Uniwersytet im. Adama Mickiewicza w Poznaniu
Wydział Matematyki i Informatyki

Praca magisterska

Sortowanie wyników wyszukiwania prac naukowych według związku z zapytaniem

Sorting by Relevance of Search Results for Scientific Papers

Paweł Kruszczyński

nr albumu: 383979
kierunek: informatyka

Promotor:
prof. UAM dr hab. Krzysztof Jassem

Poznań, 2017

Oświadczenie

Ja, niżej podpisany Paweł Kruszczyński student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt: Sortowanie wyników wyszukiwania prac naukowych według związku z zapytaniem napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[ ]* - wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[ ]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....

(czytelny podpis studenta)

Streszczenie

Celem niniejszej pracy magisterskiej jest przedstawienie zasad działania algorytmu *PageRank*, silnika wyszukiwań *Elasticsearch* oraz stworzonego systemu sortującego prace naukowe według związku z zapytaniem.

W pracy zawarte zostało: omówienie algorytmu *PageRank* oraz jego różnych implementacji, omówienie silnika wyszukiwań *Elasticsearch*, poprzedzone wprowadzeniem podstawowych zagadnień dotyczących wyszukiwarek, opis użytych technologii użytych w projekcie, przedstawienie interfejsu webowego i ewaluacja projektu autorskiego wraz z podsumowaniem.

Abstract

The purpose of this thesis is to present the principles of the *PageRank* algorithm, the *Elasticsearch* search engine and the system for ranking scientific papers based on the query.

The paper contains: discussion of the *PageRank* algorithm and its various implementations, brief summary of the *Elasticsearch* search engine, preceded by introduction of search engine basics, description of the technologies used in the project, web interface presentation and evaluation of the project alongside with a summary.

Spis treści

Wstęp

Motywacja

Sukces odniesiony przez algorytm *PageRank* w wyznaczaniu rang stron internetowych natchnął mnie do tego, aby spróbować zastosować go do wyznaczania ważności artykułów naukowych według zapytania. W Internecie nie znalazłem pracy, która połączyłaby ten algorytm oraz silnik *Elasticsearch*, który daje wiele możliwości dopasowywania zapytań do swoich potrzeb, dlatego też właśnie zdecydowałem się podjąć taki temat pracy.

Cel (hipoteza badawcza)

Możliwe jest uszeregowanie artykułów naukowych względem ich ważności przy użyciu algorytmu *PageRank* i silnika wyszukiwań *Elasticsearch*.

Układ pracy

Potwierdzenie celu wymaga wprowadzenia wiedzy o algorytmie *PageRank*, który przedstawiony jest w rozdziale 1. W rozdziale 2 omawia się podstawowe zagadnienia powiązane z silnikami wyszukiwań, a w szczególności silnikiem *Elasticsearch*, zastosowanym w projekcie autorskim. W tym samym rozdziale następuje przedstawienie dwóch artykułów naukowych: jeden z nich bada wykorzystanie *PageRank* w rangowaniu artykułów naukowych, drugi omawia wykorzystanie *Elasticsearch* w ich wyszukiwaniu. Rozdział 3 opisuje projekt autorski wykonany w ramach pracy magisterskiej, a ponadto wprowadza podstawowe informację na temat użytych technologii: *Jupyter Notebook*, *AngularJS* i *Apache Spark*. Ostatni rozdział 4 przedstawia metodę oraz wyniki ewaluacji, na podstawie których na końcu pracy wyciągnięte są wnioski.

10:2993632250

Rozdział 1

Teoria - algorytm PageRank

1.1 Podstawowe definicje

Algorytm *PageRank* jest algorytmem wyznaczającym *rangę* (*score*) dokumentu na podstawie grafu połączeń dokumentów. Pierwotnie został stworzony, aby wyznaczać rangę dla stron sieci web.

Struktura, na podstawie której wyliczany jest *PageRank*, to graf skierowany o n wierzchołkach. Przedstawia on sieć połączeń pomiędzy dokumentami. Jeśli dany dokument odnosi się do drugiego, tzn. zawiera do niego odnośnik w sekcji bibliografii, to z wierzchołka reprezentującego dokument A zostaje poprowadzona skierowana krawędź do wierzchołka B.

Głównym zadaniem algorytmu *PageRank* jest wyznaczenie prawdopodobieństwa, z jakim losowy surfer trafi na dany dokument, co przekłada się na to, w jakim stopniu ten dokument jest warty odwiedzenia. Na podstawie tego określa się czy dany dokument jest odpowiedni do zadanego zapytania.

1.2 Opis metody działania

Metoda działania algorytmu *PageRank* opiera się na koncepcji losowego surfera, przeglądającego strony internetowe, bądź dokumenty.

Założenia:

- surfer zaczyna w losowo wybranym dokumencie
- w każdym kroku surfer przechodzi do losowo wybranego dokumentu:

- jeśli bieżący dokument to ślepy zaułek (brak odnośników do innych dokumentów), to surfer przenosi się na losowo wybrany dokument

- jeśli bieżący dokument posiada odnośniki do innych dokumentów, to surfer z prawdopodobieństwem $1-\alpha$ przechodzi do jednego z odnośników, bądź z prawdopodobieństwem α przechodzi do losowo wybranego dokumentu, spośród wszystkich

- surfer nigdy nie kończy swojej wędrówki

Metoda jest wyrażona poprzez poniższy wzór:

$$PR_i = \frac{\alpha}{n} + (1-\alpha) \sum_{j \in \{1,\ldots,n\}} \frac{PR_j}{|c_{ij}|} \tag{1.1}$$

gdzie:

- PR_i to prawdopodobieństwo znalezienia się w i-tym dokumencie,

- α to prawdopodobieństwo skoku do losowego dokumentu,

- n jest liczbą dokumentów w sieci,

- $|c_{ij}|$ jest liczbą połączeń wychodzących z i-tego dokumentu do dokumentu j

1.2.1 Algorytm

Algorytm 1: PageRank

Dane wejściowe: graf $G = (V, E)$; lista połączeń c zbudowana w oparciu o graf G; liczba iteracji N; prawdopodobieństwo losowego skoku α

Wynik: wartości *PageRank* PR_i, odpowiadające V_i

```
1 P_leap ← α/n
2 for i ← 1 to n do
3 |   outDeg ← 0
4 |   for k ← 1 to n do
5 |   |   outDeg ← outDeg + |c_ik|
6 |   for j ← 1 to n do
7 |   |   P_ij ← P_leap + (1 − α) * |c_ij| / outDeg
8 PR = P^N
9 return PR_0
```

Omówienie algorytmu 1

1-7 Tworzona jest macierz prawdopodobieństw przejścia P w trzech krokach:

- 1 wyznaczanie prawdopodobieństwa losowego skoku P_{leap},
- 3-5 dla każdego z dokumentów $i \in \{1, \ldots, n\}$ liczona jest liczba połączeń wychodzących $outDeg = \sum_{k \in \{1,\ldots,n\}} |c_{ik}|$.
- 6-7 dla każdego z dokumentów $i \in \{1, \ldots, n\}$ liczone jest prawdopodobieństwo przejścia P_{ij} do dokumentu $j \in \{1, \ldots, n\}$.

8 Macierz P jest potęgowana N-tą liczbę razy.

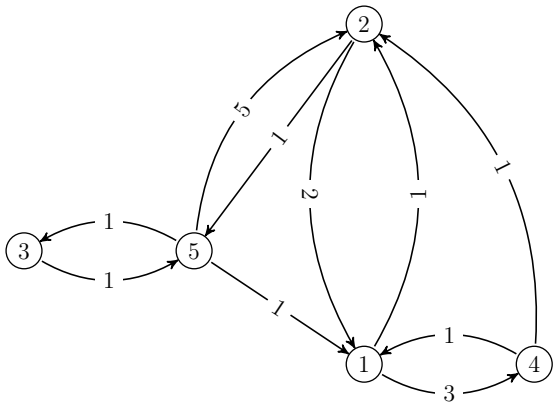
9 Wynikiem jest tablica PR, która przechowuje wartości *PageRank* dla danego dokumentu: PR_i odpowiada dokumentowi V_i.

Liczba iteracji N powinna być dopasowana do danego modelu, tak aby wartości w kolumnach zbiegały co najmniej do 2-3 miejsc po przecinku. Zwykle, przy dużym modelu jest to 50[1] iteracji, ale dla małych modeli, wystarczy około 20.

Zbieżność potęgowania macierzy P wynika z faktu, iż te zastosowanie również wykorzystuje łańcuchy Markova, ale nie wykonuje operacji bezpośrednio na nich. Wykorzystany jest fakt, iż $\pi(n) = \pi(0) * P^n$, co jest wybraniem dowolnego wiersza (oznaczonego przez $\pi(0)$, np.: $\pi(0) = (\,1\;0\;0\;0\;0\,)$) z n-tej potęgi macierzy P. Dzięki temu obliczenia nie wymagają znajomości dodatkowej wiedzy z dziedziny matematyki dyskretnej. Temat zbieżności macierzy prawdopodobieństw przejść P został przedstawiony bliżej w sekcji 1.4.1.

Przykład użycia

Poniższy przykład z rysunku 1.1 przedstawia sieć dokumentów. Dla uproszczenia, aby rysunek był bardziej przejrzysty, na krawędziach są wartości liczby połączeń pomiędzy dokumentami.



Rys. 1.1: Graf przykładowej sieci dokumentów.

- Liczba dokumentów $n = 5$
- Przyjmujemy prawdopodobieństwo przejścia do kolejnej strony $\alpha = 0.1$.
- Prawdopodobieństwo skoku wynosi $P_{leap} = \frac{\alpha}{n} = 0.02$.
- G to macierz incydencji grafu, obrazującego sieć z rysunku 1.1.

$$G = \begin{bmatrix} 0 & 1 & 0 & 3 & 0 \\ 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 5 & 1 & 0 & 0 \end{bmatrix}$$

$$P_1 = \left[0.02_{n\times n}\right]$$

$$P_2 = \begin{bmatrix} 0.0 & 0.225 & 0.0 & 0.675 & 0.0 \\ 0.6 & 0.0 & 0.0 & 0.0 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.9 \\ 0.45 & 0.45 & 0.0 & 0.0 & 0.0 \\ 0.1286 & 0.6428 & 0.1286 & 0.0 & 0.0 \end{bmatrix}$$

$$P = P_1 + P_2 = \begin{bmatrix} 0.02 & 0.245 & 0.02 & 0.695 & 0.02 \\ 0.62 & 0.02 & 0.02 & 0.02 & 0.32 \\ 0.02 & 0.02 & 0.02 & 0.02 & 0.92 \\ 0.47 & 0.47 & 0.02 & 0.02 & 0.02 \\ 0.1486 & 0.6628 & 0.1486 & 0.02 & 0.02 \end{bmatrix}$$

$$P^{50} = \begin{bmatrix} 0.3108469 & 0.28262402 & 0.03785274 & 0.22982166 & 0.13885468 \\ 0.3108469 & 0.28262402 & 0.03785274 & 0.22982166 & 0.13885468 \\ 0.3108469 & 0.28262402 & 0.03785274 & 0.22982166 & 0.13885468 \\ 0.3108469 & 0.28262402 & 0.03785274 & 0.22982166 & 0.13885468 \\ 0.3108469 & 0.28262402 & 0.03785274 & 0.22982166 & 0.13885468 \end{bmatrix}$$

Po 50 iteracjach algorytmu - 50 potęgach macierzy P wartości zbiegają nawet do 8 miejsc po przecinku, co daje wystarczający wynik przedstawiony w tabeli 1.1.

Nr dokumentu	**Wartość P_i**
1	0.3108469
2	0.28262402
4	0.22982166
5	0.13885468
3	0.03785274

Tab. 1.1: Tabela wyników - dokumenty posortowane wg wartości *PageRank*

1.3 Problemy różnych implementacji

Implementacja opisana w sekcji 1.2 jest wersją najbardziej czasowo i pamięciowo zachłanną.

Zmusza ona do przetrzymywania dwóch macierzy prawdopodobieństw P w pamięci, do wykonywania potęgowania, co przy dużej liczbie dokumentów jest

nie do wykonania. Istnieją metody cachowania danych na dysku, lecz spowalnia to znacznie czas obliczeń, ponieważ każdy odczyt wartości macierzy jest opóźniony o operacje wejścia/wyjścia. Te niedogodności sprawiają, że nie jest możliwe wykorzystanie tej implementacji dla realnych danych sieci Web. W kolejnej sekcji 1.4, przedstawię kilka implementacji, które zwiększają wydajność algorytmu.

1.4 Efektywna implementacja

Istnieje wiele implementacji algorytmu, które są stosowane w zależności od charakterystyki danych lub ich ilości. Poniżej zostaną przedstawione tylko te, które w ogólności mają wpływ na wydajność czasową i pamięciową.

1.4.1 Implementacja oparta o łańcuch Markova

Łańcuchy Markova

Łańcuch Markova to aparat matematyczny pozwalający na wyznaczenie prawdopodobieństwa zdarzeń, które są zależne od siebie, tak że kolejne wyniki prawdopodobieństwa, zależą od poprzednich. Opisuje on ciąg zmiennych losowych $(X_0, X_1, \dots)$, takich że X_n w danej chwili zależy od wszystkich poprzednich zmiennych losowych, ale bezpośrednio od poprzedniej zmiennej X_{n-1}.

Założenia

- Zbiór stanów $S = \{1, \dots, n\}$ jest skończony.
- P określa macierz prawdopodobieństw przejścia ze stanu i do stanu j z prawdopodobieństwem p_{ij}.
- Macierz prawdopodobieństw przejścia P jest macierzą stochastyczną:
  - jej wyrazy są nieujemne $\forall_{i,j \in S}\, p_{ij} >= 0$,
  - wiersze sumują się do 1: $\sum_{j=1}^{n} p_{ij} = 1$.
- $\pi(n)$ jest rozkładem prawdopodobieństwa zmiennej losowej X_n, na zbiorze stanów łańcucha Markova w chwili n.
- $\pi_i(n) = P(X_n = i)$
- $\sum_{j=1}^{k} \pi_j(i) = 1$

- $\pi(n) = \pi(n-1) * P$
- obliczając $\pi_i(n)$ otrzymujemy prawdopodobieństwo znalezienia się w stanie i w chwili n.

Powiązanie z algorytmem

Problem losowego surfera może być opisany poprzez łańcuch Markova, gdzie kolejne zmienne losowe X_n to kolejne dokumenty, do których przechodzi surfer. Może on krążyć po dokumentach w nieskończoność, dlatego też łańcuchy Markova, idealnie nadają się do wykorzystania w tym przypadku.

Macierz P spełnia założenia macierzy prawdopodobieństw przejścia.

Surfer zaczyna na losowej stronie, więc generując $\pi(0)$, rozkład początkowy, punkt startowy surfera wybierany jest dowolnie.

Następnie obliczany jest rozkład prawdopodobieństwa $\pi(n)$, po n stanach, którego wartości będą odpowiadały rangom dokumentów w algorytmie *PageRank*.

Zbieżność potęgowania macierzy prawdopodobieństw przejścia

Zbieżność potęgowania macierzy P oraz obliczania kolejnych rozkładów prawdopodobieństwa wynika z własności łańcuchów Markova. Jak już wspomniałem poprzednio $\pi(n) = \pi(0) * P^n$, dlatego też potęgowanie macierzy to obliczanie rozkładu prawdopodobieństw π po n stanach.

Na podstawie książki pt. *Introduction to Information Retrieval*[8] można stwierdzić, iż P_{leap} zapewnia nieprzywiedlność (nieredukowalność) i nieokresowość, zatem i ergodyczność, łańcucha Markova zbudowanego na podstawie problemu losowego surfera.

Twierdzenie ergodyczne mówi, iż każdy ergodyczny łańcuch Markova ma dokładnie jeden rozkład stacjonarny Π, taki że:

$$\lim_{n\to\infty} \pi(n) = \Pi \tag{1.2}$$

Rozkład stacjonarny łańcucha Markova Π to taki rozkład prawdopodobieństwa, który spełnia warunek:

$$\Pi \times P = \Pi \tag{1.3}$$

Zatem jeśli łańcuch Markova opisujący problem losowego surfera w algorytmie *PageRank* jest ergodyczny, to posiada on rozkład stacjonarny, dlatego też z (1.2) wiemy, iż przy odpowiednio dużym n, można znaleźć taki rozkład

prawdopodobieństwa π spełniający równanie (1.3) z pewną dokładnością. Jak już wcześniej było wspomniane, przy dużym modelu, wartości Pagerank zbiegają dla $n >= 50$[1].

Przykład użycia

W oparciu o przykład z sekcji 1.2.1 przedstawione zostanie inne podejście, z wykorzystaniem łańcuchów Markova, opisanych poprzednio w tym rozdziale.

- Wyznaczamy rozkład początkowy - $\pi(0) = (\,1\;0\;0\;0\;0\,)$
- Wykonujemy algorytm dla $n = 50$ iteracji.

$$\pi(50) = \begin{bmatrix} 0.3108469 & 0.28262402 & 0.03785274 & 0.22982166 & 0.13885468 \end{bmatrix}$$

Wartości π po 50 iteracjach przedstawiają rangi poszczególnych dokumentów. Widać, iż wartości *PageRank* są takie same jak wyniki z poprzedniego przykładu - tabela 1.1.

Korzyści implementacji

Dzięki tego rodzaju implementacji, nie musimy potęgować macierzy P, co zwiększa wydajność algorytmu. W tym podejściu cały czas w pamięci trzymamy stałą, niezmienną macierz P, oraz wektor π, który jest nadpisywany, z każdą iteracją.

Złożoność pojedynczej iteracji spada diametralnie. Mnożenie dwóch kwadratowych macierzy metodą szkolną (podstawową metodą mnożenia macierzy) może być obliczone w czasie $O(n^3)$, natomiast wykorzystując jedne z wydajniejszych metod - $O(n^{2.375477})$ [4]. Metoda wykorzystująca łańcuchy Markova pozwala na wykonywanie każdej z iteracji w czasie $O(n^2)$. Złożoność pamięciowa również spada - metoda pozwala na przetrzymywanie tylko 2 wektorów i 1 macierzy w pamięci, zamiast 3 macierzy, które są wykorzystywane w trakcie wykonywania operacji potęgowania/mnożenia.

1.4.2 Implementacja rozproszona

Poniższa implementacja jest wykorzystywana przez *Apache Spark* - silnik procesujący dane, pozwalający na przetwarzanie rozproszone.
Implementacja posiada dwa warianty:

1. Iterujący określoną n liczbę razy.

2. Liczący do momentu osiągnięcia określonego progru tolerancji.

Ogólna zasada działania jest taka sama jak w przykładzie z sekcji 1.2.1 z tym, że nie jest tworzona macierz prawdopodobieństw P. Algorytm wykorzystuje strukturę grafu, dzięki czemu obliczenia mogą być rozbijane na mniejsze części w celu paralelizacji (zrównoleglenia) zadań.

Wyniki algorytmu *PageRank* w tej implementacji zawierają się w przedziale $[0.0, \infty)$, nie są to wartości prawdopodobieństwa (przedział $[0.0, 1.0]$) tak jak w poprzednich implementacjach. Jeśli zależy nam na wartościach prawdopodobieństwa z danego przedziału, musimy je znormalizować, ale nie jest to konieczne w celu sortowania dokumentów według ich rangi.

Algorytm 2: PageRank - implementacja rozproszona iterująca N liczbę razy

Dane wejściowe: graf $G = (V, E)$; lista połączeń c zbudowana w oparciu o graf G; liczba iteracji N, prawdopodobieństwo losowego skoku α

Wynik: wartości *PageRank* PR_i, odpowiadające V_i

1 $PR \leftarrow [1.0]_n$
2 $oldPR \leftarrow [1.0]_n$
3 **for** $iter \leftarrow 1$ ***to*** N **do**
4 $\quad oldPR \leftrightarrow PR$
5 $\quad$ **for** $i \leftarrow 1$ ***to*** n **do**
6 $\quad\quad PR_i \leftarrow \alpha + (1 - \alpha) * \sum_{\forall_{k \in \{1 \ldots n\}} |c_{ki}| > 0} \frac{oldPR_k}{\sum_{j \in 1 \ldots n} |c_{kj}|}$
7 **return** PR

Algorytm 3: PageRank - implementacja rozproszona iterująca do osiągnięcia zbieżności *tol*

Dane wejściowe: graf $G = (V, E)$; lista połączeń c zbudowana w oparciu o graf G; wartość minimalnej zbieżności *tol*, prawdopodobieństwo losowego skoku α

Wynik: wartości *PageRank* PR_i, odpowiadające V_i

1 $PR \leftarrow [1.0]_n$
2 $oldPR \leftarrow [1.0]_n$
3 **while** $max(abs(PR - oldPR)) > tol$ **do**
4 $\quad oldPR \leftrightarrow PR$
5 $\quad$ **for** $i \leftarrow 1$ ***to*** n **do**
6 $\quad\quad PR_i \leftarrow \alpha + (1 - \alpha) * \sum_{\forall_{k \in \{1 \dots n\}} |c_{ki}| > 0} \frac{oldPR_k}{\sum_{j \in 1 \dots n} |c_{kj}|}$
7 **return** PR

Omówienie algorytmów 2 i 3

1-2 Inicjowanie wektorów prawdopodobieństw PR i $oldPR$ o długościach n wartościami 1.0.

3 W zależności od wariantu algorytmu, jeśli nie przekroczono pożądanej tolerancji *tol*, lub bieżąca liczba iteracji nie przekroczyła N, wykonywane są operację z linii 4-6.

4-6 Dla każdego dokumentu $i \in \{1 \dots n\}$ wartości $oldPR$ i PR zostają zamienione ze sobą, zostaje wyliczona wartość PR_i, której obliczenie może być rozbite na operacje cząstkowe w celu obliczenia w systemie rozproszonym.

7 Wynikiem jest tablica PR, która przechowuje wartości *PageRank* dla danego dokumentu: PR_i odpowiada dokumentowi V_i.

Korzyści implementacji

Do największych korzyści zaliczamy, wspominaną wcześniej, możliwość paralelizacji zadań, gdyż obliczenia dla każdego z połączeń w grafie mogą być wykonywane przez osobne maszyny/wątki. Drugą największą korzyścią jest zmniejszenie złożoności obliczeniowej algorytmu. Działamy tylko na wektorach, iterujemy po wektorach, liczba potrzebnych operacji jest zmniejszona do minimum.

Rozdział 2

Przegląd literatury w dziedzinie oraz istniejące rozwiązania

W tym rozdziale omówione zostanie kilka rozwiązań powiązanych z tematem pracy.

2.1 Wprowadzenie

Poniżej przybliżę podstawowe pojęcia powiązane z silnikiem wyszukiwań *Elasticsearch*, po których nastąpi jego omówienie.

2.1.1 Indeksowanie dokumentów

Indeks to struktura pozwalająca na szybki dostęp do dokumentów według danego klucza, który wskazuje na dokumenty z nim powiązane.

Indeksowanie dokumentów w kontekście wyszukiwarek [13] można podzielić na:

- zbieranie danych - dokumenty mogą być przekazywane do indeksowania poprzez jeden lub kilka określonych sposobów (np. zapytaniem restowym w formie obiektów JSON).

- parsowanie - forma przetwarzania dokumentu polegająca na sprawdzeniu, czy zgadza się on z docelowym formatem jego składowania, a następnie na wykonaniu tokenizacji, normalizacji dokumentu, w celu dalszego składowania

- składowanie danych - zapisanie dokumentu w formie wejściowej, oraz dodanie go do struktur zwiększających wydajność wyszukiwania, np. indeksu odwróconego omówionego w kolejnej sekcji.

Indeks odwrócony

Indeks odwrócony [13][14] to indeks zbudowany na zbiorze znormalizowanych słów[1] t i dokumentów d ich zawierających. Każdy wpis w indeksie daje dostęp do dokumentów, które zawierają słowo t, wraz z pozycjami w tekście, na której ono się znajduje. Struktura ta pozwala na zwiększenie wydajności wyszukiwania słów w dokumentach.

$\boldsymbol{i}$	$\boldsymbol{d_i}$
1	Ala ma kota
2	Nie lubię nowych butów
3	Mamy nowego kota
4	To kot w butach
5	Muszę kupić nowe buty dla moich nowych kotów

Tab. 2.1: Indeks dokumentów (*forward index*)

$\boldsymbol{t}$	$\boldsymbol{t \in d}$
ala	$(d_1 : 0)$
kot	$(d_1 : 7)$, $(d_3 : 12)$, $(d_4 : 3)$, $(d_5 : 39)$
but	$(d_2 : 17)$, $(d_4 : 9)$, $(d_5 : 15)$
mieć	$(d_1 : 4)$, $(d_3 : 0)$
lubić	$(d_2 : 4)$
nie	$(d_2 : 0)$
nowy	$(d_2 : 10)$, $(d_3 : 5)$, $(d_5 : 12)$
kupić	$(d_5 : 6)$
musieć	$(d_5 : 0)$

Tab. 2.2: Indeks odwrócony (*inverse index*) - dokumenty d z tabeli 2.1 zawierające słowa t

Tabela 2.2 przedstawiająca przykład indeksu odwróconego na podstawie 2.1, posiada wpisy formatu t - d, np. lubić - $(d_2 : 4)$, oznacza że dowolna forma słowa "lubić" jest poprzedzona 4 znakami i znajduje się w dokumencie d_2.

[1] słowa znormalizowane - sprowadzone do formy podstawowej, po lematyzacji, lowercasingu

2.1.2 Ocena wyników wyszukiwania z wykorzystaniem *td-idf*

Poniższa sekcja w oparciu o [7] przedstawia jedną z najpopularniejszych miar relewantności, którą jest *td-idf*.

Metoda ta wykorzystuje frekwencję wyrazów t w dokumentach d - $tf_{t,d}$. Poniżej przedstawiony jest przykład tabeli tf - 2.3.

d \ t	ala	kot	but	mieć	lubić	nie	nowy	kupić	musieć
d_1	1	1	0	1	0	0	0	0	0
d_2	0	0	1	0	1	1	1	0	0
d_3	0	1	0	1	0	0	1	0	0
d_4	0	1	1	0	0	0	0	0	0
d_5	0	1	1	0	0	0	2	1	1

Tab. 2.3: Tabela frekwencji wyrazów tf dla 2.1

Wykorzystanie samej frekwencji wyrazów nie wystarcza, aby określić relewantność dokumentu względem zapytania - długi dokument zawierający dużą liczbę wyrazów z zapytania zawsze by wygrywał w przypadku wykorzystania poniższego wzoru (2.1).

$$Score_{tf}(q,d) = \sum_{t \in q} tf_{t,d} \tag{2.1}$$

Z tego też powodu została wprowadzona dodatkowa miara - odwrotna frekwencja wyrazów w dokumencie - idf_t określona wzorem (2.3), gdzie N jest liczbą zindeksowanych dokumentów, a df_t to liczba dokumentów zawierających wyraz t, którą możemy wyznaczyć wykorzystując (2.2).

$$df_t = \sum_{d \in D} min(1, tf_{t,d}) \tag{2.2}$$

$$idf_t = \log \frac{N}{df_t} \tag{2.3}$$

Wartość idf_t dla wyrazów t występujących rzadko jest wyższa, natomiast wyrazy występujące bardzo często dają niską wartość.

Mając wyznaczone poniższe miary możemy zdefiniować $tf\text{-}idf_{t,d}$ poprzez wzór (2.4).

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t \tag{2.4}$$

Relewantność zapytania q wyznaczana jest wzorem (2.5) wykorzystującym $tf\text{-}idf_{t,d}$ poprzez sumę wartości po każdym z termów. Zapytanie q musi być znormalizowane metodami wykorzystującymi te same reguły, co dla dokumentów d.

$$Score(q, d) = \sum_{t \in q} tf\text{-}idf_{t,d} \tag{2.5}$$

Dla przykładowego zapytania "nowy kot w butach" przedstawię kolejne kroki wyznaczające relewantność dokumentów ze zbioru D.

- Tokenizacja i normalizacja zapytania daje $q = [nowy, kot, but]$[2].
- Dla każdego wyrazu t obliczana jest wartość idf_t.
  - $N = 5$
  - $t_1 =$ "$nowy$", $df_{t_1} = 3$, $idf_{t_1} = log(\frac{N}{df_{t_1}}) = log(\frac{5}{3}) \approx 0.222$.
  - $t_1 =$ "kot", $df_{t_2} = 4$, $idf_{t_2} = log(\frac{N}{df_{t_2}}) = log(\frac{5}{4}) \approx 0.097$.
  - $t_2 =$ "but", $df_{t_3} = 3$, $idf_{t_3} = log(\frac{N}{df_{t_3}}) = log(\frac{5}{3}) \approx 0.222$.
- Wyznaczany jest $Score$ dla każdego z dokumentów.

$\boldsymbol{i}$	$\boldsymbol{Score(q, d_i)}$
1	$0 * 0.222 + 1 \cdot 0.097 + 0 \cdot 0.222 = 0.097$
2	$1 * 0.222 + 0 \cdot 0.097 + 1 \cdot 0.222 = 0.444$
3	$1 * 0.222 + 1 \cdot 0.097 + 0 \cdot 0.222 = 0.319$
4	$0 * 0.222 + 1 \cdot 0.097 + 1 \cdot 0.222 = 0.319$
5	$2 * 0.222 + 1 \cdot 0.097 + 1 \cdot 0.222 = 0.763$

Tab. 2.4: Wartości $Score$ dla dokumentów z 2.1 i zapytania "nowy kot w butach"

- Zwracane dokumenty są uszeregowane według wartości $Score$ z tabeli 2.4 - im wyższa wartość, tym dokument jest bardziej relewantny, bardziej zgodny z zapytaniem.

Zatem dla zapytania "nowy kot w butach" wyszukiwarka oparta na *tf-idf* zawierająca indeks składający się z dokumentów ze zbioru D zwróci je w kolejności: 5,2,3,4,1.

[2]"w" nie występuje w q, ponieważ tzw. *stop words* - mało znaczące, w kontekście przetwarzania zapytania, wyrazy, np. spójniki, przyimki, zaimki, są odrzucane w trakcie tokenizacji

2.1.3 Elasticsearch - omówienie

Poniższe omówienie zostało stworzone w oparciu o [10].

Elasticsearch jest silnikiem wyszukiwań oraz analizy danych, który pozwala na szybkie składowanie, wyszukiwanie, oraz analizę dużych ilości danych, niemalże w czasie rzeczywistym. Cechuje go wysoka skalowalność, jego kod jest otwarto-źródłowy. Najczęściej jest wykorzystywany jako silnik wyszukiwań dla aplikacji, których zapytania są skomplikowane, tzn. złożone z różnych pól, wykorzystują sortowanie lub agregację.

Domyślną miarą relewantności w *Elasticsearch* jest metoda wykorzystująca miarę *tf–idf* w połączeniu z normalizacją względem długości pola, po którym następuje wyszukiwanie.

Komunikacja z serwerem odbywa się z wykorzystaniem zapytań REST[3], z których pomocą mamy możliwość między innymi wykonywania poniższych operacji:

- Sprawdzanie klastra, węzła oraz indeksu w kontekście statusu sprzętowego, statystyk.
- Wykonywanie operacji CRUD[4] oraz wyszukiwania na indeksach.
- Administracja danymi oraz metadanymi klastra, węzła i indeksu.
- Wykonywanie zaawansowanych zapytań wyszukujących, z możliwością użycia funkcji stronicowania, sortowania, filtrowania, skryptów, agregacji lub innych.

2.1.4 Koncepcje wykorzystywane w Elasticsearch

Klaster

Klaster to zbiór jednego lub więcej węzłów, przechowujących dane, dostarczający ujednolicone mechanizmy indeksowania i wyszukiwania na każdym z węzłów. Każdy klaster identyfikowany jest poprzez unikalną nazwę, domyślnie "elasticsearch". Nazewnictwo jest kluczowym elementem, ponieważ poprzez nazwę przypisujemy węzły do klastra po jego nazwie.

[3]REST - Representational state transfer lub RESTful to styl bezstanowej komunikacji pomiędzy zasobami serwera, lub ich reprezentacji.

[4]CRUD - podstawowe operacje na zasobie: Create, Read, Update, Delete

Węzeł

Węzeł to pojedynczy serwer, który jest częścią klastra. Przechowuje on dane i uczestniczy w indeksowaniu oraz wyszukiwaniu danych w klastrze. Tak jak w przypadku klastra, węzeł identyfikowany jest poprzez nazwę, a domyślnie przypisany jest mu losowo wygenerowany ciąg znaków.

Każdy z serwerów może dołączyć do konkretnego klastra, a domyślnie węzły dołączają do klastra o nazwie "elasticsearch". Oznacza to, że jeśli uruchomionych zostanie kilka węzłów w naszej lokalnej sieci, zakładając, że są dla siebie widoczne, automatycznie połączą się w jeden klaster o nazwie "elasticsearch".

Jeśli aktualnie w sieci nie zostanie wykryty inny węzeł, to uruchomienie serwera spowoduje stworzenie nowego klastra, z pojedynczym węzłem, o nazwie "elasticsearch".

Indeks

Indeks to zbiór podobnie skonstruowanych dokumentów. Na indeksie wykonywane są operacje wyszukiwania, indeksowania, uaktualniania danych, usuwania, względem dokumentów, które zawiera.

Klaster może zawierać dowolną liczbę indeksów.

Typ

Typy to struktury, które są definiowane dla danego indeksu. Typ to część składowa indeksu, którą definiujemy dowolnie. Głównie typy definiuje się dla dokumentów, które mają wspólne pola i należą do tej samej klasy/typu.

Dla przykładu, jeśli w indeksie składujemy dokumenty, których struktura jest w większości taka sama, to możemy wprowadzić kilka typów na jednym indeksie, w przeciwnym przypadku, lepszym rozwiązaniem jest kilka indeksów, dla każdego indeksu osobny typ.

Dokument

Dokument to podstawowa jednostka informacji, które może być zindeksowana. Dokumenty są przedstawiane jako obiekt JSON[5], który jest wszechobecnym formatem prezentowania danych w internecie.

[5] `http://json.org/`

Shard

Indeksy potencjalnie mogą składować duże ilości danych, które mogą napotkać sprzętowe ograniczenia dla pojedynczego węzła. Dla przykładu, pojedynczy indeks *Elasticsearch* z bilionem dokumentów zajmujący 1TB przestrzeni dyskowej może nie zmieścić się na dysku węzła, lub jako pojedynczy indeks, zbyt wolno radzić sobie z zapytaniami.

Z rozwiązaniem przychodzi mechanizm podziału indeksu na fragmenty (ang. *shard*). W trakcie tworzenia indeksu definiujemy liczbę shardów. Każdy fragment to w pełni funkcjonalny samodzielny "indeks", który może być wystawiony przez jakikolwiek z węzłów w klastrze - ten sam fragment może być na innym z węzłów jako replika.

Takie rozwiązanie daje nam dwie podstawowe korzyści:

- Pozwala na podzielenie obszaru danych.
- Pozwala na dystrybucję i paralelizację zadań na kilka fragmentów zwiększając wydajność/przepustowość.

Elasticsearch w pełni zarządza podziałem indeksu na fragmenty, oraz dystrybucją, paralelizacją zapytań na poszczególne fragmenty. Modyfikacja liczby shardów nie jest możliwa, definiujemy ją w trakcie tworzenia indeksu.

Replika (ang. *replica*)

Środowiska budowane w większych sieciach, bądź w chmurze, są bardziej podatne na zaistnienie błędów, z uwagi na większe zużycie, zapotrzebowanie zasobów, dlatego też niezbędny jest mechanizm, który zapobiega wystąpieniu przerw w dostawie usługi - węzłów, a dokładniej ich fragmentów. Takim mechanizmem jest replikacja, która pozwala na stworzenie jednej lub więcej kopii shardów w danym indeksie, które zwane są replikami.

Główny shard to pierwszy egzemplarz fragmentu, taki, z którego tworzone są kopie.

Replikacja daje nam 2 podstawowe korzyści:

- Zwiększa dostępność, w przypadku uszkodzenia fragmentu/węzła. Warto wspomnieć, że replika nie powinna znajdować się na tym samym węźle, co główny shard, którego replikuje.

- Pozwala na zwiększenie wydajności, ponieważ zapytania mogą być wykonywane równolegle na wszystkich replikach.

Liczba replik może być zmieniona po utworzeniu indeksu, zatem w każdej chwili możemy dodać kolejne repliki, aby zwiększyć dostępność danych. Domyślnie każdy index *Elasticsearch* jest alokowany na pięciu głównych fragmentach i jednej replice.

2.1.5 Przykład działania

W tej sekcji przedstawię kilka podstawowych operacji na *Elasticsearch* z wykorzystaniem komendy *curl*[6].

Sprawdzanie stanu klastra

Aby sprawdzić stan klastra, w najprostszy sposób, należy wykonać poniższe zapytanie *curl*, lub też po prostu odwiedzić dany adres w przeglądarce.

```
1 curl "localhost:9200/_cat/health?v"
```

Blok kodu 2.1: Komenda sprawdzająca stan klastra

Dodawanie indeksu

W celu dodania nowego indeksu, wykonujemy zapytanie PUT na działającym serwerze:

```
1 curl -XPUT 'localhost:9200/test?pretty'
```

Blok kodu 2.2: Komenda tworząca indeks

gdzie "test" to nazwa naszego indeksu. Tak jak było wspomniane w poprzedniej sekcji, domyślnie indeks jest tworzony z 5 głównymi shardami i 1 repliką. Możliwe jest podanie dodatkowych ustawień dla indeksu, w formie obiektu JSON:

```
1 curl -XPUT 'localhost:9200/twitter?test' -H 'Content-Type:
      application/json' -d'
2 {
3     "settings" : {
4         "index" : {
```

[6]cURL - biblioteka, narzędzie pozwalające na transmisje danych wykorzystując różne z protokołów, w tym HTTP REST API

```
            "number_of_shards" : 8,
            "number_of_replicas" : 2
        }
    }
}'
```

Blok kodu 2.3: Komenda tworząca indeks z dodatkowymi ustawieniami

Dodawanie dokumentu

Poniżej znajduje się dokument 2.4, który zostanie dodany do indeksu *test* za pomocą komendy 2.5. Odpowiedź z serwera 2.6 na dodanie dokumentu z *id* równym 1 o typie *default* jest pozytywna - dokument został poprawnie dodany do indeksu *test*.

```
{
    "firstname": "John",
    "lastname": "Doe",
    "age": 25,
    "gender": "M"
}
```

Blok kodu 2.4: Przykładowy dokument o nazwie *inputData.json*

```
curl -XPOST 'localhost:9200/test/default/1?pretty' --data-
   binary @inputData.json
```

Blok kodu 2.5: Komenda dodająca dokument do indeksu.

```
{
  "_index" : "test",
  "_type" : "default",
  "_id" : "1",
  "_version" : 1,
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "created" : true
}
```

Blok kodu 2.6: Odpowiedź z serwera po wykonaniu komendy 2.5

Zapytanie o dokument

Znając id dokumentu, możliwe jest wykonanie zapytania zwracającego zawartość danego dokumentu.

```
curl -XGET 'localhost:9200/test/default/1?pretty'
```

Blok kodu 2.7: Komenda z zapytaniem o dokument

```
{
  "_index" : "test",
  "_type" : "default",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : {
    "firstname" : "John",
    "lastname" : "Doe",
    "age" : 25,
    "gender" : "M"
  }
}
```

Blok kodu 2.8: Odpowiedź z serwera po wykonaniu komendy 2.7

Usuwanie dokumentu

Analogicznie do zapytania o dokument, aby usunąć go wystarczy wykonać zapytanie DELETE, na dany adres.

```
curl -XDELETE 'localhost:9200/test/default/1?pretty'
```

Blok kodu 2.9: Komenda usuwająca dokument

Proste zapytanie wyszukujące

Jest wiele możliwości stworzenia zapytania wyszukującego, ale poniżej przedstawiona zostanie jedna z podstawowych jego postaci. Poniższa komenda wyszukuje, po wszystkich polach w indeksie *test*, ciągów znaków które zaczynają się od litery "J".

```
curl 'localhost:9200/test/_search?q=J*&pretty'
```

Blok kodu 2.10: Komenda wyszukująca dokumenty

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 2,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "test",
      "_type" : "default",
      "_id" : "2",
      "_score" : 1.0,
      "_source" : {
        "firstname" : "Jane",
        "lastname" : "Doe",
        "age" : 24,
        "gender" : "F"
      }
    }, {
      "_index" : "test",
      "_type" : "default",
      "_id" : "1",
      "_score" : 1.0,
      "_source" : {
        "firstname" : "John",
        "lastname" : "Doe",
        "age" : 25,
        "gender" : "M"
      }
    } ]
  }
}
```

Blok kodu 2.11: Odpowiedź z serwera po wykonaniu komendy 2.10

2.2 Algorytm PageRank i artykuły naukowe

Zasada działania algorytmu *PageRank* została przedstawiona dokładnie w rozdziale 1, natomiast w tej sekcji przybliżona zostanie jedna z prac na temat wykorzystania algorytmu *PageRank* w ocenianiu artykułów naukowych.

Publikacja naukowa pt. *Finding Scientific Gems with Google*[6] jest jedną z pierwszych publikacji przedstawiających wykorzystanie algorytmu *PageRank* bezpośrednio w kontekście oceny artykułów naukowych.

Artykuł opisuje zastosowanie algorytmu *PageRank* w celu określenia ważności artykułów naukowych w publikacjach z dziedziny fizyki z lat 1893-2003. Wartość *PageRank* i liczby cytowań danej publikacji, są bezpośrednio skorelowane, jednakże pomimo tego można wyszczególnić niezwykłe dzieła lub "perełki", które są uniwersalnie wykorzystywane w tej dziedzinie.

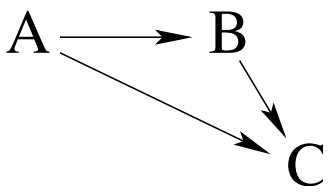
Autorzy ewaluują ten algorytm na bazie sieci złożonej z 353.268 węzłów reprezentujących artykuły oraz 3.110.839 połączeń pomiędzy nimi.

Główną obserwacją w tej pracy są artykuły cytowane małą liczbę razy, posiadające dużą wartość *PageRank*. Jeden z analizowanych artykułów, "The Theory of Complex Spectra", J. C. Slater, znajdujący się aż na 10 pozycji w bazie wg *PageRank*, jest cytowany tylko 114 razy (największa wartość cytowań to 3227), będąc na 1853 pozycji według liczby cytowań. Ta publikacja wprowadziła ostateczną formę wielociałowej funkcji falowej, która jest tak często używana w literaturze, pomimo że mało który artykuł cytuje bezpośrednio publikacje Slatera. Ranga wyznaczona przez algorytm *PageRank* wykryła ten artykuł jako "ukrytą perełkę" wśród innych, ponieważ wiele artykułów odnoszących się właśnie do publikacji Slatera, jest dobrych - ma wiele cytowań.

Potwierdza to fakt, iż bazowanie na liczbie cytowań do artykułu jest niewystarczające, aby stwierdzić czy publikacja jest dobra. Zatem można stwierdzić, iż *PageRank* może byc nową użyteczną miarą w mierzeniu jakości naukowej publikacji.

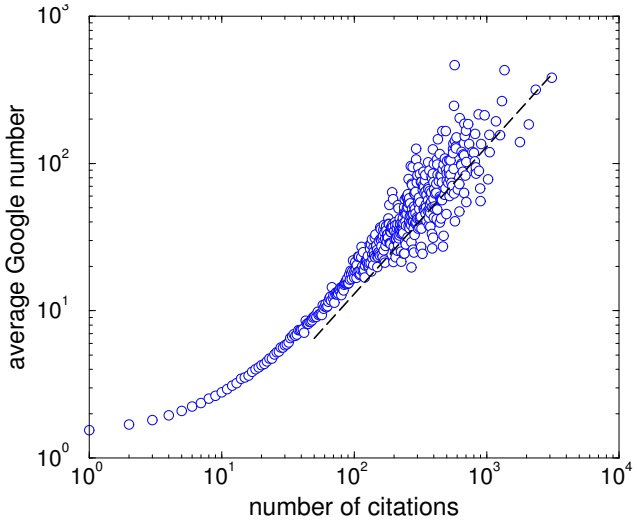
Autorzy również zastanawiają się nad tym, jaka wartość α przynosi najlepsze rezultaty i jaki ma wpływ na wyniki. Przytoczona jest publikacja nt. algorytmu *PageRank*[12], gdzie parametr α został ustalony na wartość 0.15, jednakże autorzy twierdzą, że dla sieci cytowań lepszym rozwiązaniem jest przyjęcie $\alpha = 0.5$. Wywnioskowali to z faktu, iż sieci stron web są zbudowane inaczej niż sieci cytowań/referencji do artykułów. Sieci artykułów są bardziej ściśle powiązane ze sobą, gdzie w sieci web możemy mieć więcej przestrzeni pomiędzy połączeniami. Brin i Page przyjęli wartość $\alpha = 0.15$, ponieważ $\alpha = 1/6 \approx 0.15$, 6 jest przyjętą typową liczbą linków na stronie, za którą podąży surfer, zanim

zostanie znudzony lub sfrustrowany i zmieni swoje zapytanie w wyszukiwarce. Przyjęcie wartości $\alpha = 1/2 = 0.5$, było spowodowane tym, że domyślna lista pętli w referencjach najczęściej odnosi się do artykułów w dużej mierze bliższych sobie. Patrząc na rysunek 2.1, możemy stwierdzić, iż aby dojść alternatywną ścieżką z artykułu A do C, przechodząc przez B, potrzebujemy tylko 2 kroków, a w przypadku stron sieci web zakładano, iż kroków będzie więcej.



Rys. 2.1: Pętla cytowań: Publikacja A cytuje obydwie publikacje B i C. Około 50% referencji do B w publikacji A cytuje co najmniej jeden kolejny raz artykuł C, który znajduje się w tej samej liście. Źródło: [6]

Godne uwagi jest również stwierdzenie iż, w miarę jak α dąży do 1, wynik *PageRank* zbliża się do wartości ważności dokumentów ustalonej wg liczby cytowań. Na wykresie 2.2 można zauważyć, iż wartości zbliżają się do przebiegu linowego, co może również wynikać z charakterystyki danych wejściowych.



Rys. 2.2: Wykres ukazujący zależność wartości algorytmu *PageRank* dla $\alpha = 0.5$ (average Google number) od liczby cytowań. Źródło: [6]

Na końcu pracy następuje wniosek, iż *PageRank* jest obiecującą metodą do wykorzystania w ocenianiu artykułów naukowych. Mógłby być rozszerzeniem dla istniejących metod oceny ważności publikacji. Naukowcy również znajdują powiązane publikację poprzez przeglądanie artykułów cytowanych przez inne, co przypomina podążenia po linkach przez losowego surfera. Jedyną różnicą jest fakt, iż referencje w artykule nie mogą być zmienione po jego publikacji, natomiast sieć WWW stale ewoluuje razem z linkami do innych stron.

2.3 Elasticsearch w wyszukiwaniu artykułów naukowych

Sekcja ta przedstawia rozwiązanie wykorzystujące silnik wyszukiwań *Elasticsearch*, w kontekście artykułów naukowych.

Publikacja pt. *Research Document Search using Elastic Search*[9] przedstawia wykorzystanie silnika wyszukiwań *Elasticsearch* w wyszukiwaniu artykułów naukowych.

We wprowadzeniu autorzy przedstawiają problem "big data", ogromnych ilości danych, z którymi relacyjne bazy danych nie mogą sobie poradzić. RDBMS[7] są nastawione na to, aby dane były usystematyzowane według ustalonego schematu. Wspierają metody procesowania danych, oraz transakcyjność, natomiast dzisiejsze dane często są częściowo lub w ogóle nie ustrukturyzowane. Zjawisko to spowodowało powstanie baz NoSQL, które radzą sobie z dużą ilością danych, są skalowalne i wysoce dostępne, kosztem braku relacji, struktur, transakcyjności.

W następnym rozdziale następuje krótkie wprowadzenie historii powstania *Elasticsearch*, po czym następuje opis działania silnika, oraz "workflow" operacji wyszukiwania.

Dane wykorzystane w tej publikacji pochodzą z różnych źródeł, ale część z nich została zaczerpnięta z *ACM Digital Library* oraz *Google Scholar*.

Projekt stworzony w tym artykule wykorzystuje framework *Django* w implementacji interfejsu graficznego, dającego dostęp z zewnątrz do wyszukiwań w *Elasticsearch*.

Wyszukiwanie artykułów może odbywać się w dwóch trybach:

[7] Relational Database Management System - systemy zarządzania relacyjną bazą danych

- wyszukiwanie normalne - szuka dokumentów, wg wprowadzonego tekstu, po tytule,
- wyszukiwane zaawansowane - pozwala na wyszukiwanie po autorze, dacie publikacji i słowach kluczowych, które przeszukują samą treść dokumentu.

Dopasowane artykuły prezentowane są na stronie z wynikami wyszukiwania wraz ze swoimi parametrami: tytuł, autor, data publikacji, podgląd artykułu. W artykule brakuje informacji o sortowaniu wyników, więc można przyjąć, że są one ułożone według domyślnej miary relewantności zapytania w *Elasticsearch*, o której była mowa w sekcji 2.1.3.

We wnioskach pada stwierdzenie, że *Elasticsearch* jest łatwo można przystosować do konkretnego przypadku użycia.

Rozdział 3

Opis rozwiązania autorskiego

W rozdziale 2 opisane zostały dwa rozwiązania związane z tematem pracy. Pierwsze z nich (2.2) mówiło o wykorzystaniu algorytmu *PageRank* w kontekście badania ważności artykułów naukowych, natomiast drugi z artykułów (sekcja 2.3) omawiał wyszukiwaniu dokumentów naukowych z użyciem *Elasticsearch*. Pomimo długich poszukiwać nie udało się znaleźć artykułu, który łączyłby te dwa zagadnienia, dlatego też rozwiązanie zawarte w tej pracy, wydaje się unikalne. Poniżej przedstawię opis rozwiązania wykorzystującego algorytm *PageRank* i silnik wyszukiwań *Elasticsearch* w kontekście artykułów naukowych.

Warto nadmienić, że artykuł z rozdziału 2.2 został znaleziony z wykorzystaniem rozwiązania autorskiego.

3.1 Opis użytych technologii

Sekcja ta krótko przybliży informacje o technologiach, bądź systemach, użytych w projekcie, które nie zostały omówione wcześniej.

3.1.1 Jupyter Notebook

Jupyter Notebook[1] to otwarto-źródłowa aplikacja webowa, pozwalająca na tworzenie oraz dzielenie się dokumentami, które zawierają: kod uruchamiany w czasie rzeczywistym, równania matematyczne i wizualizację. Aplikacja jest często

[1] `http://jupyter.org/`

wykorzystywana w symulacjach numerycznych, statystyce, uczeniu maszynowym i wielu innych.

W skład architektury aplikacji wchodzą m.in.:

- notatnik, którego format bazuje na obiektach JSON, zawierających pełny wgląd w ostatnią sesję użytkownika na dokumencie, bloki kodu, tekst, definicję równań matematycznych oraz ostatnie wyjście uruchamianego wcześniej kodu,

- protokół obliczeniowy, który wysyła obiekty JSON za pośrednictwem ZMWQ[2] lub WebSockets[3] do jednostek obliczeniowych (ang. *kernel*),

- kernel, który odpowiada za procesy uruchamiające kod w czasie rzeczywistym, w wybranym języku, którego wywołanie zwróci wynik do użytkownika.

System wspiera kernele działające na takich językach programowania jak: Python, R, Julia oraz Scala. Kernel może również działać we współpracy z silnikiem *Apache Spark*, z którym komunikujemy się za pośrednictwem języków: Python, R lub Scala. Ponadto, stosując dany język, można korzystać ze wszystkich bibliotek zainstalowanych w środowisku uruchomieniowym.

Aplikacja zapewnia duże wsparcie w tworzeniu wykresów w bibliotece *matplotlib*, pozwalającej na tworzenie wykresów w składni przypominającej tę ze środowiska MATLAB[4], oraz pisaniu równań matematycznych w składni *LaTeX*. Właśnie przez te cechy, aplikacja stosowana jest przez pracowników naukowych, którzy mogą tworzyć modyfikowalne i interaktywne prezentacje podwyższające jakość zajęć przez nich prowadzonych.

Jupyter Notebook został wykorzystany w rozwiązaniu autorskim do ewaluacji danych, generowania wykresów, pracy na silniku *Apache Spark* oraz wstępnej obróbki artykułów naukowych.

[2]ZMQ - biblioteka pozwalająca na asynchroniczną komunikację za pomocą wiadomości w aplikacjach stawiających na współbieżność.

[3]WebSockets - protokół połączeniowy pomiędzy przeglądarką a serwerem webowym pozwalający na przesyłanie i odbieranie danych bez przeładowania strony web.

[4]MATLAB - środowisko obliczeniowe, pozwalające na wykonywanie matematycznych operacji, często używane w środowisku akademickim, oraz przez inżynierów, ekonomistów.

3.1.2 Apache Spark

Apache Spark[5] to otwarto-źródłowy silnik przetwarzania danych, zbudowany z myślą o łatwości użycia, wysokiej wydajności oraz użyciu wyspecjalizowanych metod analitycznych. Wykorzystywany jest przez wiele organizacji, m.in. takich jak eBay, Netflix, Yahoo, których klastry sięgają 8000 lub więcej węzłów.

Architektura silnika przypomina *Elasticsearch* - wiele węzłów połączonych w jeden klaster obliczeniowy.

Zapytania wyodrębniające dane z *DataFrames* (baza danych w *Apache Spark*) mają składnię SQL. Wykonywane operacje na silniku *Spark* mogą być zdefiniowane z wykorzystaniem takich języków jak: R, Python, Scala oraz Java. Silnik wspiera również operację na grafach z wykorzystaniem biblioteki *GraphX*. Może być zintegrowany ze środowiskiem *Jupyter Notebook*.

W tej chwili *Apache Spark* jest najszybszym, najwydajniejszym silnikiem obliczeniowym, który może być wykorzystywany do wielu rozwiązań, takich jak wyszukiwanie korelacji w ogromnych bazach danych, np. korelacja stanu pogody, godziny i liczby wypadków w kraju.

3.1.3 AngularJS

AngularJS[6] to framework aplikacji webowych napisany w języku JavaScript, bazujący na architekturze MVW[7]. Framework pozwala na rozszerzenie HTML, o dodatkowe obiekty, zdefiniowane wcześniej przez framework lub przez aplikację, dzięki czemu kod aplikacji jest czytelny, łatwy w zrozumieniu i dalszym rozwijaniu. Każda część frameworku może być rozszerzona, lub nadpisana, co daje pełną swobodę programiście.

Strony HTML to widoki, które mogą być zarówno przypisane do całego kontrolera - ścieżki dostępu, jak i mogą być *partialami*, czyli widokami jednej z dyrektyw, kontrolerów podrzędnych.

Kontrolery definiują połączenie pomiędzy widokami a modelem. Każda aplikacja musi posiadać główny kontroler. Preferowanym jest również, aby każdy widok miał swój własny kontroler.

Dyrektywy to definicje elementów, które są często używane, w celu zapobiegnięcia redundancji, duplikacji kodu. Mogą być zdefiniowane jako element

[5] `https://spark.apache.org/`

[6] `https://angularjs.org/`

[7] Model-View-Whatever - architektura dająca developerowi swobodę nad wyborem typu architektury, który będzie najbardziej odpowiedni, np. Model-View-ViewModel.

HTML, bądź atrybuty elementu. Tak zdefiniowane obiekty, mają swoją własną logikę, która wykonuje operację poprzez przekazane parametry. Framework po prostu dołącza zawartość dyrektywy w miejsce jej deklaracji w widoku aplikacji.

Framework daje również możliwość definiowania fabryk, providerów oraz serwisów, za pomocą których możemy izolować kod, który jest niezależny od kontrolerów, co zwiększa czytelność kodu aplikacji. Te obiekty są wstrzykiwane do kontrolera za pomocą mechanizmu zwanego *dependency injection*. *AngularJS* tworzy obiekt, i wstrzykuje go w trakcie tworzenia obiektu, w którym chcemy go wykorzystać.

AngularJS jest aktualnie wykorzystywany przez wiele serwisów. Jednym z wartych uwagi jest *Endomondo* - serwis do dzielenia się swoimi osiągnięciami sportowym, który jest przykładem prawdziwego wykorzystania frameworka, w aplikacji, z której korzystają miliony użytkowników. Te wykorzystanie daje potwierdzenie, iż *AngularJS* jest dobrym frameworkiem do wykorzystania w środowisku złożonym z ogromnej liczby użytkowników, przy czym nie traci on na atrakcyjności, czy wydajności.

3.2 Projekt wyszukiwarki artykułów

3.2.1 Pochodzenie i charakterystyka danych

Baza danych została stworzona na podstawie danych pochodzących z elektronicznego archiwum arXiv[8], a dokładniej ze zbioru danych, na podstawie którego zbudowany jest serwis Paperscape[3], udostępniający graf połączeń pomiędzy artykułami naukowymi.

Sieć stworzona z poprzednio wspomnianej bazy zawiera 1.106.142 węzłów (artykułów) oraz 10.498.461 połączeń między nimi. Połączenia są binarne, tzn. że nie uwzględnione są liczby cytowań danej publikacji w artykule. Jeśli artykuł jest cytowany przez drugi, to jest między nimi połączenie o wadze 1.

Eksperymentalny dobór α

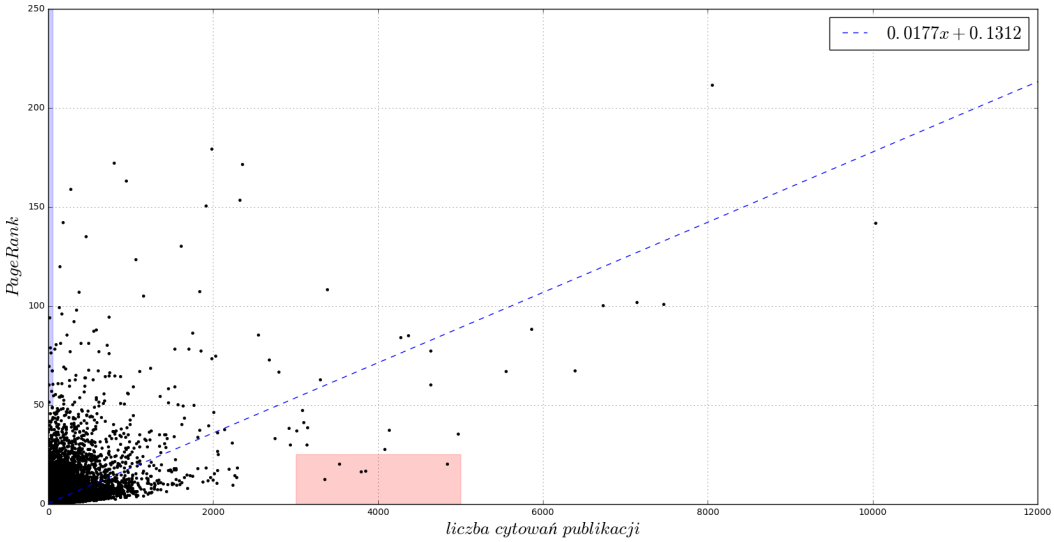
Jak już wspomniałem wcześniej, w pierwszym z dokumentów Brina i Page'a nt. algorytmu *PageRank* [12], parametr α został ustalony na wartość 0.15, ale w artykule opisanym w sekcji 2.2 jest mowa, że dla sieci artykułów naukowych, lepsze wyniki daję wartość $\alpha = 0.5$. Poddałem to własnej ewaluacji, którą przedstawiam poniżej.

[8] `https://arxiv.org`

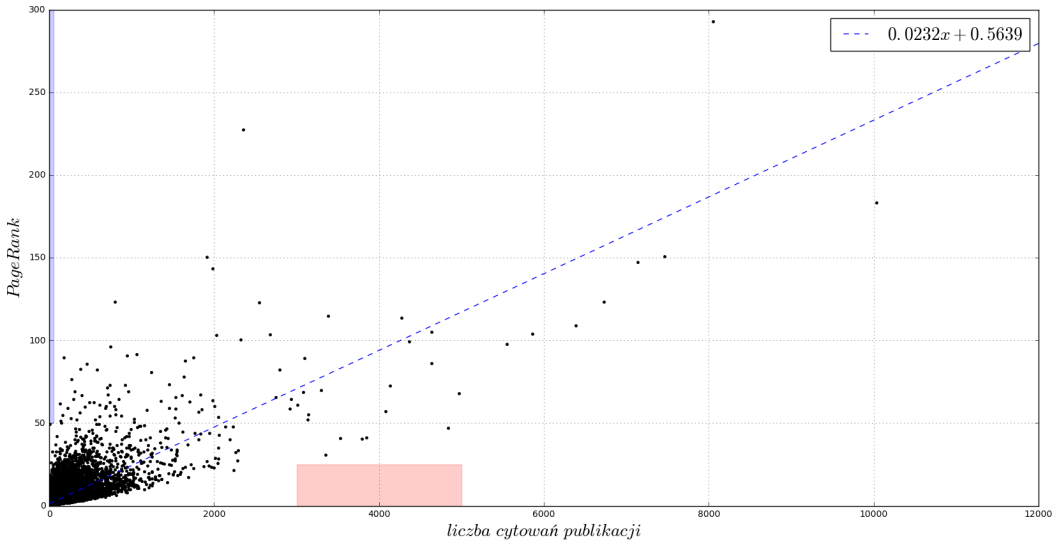
	$PR(\alpha = 0.10)$	$PR(\alpha = 0.15)$	$PR(\alpha = 0.20)$	$PR(\alpha = 0.50)$
$PR(\alpha = 0.10)$	1.	0.99999671	0.99999518	0.99999647
$PR(\alpha = 0.15)$	0.99999671	1.	0.99999673	0.99999647
$PR(\alpha = 0.20)$	0.99999518	0.99999673	1.	0.99999646
$PR(\alpha = 0.50)$	0.99999647	0.99999647	0.99999646	1.

Tab. 3.1: Wartości korelacji Spearmana dla wyników algorytmu *PageRank* dla różnego α

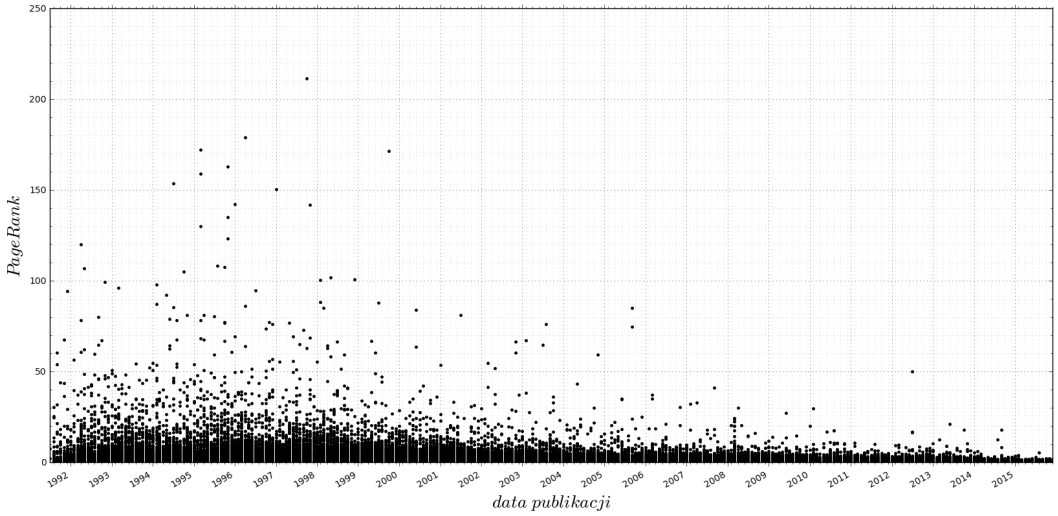
Z tabeli 3.1 wynika, że wartości dla różnych α są mocne skorelowane, jednakże jeśli spojrzymy na wykres 3.1 oraz 3.2, możemy dostrzec, że zgodnie z faktem podanym w sekcji 2.2, wraz ze wzrostem α, wartości *PageRank* przybliżają się do liczby cytowań, ku regresji liniowej wyprowadzonej na podstawie wszystkich danych. Wykresy 3.3 oraz 3.4 prezentują zależność *PageRank* od daty, tutaj też widać nieznaczne pochylenie wartości ku poziomowi dla $\alpha = 0.5$ w porównaniu do $\alpha = 0.1$.



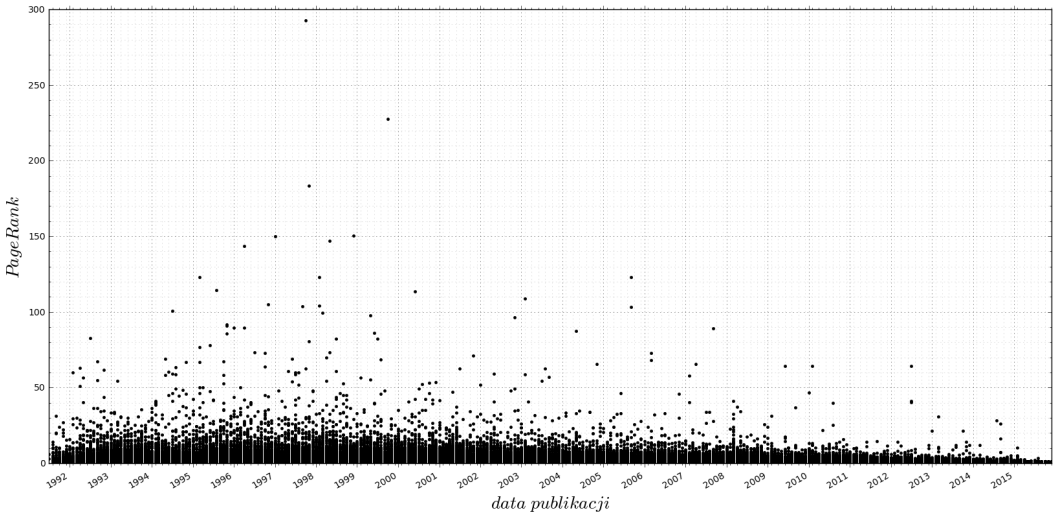
Rys. 3.1: Wykres wartości *PageRank* artykułu od liczby jego cytowań dla $\alpha = 0.10$



Rys. 3.2: Wykres wartości *PageRank* artykułu od liczby jego cytowań dla $\alpha = 0.50$



Rys. 3.3: Wykres wartości *PageRank* artykułu od daty jego publikacji dla $\alpha = 0.10$



Rys. 3.4: Wykres wartości *PageRank* artykułu od daty jego publikacji dla $\alpha = 0.50$

Niestety wykresy 3.3 oraz 3.4 nie dają jednoznacznej informacji na temat zależności *PageRank* do daty publikacji artykułu w stosunku do zmiany parametru α.

Kolejna obserwacja to duża liczba artykułów o niskiej wartości *PageRank*, która może być spowodowane dużą liczbą nietrafionych artykułów, które nie odniosły sukcesu, albo jeszcze nie zostały dostrzeżone jako dobre - wysoko cytowane.

Zaznaczony obszar na wykresach 3.1 oraz 3.2 obejmujący wartości funkcji w przedziale $x \in \{3000 \dots 5000\}, y \in \{0 \dots 25\}$ zawiera kilka dokumentów posiadających dosyć niski *PageRank* dla $\alpha = 0.1$ w stosunku do liczby cytowań. Wszystkie z tych dokumentów są związane z raportami amerykańskiej sondy kosmicznej *WMAP*. Raporty te były bardzo często cytowane, zapewne w pracach badawczych odnoszących się do zawartych w nich danych. Natomiast dla $\alpha = 0.5$ widać, iż obszar stał się pusty, a dokumenty te otrzymały wyższą wartość *PageRank* z uwagi na wysoką liczbę cytowań.

Z drugiej strony punkty w obszarze obejmującym wartości $x \in \{0 \dots 50\}$ i $y \in \{50 \dots \infty\}$ z artykułami posiadającymi niską liczbę cytowań, ale wysoką wartość *PageRank* dla $\alpha = 0.1$, na drugim wykresie zniknęły. Wartość $\alpha = 0.5$ spowodowała większą korelację z liczbą bezpośrednich cytowań, dlatego też wartość algorytmu *PageRank* dla tych dokumentów spadła diametralnie. Na drugim z wykresów nie jest możliwe odszukiwanie tych punktów, dlatego przedstawię ich wartości PR w tabeli 3.2.

identyfikator arXiv	liczba cytowań	$PR(\alpha = 0.10)$	$PR(\alpha = 0.50)$
cond-mat/9612237	34	56.8769440837	12.6870402353
hep-th/9406058	20	78.9683544698	26.1697089515
cond-mat/9706261	6	69.3832654892	7.26493987608
cond-mat/9612007	32	76.2314955827	12.4196200547
hep-th/9112067	12	94.1818001183	15.5385923379
nucl-th/9210014	42	67.2286067023	23.0890704081
hep-th/9111016	46	67.3731763698	22.3265814529
physics/9612013	5	51.4637701754	7.19672726776
math/0211065	3	60.210815658	49.2286609369

Tab. 3.2: Tabela przedstawiająca punkty z obszaru $x \in \{0 \dots 50\}$ i $y \in \{50 \dots \infty\}$ z wykresów 3.1 oraz 3.2.

Myślę, że *PageRank* powinien być związany z liczbą cytowań w większym stopniu niż dla $\alpha = 0.1$, dlatego też mój wybór padł na $\alpha = 0.5$. Dzięki temu artykuły mające dużą liczbę cytowań, będące na pewno dobrymi wyborami, nie są poszkodowane poprzez algorytm. Kolejnym efektem doboru takiego α jest zmniejszenie wartości *PageRank* dla artykułów, które zyskały "popularność", będąc cytowanymi przez garstkę artykułów, których podsieć cytowań jest bardzo duża i rozrastająca się. Artykuły te są bazowymi referencjami dla dobrego artykułu. Bazowe referencje mogą zawierać metody użyte w pracach, definicje notacji, ale nie informacje, których szukamy. Dlatego też algorytm *PageRank* powinien oceniać lepiej bezpośrednio cytowane artykuły.

3.2.2 PageRank

Algorytm *PageRank* zostal uruchomiony na grafie artykułów z portalu arXiv, wcześniej zbudowanego przez serwis Paperscape[3]. Udostępniona baza w formacie dokumentów *csv* wymagała przetworzenia - podział na wierzchołki, krawędzie w celu wczytania ich jako graf przez *Apache Spark*.

Domyślną implementacją *PageRank* w *Apache Spark* jest implementacja rozproszona, opisana w sekcji 1.4.2.

3.2.3 Elasticsearch

Zasada działania silnika wyszukiwań *Elasticsearch* została przedstawiona w sekcji 2.1.3, natomiast tutaj chciałbym się skupić na dostrojeniu wartości funkcji *Score*, względem algorytm *PageRank*.

W systemie został utworzony indeks o nazwie *arxiv* przechowujący dokumenty o typie *paper*, których pola przedstawione są w tabeli 3.3.

pole	**typ**
id	`String`
pagerank	`Double`
title	`String`
content	`String`
authors	`[String]`

Tab. 3.3: Pola typu *paper* w indeksie *arxiv*

Zapytanie wyszukujące uruchamiane jest na polach: *title* i *content*, dla których wyznaczany jest domyślny *Score*. Funkcja *Score* została nadpisana, aby brać pod uwagę wartości *PageRank*.

$$Score(q,d) = \frac{Score_{tf\text{-}idf}(q,d) + PRWeight \cdot PR_d}{1 + PRWeight} \tag{3.1}$$

$PRWeight$ to waga z jaką wartość *PageRank* będzie brana pod uwagę w rangowaniu dokumentów, która może być dobierana do każdego zapytania.

Aplikacja kliencka, dająca dostęp do wyszukiwania w *Elasticsearch*, została opisana w sekcji 3.2.4.

3.2.4 Interfejs użytkownika

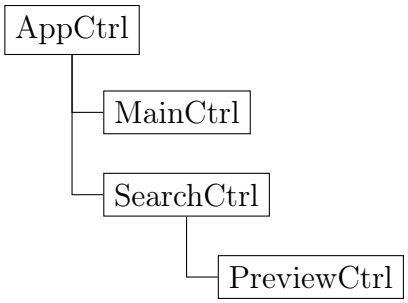
Aplikacja kliencka to webowa aplikacja zbudowana w oparciu o framework *AngularJS* opisany w sekcji 3.1.3.

Wygląd aplikacji opiera się na frameworku *Bootstrap*[9], dzięki czemu jest on responsywny[10], co widać na 3.8, gdzie ta sama aplikacja na urządzeniu mobilnym przystosowuje się do mniejszego urządzenia. Kolejnym przystosowaniem do urządzenia mobilnego jest przycisk *Preview*, który zamienia się na *Download*, ponieważ nie ma możliwości podglądu dokumentu *PDF* na urządzeniu mobilnym.

[9]front-end framework zawierający style *CSS*, dzięki którym zbudowanie layoutu strony internetowej jest szybkie i proste - `https://getbootstrap.com/`

[10]aplikację webową, która została stworzona docelowo na komputer osobisty, ale potrafiąca dostosować swoje wymiary do urządzenia mobilnego nie tracąc na przejrzystości, nazywamy responsywną

Poniżej, na rysunku 3.5, przedstawia się struktura aplikacji, której komponenty zostaną omówione w kolejnych sekcjach.



Rys. 3.5: Struktura aplikacji.

AppCtrl

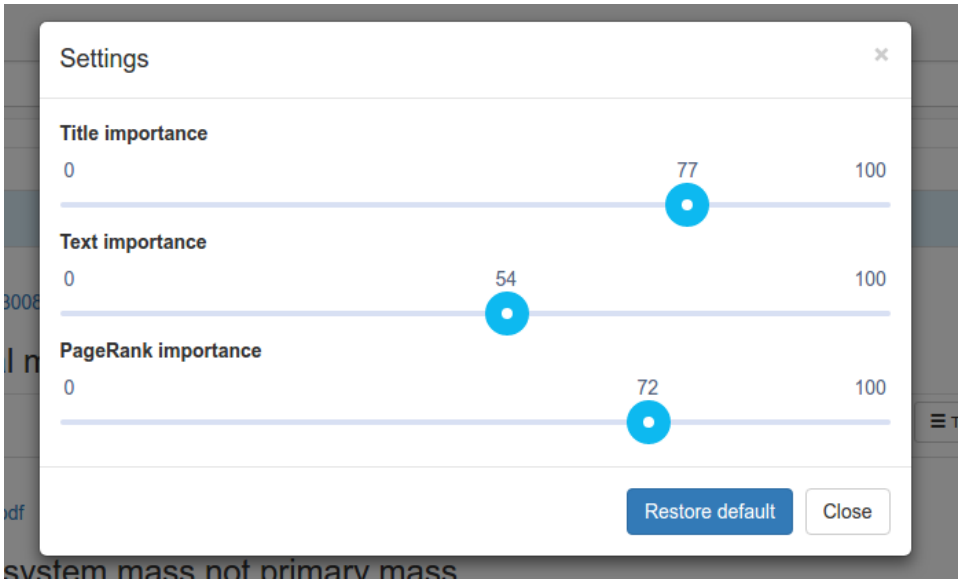
Główny kontroler aplikacji *AppCtrl* jest odpowiedzialny za zarządzanie ustawieniami wyszukiwania, przechowywanie ich w *local storage*[11] przeglądarki,

AppCtrl jako główny kontroler całej aplikacji, nie ma przypisanej ścieżki dostępu, ponieważ jest uruchamiany przy każdej ze ścieżek, jako kontroler nadrzędny.

Bezpośrednio z głównego kontrolera uruchamiany jest widok ustawień zapytania, widoczny na rysunku 3.6. Każda zmiana ustawień suwaków w widoku zapisywana jest do *local storage* i odczytywana w przypadku ponownego uruchomienia widoku. Nie jest możliwe ustawienie wartości ważności tekstu i tytułu na 0, gdyż jedna z tych wartości musi być większa lub równa 1. Wyszukiwanie jest uruchamiane na danym polu jeśli jego wartość ważności wynosi co najmniej 1, natomiast w przypadku, gdyby dwa pola miały wartość 0, wtedy wyszukiwanie tekstowe nie miałoby sensu, bo nie byłoby ustalone, które pola mają być rozpatrywane w wyszukiwaniu. *PageRank importance* odpowiada wartości *PRWeight* z sekcji 3.2.3, natomiast *Title importance* i *Text importance* są obsługiwane przez *Elasticsearch* jako *boosting operator*, który zwielokrotnia wartość *tf-idf* dla danego pola.

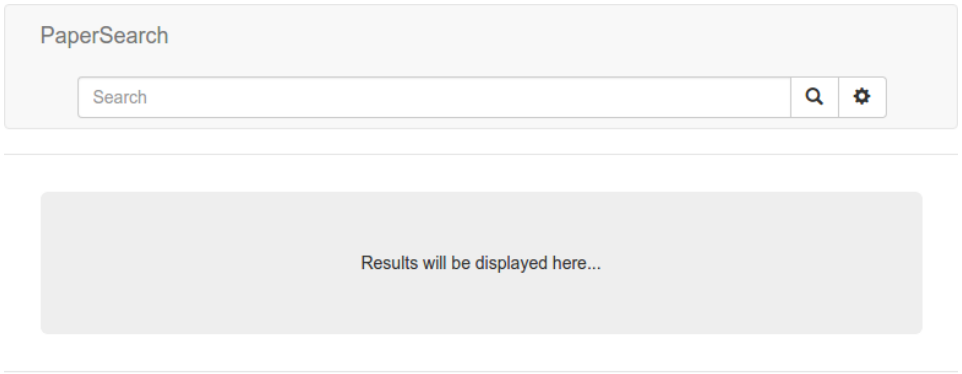
Zawiera on również pole tekstowe, którego zawartość przesyłana jest jako zapytanie do widoku wyszukiwania - *SearchCtrl*. Po wciśnięciu przycisku z lupą, bądź naciśnięciu klawisza ENTER na klawiaturze będąc w tym właśnie polu przechodzimy do widoku wyszukiwania.

[11] lokalny zasób przeglądarki, przechowujący dane aplikacji webowej, w odróżnieniu od cookies, nie wymaga wysyłania zapisanych informacji do serwera, dostęp do nich odbywa się bezpośrednio ze skryptów JavaScript



Rys. 3.6: Widok ustawień parametrów wyszukiwania.

MainCtrl

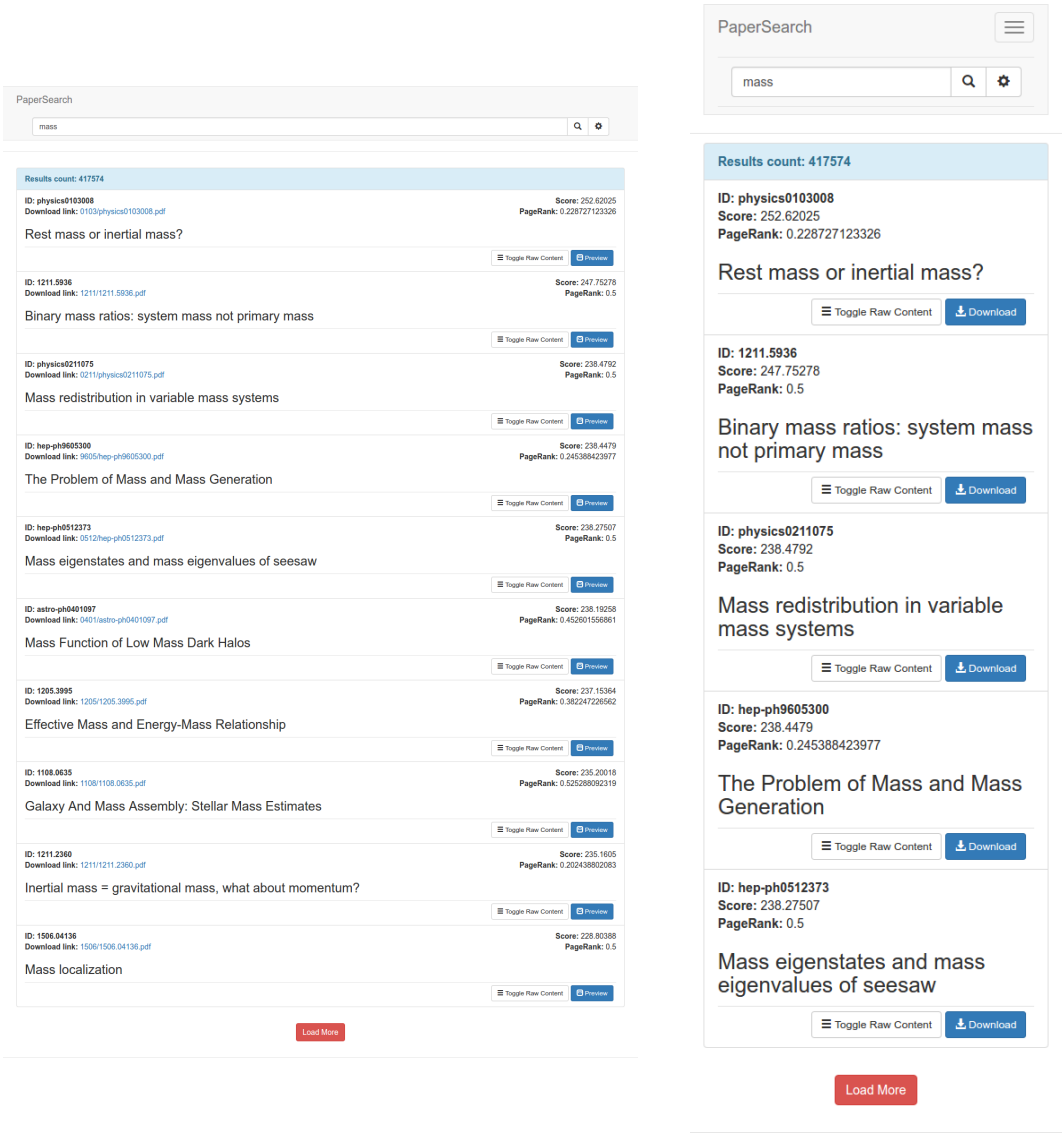


Rys. 3.7: Widok strony głównej.

Kontroler *MainCtrl* odpowiedzialny jest za wyświetlanie strony startowej. Na rysunku 3.7 prezentuje się widok strony głównej obsługiwany przez ten kontroler.

Ścieżka dostępu do kontrolera to `/`.

SearchCtrl



Rys. 3.8: Prezentacja wyników wyszukiwania - z lewej strony wersja na komputer, z prawej na urządzenia mobilne.

Kontroler wyszukiwania, przyjmuje parametr *query*, będący treścią zapytania. Ścieżka dostępu do kontrolera to `/search/query`. Parametr ten jest wykorzystywany do dwóch zapytań:

- *count* - zapytanie liczące liczbę dokumentów, które zwróci dane zapytanie do *Elasticsearch*. Dzięki temu zapytaniu możemy wyliczyć, ile stron dokumentów jest w stanie wyświetlić nasza aplikacja, czy też czy możliwe jest załadowanie kolejnych dokumentów na dole strony poprzez przycisk *Load More*.

- *search* - zapytanie wyszukujące, uruchamiane jeśli z poprzedniego zapytania *count* została zwrócona pozytywna wartość. Uruchamiane jest z funkcją *Score* opisaną w sekcji 3.2.3 i zwraca 10 dokumentów na każdą z załadowanych stron. Pierwsze uruchomienie widoku ładuje pierwszą stronę, kolejne mogą być załadowane poprzez akcję *Load more*, jeśli liczba dokumentów zwrócona przez zapytanie *count* jest większa niż 10.

Wyniki wyszukiwania wyświetlane są jako lista dokumentów, która zawiera: id, link do pobrania, *Score* oraz wartość PageRank dokumentu.

Kolejnymi elementami interfejsu są:

- przycisk *Preview* otwierający modalne okno przechodząc do podglądu - (*PreviewCtrl*),

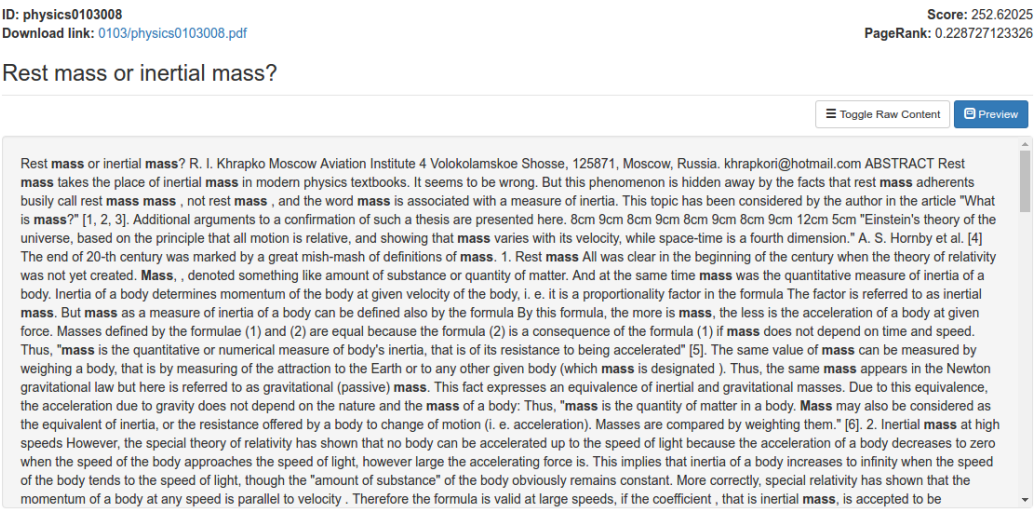
- Akcja *Toggle Raw Content* dająca możliwość podejrzenia zindeksowanego tekstu dokumentu z wyszczególnionymi, poprzez pogrubienie, wyrazami, które są związane bezpośrednio z zapytaniem. Na przykładzie rysunku 3.9 widać pogrubione wystąpienia słowa z zapytania - "mass".

PreviewCtrl

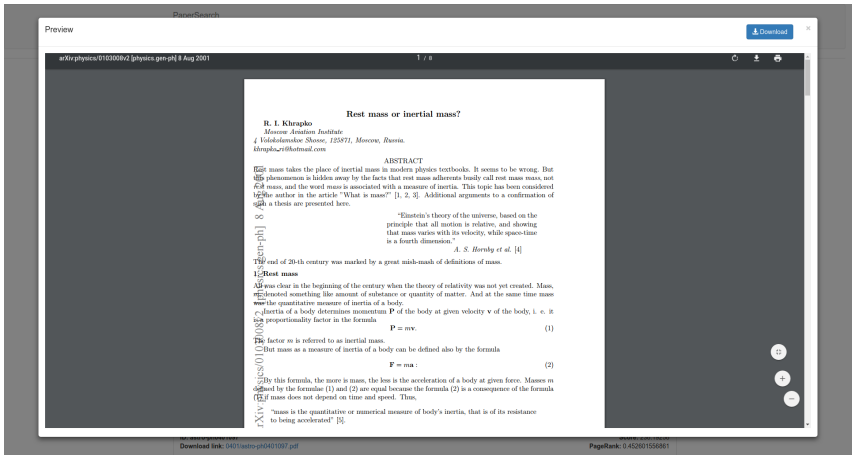
Kontroler obsługujący podgląd dokumentu. Uruchamiany bezpośrednio z widoki prezentującego wyniki wyszukiwania. Wymaga przekazania parametru *path* będącego ścieżką do pliku, którego pogląd ma wyświetlić.

Rysunek 3.10 prezentuje wygląd widoku przypisanego do kontrolera *PreviewCtrl* - jest to okno modalne wywoływane bezpośrednio z *SearchCtrl*.

Ścieżka dostępu do kontrolera to `/search/query/preview`. W przypadku braku parametru *path*, aplikacja cofa użytkownika do ścieżki `/search/query`.



Rys. 3.9: Widok zindeksowanego tekstu dokumentu z wyszczególnionymi wyrazami związanymi z zapytaniem “mass”.



Rys. 3.10: Widok podglądu dokumentu.

3.3 Konserwacja i inżynieria wtórna

System, działający na tej samej kolekcji danych, nie wymaga żadnych konserwacji.

Niestety w trakcie testowania systemu ujawniła się pewna wada: dodanie nowych dokumentów wymaga przeliczenia *PageRank* dla wszystkich dokumen-

tów oraz uaktualnienie wartości uprzednio dodanych dokumentów. Uaktualnienie tych wartości jest możliwe poprzez wykonanie zapytań batch na indeksie.

Istnieją metody aproksymacji *PageRank* bazujące na istniejących dokumentach, do którego odnosi się nowo dodawany dokument, aczkolwiek to nie było głównym przedmiotem tematu i nie zostało zaimplementowane w ramach projektu autorskiego.

Domyślna miara relewantności *tf-idf* w najnowszej wersji *Elasticsearch* została zastąpiona poprzez *Okapi BM25*. Być może warto byłoby przejść na tę właśnie wersję w celu ulepszenia relewantności zapytań [11].

Rozdział 4

Ewaluacja i podsumowanie

Niniejszy rozdział przedstawia metodę ewaluacji oraz jej wyniki, a w ostatniej sekcji zawarte jest podsumowanie wyników.

4.1 Metoda ewaluacji

Ewaluacja pracy została wykonana na podstawie korelacji wyników zapytania z innymi wyszukiwarkami artykułów takimi jak *Inspire*, *CiteSeer*, *Google*. Korelacja wyznaczana jest metodą Pearsona, którą przedstawię poniżej.

Korelacja Pearsona

Współczynnik korelacji liniowej służący do badania zależności między danymi. Dla przykładu, zakładamy, że mamy zbiór danych dotyczący liczby wypadków (X) oraz ilości opadów na metr kwadratowy (Y) w danym dniu. Wykorzystując współczynnik korelacji Pearsona, możemy określić czy wraz ze wzrostem opadów, proporcjonalnie rośnie liczba wypadków.

Korelację definiuję się poniższym wzorem:

$$r_{xy} = \frac{\sum(X_i - \overline{X}) \cdot (Y_i - \overline{Y})}{\sqrt{\sum(X_i - \overline{X})^2 \cdot \sum(Y_i - \overline{Y})^2}} = \frac{\frac{1}{n}\sum X_i Y_i - \overline{X}\ \overline{Y}}{\sigma_X \cdot \sigma_Y} \tag{4.1}$$

gdzie:

- n - ilość obserwacji,

- X_i, Y_i - i-te wartości obserwacji z populacji X i Y,
- $\overline{X}$, $\overline{Y}$ - średnie z populacji X i Y,
- σ_x, σ_y - odchylenia standardowe populacji X i Y,

Siła korelacji r może być interpretowana w następujący sposób - dla $|r|$:

- $< 0,2$ - brak związku liniowego,
- $0,2 - 0,4$ - słaba zależność,
- $0,4 - 0,7$ - umiarkowana zależność,
- $0,7 - 0,9$ - dość silna zależność,
- $> 0,9$ - bardzo silna zależność

Interpretacja siły korelacji oczywiście zależy od charakterystyki danych i ich ilości.

Identyfikacja dokumentów

W trakcie ewaluacji zauważono, iż identyfikacja wyników wyszukiwań artykułów na różnych serwisach nie jest zunifikowana - większość z artykułów nie posiada wspólnego identyfikatora, za pomocą którego można stwierdzić, iż dwa dokumenty są takie same. Problem ten został rozwiązany z wykorzystaniem odległości Levenshteina, za pomocą, której wybierany jest najbardziej podobny dokument.

Dokumenty są porównywane ze sobą w następujący sposób:

- zbiór A zawiera wyniki wyszukiwania z serwisu *Inspire*
- zbiór B zawiera wyniki wyszukiwania z projektu na bazie *Elasticsearch*
- dla każdego z elementów ze zbioru A znajdowany jest najbardziej podobny artykuł ze zbioru B
- znajdowanie odbywa się poprzez obliczenie podobieństwa $sim(4.3)$ dla każdego elementu ze zbioru B w stosunku do wcześniej wybranego dokumentu ze zbioru A

- dokumenty podobne do siebie z prawdopodobieństwem większym niż $0,9$, są dodawane do zbioru sim_{A_i}, z którego następnie wybierany jest element z największym podobieństwem.

- jeśli zbiór sim_{A_i} jest pusty, to znaczy, że w wynikach wyszukiwania A i B istnieją punkty rozłączne, które nie są poddawane dalszej ewaluacji - są odrzucane.

Tak znaleziony wspólny zbiór dokumentów poddawany jest dalszym badaniom poziomu korelacji, której wyniki są przedstawione w sekcji 4.2.

Odległość Levenshteina

Odległość Levenshteina to miara pozwalająca na szacowanie różnic pomiędzy dwoma ciągami znaków. Miara ta wyznacza minimalną liczbę zmian pojedynczych znaków (dodanie, usunięcie, podmianę znaku) wymaganą, aby z jednego słowa otrzymać drugie.

Odległość Levenshteina może być zdefiniowana za pomocą wzoru:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{jeżeli } min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j) + 1_{(a_i \neq b_j)} \end{cases} & \text{w przeciwnym wypadku} \end{cases} \tag{4.2}$$

gdzie a i b to porównywane ciągi znaków, $1_{(a_i \neq b_j)}$ to funkcja zwracająca 1 jeśli $a_i \neq b_j$, w przeciwnym wypadku 0. $lev_{a,b}(i,j)$ to odległość pomiędzy pierwszymi i znakami ciągu a i pierwszymi j znakami ciągu b.
Powyższy wzór można wykorzystać do zdefiniowania miary podobieństwa:

$$sim_{a,b} = 1 - \frac{lev_{a,b}(|a|,|b|)}{max(|a|,|b|)} \tag{4.3}$$

gdzie $|a|$ i $|b|$ to długość ciągów znaków a i b.

Miara podobieństwa sim zwraca wartości prawdopodobieństwa z przedziału $[0,1]$, co może być łatwo wykorzystane do porównywania dokumentów.

4.2 Wyniki

Porównanie pomiędzy innymi serwisami

Pierwsza ewaluacja polegała na porównaniu wyników wyszukiwania artykułów w projekcie autorskim z portalem *Inspire*. Proces wymagał pobrania $10,000$ wyników wyszukiwania dla kilku różnych zapytań, a następnie znalezieniu w nich wspólnych punktów i wyznaczeniu współczynnika korelacji między nimi.

Wykres 4.1 prezentuje korelację pozycji dokumentów w portalu *Inspire* oraz projekcie autorskim, widocznych w tytule wykresu. Tabela 4.1 reprezentuje współczynniki korelacji Pearsona, dla każdego z zapytań.

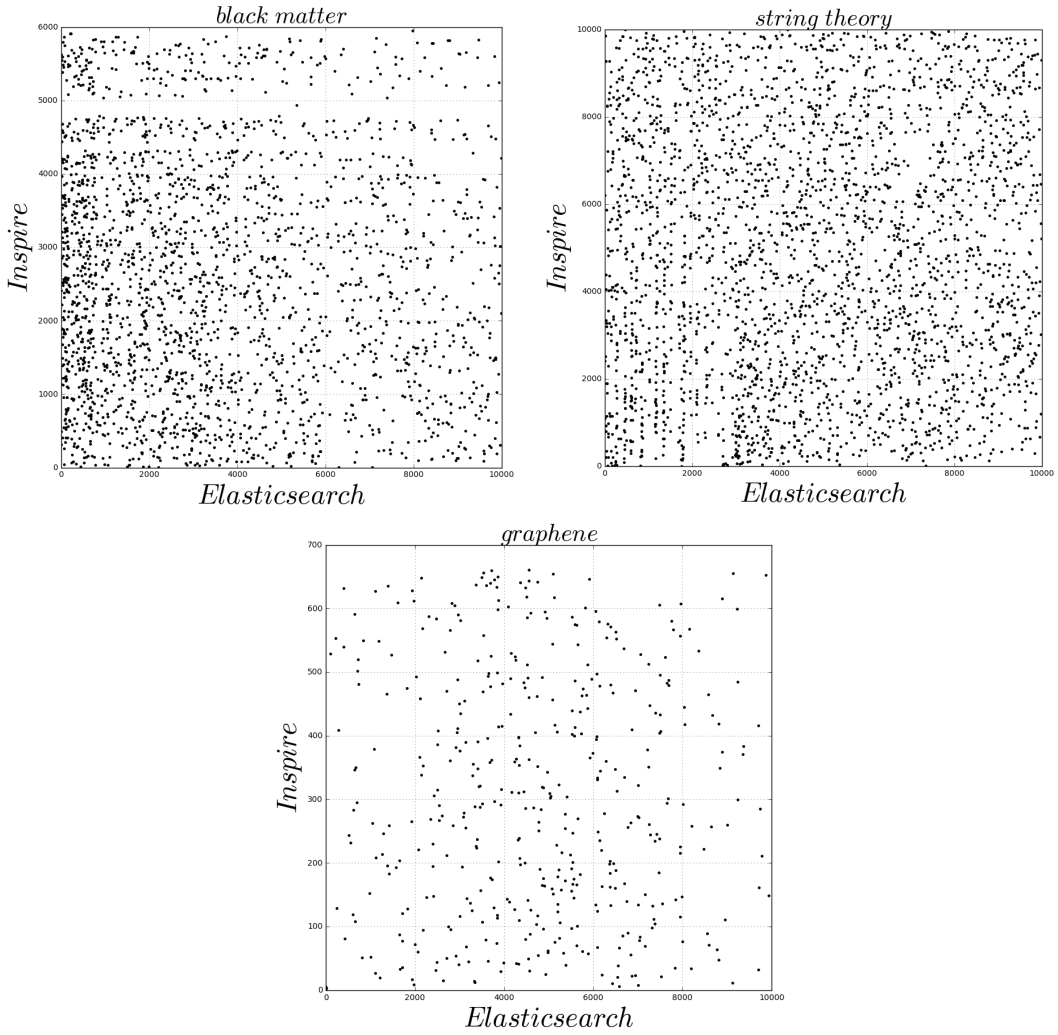
Jak widać, współczynnik oraz wykresy wskazują, że nie istnieje korelacja pomiędzy wynikami wyszukiwania z projektu autorskiego a tymi z portalu *Inspire*.

Test *PRWeight*

Kolejna ewaluacja polegała na porównaniu pozycji wyników wyszukiwania w projekcie autorskim przy różnym parametrze $PRWeight$, określającym ważność *PageRank* w wynikach wyszukiwania.

Wykres 4.2 oraz tabela 4.2 przedstawia korelację pomiędzy pozycjami wyników wyszukiwania dla zapytania "black matter" i $PRWeight = 1$ w stosunku do $PRWeight \in \{2, 5, 10, 20\}$. Ten sam eksperyment został ponowiony dla zapytania "graphene", co jest zobrazowane na wykresie 4.3 oraz tabeli 4.3.

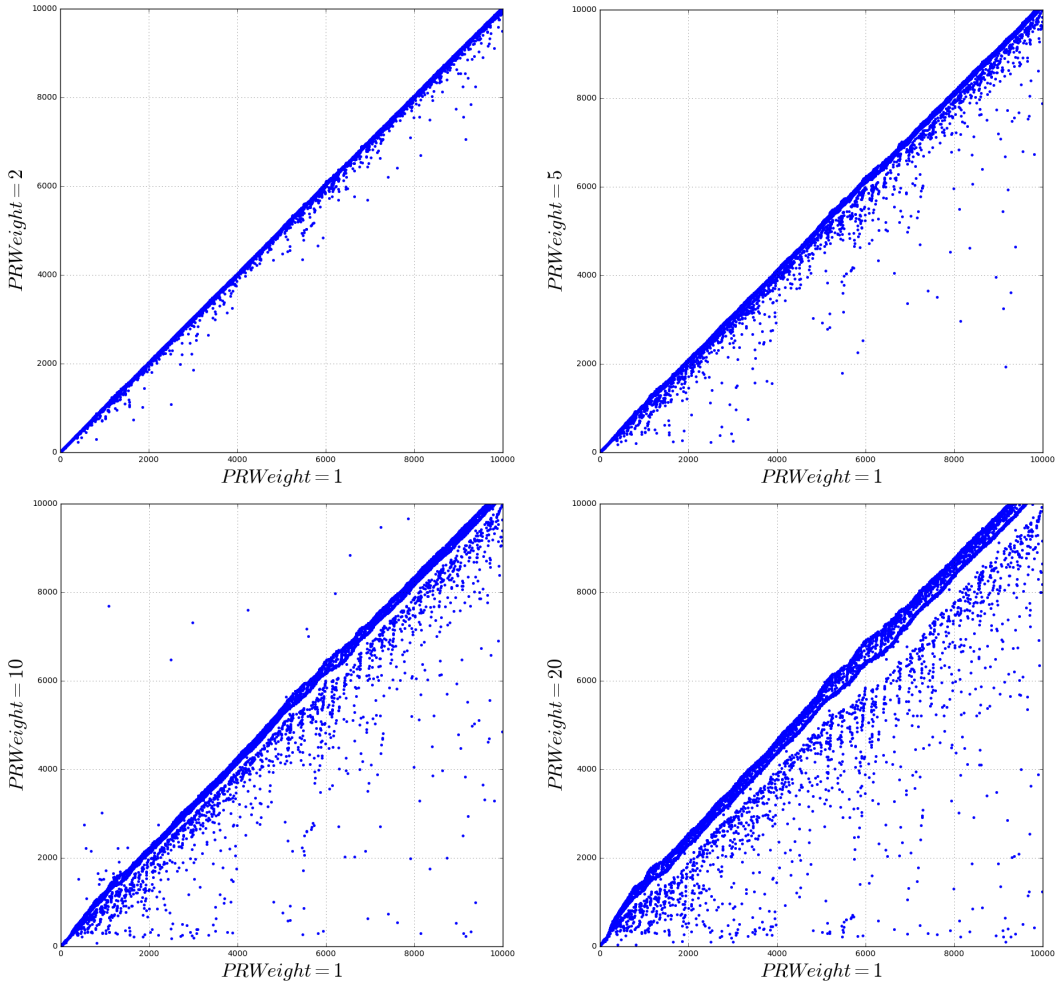
Na wykresach wyraźnie widać, iż wraz ze wzrostem $PRWeight$ coraz więcej artykułów zwiększa swoją pozycję w wynikach wyszukiwania - w dolnej części wykresu pojawia się więcej punktów. Wartość współczynnika korelacji, widoczna w tabelach, spada wraz ze wzrostem $PRWeight$, z uwagi na zmianę pozycji wyników wyszukiwania.



Rys. 4.1: Wykresy prezentujące korelację pozycji wyników wyszukiwania projektu autorskiego z portalem *Inspire* dla różnych zapytań.

zapytanie	**korelacja Pearsona**
"black matter"	-0.064099698969655505
"string theory"	0.039817223285069508
"graphene"	$-0,015440329940013266$

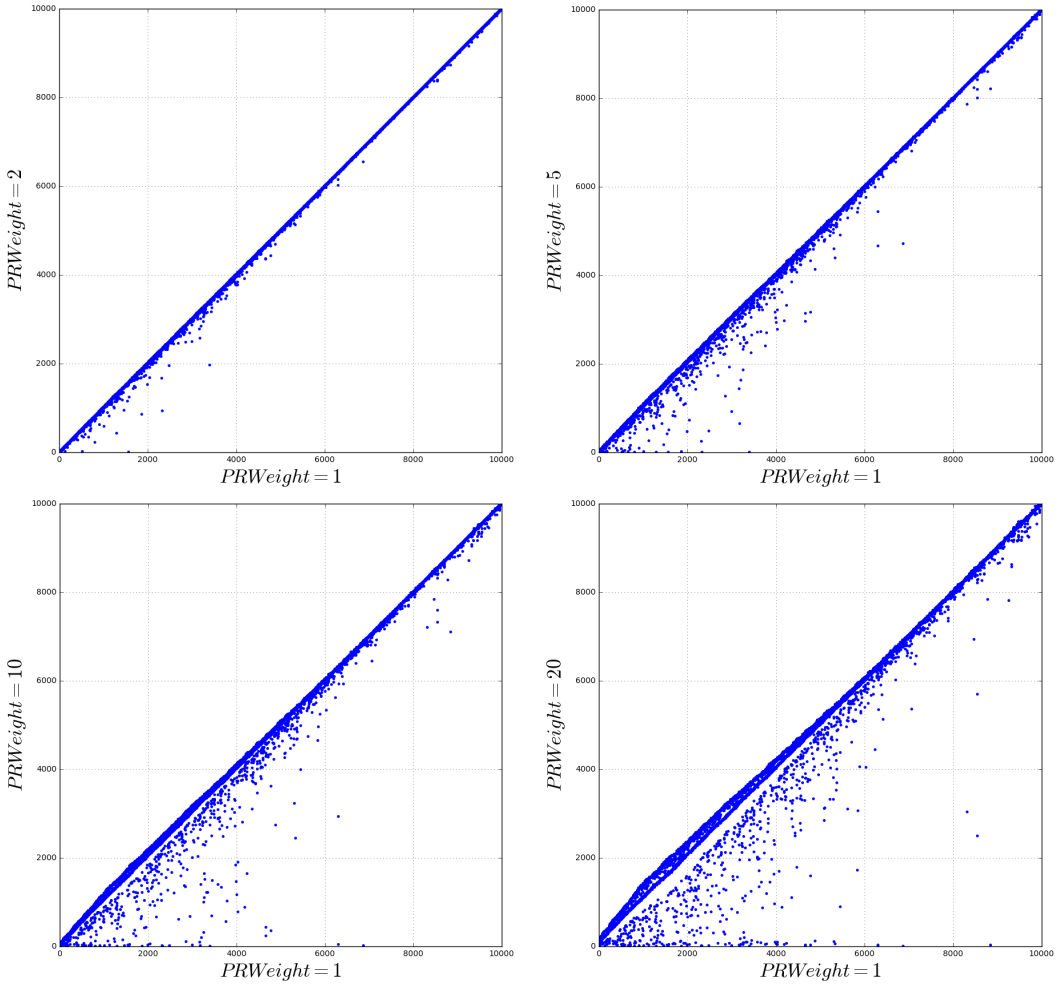
Tab. 4.1: Korelacja Pearsona pozycji wyników zapytań dla różnych zapytań.



Rys. 4.2: Wykresy prezentujące korelację wyników wyszukiwania z parametrem $PRWeight = 1$ do $PRWeight \in \{2, 5, 10, 20\}$ dla zapytania "black matter".

PRWeight	**korelacja Pearsona**
2	0.99960559628735302
5	0.99490360044539805
10	0.98009479475713501
20	0.95279571915930727

Tab. 4.2: Tabela prezentuję wyniki korelacji Pearsona dla wyników wyszukiwania z parametrem $PRWeight = 1$ w stosunku do $PRWeight \in \{2, 5, 10, 20\}$ dla zapytania "black matter".



Rys. 4.3: Wykresy prezentujące korelację wyników wyszukiwania z parametrem $PRWeight = 1$ do $PRWeight \in \{2, 5, 10, 20\}$ dla zapytania "graphene".

PRWeight	**korelacja Pearsona**
2	0.99990184899049017
5	0.99913745619031680
10	0.99641861446363111
20	0.99064437859144583

Tab. 4.3: Tabela prezentuję wyniki korelacji Pearsona dla wyników wyszukiwania z parametrem $PRWeight = 1$ w stosunku do $PRWeight \in \{2, 5, 10, 20\}$ dla zapytania "graphene".

4.3 Podsumowanie wyników

Z uwagi na ograniczenia wyszukiwarek takich jak *Google*, *CiteSeer*, ewaluacja była ograniczona tylko do portalu *Inspire*, który umożliwia pobranie $10,000$ wyników wyszukiwania. Ograniczenia dla każdego z nich to:

- *Google* - API dostarczające wyniki wyszukiwania poprzez zapytania RESTowe pozwala na wyodrębnienie tylko 100 pierwszych wyników wyszukiwania,
- *CiteSeer* - portal pozwala na przeglądanie tylko 500 pierwszych wyników wyszukiwania

W tak małych zbiorach danych nie jest możliwe znalezienie wystarczającej ilości punktów wspólnych, aby wyciągnąć jakiekolwiek wnioski.

Pierwsza ewaluacja pokazuje, iż projekt autorski nie ma powiązania pomiędzy portalem *Inspire* pod względem pozycji wyników wyszukiwania.

Druga z ewaluacji, skupiająca się $PRWeight$, wskazuję na skuteczność sterowania wyników tym parametrem, który wyszczególni artykuły posiadające wyższą wartość *PageRank*.

Podsumowując: projekt autorski można uważać za jedyny w swoim rodzaju, ponieważ nie jest on porównywalny z żadnym innym istniejącym portalem, natomiast ewaluacja wyników wyszukiwania musiałaby się odbyć we współpracy ze specjalistę z danej dziedziny nauki, który oceniłby czy system zwraca poprawne wyniki wyszukiwania, w oczekiwanej kolejności.

Bibliografia

[1] Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, 1999

[2] Bryan Kurt, Leise Tanya: The $ 25,000,000,000 Eigenvector. The

[3] Damien P. George and Robert Knegjens, Paperscape, `http://paperscape.org`

[4] D. Coppersmith, S. Winograd. *M*atrix Multiplication via Arithmetic Progressions, *J*ournal of Symbolic Computation, 9(3):251-280, 1990.

[5] Austin David: How Google Finds Your Needle in the Web's Haystack. (`http://www.ams.org/samplings/feature-column/fcarc-pagerank`).

[6] P. Chen, H. Xie, S. Maslov, S. Redner, *Finding Scientific Gems with Google*, 2006

[7] Christopher D. Manning, Prabhakar Raghavan i Hinrich Schütze, *Introduction to Information Retrieval*, Section 6.2, p. 117-119, Cambridge University Press. 2008

[8] Christopher D. Manning, Prabhakar Raghavan i Hinrich Schütze, *Introduction to Information Retrieval*, Section 21.2, p. 464-474, Cambridge University Press. 2008

[9] R. Vidhya1, G. Vadivu, *Research Document Search using Elastic Search*, 2016

[10] Elasticsearch Reference, version 2.3 `https://www.elastic.co/guide/en/elasticsearch/reference/2.3/index.html`

[11] Konrad Beiske, *BM25 vs Lucene Default Similarity*, 2014 `https://www.elastic.co/blog/found-bm-vs-lucene-default-similarity`

[12] S. Brin and L. Page, *Computer Networks and ISDN Systems*, 30, 107, 1998.

[13] S. Brin, L. Page, *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, 1998

[14] Paul E. Black, "inverted index", in *Dictionary of Algorithms and Data Structures* [online], Vreda Pieterse and Paul E. Black, eds. 25 March 2016, `https://www.nist.gov/dads/HTML/invertedIndex.html`

Spis rysunków

Spis tabel

Spis algorytmów

Spis bloków kodu