Sprawozdanie - Latent Semantic Indexing

Arkadiusz Kraus

21 maja 2019

1 Zadanie

Zadanie polegało na zastosowaniu metody Latent Semantic Indexing (LSI), aby w zbiorze tektów móc wyszukiwać te związane z podaną frazą.

2 Zbiór danych

Zbiór danych jaki zastosowałem pochodzi ze strony `https://www.kaggle.com/snapcrack/all-the-news`. Jest to zbiór artykułów z amerykańskich gazet z lat 2016-2017 o różnej tematyce. Jest złożony z ponad 140 tys. tekstów.

3 Zastosowane przekształcenia

Na początku przetwarzamy wstępnie przetwarzamy zbiór artykułów, aby później móc skuteczniej w nim wyszukiwać:

3.1 Zbiór użytych słów

Dla każdego artykułu bierzemy wszyskie użyte słowa i tworzymy wektor słów dla wszystkich tekstów. Dla kolejnych podzbiorów artykułów posiada on następujące wielkości.

liczba artykułów	rozmiar wektora
1000	
10000	140 tys.
50000	230 tys.
wszystkie(142572)	400 tys.

3.2 Wstępne przetworzenie

Liczba słów uzyskana w poprzednim podpunkcie jest bardzo duża, aby uczynić algorytm wydajniejszym staramy się zredukować ten wektor. We wstępnym przetworzeniu pomijamy wielkość liter oraz znaki interpunkcyjne.

3.3 Stemming

Wiele słów może występować w różnych formach np. take i taking. Nas jednak interesuje sam fakt czynności, a nie koniecznie np. czas w jakim była wykonywana. Innym przypadkiem jest też liczba mnoga rzeczownika, gdzie nadal interesuje nas sam rzeczownik. Dlatego stosujemy algorytm stemmingu (dokładniej Porter Stemmer), doprowadzamy słowa do wspólnego rdzenia i usuwamy duplikaty. W połączeniu z poprzednim krokiem pozwala to znacząco zredukowac rozmiar wektora:

liczba artykułów	rozmiar wektora
1000	24 tys.
10000	70 tys.
wszystkie(142572)	234tys.

3.4 Stopwords

Stopwords to zbiór słów takich jak przyimki itp, które występują bardzo często we wszelkich tekstach. Nie mają one większego znaczenia dla treści dlatego również można je pominąć. Nie redukuje to wielkości wektora znacznie (o około 150 słów), jednak zwiększa jakość wyszukiwania.

3.5 Tworzenie macierzy rzadkiej

Chcemy teraz utworzyć macierz, która jako wiersze będzie przyjmować kolejne artukułu, a jako kolumny kolejne słowa ze zbioru. Wartością w komórce a_ij macierzy będzie liczba wystąpień j-tego słowa w i-tym artykule. Zauważmy, że będzie to macierz rzadka ponieważ artyukuły mają ok. 4000 tys słów, a wektor słów jest znacznie większy. Dodatkowo jeśli artykuł jest na jakis temat to słowa często się w nim powtarzają. Aby przyśpieszyć obliczenia (i w ogóle je umożliwić) stosujemy więc przechowywanie w postaci macierzy rzadkiej (dokładniej CSR - Compressed Sparse Row).

3.6 IDF

W celu zwiększenia jakości wyszukiwania stosujemy Inverse Document Frequency (IDF). Pozwala on nam zmniejszyć znaczenie słów, które występują w wielu artykułach i zwiększyć tych, które rzadko.

3.7 Normalizacja

Normalizacja pozwala uniezależnić korelację od długości tekstu. Wykonujemy ją od razu, aby potem przyspieszyć wyszukiwanie, aby nie trzeba było obliczać normy za każdym razem.

3.8 Odszumianie

W celu redukcji szumów stosujemy algorytm SVD i wybieramy k najciekawszych wartości własnych. Co ciekawe zwiększa to liczbę niezerowych elementów macierzy z ok 1-2 mln do 87 mln dla 10 tys. artykułów.

4 Obliczenia

Problem ten jest bardzo wymagający obliczeniowo oraz pamięciowo. Przechowanie macierzy wielkości *liczba artykułów * liczba słów* dla większej ilości artukułów skutkuje skończeniem pamięci. W pliku *logs* znajdują się czasy wstępnego przetworzenia dla 50000 i wszystkich artykułów. Wyszukiwanie po przetworzeniu i odczytaniu wcześniejszej macierzy jest również zależne od ilości artykułów i dla większej ilości jeszcze trochę mu brakuje względem czasów osiągnych przez Google.

liczba artykułów	rozmiar wektora	rozmiar wektora po przetworzeniu	czas przetworzenia	czas wyszukania
1000		24 tys	natychmiast	natychmiast
10000	140 tys	70 tys	1 min	3 s
50000	230 tys		35 min	
wszystkie(142572)	400 tys	234 tys	2,5h	10 min

Niestety dla 50000 oraz dla wszystkich artykułów stworzenie macierzy po dekompozycji SVD okazało się zbyt czasochłonne. Przy próbie utworzenia normalnej macierzy po dekompozycji dla np. 200 wartości własnych otrzymywałem brak pamięci, natomiast dla macierzy rzadkich obliczenia trwały w nieskończoność (obliczenia na nich są czasochłonnymi operacjami). Dlatego dla tych wielkości wyszukiwarka używa macierzy bez odszumienia.

5 Wyniki

6 Wnioski