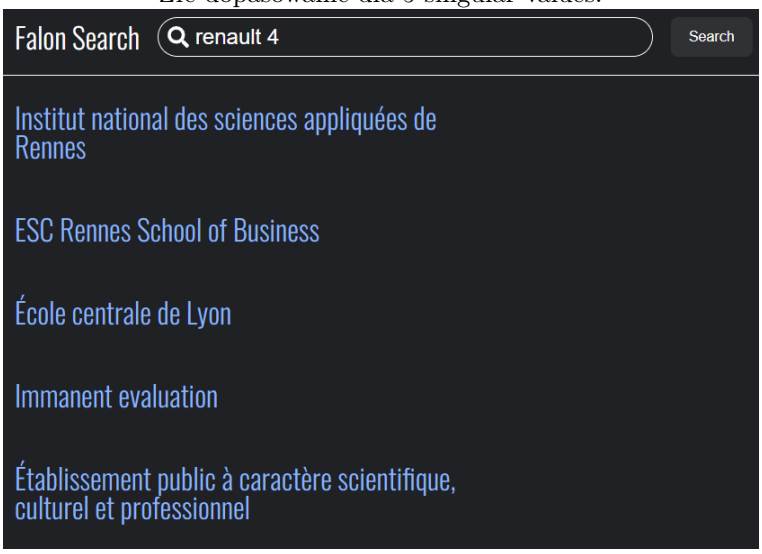
8 Rezultaty

Dla około 13 tysięcy dokumentów i 8 tysięcy słów czas wynosi 8 sekund, a po redukcji szumów (do 5 singular values) około 5 sekund



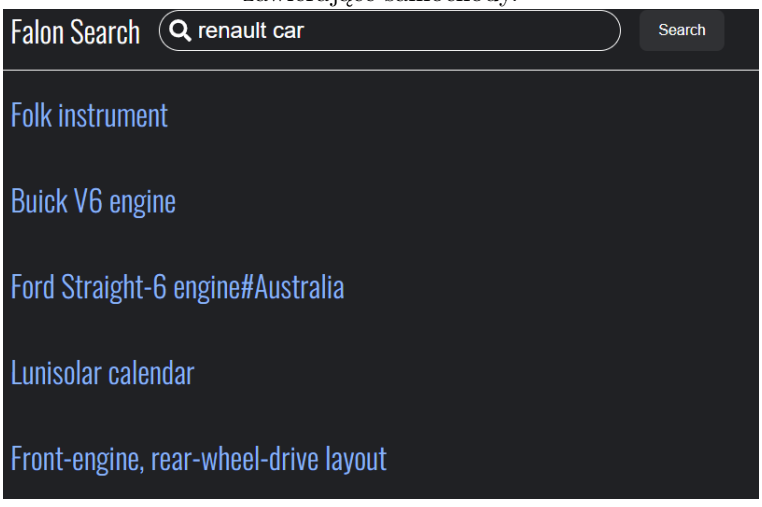
Rysunek 1: Dopasowanie do frazy 'renault car' bez redukcji szumów.

Złe dopasowanie dla 5 singular values:



Rysunek 2: Dopasowanie do frazy 'renault car' po redukcji szumów do 5 singular values.

Słabe dopasowanie dla 8 singular values, ale przynajmniej zwraca artykuły zawierające samochody.



Rysunek 3: Dpasowanie do frazy 'renault car' po redukcji szumów do 8 singular values.

Coraz lepsze dopasowanie, tym razem dla 16 singular values. Tym razem zwraca marki samochodów, ale różne od Renault. Dla kolejnych 32, 64, 128, 256, 512, 1024 nie zwraca linków odnoszących się do Renualt, ale zwracane są w temacie silników i marek samochodów.

Przy 1024 singular values zwraca artykuły z Renault, ale na pozycji 4 i 5.

Po zwiększeniu do 2048 signular values i zmniejszeniu słownika zwraca artykuły zawierające słowo Renault w linku, co jest satysfakcjonujące. Czas oczekiwania to 3.9 sekundy.

Rysunek 4: Dpasowanie do frazy 'renault car' po redukcji szumów do 2048 singular values.

9 Wnioski

q

Implementacja ta nie jest state-of-art, ponieważ oczekiwanie na wynik zajmuje zbyt dużo czasu, jednakże zwraca wyniki podobne do wyszukiwanej frazy. Redukcja szumów dodatkowo skróciła czas wyszukiwania do 4 sekund.

Alternatywnym podejściem do problemu byłoby najpierw sprawdzenie, do której z kategorii należy zapytanie, np. (nauka, muzyka, sport) i wtedy dla niej porównywać z artykułami, które znajdują się w danej kategorii.