Uniwersytet Ekonomiczny w Krakowie
Kolegium Nauk o Zarządzaniu i Jakości
Instytut Zarządzania
Kierunek: Informatyka stosowana
Specjalność: Inżynieria oprogramowania

Mateusz Puto
Nr albumu: 219123

Porównanie metod wyszukiwania informacji opartych o uczenie głębokie

Praca licencjacka

Promotor
dr Paweł Konkol

Kraków 2023

Spis treści

Wstęp

Wraz z rozpowszechnieniem się komputerów, powstały różne modele wyszukiwania informacji, jak i ich implementacje. Zostaną one opisane w **Rozdziale 1**. Techniki te, trwały przez wiele lat, jako główne modele wyszukiwania informacji. Chociażby Okapi BM25 stworzone w latach 80-tych i 90-tych XX wieku jest stosowane do dziś. Zmiana paradygmatu zaczęła w dużej mierze zachodzić dzięki udanemu zastosowaniu technik uczenia maszynowego. Szczególną rolę miał tutaj model uczenia głębokiego *Transformer*, który ostatnimi laty zaczął odgrywać znaczącą rolę w różnych gałęziach uczenia głębokiego. **Rozdział 2** służy ogólnemu przedstawieniu formalizmów, modeli, zbiorów danych, sposobów trenowania, jak również zgrubnemu opisowi przykładowych możliwych zastosowań. Rozdział ten ma charakter wprowadzający do tematyki. W efekcie niekwestionowanych sukcesów na polu wykorzystania modeli neuronowych, nastąpiła proliferacja badań w kierunku trenowania sieci do celów wyszukiwania informacji. Swoiste Zoo architektur z różnymi charakterystykami, rodzi pytanie o to, które podejście jest najlepsze oraz jakie perspektywy stoją przed tą szybko rozwijającą się dziedziną. **Rozdział 3** dotyczy dokładnego scharakteryzowania rozważanych w obszarze IR systemów, przedstawienie charakterystycznych typów architektur oraz przedstawieniu ich zalet oraz wad w zakresie efektywności, szybkości działania itd. W rozdziale tym rozważamy też, możliwe rozszerzenie zakresu wyszukiwania informacji o problematykę odpowiadania na pytania (QA), jak również rozważamy systemy agentowe korzystające z uczenia przez wzmacnianie (RL). Ostatecznie w **Rozdziale 4** prezentujemy system wyszukiwania informacji stworzony przez autora pracy, z wykorzystaniem przedstawionej wiedzy, którego charakterystyka zostanie również omówiona.

Rozdział 1

Klasyczne techniki i zastosowania wyszukiwania informacji

1.1. Nota historyczna

Początki gromadzenia i systematycznego przetwarzania informacji są ściśle związane z pojawieniem się pisma oraz trwałych materiałów piśmienniczych. Jednak aby mówić o procesie wyszukiwania informacji przyszło nam poczekać do powstania antycznych bibliotek. Biblioteka Aleksandryjska, chociaż na pewno nie była pierwszym tworem posiadającym duży zbiór tekstów, wyróżnia się nie tylko swoją częściowo legendarną historią. Dawne podania mówią, że Aleksander Wielki po zobaczeniu biblioteki Ashurbanipala w Niniwie powziął się stworzenia uniwersalnej biblioteki zawierającej wszystkie dzieła podbitych narodów (Phillips, 2010). Szacuje się, że kolekcja składała się, w swoim szczytowym okresie, z od 400,000 do 700,000 zwojów, pozyskanych poprzez zakupy, kopiowanie, konfiskaty z przepływających statków czy nawet podstęp jak w przypadku dzieł Ateńskich (Phillips, 2010). Przy tak dużej ilości materiałów źródłowych organizacja przechowywania i wyszukiwania tekstów nabiera niebagatelnego znaczenia. Jednym z patronów takiego systemu jest Zenodot, pierwszy bibliotekarz Wielkiej Biblioteki Aleksandryjskiej. Wprowadził on alfabetyczną organizację dzieł po pierwszej literze imienia autora, co było rozwiązaniem w tamtym czasie niespotykanym (Phillips, 2010). To jednak Kallimach z Cyreny, uważany jest za twórcę *pinakes* - 120 zwojowego katalogu autorów i ich dzieł. Informacje o autorze były dodatkowo poszerzone o notę biograficzną, natomiast wpisy o dziełach, zawierały pierwsze słowa utworu, jak i liczbę linijek, które się na niego składały. (Phillips, 2010).

Systemy katalogowe były szeroko używane w zastosowaniach bibliotecznych, lecz

proces ten nie został zautomatyzowany przez długi okres. Pierwsze rudymentarne techniki wyszukiwania przy użyciu narzędzi pojawiają się na początku XX wieku. Soper wniósł w 1918 r. o patent na przeszukiwanie katalogu przy użyciu kart z dziurkami i światła, które przechodziło przez odpowiednio umieszczone za sobą karty (Sanderson & Croft, 2012). Goldberg stworzył w latach 20-tych i 30-tych XX wieku urządzenie pozwalające na zautomatyzowane porównywanie strony dokumentu z jego negatywem na rolce filmowej. Kiedy dokładna zgodność została zarejestrowana na fotokomórce, urządzenie wyświetlało zatrzymany mikrofilm (Sanderson & Croft, 2012). W roku 1950 zbudowano system wyszukiwania oparty o karty perforowane, działający z szybkością 600 kart na minutę. Wtedy też Calvin Mooers, prezentujący pracę naukową, użył jako pierwszy terminu 'wyszukiwanie informacji' (ang. 'information retrieval') (Sanderson & Croft, 2012). W brytyjskim Royal Society rozważano już jednak dwa lata wcześniej możliwość zastosowania technologii, która wkrótce miała się okazać przełomowa dla wszelkich zadań obliczeniowych. Holmstrom opisywał wtedy komputer UNIVAC, mogący wyszukiwać odniesienia w tekście umieszczonym na taśmie magnetycznej, na podstawie powiązanych z nimi kodów(Sanderson & Croft, 2012).

1.2. Wyszukiwanie informacji

Najprostszym, niedojrzałym sposobem wyszukiwania informacji, w pewnym zbiorze dokumentów, jest wyszukiwanie po jasno zdefiniowanych meta-informacjach. Chociaż oczywistym problemem, w korzystaniu z takiego systemu, jest fakt, iż musimy znać wcześniej odpowiednie informacje o wyszukiwanym dokumencie, takie jak nazwisko autora, tytuł czy numer, to tego rodzaju systemy były wykorzystywane nie tylko w urzędach czy bibliotekach, ale również w wyszukiwaniu internetowym do lat 90-tych.

Jako bardziej wygodna metoda wyszukiwania, wykorzystanie znalazł model boolowski. Zakłada on, że dokument jest zbiorem wyrazów, a więc pomija ilość wystąpień, kolejność występowania oraz powiązania między wyrazami. Na jego podstawie można stworzyć system, który po uprzednim zindeksowaniu wyrazów, pozwoli na wyszukiwanie wspólnych wystąpień wyrażeń w tekście przy pomocy powiązania ich operatorami AND, OR a nawet NEAR. W roku 2005 system oparty o model boolowski był nadal wykorzystywany jako domyślny sposób wyszukiwania w systemie wyszukiwania dokumentów prawnych firmy Westlaw (Manning, 2009, s. 38-56). Wykorzystanie tego rozwiązania wiąże się jednak z koniecznością wcześniejszego

przetworzenia informacji w dokumentach do postaci wyszukiwalnego odwróconego indeksu. Składa się on z pól *klucz*: *wartość*, gdzie kluczem jest słowo, unikalny identyfikator słowa, ewentualnie pewna kombinacja słów taka jak n-gram. Ponadto dla każdego klucza mogą być przechowywane dodatkowe informacje, takie jak częstość występowania. W polu wartości przechowywane są powiązane z danym słowem dokumenty, a raczej ich identyfikatory, które znajdują się w pewnego rodzaju kolekcji. Kolekcja jest często posortowana, a każdy zapis związany z dokumentem może posiadać także dodatkowe informacje, na przykład pozycyjne określenie umiejscowienia wystąpienia danego słowa w dokumencie. Wartości znajdujące się w odwróconym indeksie są zazwyczaj posortowane, co przyspiesza operacje, które chcielibyśmy na nim wykonywać (Manning, 2009, s. 104-122).

Kluczowym uwarunkowaniem dla konstrukcji odwróconego indeksu, dla dużych zbiorów danych, jest przechowywanie struktur danych w pamięci. Jeśli indeks jest niewielki to możemy umieścić wszystkie dane w pamięci operacyjnej podczas tworzenia, modyfikowania i dostępu, co pozwoli na optymalny dostęp do aktualnych informacji. Jeśli nasze wymagania są większe, to możliwe, że część danych będziemy musieli składować na dysku, co stwarza dodatkowe wyzwania. Kompresja indeksu pozwala na, przechowywanie większej ilości zindeksowanych danych na dysku, a także szybszy przesył pomiędzy dyskiem a pamięcią operacyjną. Gdy posiadamy szybki algorytm kompresujący, opłacalnym może być również kompresowanie części cache'u indeksu, która jest przechowywana w pamięci, co pozwoli na przechowywanie większej ilości często używanych wyrażeń bliżej procesora, tym samym przyspieszając działanie systemu.

Zapytania, które interesują użytkownika są często bardziej złożone niż logiczna kombinacja słów kluczowych. Jednym z usprawnień, które wyszukujący chce mieć do swojej dyspozycji, jest wyszukiwanie parametryczne. Pozwala ono na wybranie pewnego interesującego nas przedziału wyszukiwanej wartości. Takie określenie problemu może się sprawdzić gdy na przykład, chcemy wyszukać informacje o podanej dacie. Aby przechowywać te dodatkowe dane, konieczny jest indeks parametryczny, który może być B-drzewem (Manning, 2009). Innym możliwym usprawnieniem dla wyszukiwania meta-informacji jest przeszukiwanie tzw. stref dokumentu (ang. 'zones'). Umożliwia ono wyszukiwanie informacji występujących w konkretnym miejscu dokumentu, chociażby w jego tytule. Dzięki algorytmowi określania rangi dokumentu za pomocą ważenia informacji z jego stref (ang. 'weighted zone scoring'), można przypisać wagę do wystąpienia danej informacji w pewnej części dokumentu,

a następnie sumując te wagi, dla wyszukiwanych informacji we wszystkich strefach, otrzymać wynik określający dopasowanie zapytania do dokumentu.

Innym, bardzo znaczącym pomysłem, jest zliczanie wystąpień danego wyrazu w tekście. Przyjmując założenie, iż kolejność wystąpień nie ma znaczenia, otrzymujemy model 'bag of words', w dosłownym tłumaczeniu jest to model 'worka wyrazów'. Pomija się w nim kolejność występowania wyrazów, traktując dokument jako zbiór termów wraz z przypisaną im ilością wystąpień (ang. 'term frequency'). Najlepiej dopasowane dokumenty w takim modelu to te, w których wyszukiwane słowa kluczowe pojawiły się najwięcej razy (Manning, 2009). Częstość w dokumentach (ang. 'document frequency') jest miarą określającą w jakiej ilości dokumentów, spośród ogólnej ich liczby, wystąpił dany term. Jest to wielkość, która określa ilość wystąpień we właściwszy sposób niż częstość w korpusie (ang. 'collection frequency'), z powodu możliwych korelacji częstości występowania słów w pojedynczym dokumencie. Na podstawie tej miary tworzy się odwrotną częstość w dokumentach (ang. 'inverse document frequency') zdefiniowaną jako logarytm z ułamka dzielącego wszystkie dokumenty w korpusie przez ilość tych, w który wystąpił dany term. Rzadsze wyrazy otrzymują więc wyższy wynik *idf*, niż te które występują częściej. Na podstawie częstości wystąpień i odwrotnej częstości w dokumentach, zdefiniowana jest miara *tf-idf* (ang. 'term frequency - inverse document frequency') określona jako iloczyn dwóch wartości (*tf* oraz *idf*), dla danego termu w podanym dokumencie. Korzystając z tej wielkości możemy przypisać numeryczny wynik dopasowania, poprzez sumację *tf-idf* każdego z termów w zapytaniu odwołującym się do zadanej kolekcji dokumentów (Manning, 2009, s. 146-172).

$$(tf^{-}idf)_{i,j} = tf_{i,j} \times idf_{i,j} \tag{1.1}$$

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1.2}$$

$$idf_i = \log \frac{\|D\|}{d_{t_i}} \tag{1.3}$$

Wychodząc ponad model boolowski możemy przyjąć założenie mówiące, że każdy z dokumentów w kolekcji nad którą wyszukujemy może być reprezentowany przez wektor w przestrzeni 'słów'. Aby skonstruować odpowiadające wektory można skorzystać z pewnej miary będącej mapowaniem z termu do liczb naturalnych. Przykładową transformacją tego typu może być *td-idf*. Wektor dokumentu będzie się składał z wartości odpowiadających wystąpieniom kolejnych elementów posortowanego słownika termów. Taka reprezentacja nazywana jest modelem

przestrzeni wektorowej (ang. 'vector space model'). W przestrzeni wektorowej możemy obliczać bliskość dla dokumentów za pomocą produktu wewnętrznego pomiędzy wektorami znormalizowanymi do wektora jednostkowego, co odpowiada kosinusowi kąta (ang. 'cosine similarity') (Manning, 2009, s. 157-160).

Statyczna analiza jakości dokumentu pozwala na określenie wartości dokumentu bez zagłębiania się w jego zawartość, natomiast wykorzystując sygnały o jego jakości. Sygnałem może być liczba cytować, pozytywnych reakcji, czy popularność. Przy określaniu rangi taki statyczny sygnał o jakości może zostać dodany do wartości miary trafności dopasowania (Manning, 2009, s. 175-177). Naturalne wykorzystanie statycznego sygnału jakości występuje w modelu probabilistycznym, przedstawionym w dalszej części, gdzie możemy go interpretować jako prawdopodobieństwo wystąpienia danego dokumentu [P(d)].

Mierzenie wpływu (ang. 'impact ordering') polega na określaniu dopasowania, między zapytaniem a dokumentami, skupiając się na wpływie kolejnych termów, zaczynając od tych z największą wartością *idf* a więc mających potencjalnie największe znaczenie, ze względu na rzadkość występowania. Następnie dokumenty z kolekcji są sortowane po częstości wystąpienia termu *tf*, aż do momentu osiągnięcia określonej wartości progowej lub poprzestając na n najlepszych dopasowaniach. Proces rankingowy jest powtarzany dla kolejnych termów, akumulując wpływy z pojedynczych słów. Kiedy algorytm przejdzie wszystkie słowa kluczowe otrzymujemy ostateczny wynik dla zapytania (Manning, 2009, s. 177-178).

Podstawowym pytaniem, które należy sobie zadać przy tworzeniu systemu wyszukiwania mierzącego stopień dopasowania, jest kwestia tego, co chcemy osiągnąć poprzez wybrany system rankingu. Z filozoficznego punktu widzenia można założyć, że chcielibyśmy zmaksymalizować stopień zadowolenia użytkownika wyszukiwarki poprzez dostarczanie jak najbardziej przydatnych mu informacji, jako pierwszych wyników. Ważną kwestią, w takim przypadku, jest określenie czy maksymalizujemy bezwzględną przydatność, pomijając zbieżności, które mogą występować między dokumentami, czy preferujemy optymalizować krańcową przydatność, a więc taką która bierze pod uwagę zawartość informacyjną innych wysoko pozycjonowanych odpowiedzi do zapytania. Ponadto użytkownik oczekuje szybkości działania, ekspresywnych sposobów wyszukiwania, czytelnej prezentacji wyników, czy też dużej bazy plików. Wszelkie dodatkowe cechy mogą mieć decydujące znaczenie dla osoby korzystającej z danego rozwiązania silnika wyszukiwarki, a jednocześnie niekoniecznie są one związane z

centralnym sposobem działania (Manning, 2009, s. 205-207).

Wydaje się, iż dobrym sygnałem dopasowania rezultatów wyszukiwania do zadanego zapytania byłoby ocenienie jakości wyników przez użytkownika (ang. 'relevance feedback'). Dane te można by wykorzystywać do poprawiana jakości systemu wyszukującego wyniki, ale również w celu interaktywnej pracy z wyszukiwarką. W takim układzie użytkownik mógłby wybierać zwrócone wyniki, które okazały mu się przydatne, natomiast wyszukiwarka próbowałaby ulepszyć kolejne zwrócone wyniki na podstawie tego sygnału zwrotnego. W przestrzeni wektorowej wykorzystanie tego pomysłu opiera się na teorii mówiącej, iż aby otrzymać wektor, według którego będziemy wyszukiwać, należy do początkowego wektora zapytania dodać średnią ważoną wektorów dokumentów dopasowanych, a następnie odjąć od niej średnią ważoną niedopasowanych wektorów wynikowych. Każdy z elementów składowych zostaje przemnożony przez wagę, którą można zmieniać w zależności od zastosowania. Algorytm Rocchia działa dla kolejnych zapytań następujących po pierwszym z nich. Chociaż rozwiązanie takie wydają się interesujące to dużym problemem, na który napotyka, jest niechęć użytkowników do udzielania informacji zwrotnej. Pewne badania pokazują, że tylko 4% użytkowników korzystało z funkcji pozwalającej wyszukać więcej rezultatów podobnych do wyświetlonego (opcja 'more like this') (Manning, 2009, s. 222).

Joachims argumentuje, że aby wykorzystać sygnał zwrotny od użytkownika nie trzeba się go o to pytać bezpośrednio, lecz wystarczające informacje znajdują się już w logach zbieranych przez wyszukiwarki internetowe. Formalnie możemy myśleć o danych dotyczących kliknięć (ang. 'clickthrough data') jako o matematycznej trójce (q, r, c) zbierającej zadane zapytanie q, odpowiadający mu zwrócony ranking wyników r, oraz kliknięte rezultaty c. Ponieważ generalnie, linki zwrócone jako pierwsze powinny posiadać wyższe prawdopodobieństwo kliknięcia w nie, należałoby myśleć o zebranej informacji raczej jako relatywnym rankingu oceniającym kolejność preferowaną przez użytkownika względem tej wyświetlonej. Autor użył modelu uczenia maszynowego SVM do modelowania tych zależności. W małym badaniu próbującym ocenić, o ile więcej kliknięć generuje nauczona funkcja od wyników z porównywanych wyszukiwarek, m. in. Google, wyniki świadczą na korzyść nauczonej funkcji ze znaczeniem statystycznym (Joachims, 2002).

Ważnym narzędziem pozwalającym pozbyć się problemu różnego słownictwa (ang. 'vocabulary mismatch'), między zapytaniem a dokumentami zawierającymi

wyszukiwane informacje, jest tzw. rozszerzanie zapytania (ang. 'query expansion'). W tym celu często wykorzystywany jest tezaurus, który może być wygenerowany automatycznie, pokazujący powiązania między wyrazami takie jak synonimy. Te dodatkowe słowa mogą zostać dodane do zapytania. Przykładowo gdy wpiszemy *samochód*, dodany zostanie wyraz *auto* (Manning, 2009).

Rozważając przypadek wyszukiwania w plikach dostępnych przez sieć internetową, należy mieć na uwadze fakt, iż większość dostępnych dokumentów jest plikami HTML, plikami CSS, skryptami JavaScript, a nie plikami tekstowymi. Pliki tekstowe, ale również obrazy, filmy, pliki audio są zazwyczaj dostępne jako część struktury DOM pliku HTML. XPath jest językiem zapytań dla plików XML, który może być również rozszerzony o możliwość nawigacji po DOM innych podobnych języków znaczników takich jak HTML5.

Trzecim typem modeli wyszukiwania informacji obok modeli boolowskich i algebraicznych są modele probabilistyczne. Ich teoretycznego uzasadnienia można szukać w probabilistycznej zasadzie szeregowania (ang. 'Probabilistic Ranking Principle') mówiącej, że dokumenty powinny być uszeregowane w kolejności prawdopodobieństwa dopasowania zapytania do dokumentu (Robertson, 1977). Nieznając tych prawdopodobieństw, próbujemy je szacować, wykorzystując zasadę Bayesa. Tradycyjnym modelem używanym w tym kontekście jest binarny model z niezależnymi zmiennymi (ang. 'Binary Independence Model'), zakładający brak zależności statystycznej pomiędzy elementami wektora opisującymi dokumenty. Założenie to daje początek klasyfikatorowi nazywanemu naiwny Bayes (ang. 'naive Bayes'). Założenie takie jest oczywiście nieprawdziwe, lecz pozwala na pewne kluczowe uproszczenia. Model probabilistyczny daje podstawę teoretyczną podpierającą *idf*, przy dodatkowym założeniu, że dokumenty określane jako pasujące do zapytania, stanowią zerową część wszystkich dokumentów, co zresztą jest często bliskie prawdy. Jednak najbardziej znanym modelem probabilistycznym jest Okapi BM25. Wielkością służącą do estymowania jakości dopasowania w modelach probabilistycznych jest RSV (ang. 'Retrieval Status Value'). Równanie BM25 jest podobne do *tf-idf*, tyle że w równaniu dodane są parametry konfigurujące ważność *tf* dla dokumentu jak i zapytania, oraz parametr modyfikujący wartość RSV dla różnych długości dokumentów (Manning, 2009).

$$BM25(q,d) = \sum_{i=1}^{|q|} idf(q_i) \cdot \frac{tf(q_i,d) \cdot (k+1)}{tf(q_i,d) + k \cdot (1-b) + b \cdot \frac{|d|}{avgdl}} \tag{1.4}$$

Jeszcze innym podejściem do wyszukiwania są modele językowe (ang. 'Language Model'). Model językowy generuje rozkład prawdopodobieństwa nad słowami ze słownika. Może być zrealizowany jako automat skończony, z którego możemy odczytać prawdopodobieństwa sekwencji. Wygenerowany, dla konkretnego dokumentu, model językowy jest używany do określenia prawdopodobieństwa warunkowego na otrzymanie zapytania pod warunkiem, że zaszedł warunek wystąpienia modelu językowego dla dokumentu. Biorąc znowu pod uwagę twierdzenie Bayesa wiemy, że znając prawdopodobieństwo na wystąpienie modelu językowego [P(Md)] i prawdopodobieństwo zapytania [P(q)], możemy przekształcić otrzymane prawdopodobieństwo warunkowe na zaistnienie zapytania pod warunkiem modelu językowego [P(q|Md)] do prawdobodobieństwa na zaistnienie modelu językowego pod warunkiem wystąpienia zapytania [P(Md|q)]. Jeśli traktujemy mapowanie z dokumentu do modelu jako zachowujące to prawdopodobieństwo [P(Md|q) ~P(d|q)] to w wyniku mamy wynik dopasowania w probabilistycznym podejściu. Najprostszym sposobem wyszukiwania tego rodzaju jest model unigramowy zakładający niezależność wystąpień termów w ich sekwencji. Możliwe są również modele bigramowe jak i n-gramowe, reprezentujące powiązania między różnymi wyrazami (Manning, 2009). Nowe osiągnięcia w dziedzinie sieci neuronowych umożliwiły również stworzenie wyrafinowanych modeli języka, co zostało opisane w rozdziale drugim.

$$P(\frac{Md}{q}) = \frac{P(\frac{q}{Md}) * P(q)}{P(Md)} \tag{1.5}$$

$$P(\frac{d}{q}) \sim P(\frac{Md}{q}) \tag{1.6}$$

W tzw. wyszukiwaniu gęstym (ang. 'dense retrieval'), w przeciwieństwie do wyszukiwania informacji typu rzadkiego (ang. 'sparse retrieval'), wyszukujemy bezpośrednio w reprezentacji, która może przyjmować pewne spektrum obecności danej właściwości. Jako przykład weźmy model przestrzeni wektorowej. Aby odnaleźć inne wektory w podobnej odległości, będziemy musieli porównać, w najgorszym przypadku, wszystkie inne wektory reprezentujące dokumenty. Aby zapobiegać tak niekontrolowanemu wzrostowi złożoności obliczeniowej wprowadza się metody ograniczające początkową listę kandydatów. Algorytmy analizy skupień pozwalają podzielić przestrzeń wyników w regiony o podobnych reprezentacjach. Dzięki takiemu

podejściu, możemy odnaleźć najpierw najbliższe skupienia, a kolejno ograniczyć wyszukiwanie do dokumentów znajdujących się w środku danej grupy. Jednym z najprostszych algorytmów tego rodzaju jest algorytm centroidów (ang. 'k- means'). Polega on na zainicjowaniu k różnych centroidów, czyli punktów środka masy. Kolejno do każdego z punktów, które chcemy przypisać, wybieramy najbliżej leżący centroid. Kiedy wszystkie punkty są przypisane, obliczamy ponownie nowe środki masy względem punktów przypisanych do danego skupienia. Proces powtarzamy aż osiągniemy zadaną dokładność. Algorytm jest zbieżny do minimum lokalnego. Kwestią decyzji pozostaje wybór ilości skupień. Chociaż istnieją heurystyczne metody wyboru tego parametru, to są one jedynie przypuszczeniami na temat zasadniczego zbioru danych. Algorytmy analizy skupień są algorytmami uczenia bez nadzoru, a więc szukają wzorców na własną rękę, albo mówiąc inaczej: rozwiązują problem maksymalizacji będący zastępcą prawdziwie interesującego nas problemu. Często takie przeformułowanie związane jest z wykorzystaniem pewnego rodzaju złożoności, biorąc pod uwagę fakt preferowania prostszych rozwiązań nad bardziej skomplikowane, w obliczu braku dodatkowych przesłanek. Jednak w wyniku takiej procedury nie możemy być pewni, że znalezione skupienia, choćby były najlepiej skoncentrowane, odpowiadają rzeczywistym interesującym nas klasom dokumentów. Potencjalnie możliwa jest również hierarchiczna analiza skupień. Efektywnym rozwiązaniem do analizy skupień jest biblioteka Faiss od Facebook AI Research (obecnie Meta AI Research), pozwalająca na oszczędne zasobowo obliczenia dotyczące skupień (Johnson et al., 2019).

1.3. Zastosowania w wyszukiwaniu pełnotekstowym

Przed powstaniem wyszukiwarki internetowej Google istniały inne rozwiązania służące do poruszania się w sieci. Wiele z pierwszych rozwiązań korzystało jednak tylko z tytułów stron albo meta-tagów, które zostały zdefiniowane między innymi do tego właśnie wykorzystania (Berners-Lee, 1995). Jednak problemem stał się stopniowo brak wiarygodności osób i instytucji dodających swoje witryny internetowe, które próbując zwiększyć wyszukiwalność swojej strony nadużywały możliwości m.in. w zakresie definiowania zawartości swojej strony w meta-tagach — proceder zwany spamdexowaniem (ang. 'spamdexing'). Pierwszą wyszukiwarką, która korzystała z indeksowania pełnotekstowego, była AltaVista (Broadley, 2023). Indeksowanie całej zawartości strony jest trudniejsze, wymaga też większych zasobów przeznaczonych na przeczesywanie stron, jednak daje dużo bardziej wiarygodne rezultaty. Obecnie technologia wyszukiwania pełnotekstowego jest wykorzystywana również przez witryny, które dodają możliwość przeglądania swoich zasobów za pomocą okienka

wyszukiwarki. Techniki te w odróżnieniu od wyszukiwarek internetowych działają na ograniczonej dziedzinie zasobów, na które składa się pewna kolekcja plików tekstowych specyficzna dla danej witryny. Istnieje wiele rozwiązań umożliwiających dodanie takiego wyszukiwania (Dutton, 2018). Wśród nich można wyróżnić np. otwartoźródłowe oprogramowanie Apache Lucene, wraz z projektami o nie opartymi takimi jak Solr. Dostępnych jest również wiele innych rozwiązań pochodzących od różnych dostawców takich usług. Można tu wyróżnić: Google Programmable Search Engine, AWS CloudSearch, Elasticsearch, czy też Algolię.

Wyszukiwarki pełnotekstowe są często zbliżone w swoim działaniu do pełnowymiarowych wyszukiwarek internetowych, a jednocześnie czasem dzielą się szczegółami dotyczącymi działania swoich systemów. Wyszukiwanie pełnotekstowe korzysta u swoich podstaw z rozwiązań dziedziny wyszukiwania informacji (IR) i przetwarzania języka naturalnego (NLP). Kluczowym dla działania tych wyszukiwarek są parsowanie, stemming, indeksowanie (Dutton, 2018). Aby zapewnić odpowiednią wygodę użytkowania, stosowanych jest wiele udogodnień i usprawnień algorytmicznych. Przykładowym rozwiązaniem, którego elementy składowe opiszę jest Algolia, której pracownicy prowadzą blog dotyczący inżynierii w tej firmie.

W serii artykułów poświęconych rozwiązaniom wykorzystywanym w ich wyszukiwarce zwracają uwagę na wyścig o zasoby, który ma miejsce pomiędzy zadaniem wyszukiwania w indeksie a procesem ciągłego indeksowania. Konkurencja ta ujawnia się pod postacią wykorzystania zasobów CPU oraz pojawienia się braku możliwości wykonania operacji zapisu, a także odczytu na dysku. Algolia rozwiązuje te problemy poprzez utworzenie dwóch procesów systemowych o różnych priorytetach oraz przechowywaniem całego indeksu w pamięci operacyjnej (Lamoine, 2016a, a). Użytkownik wyszukiwarki oczekuje, że pole wyszukiwania będzie mu podpowiadać możliwe kolejne słowa w zapytaniu. Takie rozwiązanie nosi nazwę przyrostowego wyszukiwania (ang. 'incremental search') lub sugestii w czasie rzeczywistym (ang. 'real-time suggestion', 'search-as-you-type'). Możliwe rozwiązania algorytmiczne pozwalające zastosować taki mechanizm to skorzystanie z wcześniej utworzonego drzewa Trie przechowującego przedrostki do indeksowanych wyrazów, w taki sposób, że łatwo można podać możliwe kontynuacje wyrazów, gdy znamy ich przedrostki. Algolia wykorzystuje skompresowane drzewo trie (ang. 'radix tree'), które oblicza wykorzystując minimalną możliwą ilość pamięci operacyjnej, aby zwolnić miejsce dla strukury indeksującej (Lamoine, 2016b). Przy wpisywaniu danego zapytania użytkownicy popełniają błędy we wprowadzonych literach. Aby temu zapobiec silnik

Algolii oblicza dystans Damerau-Levenshtein'a i przeszukuje słownik dla ograniczonej wielkości błędu w celu znalezienia poprawnej pisowni słowa (Lamoine, 2016c). Ponadto stosowanymi rozwiązaniami jest lematyzacja i powiększanie zapytania o synonimy. Kolejnym stosowanym rozwiązaniem jest podkreślanie wyrazów w wyświetlanym wycinku. Kiedy znalezione zostaną dokumenty, które pasują do zadanego zapytania, problemem może się okazać kolejność ich wyświetlenia. Użytkownik oczekuje, iż dokumenty najbardziej pasujące do wyszukiwanego zapytania znajdą się u góry sekcji wyników. W tym celu Algolia wykorzystuje kilka metryk określających dopasowanie dokumentu do zapytania. Brane pod uwagę kryteria to m.in.: ilość błędów występujących między słowami zapytania a tymi w dokumencie, ilość dopasowanych słów, bliskość występowania pomiędzy szukanymi słowami, ważność atrybutu, dokładność - rozumiana jako dopasowanie całkowite albo przedrostkowe (Lamoine, 2016d). Zamiast obliczać składową wartość rankingu dla każdego atrybutu, silnik Algolia stosuje algorytm typu 'tie-breaking', który sortuje wyniki według pierwszego kryterium, a następnie, jeśli zachodzi remis dla pewnych pozycji to sortuje je po kolejnych kryteriach aż do momentu, gdy nie występują impasy. Takie podejście zapobiega porównywaniu wartości numerycznych charakteryzujących dopasowanie, gdzie może zachodzić sytuacja wzajemnego przeważania się kryteriów.

1.4. Zastosowania w wyszukiwarkach internetowych

Istnieją różne rozwiązania pozwalające na wyszukiwanie w sieci, wśród nich: rozwiązanie Microsoftu Bing – wbudowane w wyszukiwarkę Edge, Yahoo – posiadające kilkunastoprocentowy udział na rynku wyszukiwarek w Japonii (Statcounter, 2022), czy Baidu ze swoim dominującym udziałem w Chinach (MarketMeChina, 2022). Jednak niekwestionowanym liderem na globalnym rynku wyszukiwania internetowego pozostaje Google ze swoim ponad 90% udziałem (Kinsta, 2022). Mimo, że nie była pierwszą szeroko używaną wyszukiwarką internetową, założona przez Larrego Page'a i Siergeya Brin'a firma tworzy technologię, która stała się de facto standardem. Wyszukiwarka internetowa działa na różnych płaszczyznach, integrując ze sobą wiele procesów informacyjnych. Do działania potrzebuje co najmniej: procesu skanowania zasobów dostępnych w internecie przy użyciu automatycznych robotów zwanych pająkami (ang. 'crawler'), procesu indeksowania zapamiętującego w efektywny sposób słowa kluczowe i inne informacje o stronie, procesu analizy zapytań oraz wyświetlania wyników (Google Developers, 2023).

Kluczowym aspektem dla działania wyszukiwarki internetowej jest zebranie

danych o stronach, które będzie można później zwrócić jako wyniki. Dane te pomimo posiadania swojej wewnętrznej logiki, są semistrukturalne lub nieustrukturyzowane, a więc wymagają różnorodnego przetworzenia, aby wydobyć z nich użyteczne informacje. Jest to główny czynnik, obok samej skali działania, odróżniający wyszukiwanie internetowe od wyszukiwania pełnotekstowego. Przeczesywanie stron internetowych za pomocą robotów, w kontekście wyszukiwania informacji, ma na celu odnalezienie, przetworzenie i zapamiętanie informacji o stronie, potrzebnych do jej odnalezienia podczas późniejszego wyszukiwania. Pająk sieciowy zaczyna swoje poszukiwania od pewnego zbioru adresów URL, zwanych ziarnem (ang. 'seed URLs'). W klasycznej teorii przeczesywania stron zakłada się, że każda strona w sieci jest osiągalna przez pewien adres URL. Za adresy ziarna wybierane zaś były strony posiadające odnośniki do wielu innych stron. Jednak obecnie, założenie o osiągalności każdej strony przez unikalny adres URL, do którego można dotrzeć z głównej strony danej witryny, jest wątpliwe. Istnieją witryny generujące swój widok interaktywnie w odpowiedzi na akcje użytkownika i przy tym nieodsyłające do nowego adresu. Problem ten adresuje nowy kierunek badań nazywany po angielsku 'deep web-crawling' (Mirtaheri et al., 2014). Głównymi cechami definiującymi użyteczność pająka są pokrycie (ang. 'coverage') i świeżość (ang. 'freshness'), które określają kolejno jaki udział wszystkich stron został odwiedzony przez pająka oraz jak dawno miało to miejsce. Dodatkowo ważnym atrybutem dla nowych, głębokich pająków jest pełność (ang. 'completeness') pokrycia zasobów z danej witryny (Mirtaheri et al., 2014). Ponieważ pająki mogą odwiedzać strony internetowe o wiele szybciej niż ludzie, to aby nie nadwyrężać zasobów serwera, i potencjalnie narażać się na ograniczenia dostępu do niej, pająki stosują zasadę uprzejmości (ang. 'politeness policy'), która opóźnia wysyłanie kolejnego żądania o przynajmniej kilka do kilkudziesięciu sekund. Każdy pająk posiada swoją strategię poruszania (ang. 'crawl policy'). Definiuje ona, w jakiej kolejności powinny być odwiedzane linki, które zostały zebrane do kolejki. Kiedy robot odwiedza stronę uzupełnia kolejkę o linki do stron z niej wychodzących w celu ich sukcesywnego przeczesania. Zazwyczaj pająki internetowe korzystają ze strategii rozwijania najpierw najpłytszych rozgałęzień (ang. 'breadth-first'), albo też zaczynając od rozwijania jednego rozgałęzienia do końca (ang. 'depth-first'). Niedawne badania sugerują strategię chciwą, jako lepszą alternatywę dla poruszania się po bogatych aplikacjach internetowych (RIA) (Mirtaheri et al., 2014). W celu utrzymania wysokiej świeżości stron roboty odwiedzają witryny ponownie. Proces ten jest kontrolowany przez strategię ponownych odwiedzin (ang. 're-visit policy'). Pająki identyfikują się podczas korzystania z zasobów poprzez pole 'User-agent' w nagłówku zapytania. Uprzejmy pająk wchodzący na stronę sprawdza plik 'robots.txt',

który zawiera specyfikację udzielonych zezwoleń na dostęp do zasobów dla określonych robotów. Plik ten jest zdefiniowany według reguł Robots Exclusion Protocol (Google, 2023c). Zbiór reguł zawartych w 'robots.txt' może chronić przed nieumyślnymi pułapkami na roboty (ang. 'spider trap'), która występuje, gdy działanie pająka zapętla się na danych podstronach. Taka sytuacja ma miejsce, chociażby przy nieskończonych, generowanych zasobach, takich jak te występujące przy korzystaniu z automatycznie generowanego internetowego kalendarza. Ważnymi elementami architektury klasycznego pająka internetowego jest używanie wielowątkowego pobierania w celu zwiększenia przepustowości, czarna lista witryn znanych jako nieprzyjazne dla robotów, program do normalizacji adresów URL – konieczny, żeby nie odwiedzać identycznego adresu dwoma różnymi ścieżkami (Mirtaheri et al., 2014). Architektura nowoczesnego głębokiego pająka uruchamia RIA na silniku JavaScript. Następnie moduł 'DOM-Seen' sprawdza, czy dany obiektowy model dokumentu (DOM) był już widziany. Jeśli nie był, to przeprowadzana jest ekstrakcja wszystkich JavaScriptowych zdarzeń, których kolejność uruchomienia określana jest przez strategię (Mirtaheri et al., 2014). Google stosuje dwa rodzaje robotów: mobilny i desktopowy. Są one uruchamiane na tysiącach komputerów w lokalizacjach geograficznych ułatwiających dostęp do treści (Google, 2023b). Po uzyskaniu zawartości witryny Google próbuje ją zindeksować, biorąc pod uwagę: tekst, tagi, atrybuty treści oraz inne zasoby takie jak obrazy czy filmy. Sprawdzane też, jest to, czy dana strona jest duplikatem innej już zindeksowanej strony (Google, 2023d). Różne informacje na temat zbieranych informacji oraz wyszukiwań, dzięki którym użytkownik odnalazł naszą stronę, można uzyskać w Google Search Console.

Wyszukiwarka Google została stworzona początkowo, aby przetestować algorytm PageRank. Był on motywowany różnorodną jakością witryn internetowych i próbą zmierzenia ich jakości za pomocą łączących je linków — podobnie jak w publikacjach naukowych. PageRank oblicza w sposób iteracyjny rekursywne równanie na ranking strony. Interpretacją tego procesu jest błądzenie losowe po stronach (ang. 'random walk', 'random surfer'). Ponadto do równania dodajemy wektor będący odpowiednikiem znudzenia się użytkowania i przejścia do losowej strony. Ten mechanizm zapobiega drenażowi rankingu (ang. 'rank sink'). Sytuacji, w której linkujące tylko do siebie strony zbierają niewspółmierną ilość rankingu. PageRank znacząco poprawia precyzję uzyskiwanych wyników, co jest ważne dla niedodefiniowanych zapytań. Algorytm może być również przydatny przy określaniu kolejności odwiedzania stron przez pająki internetowe, będąc lepszym predyktorem ilości cytowań od samej liczby znanych cytowań (Page, 1998).

Wykorzystywanym w Wyszukiwaniu Google narzędziem jest NoSQL-owa baza danych BigTable, która jest wielowymiarową posortowaną tablicą asocjacyjną indeksowaną w oparciu o klucz rzędu oraz kolumny i posiadającą związany z nią znacznik czasu. Dane przechowywane są jako nieinterpretowany ciąg bajtów. Każdy zapis, jak i odczyt z pojedynczego rzędu jest atomiczny. Rekordy są przechowywane w kolejności leksykograficznej. Kolumny posiadają swój rodzaj zwany rodziną, który pozwala na optymalizację zapisu. BigTable jest wyposażona w możliwość przechowywania do **n** wersji dokumentu z różnymi czasami dodania pod jednym kluczem. Rozwiązanie to używa Google File System do magazynowania logów i plików, może być również używane wraz z rozwiązaniem MapReduce umożliwiającym paralelizm obliczeń na wielu komputerach (F. Chang et al., 2008). Formatem przechowywania danych jest wewnętrzny Google format SSTable. Występuje oddzielenie struktur danych przechowywanych w systemie plików Collosus, a samymi klastrami obliczeniowymi, które wykonują operacje dla serwera frontendowego (Google, 2023a). Rozwiązanie opiera się na architekturze 'master-slave' gdzie jeden serwer zarządza zadaniami, zajmuje się balansowaniem obciążenia serwerów, odśmiecaniem pamięci, a reszta odpowiada za przechowywanie oraz obsługę żądań. BigTable korzysta z kompresji, która dzięki lokalizacji podobnych plików z jednej witryny niedaleko siebie, osiąga poziom 10 do 1. Filtry Blooma są preferowane ponad wyszukiwanie binarne, żeby przyspieszyć sprawdzanie istnienia rekordów (F. Chang et al., 2008).

Rozdział 2

Nowoczesne narzędzia wyszukiwania informacji

2.1. Komputerowe reprezentacje tekstu

2.1.1. Wektory słów

Chociaż w niektórych rozwiązania wystarczającą techniką kodowania słów jest potraktowanie ich jako niezależnych wystąpień w słowniku, to istnieją sytuacje, w których możliwość mierzenia odległości syntaktycznej lub semantycznej jest niezaprzeczalnie przydatna. Do tworzenia takich rozproszonych reprezentacji można używać technik sztucznych sieci neuronowych.

Praca naukowa pod przetłumaczonym tytułem 'Efektywna estymacja reprezentacji słów w przestrzeni wektorowej', zdobyła duży rozgłos (ponad 32 tysiące cytowań) dzięki wprowadzeniu interesującego testu na efektywność wektorów słów. We wcześniejszych pracach popularną techniką porównywania jakości nauczonych reprezentacji było mierzenie bliskości zakodowanych słów, które są do siebie najbardziej podobne. Wprowadzona przez autorów technika pozwala na bardziej zniuansowany sposób porównywania takich systemów. Test opiera się na obserwacji mówiącej, iż rozwiązanie zadania określania niektórych typów relacji między słowami można sprowadzić do dodawania i odejmowania wektorów ich reprezentacji. I tak, na przykład, aby zmierzyć rozumienie relacji "Jaki jest wyraz podobny do *mały* w sensie takim jak *największy* jest podobny do *dużego*?", autorzy obliczają wektor X uzyskany z reprezentacji w następujący sposób: X = *wektor('duży') - wektor('największy') + wektor('mały')*. Zaproponowane zadanie składa się z prawie 9,000 pytań semantycznych oraz ponad 10,000 pytań syntaktycznych w formie, gdzie na podstawie jednej pary wyrazów

trzeba wywnioskować drugi wyraz z drugiej pary. Typem użytej relacji semantycznej są przykładowo: państwo-stolica czy państwo-waluta, natomiast syntaktycznej: przeciwieństwa i zmiany czasu w jakim występuje dany wyraz. Oryginalna metoda proponuje aby uznawać za prawidłowe odpowiedzi tylko te gdzie reprezentacja jest najbliższa do reprezentacji poprawnego wyrazu (Mikolov et al., 2013).

Autorzy zaproponowali również dwa nowe oraz porównali dwa istniejące modele kodowania słów do wektorów. Jednym z wcześniej używanych do tego celu modeli była sieć neuronowa do modelowania języka (NNLM), składająca się z warstw wejścia, projekcji, ukrytej oraz wyjścia. Na wejściu, N poprzednich słów jest kodowanych przy pomocy kodowania 1 z n (ang. 'one-hot encoding'), gdzie n jest rozmiarem słownika. Następnie warstwa projekcji jest używana w celu przetworzenia danych wejściowych w gęstą reprezentację. O warstwie tej można myśleć jak o wybierającej, w sposób deterministyczny dla danego słowa, związaną z tym słowem kolumnę w macierzy odwzorowania. Kolejno połączone gęste wektory słów są przetwarzane przez warstwę ukrytą sieci, aby zostać zwrócone jako dystrybucja prawdopodobieństwa nad słowami ze słownika. Z powodu różnej częstotliwości występowania słów w języku, przydatnym dla autorów okazało się użycie kodowania Huffmana, gdzie częściej występujące wyrazy otrzymują krótsze kody, tym samym zmniejszając ogólną objętość wstępnie zakodowanego tekstu. W celu przyspieszenia działania tej sieci użyto w tym wypadku również hierarchicznej warstwy *softmax* stworzonej na podstawie drzewa binarnego odpowiadającego kodowaniu Huffmana. Nowe zaproponowane modele to CBOW (ang. 'Continuous Bag-of-Words Model') oraz CSM (ang. 'Continuous Skip-gram Model'). Pierwszy z nich jest podobny do modelu neuronalnego bez warstwy ukrytej. Kody czterech wyrazów z otoczenia lewo, jak i prawostronnego przewidywanego słowa są mapowane przez macierz odwzorowania. Ich reprezentacje zostają połączone przez uśrednienie. Na tych danych działa klasyfikator, próbujący dopasować przesłonięte środkowe słowo. Drugim z modeli jest CSM, który uczy się odwrotnej zależności. Zadanie polega w tym przypadku na wygenerowaniu słów kontekstu. Autorzy użyli, w tym przypadku, klasyfikatora działającego na zmiennej długości lewego i prawego kontekstu wynoszącego od 1 do 10 słów (Mikolov et al., 2013).

2.1.2. Wektory paragrafów

Nauczone reprezentacje słów mogą zachowywać relacje między słowami danego języka, a także być przydatne jako dobre początkowe kodowania, na których będzie uczony nowy klasyfikator lub model językowy. Jednak aby w sposób jawny można identyfikować zależności między większymi częściami tekstu, przydatnym byłby

sposób osadzania całych zdań, a nawet paragrafów o zmiennej długości. Przed erą nowoczesnych dużych modeli językowych (LLM) udanym rozwiązaniem, które pozwalało na uczenie takich zależności był system *Doc2vec* zaprezentowany w pracy 'Dystrybuowane reprezentacje zdań i dokumentów' (ang. 'Distributed Representations of Sentences and Documents').

Wektory paragrafów są motywowane koniecznością reprezentowania danego tekstu jako wektora o określonej długości dla wielu zastosowań takich jak klasyfikacja tekstu. Było to szczególne istotne w latach, gdy nie istniały modele pozwalające na sekwencyjne przetwarzanie dokumentu poprzez obliczenia na pewnej grupie tokenów znajdujących się w kontekście o zadanej wielkości. Mimo istnienia wcześniejszych metod reprezentowania wycinków tekstu były one niezadowalające ze względu na swoje ograniczenia: tracenie ważnych informacji albo ograniczenie do pojedynczych zdań (Le & Mikolov, 2014).

Opisywany algorytm działa na podobnej zasadzie jak wcześniejszy model osadzania słów w wektorach. Słowa są mapowane przy użyciu kolumny macierzy W do wektora. Dodatkowo *id* paragrafu jest mapowane wykorzystując macierz D. Wiąże się to z rozmiarem tej macierzy, która posiada ilość kolumn równą liczbie paragrafów w zbiorze uczącym. Powstały wektor paragrafu służy jako pamięć, która jest w stanie zapisać informacje o brakującym kontekście. Następnie reprezentacje słów, z kontekstu liczącego od kilku do kilkunastu słów, są łączone z wektorem paragrafu poprzez konkatenację lub uśrednianie. Na podstawie tych danych obliczana jest reprezentacja przekrytego słowa. Macierze transformacji są uczone na zbiorze, który powstaje poprzez próbkowanie kontekstu z paragrafu używając stochastycznego algorytmu propagacji wstecznej. Podczas fazy wnioskowania modelu reprezentacja nowego paragrafu musi zostać nauczona, przy innych wagach pozostających bez zmiany. Autorzy używają tej metody w połączeniu z wektorami nauczonymi na zadaniu przewidywania kontekstu z samego wektora paragrafu (Le & Mikolov, 2014).

Algorytm uczenia wektorów paragrafów okazał się sukcesem przy użyciu na zbiorach danych dotyczących przewidywania sentymentu: Stanford Sentiment Treebank Dataset, a także zbiorze recenzji IMDb. Metoda osiągnęła statusu najlepszego dostępnego w tamtym czasie rozwiązania na tych zbiorach. W kontekście wyszukiwania informacji ważniejszy jest jednak fakt udanego zastosowania tego modelu do wyszukiwania informacji. Na podstawie niedostępnego publicznie zbioru miliona najpopularniejszych zapytań do wyszukiwarki Google i odpowiadających im

dziesięciu najwyżej zwróconych rezultatów, autorzy zaproponowali zadanie podobne w swojej strukturze do późniejszych rozwiązań tego typu. Spośród rezultatów jest wybierana trójka, tak że dwa paragrafy pochodzą z rezultatów jednego zapytania natomiast trzeci z dowolnego paragrafu kolekcji. Celem jest odpowiednie określenie, które z elementów są rezultatami pojedynczego zapytania. Wektory paragrafów są w stanie osiągnąć na tym zbiorze rezultat lepszy od porównywanej metody *bag-of-bigrams* wraz z ważeniem przy pomocy TF-IDF (Le & Mikolov, 2014).

2.2. Modele uczenia maszynowego

2.2.1. Transformery

Było to w pracy naukowej zatytuowanej *Attention is all you need*, gdy Vaswani et al., 2017 zaprezentowli model sieci neuronowej nazwany **transformerem**. Składał się on z enkodera i dekodera, złożonego z kilku warstw, gdzie każda posiadała dwie warstwy wewnętrzne: mechanizm uwagi oraz warstwę jednokierunkowej sieci neuronowej. Użyto połączeń rezydualnych, a także normalizacji warstw. Bloki dekodera zostały rozszerzone o dodatkowy mechanizm uwagi, który przy odpowiednim maskowaniu zbiera również informację o kodowaniu z odpowiadających warstw enkodera. Kluczową innowacją był użyty model uwagi pozwalający na łączenie informacji z różnych części zdania, co było utrudnione w rozwiązaniach wykorzystujących inne modele obliczeniowe. Popularne wcześniej sieci rekurencyjne mogą się odnosić oraz propagować gradient tylko względem poprzedzającego tokenu co znacząco utrudnia uczenie długich zależności ze względu na problem zanikających gradientów. Implementacją modelu uwagi, która została użyta jest *uwaga typu zeskalowanego produktu iloczynowego (ang. scaled dot product attention)*. Działa ona zgodnie z poniższym równaniem:

$$Uwaga(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

Macierze **Q, K, V** składają się kolejno z wielu wektorów zapytania, klucza i wartość. Parametr $\mathbf{d_k}$ odpowiada za skalowanie ze względu na rozmiar macierzy.

Autorzy używali również wielogłowicowej uwagi (ang. multi-head attention), gdzie każda z głowic operowała na projekcji zapytania, klucza i wartości, które zostały otrzymane przez wymnożenie ich z parametrycznymi macierzami. Wyniki były łączone przez konkatynację i mapowane przez macierz $\mathbf{W^o}$.

Sieć transformera korzysta z nauczonych reprezentacji wektorowych, które odpowiadają wybranej wewnętrznej wielkości reprezentacyjnej modelu $\mathbf{d_{model}}$. Wyjście może natomiast być przetworzone przez warstwę *softmax*, aby zwracało prawdopodobieństwo nad słownikiem. Ponieważ model nie zachowuje naturalnie kolejności tokenów, twórcy używają funkcji sinus i kosinus różnych częstotliwości aby zakodować położenie tokenu.

Już w tej pierwszej pracy na temat nowej architektury autorzy udowodnili jej przydatność osiągając najlepszy w tamtym czasie wynik w translacji pomiędzy francuskim a angielskim, jak i niemieckim a angielskim. Sieć była też bardziej efektywna pod względem zużytych zasobów obliczeniowych do trenowania od istniejących alternatyw. Wytrenowanie jej do poziomu najlepszego istniejącego rozwiązania zajęło autorom zaledwie 3,5 dnia na ośmiu procesorach GPU P100 (Vaswani et al., 2017).

2.2.2. BERT: Bidirectional Encoder Representations from Transformers

BERT, w odróżnieniu od wcześniejszego GPT, jest modelem językowym trenowanym na zarówno lewy jak i również prawym kontekście. Ta właściwość zapewnia potencjalną przewagę nad modelami, które potrafią przewidywać tylko następny token. Zadanie treningowe dla BERT'a jest oparte o test luk (ang. Cloze task). W tej wersji, celem jest wypełnienie losowych luk powstałych poprzez zakrycie tokenów pochodzących z wejściowego ciągu, tak aby zachodziła zgodność między oryginalnymi nieznanymi dla modelu słowami, a tymi które zostały przewidziane przez model.

Styl uczenia się BERT'a oparty jest na dwóch fazach: treningu wstępnego oraz dostrajania. Wzorując się na poprzednich osiągnięciach, które pokazują korzyści płynące z treningu na ogólnym zadaniu z danej dziedziny oraz późniejszym dotrenowaniu architektury do konkretnego zadania końcowego, autorzy proponują rozwiązanie oparte na wspólnym modelu bazowym dającym początek dostrojonym rozwiązaniom różniącym się jedynie dostosowanymi parametrami, a współdzielącymi ogólną architekturę.

Model ten został oparty o sieć transformera. Różni się od oryginalnego, według autorów, praktycznie wyłącznie hiperparametrami. BERT jest przystosowany do rozumienia danych wejściowych w formie krotki: pytanie, odpowiedź. Osiąga to dzięki

specjalnemu tokenowi [SEP], który rozdziela zdania oraz zakodowanej reprezentacji, sygnalizującej przynależność do konkretnej części wejścia, która jest dołączona do każdego tokenu. Dodatkowy token klasyfikacji [CLS] jest dołączany do każdej sekwencji wejściowej. Ponieważ transformer potrzebuje jako wejścia sekwencji tokenów, a nie słów, to użyte zostały osadzenia podsłów *WordPiece* (Devlin et al., 2018).

Procedura treningowa oparta została na dwóch zadaniach. Pierwsze, odpowiada maskowanemu zadaniu modelowania językowego. Dla 15% tokenów zakrytych przez specjalny token [MASK] zadaniem jest przewidzenie nieznanych wyrazów. Funkcją nagrody jest entropia krzyżowa. Drugim z wykorzystywanych zadań jest predykcja następującego zdania. Następujące zdanie jest kolejnym zdaniem pochodzącym z tekstu przy prawdopodobieństwie równym 50%, natomiast w 50% przypadków zdanie następujące jest losowym zdaniem pochodzącym ze wszystkich dostępnych zdań z korpusu. Problem polega na przewidzeniu czy wybrane dwa zdania następują po sobie. Do treningu wstępnego wykorzystano korpus tekstowy *BooksCorpus*, jak również angielską Wikipedię.

Model oryginalnie dostrojono do aż 11 różnych zadań z zakresu przetwarzania języka naturalnego. Dostrajanie jest wykonywane w procedurze *end-to-end* i jest znacząco mniej kosztowne obliczeniowo od procedury wstępnego treningu. BERT osiągnął również wynik nieosiągalny przedtem na zbiorach danych dotyczących rozumienia tekstu naturalnego: GLUE, SQuAD v1.1, SQuAD v.2.0 i SWAG (Devlin et al., 2018).

2.2.3. SBERT

Model BERT pozwolił na osiągnięcie nowych rekordów w zadaniu semantycznego porównywania tekstu (STS), jednak nie jest przystosowany do wykonywania tego zadania. Głównym problemem uwydatnionym w pracy na temat *Sentence-BERT'a* jest uzyskanie poprawy wydajności obliczeniowej względem BERT'a o kilka rzędów wielkości. Jeśli chcemy badać podobieństwo między częściami tekstu za pomocą podstawowej sieci transformera takiej jak BERT, to musimy podać jej obie części tekstu, a następnie wytrenować klasyfikator na pewnej próbce z tokenów wyjściowych. Problemem jest tu sytuacja gdy, na przykład, chcemy badać podobieństwo między tysiącami zdań. Oznaczałoby to, że musielibyśmy obliczyć wynik podobieństwa dla każdej pary zdań. Kwadratowa złożoność obliczeniowa jest niepożądana i możemy jej uniknąć dzięki modelowi reprezentacyjnemu, takiemu jak SBERT. Drugim

podniesionym defektem BERT'a jest fakt, iż nie jest jasnym w jaki sposób należy próbkować jego tokeny wynikowe, żeby otrzymać najlepszą reprezentację. Autorzy próbują kilka klasycznych funkcji: maksimum z tokenów, średnia z tokenów i reprezentacja na specjalnym tokenie [CLS], spośród których metoda uśredniania okazuje się najskuteczniejsza (Reimers & Gurevych, 2019).

Sentence-BERT albo krótko SBERT korzysta z syjamskiej (lub potrójnej) architektury sieci neuronowych gdzie blokiem budulcowym każdego z ramion sieci jest wstępnie wytrenowana sieć BERT. Dwa porównywane zdania są przetwarzane przez te sieci, a następnie przy pomocy wybranej strategii próbkowania reprezentacji z niezdekodowanych tokenów wyjściowych otrzymujemy dwie reprezentacje dla każdego ze zdań wejściowych. Wyjście jest dodatkowo rozszerzane o bezwzględną różnicę między reprezentacjami, co poprawia osiągane wyniki. Sieci są dotrenowywane na zadaniu klasyfikacji lub pewnego rodzaju regresji, w zależności od zbioru treningowego. W badaniu SBERT okazał się najlepszą prezentowaną metodą na kilku z zadań STS. Chociaż autorzy zaznaczają, iż SBERT nie jest przeznaczony do tzw. *transfer learningu* to okazuje się, że metoda osiąga najlepsze porównawcze wyniki na większości z wybranych przez nich 7 zadaniach SentEval, dla których sieć nie była trenowana (Reimers & Gurevych, 2019).

2.2.4. T5

Sukcesy ostatnich kilku lat na polu budowania dużych modeli języka, które subiektywnie wydają się być w stanie przejść test Turinga, wynikają w dużej mierze z prostych praw skalowania. Jedną z prób zbadania zależności pomiędzy rodzajem zadania treningowego, zbiorem korpusu tekstów, na których trenuje model, wielkością i budową modelu a osiągniętymi rezultatami na końcowych zadaniach językowych jest T5 czyli 'Text-to-Text Transfer Transformer'. Autorzy tego modelu wykonali ogromną liczbę eksperymentów próbując określić, które z parametrów sprawdzają się najlepiej dla trenowania sieci tego typu. Porównywane wyniki opierają się o pomiar efektów działania na zbiorze zadań takich jak: GLUE i SuperGLUE, SQuAD, abstrakcyjne streszczanie CNN/Daily Mail oraz zbiorach translacji WMT angielsko-niemieckich, ang.-francuskich i ang.-rumuńskich (Raffel et al., 2020). Zgodnie z nazwą zaproponowanego modelu, wszystkie zadania są wprowadzane jako tekst przy użyciu odpowiednich przedrostków, na przykład dla tłumaczenia 'translate English to German: That is good.'. Takie podejście upraszcza użycie jednolitego modelu dla wszystkich zadań i parametrów. Finalnie wybrana architektura, która została wytrenowana na największej liczbie tokenów jest rodzajem Transformera składającego

się z enkodera, jak i dekodera. Wytrenowano go wstępnie na zbiorze nazwanym C4, który jest wyczyszczonym zbiorem danych pochodzących z inicjatywy Common Crawl zbierającej ogromne ilości informacji ze stron internetowych, które są upubliczniane każdego miesiąca. Model używa w pewnym stopniu ulepszonego, względem BERT'a, zadania uczenia nienadzorowanego do treningu wstępnego. Polega ono na losowym maskowaniu pewnej części tokenów, które model uczy się przewidywać. Maskowanie jest przeprowadzane tak aby liczba przekrytych słów obok siebie wynosiła średnio trzy. Ulepszono też, cel przewidywań zastępując długie konsekutywne części wejścia specjalnymi tokenami pominięcia, co odciąża model z uzupełniania docelowego tekstu podanymi już wcześniej na wejściu tokenami. T5 jest dostrajany w specjalnym reżimie, w którym widzi najpierw miksturę wszystkich dostrajanych zadań wmieszanych w zbiór C4, a dopiero później jest do nich pojedynczo dostrajany, co nie ma jednak znaczącego wpływu na wynik końcowy względem klasycznego dostrajania (Raffel et al., 2020).

2.3. Zbiory danych

2.3.1. MS MARCO

Drugim najważniejszym wydarzeniem, oprócz pojawienia się nowego rodzaju sieci do przetwarzania języka, mającym wpływ na zwiększone, w ostatnich kilku latach, zainteresowanie badaniami nad wyszukiwaniem informacji był zbiór danych MS MARCO. Zawiera on 1,010,916 pytań pozyskanych z logów wyszukiwarki Bing. Każde z pytań zawiera odpowiedź napisaną przez człowieka. Co więcej, zbiór ten składa się również 8,841,823 paragrafów pochodzących z 3,563,535 dokumentów, które pozyskano dzięki algorytmom wyszukiwania wyszukiwarki. Dane te są niewątpliwie atrakcyjne dla badaczy opracowujących nowe techniki i modele z dziedziny rozumienia tekstu oraz wyszukiwania informacji (Bajaj et al., 2016).

Proces tworzenia tego zbioru polegał na zebraniu dużej puli zapytań wysłanych do wyszukiwarki. Zostały one poddane automatycznemu filtrowaniu, tak aby zawierały jedynie pytania. Następnie nowoczesny system wybrał z uzyskanych dokumentów internetowych odpowiednie paragrafy. Każde z zapytań zostało przeanalizowane przez człowieka, który na podstawie danych zawartych w dołączonych paragrafach stworzył odpowiedź oraz zaznaczył przydatne paragrafy. Część odpowiedzi została oznaczona do poprawy w procesie recenzji. Takie pytania otrzymały osobną odpowiedź stworzoną przez recenzenta. Do każdego z paragrafów dołączone są: URL, tytuł i tekstowa treść strony, z których pochodzi dany wycinek. Pytania zostały również

podzielone na kilka klas przy pomocy klasyfikatora wykorzystującego uczenie maszynowe (Bajaj et al., 2016).

MS MARCO (ang. Microsoft MAchine Reading COmprehension) stworzony został aby odpowiadać ogólnie rozumianym potrzebom mierzenia rozumienia tekstu czytanego przez maszyny. Autorzy proponują trzy różne zadania opracowane dla tego zbioru danych. Pierwsze polega na przewidzeniu, przy wykorzystaniu paragrafów, czy można sformułować odpowiedź na ich podstawie oraz na syntezie tejże odpowiedzi. Drugie z zadań ma na celu wygenerowanie poprawnie sformułowanej odpowiedzi, która może być zrozumiana w kontekście pytania i dołączonych paragrafów. Trzecie z zadań, najbliższe obecnym wymaganiom wyszukiwarek internetowych, polega na stworzeniu rankingu paragrafów dla zadanego pytania (Bajaj et al., 2016).

2.3.2. SQuAD 2.0

Nowa wersja znanego zbioru danych SQuAD 1.1 rozszerza swój zakres o pytania, na które nie ma odpowiedzi w podanym kontekście. Taka zmiana pozwoliła znacząco utrudnić zadanie stawianie przed systemem MRC. SQuAD składa się z pytań pozyskanych od zakontraktowanych pracowników społecznych, którzy zadawali pytania do artykułów znajdujących się na stronie Wikipedia, gdzie odpowiedzią jest część tekstu lub pytanie może nie posiadać odpowiedzi. SQuAD 2.0 łączy 100 tysięcy pytań ze zbioru SQuAD 1.1 z 50 tysiącami stworzonych w sposób antagonistyczny pytań, które wyglądają podobnie do reszty pytań, ale na które nie ma odpowiedzi w tekśce. Aby radzić sobie w tym zbiorze, model musi nie tylko umieć odpowiadać na pytania, ale również potrafić zdecydować czy posiada wystarczające informacje, żeby udzielić odpowiedzi.

2.4. Przechowywanie gęstych reprezentacji

2.4.1. FAISS

Faiss jest biblioteką służącą do wyszukiwania podobieństw i analizy skupień operującą na wektorach o gęstych reprezentacjach (Meta AI Research, 2023). Główną zaletą, stworzonego przez zespół Facebook AI (obecnie Meta AI) rozwiązania, są jej możliwości zrównoleglania obliczeń dzięki wykorzystaniu procesora graficznego (GPU). Rozwiązania wykorzystujące jedynie CPU wykonują porównania sekwencyjnie lub z małym poziomem współbieżności co powoduje nieprzystępność takiego podejścia dla wielowymiarowych zbiorów danych takich jak zdjęcia czy wideo, które mogą składać

się nawet z miliardów punktów danych (Johnson et al., 2019).

Gdy mamy do czynienia z wielowymiarowymi wektorami reprezentacji, porównywanie ich metodą siłową (ang. 'brute force') w celu znalezienia wektorów najbliżej położonych może stawać się niemożliwe. Zakładając, że wykonujemy wcześniej grupowanie wektorów przy pomocy metody k-centroidów moglibyśmy ograniczyć region wyszukiwania do punktów znajdujących się w pobliżu centroidu, od którego jest najbliżej do wektora zapytania. Ta metoda pozwoli na ograniczenie ilości wymaganych porównań.

Technika kwantyzacji zwana 'Product Quantization' (PQ) polega na rozbiciu jednego długiego wektora na wiele podwektorów. W każdej przestrzeni, którą tworzy *i-ta* część wektora wykonujemy algorytm k-centroidów, otrzymując w ten sposób pewną liczbę rozdzielnych podziałów przestrzeni. Reprezentacja każdego z podwektorów jest zastępowana *id* centroidu, do którego należy. W ten sposób po połączeniu numerów id dla i części otrzymujemy nowy wektor kodujący, który jest krótszy od oryginalnego. Jest on zwany kodem PQ. Dzięki tym kodom otrzymujemy logarytmiczne zmniejszenie reprezentacji względem długości podstawowych wektorów, co znacząco zmiejszy konieczną wielkość przechowywanych danych (P. Chang, 2022a).

Wyszukiwanie podobnych wektorów, czy to według najmniejszej odległości L2 czy też przy użyciu podobieństwa kosinusowego, składa się z kilku kroków w systemie IVFADC ('Inverted file system with asymmetric distance computation'). Generalna idea polega na przybliżaniu wektora za pomocą podwójnej kwantyzacji.

$$y \approx q(y) = q_1(y) + q_2(y - q_1(y)) \tag{2.2}$$

Pierwszy z nich (q_1) zwany zgrubnym kwantyzerem, drugi (q_2) jest kwantyzerem dokładnym przyjmującym jako wejście rezydualne różnice między pierwszą kwantyzacją a danymi wejściowymi.

Grupy kodów PQ dla wszystkich punktów są zapisane w odwróconym indeksie, którego kluczami są centroidy uzyskane poprzez algorytm k-centroidów, gdzie k będzie wynosić w tym przypadku około pierwiastek z ilości wszystkich punktów (Raffel et al., 2020). Jednocześnie te przypisania będą służyły za pierwszy, zgrubny poziom kwantyzacji, który będzie określał wartość każdego z punktów jako równą centroidowi, do którego jest przypisany. Drugi poziom kwantyzacji operuje na różnicy między wektorami a ich wartością obliczoną za pomocą pierwszego kwantyzera. Pierwszym

powodem dla którego używamy różnic między wektorem a wartością centroidu skupienia, do którego należy jest fakt, iż jest to niedokładność na której operujemy po użyciu pierwszego przybliżenia. Dodatkową przewagą takiego podejścia jest to, że gdy po grupowaniu skupienia leżą w różnych częściach przestrzeni, to po obliczeniu rezydualnej różnicy ich środki zostają nałożone na siebie a pozostałe wartości uzyskanych wektorów są równe oddaleniu od środków dla odpowiednich skupień (P. Chang, 2022b). Aby znaleźć odległości między wektorami rezydualnymi zapisanymi przy pomocy kodów PQ wykorzystywany jest dystans asymetryczny (ADC). Jest on obliczany jako suma z odległości L2 między wektorem a centroidem kwantyfikacji do którego należy. Ponieważ w kodach PQ każdy z wektorów jest podzielonych na wiele części to odległość będzie się składać z sumy odległości dla każdego z podziałów między wektorem zapytania a zkwantyzowanym wektorem zapisanym w bazie danych. Dopiero po obliczeniu tych wielkości wykonywane jest odpowiednie wyszukiwanie (P. Chang, 2022b) (P. Chang, 2022a).

Autorzy biblioteki Faiss pokazują w kilku eksperymentach na dużych zbiorach danych, że ich przybliżony sposób wyszukiwania najbliższych sąsiadów jest: znacząco szybszy niż odpowiadające rozwiązanie działające na procesorze centralnym (CPU), jest kilkukrotnie efektywniejsze niż wyniki wyszukiwania przeprowadzone innym systemem i w końcu, osiąga czułość większą niż w innym porównywanym badaniu (Raffel et al., 2020). Na podstawie tych wyników można stwierdzić, że biblioteka ta nadaje się do skutecznego wyszukiwania w zbiorach osiągających nawet miliardy wektorów.

2.5. Przyszłość wykorzystania dużych modeli językowych w wyszukiwarkach

Największy z graczy na rynku usług wyszukiwania rozpoczął wprowadzanie rozwiązań opartych o uczenie maszynowe już kilka lat temu. W 2015 Google rozpoczął wykorzystywanie algorytmu RankBrain. Mimo iż działanie tego systemu rankingowego nie jest oficjalnie znane, to firma przyznała, że służy on do łączenia wyrazów z ich znaczeniami w sposób pozwalający zwracać lepsze wyniki. Inny wprowadzony system nazywany 'neural matching' ma dopasowywać zapytania do stron, biorąc pod uwagę całą zawartość tekstu. W 2019 roku firma wprowadziła BERT'a jako część swojego systemu w dwóch kluczowych miejscach: wyszukiwaniu (ang. 'retrieval') i szeregowaniu (ang. 'ranking'). W dalszej części pracy będzie przeanalizowane jak BERT może być użyty do osiągania tych celów. Innym z rozwiązań wprowadzonych do usługi na

smartfony Google Lens, jest MUM (' Multitask Unified Model'). Jest to model potrafiący rozumieć i generować język w 75 językach, a dodatkowo wykorzystujący inne modalności takie jak rozumienie obrazu (Nayak, 2022). Aplikacja pozwala m. in. na rozpoznawanie zdjęć, wyszukiwanie wizualne, tłumaczenie sfotografowanego tekstu.

Microsoft jest wyraźnie zainteresowany podjęciem konkurencji w obszarze wyszukiwania, a jego potencjał wzmacniać mają technologie oparte o AI. Jednym z przykładów tego wysiłku jest stworzenie zbioru MS MARCO. Jednak ostatnimi czasy głośno o wyszukiwarce Bing jest głównie z jednego powodu. Dzięki podjęciu współpracy z OpenAI, firma wypuściła nową wersję swojej wyszukiwarki wzmocnionej o kolejne wcielenie GPT. Sam Microsoft nazywa te ulepszenia kopilotem dla wyszukiwania. I rzeczywiście, oprócz samych linków po wpisaniu zapytania pojawiają się również okienka z wyciągiem najważniejszych odnalezionych informacji. Wprowadzono też opcję czatu znaną z rozwiązań takich jak ChatGPT. Możemy w niej naturalnie, bo przy pomocy języka naturalnego, komunikować się z modelem. Przy jego wykorzystaniu możemy wygenerować różnego rodzaju tekst, jak i zadać pytanie a następnie otrzymać odpowiedź wraz z odsyłaczami do źródeł danych z sieci, które uzasadniają określone fragmenty (Mehdi, 2023). Rozwiązanie to wydaje się oferować ciekawą funkcjonalność, która przynosi powiew świeżości do scementowanego wcześniej świata wyszukiwarek.

Google nadal posiada wiele przewag konkurencyjnych wraz z byciem, wbrew narracji medialnej, w forpoczcie tworzenia rozwiązań uczenia maszynowego dla wyszukiwania. Jednak bezapelacyjnie szybkość wprowadzenia modeli OpenAI do Bing'u może budzić obawy o to co wydawało się niepodważalnym monopolem. Stąd na początku roku Sundar Pichai ogłosił zwiększone zaangażowanie Alphabet'u w rozwój AI dla wyszukiwania, a szczególnie w ogromne modele językowe. Firma zaprezentowała usługę konkurencyjną do tej oferowanej przez Microsoft. Bard, bo tak nazywa się to rozwiązanie, jest serwisem opartym o model LaMDA. Na razie pozostaje on jednak w fazie testów (Pichai, 2023).

2.5.1. LaMDA

W następnych rozdziałach podejmiemy się opisania i uruchomienia nowoczesnego systemu wyszukiwania. Należy sobie jednak zadać pytanie czy przyszłość dostępu do informacji opiera się na skalowaniu i ulepszaniu obecnych systemów, czy też będzie wymagać fundamentalnej zmiany podejścia. Jeśli popularność jest jakkolwiek dobrym predyktorem przyszłego sukcesu, to należy się spodziewać, że ChatGPT ma przed sobą świetlaną przyszłość. Szacuje się, że aplikacja osiągnęła 100 milionów użytkowników

w styczniu 2023. Zajęło jej to tylko dwa miesiące od premiery, co ustanowiło nowy rekord szybkości adopcji technologii w historii Internetu (Hu, 2023).

Ponieważ autorzy ChatuGPT nie udostępnili zweryfikowanej pracy naukowej, która tłumaczyłaby działanie tego rozwiązania, przyjrzymy się pracom prezentującym podobne systemy, których przeznaczeniem jest dialog w języku naturalnym. Jednym z nich jest LaMDA, czyli model językowy przeznaczony do zastosowań dialogowych ('Language Models for Dialog Applications'). Model opisany na początku 2022 roku został oparty o architekturę sieci transformera z samym dekoderem, o wielkości do 137 miliardów parametrów i został wstępnie wytrenowany na 1.56T słów (Thoppilan et al., 2022). LaMDA jest trenowana początkowo jako model językowy ogólnego przeznaczenia na danych składających się z dokumentów tekstowych i dialogów lub ich części. Następnie badacze dotrenowywują model w celu zwiększenia trzech interesujących ich parametrów: jakości, bezpieczeństwa i zaczepienia w faktach. Do tego celu zostają pozyskane interakcje z pracownikami kontraktowymi, którzy przeprowadzają interakcję z systemem oraz oceniają jego odpowiedzi. Następnie, model uczy się przewidywać cechy wygenerowanego tekstu, co umożliwia filtrowanie niepożądanych wyników. Dodatkowo dane dialogowe, które zostały wykorzystane do wczesnego trenowania, zostają ocenione w ten sposób. Następnie model jest dostrajany na tym zbiorze danych (Thoppilan et al., 2022). Aby poprawić zgodność odpowiedzi z rzeczywistością model jest rozszerzony o możliwość używania narzędzi: kalkulatora, tłumacza i narzędzia wyszukiwania informacji. Zwracają one na podstawie zadanego zapytania listę pasujących odpowiedzi. Na bazie zbioru danych zawierającego interakcje z człowiekiem wersja modelu nazwana LaMDA-Reasearch uczy się wyszukiwać odpowiednie informacje. Proces generowania odpowiedzi przebiega więc następująco: podstawowy model LaMDA odpowiada na zadane zapytanie, później LaMDA-Reasearch pyta się o fakty, otrzymuje odpowiedź z narzędzia wyszukiwania, na końcu model generuje ostateczną odpowiedź (Thoppilan et al., 2022). Podsumowując, efekt wykorzystania dodatkowych zadań treningowych ma widoczny wpływ na mierzone metryki takie jak: sensowność, specyficzność, bezpieczeństwo, zakotwiczenie, ciekawość, informatywność.

2.5.2. WebGPT

WebGPT jest systemem otwartego odpowiadania na pytania w długiej formie. Model jest dostrojoną wersją GPT-3, która została wyposażona przy pomocy API o możliwość korzystania z Microsoft Bing. Jest to rozwinięcie pomysłu, o

którym wspomnimy w następnym rozdziale, polegającego na wykorzystywaniu modeli językowych do syntezowania odpowiedzi na podstawie informacji, do których mają one dostęp. Środowisko operowania tego systemu, to odpowiadanie na pytania, gdzie model może wykonać kilka różnych akcji: wyszukać daną frazę, wybrać link, znaleźć dany tekst na stronie, zacytować informację, przewijać witryny internetowe oraz udzielać odpowiedzi. Akcje te model może wykonywać za pomocą generowania odpowiednich szablonów tekstowych. Zbiorem danych treningowych są pytania i odpowiedzi z podstrony ELI5 będącej częścią witryny Reddit. Na podstawie tych danych powstało kilka zadań treningowych. Anotacja ludzka została wykorzystana aby porównywać preferencje między odpowiedziami udzielonymi przez człowieka a wygenerowanymi przez model. Na zadania ćwiczebne składa się: klonowanie behawioralne - czyli wykorzystanie zbioru do uczenia nadzorowanego, uczenie modelu nagrody - model jest trenowany, żeby przewidywać która z odpowiedzi jest preferowana przez człowieka, uczenie ze wzmocnieniem przy użyciu algorytmu PPO. W końcu wykorzystywane jest próbkowanie odrzuceniowe. Wybieranych jest w nim n próbek spośród których wybierane są te preferowane przez model nagrody (Nakano et al., 2021). Dzięki temu, że system trenowany jest przy użyciu uczenie ze wzmocnieniem, możliwym staje się uzyskanie preferencji oceniających ponad bazowe odpowiedzi udzielane przez człowieka. Uzyskuje on porównywalne wyniki do ludzkiej demonstracji przebijając ją w 56% przypadków jeśli chodzi o ogólną użyteczność. Na zbiorze ELI5 69% pytanych preferuje odpowiedzi WebGPT w porównaniu do najwyżej ocenianego komentarza z danego wątku (Nakano et al., 2021).

Rozdział 3

Porównanie metod wyszukiwania informacji

Od pewnego czasu sieci neuronowe są nieodzowną częścią skutecznej architektury wyszukiwania informacji. Może najbardziej widocznym z miejsc użycia jest wykorzystanie ich do tworzenia finalnego rankingu stron, problem zwany w angielskiej literaturze LTR (ang. 'Learning To Rank'). Rozwiązania, korzystające zazwyczaj z sieci transformera, są najlepszymi rozwiązaniami obecnego stanu techniki w tym zakresie. Drugim z bardziej oczywistych zastosowań uczenia głębokiego jest uczenie reprezentacji dokumentów i zapytań. (Tonellotto, 2022) Poza tymi, bardziej klasycznymi tematami wyszukiwania informacji, można by rozszerzyć badaną dziedzinę potencjalnych zastosowań uczenia maszynowego o dodatkową problematykę. Podstawowym ulepszeniem, które wydaje się obecnie zyskiwać na popularności, jest wykorzystanie modeli językowych do syntezowania ostatecznej odpowiedzi. Użytkownik otrzymuje wtedy oprócz listy linków, pewien rodzaj zredagowanego tekstu będącego odpowiedzią na zadane pytanie. Co więcej, możliwym wydaje się jest sformułowanie problemu wyszukiwania tak, aby określić go jako rodzaj problemu uczenia ze wzmocnieniem gdzie możliwym byłoby wykorzystanie technik tego działu wiedzy, które odniosły już sukcesy w grze Go - AlphaGo, szachach - AlphaZero czy grze komputerowej StarCraft - AlphaStar. Ten kierunek badań jest na razie ograniczony, niemniej jednak został rozpoczęty. Wydaje się to być interesujące pole dla przyszłych prac naukowych.

3.1. Modele wyszukiwania informacji

3.1.1. Dostrajanie BERT'a

Jednym z pierwszych udanych zastosowań BERT'a do wyszukiwania informacji był system zaproponowany przez Nougeirę i Cho. Dostroili oni model na zbiorze anotowanych paragrafów MS MARCO, tak aby po uprzednim zwróceniu rezultatów przez BM25 oceniał czy zadany paragraf pasuje do zapytania, czy też nie. Pozwoliło im to osiągnąć 27% relatywną poprawę względem najlepszego wtedy rezultatu, jeśli chodzi o miarę MRR@10 dla rerankingu (R. Nogueira & Cho, 2019).

3.1.2. doc2query i docTTTTTquery

Znaną techniką ulepszania wyników wyszukiwania jest rozszerzanie zapytania lub dokumentu o pewne słowa kluczowe, które nie pojawiają się bezpośrednio w tekście, a dobrze opisują zawartość danego zapytania lub też dokumentu. Autorzy rozwiązania *Doc2query* proponują rozszerzać dokumenty o pytania przy użyciu modelu seq-to-seq wytrenowanego na zbiorze składającym się z dokumentów i pasujących do ich zawartości pytań. Wytrenowany w ten sposób transformer jest używany do predykcji 10 pytań przy użyciu metody próbkowania modeli językowych nazywaną próbkowaniem k-najlepszych (ang. 'top k-sampling'), która wybiera k najbardziej prawdopodobne ciągi tokenów. Wygenerowane w ten sposób pytania są dołączane bezpośrednio do dokumentów, a następnie indeksowane i wyszukiwane przy pomocy algorytmu BM25. Dodatkowym krokiem, używanym opcjonalnie przez autorów, jest ponowne obliczenie rankingu przy pomocy rozwiązania opartego o BERT. To proste rozwiązanie osiągnęło w czasie publikacji wynik zbliżony do rekordowych wyników na tablicy wyników dla zbioru MS MARCO (R. Nogueira et al., 2019).

DocTTTTTquery jest bezpośrednim ulepszeniem doc2query, które wykorzystuje model T5 do generowania pytań. Parametry i sposób treningu są bardzo podobne do wcześniejszego podejścia dlatego pomijamy ich opis (L. Nogueira, 2019). Prosta modyfikacja pozwala znacząco poprawić wyniki działania całego systemu. Jednocześnie praca ta empirycznie potwierdza sensowność użycia T5 do generowania pytań odnoszących się do bazowego tekstu.

3.1.3. ColBERT

Chociaż model BERT może być dostrojony do zadania wyszukiwania informacji i osiągać wyniki wykraczające poza klasyczne metody, to często wiąże się to z nadmiernym dodatkowym kosztem obliczeniowym. Nawet 100-1000 większym niż poprzednie metody, co jest związane z koniecznością uruchomienia pełnego modelu dla każdej pary zapytania i dokumentu, której zgodność sprawdzamy. ColBERT, wykorzystujący model BERT u swojej podstawy, buduje dwie oddzielne reprezentacje, które łączy w kroku późnej interakcji. Dzięki takiej konstrukcji obliczenie reprezentacji dokumentów może odbywać się jako krok wstępnego przetwarzania, co znacząco przyspiesza działanie podczas szukania. Aby obliczyć dopasowanie, autorzy ColBERT'a proponują operację sumy z maksimum operacji podobieństwa będącej kosinusem między wektorami reprezentacji (maxSim). Kiedy dokumenty zostaną zindeksowane poprzez zapisanie wartości ich reprezentacji na dysku, przydatnym dla dużych zbiorów danych może być rozdzielenie procesu wyszukiwania na dwa etapy. W pierwszym, przy pomocy biblioteki do obliczania podobieństwa i klastrów FAISS wszystkie reprezentacje są rozdzielane na 1000 klastrów, z których zawartość najbliższych 10 jest przesyłana do kolejnego etapu. W drugiej fazie przetwarzania, wszystkie otrzymane elementy są szeregowane poprzez bezpośrednie obliczenie dopasowania nazywanego maxSim. ColBERT oferuje znaczące przyspieszenie na zadaniu ponownego rankingu wyników względem modelu opartego o sam BERT, jednocześnie zachowując konkurencyjną jakość działania w porównaniu do starszych modeli BM25, KNRM czy Duet (Khattab & Zaharia, 2020).

3.1.4. DeepImpact

DeepImpact jest modelem mającym zastosowanie jako pierwszy stopień wyszukiwania. Rozwiązanie wykorzystuje rozszerzanie dokumentu przez DocT5Query. System ten działa podobnie do Doc2query, ale oparty jest o model językowy T5. Dokument, który został rozszerzony poprzez dodanie dodatkowych wyrażeń po tokenie separującym, jest przetwarzany przez model językowy BERT, aby otrzymać osadzone wektory. Tokeny pojawiające się ponownie są maskowane, a następnie każde pierwsze pojawienie się tokenu jest przetwarzane przez sieć MLP z dwoma warstwami. W wyniku otrzymujemy pojedynczą wartość dla każdego z tokenów reprezentującą jego wpływ na ważność dokumentu. Te wartości można zsumować, aby otrzymać końcowe dopasowanie. Dla danego zapytania, jego dopasowanie jest modelowane jako suma wartości wpływu wszystkich słów, które pojawiają się w zapytaniu, jak również w dokumencie. DeepImpact jest wytrenowany na paragrafowym zbiorze MS MARCO,

gdzie danymi treningowymi jest trójka zapytanie, pasujący i niedopasowany paragraf. Nauczone w ten sposób wartości wpływu słów mogą być wykorzystane przy konstrukcji indeksu. Autorzy porównują wynik swojego modelu do DocT5Query uzyskując kilkunastoprocentową przewagę, a dodatkowo przy zastosowaniu jako pierwszy stopień wyszukiwania wraz z ColBERT'em osiągają porównywalne wyniki efektywności przy około 5-krotnym przyspieszeniu działania (Mallia et al., 2021).

3.1.5. PROP

Chociaż rozwiązania oparte o BERT są mocnym modelem porównawczym, to pierwotne zadania treningowe dla tego modelu: wypełnianie luk oraz przewidywanie następnego pasującego zdania, są bardziej przydatne przy tworzeniu rozwiązań dotyczących rozumowania w języku naturalnym, niż odpowiadające zadaniu wyszukiwania informacji. Ma et al., 2021 proponują PROP jako metodę wstępnego treningu sieci Transformer w oparciu o zadanie przewidywania prawdopodobieństwa zapytania. Użyto dwóch zadań. W pierwszym, dla dokumentu w zbiorze uczącym, obliczany jest klasyczny probabilistyczny model językowy. Na jego podstawie generowane są słowa zapytania poprzez modelowanie prawdopodobieństwa warunkowego P(Q|D) wykorzystując, chociażby model unigramowy. W zadaniu treningowym generowane są dwa takie zapytania wraz z odpowiadającymi im prawdopodobieństwami, a model jest uczony przewidywać na tokenie "[CLS]", przetworzonym kolejno przez dodany perceptron wielowarstwowego, prawdopodobieństwo każdego z zapytań. Ponieważ PROP jest trenowany od zera, to dodatkowym zadaniem jest maskowane modelowanie języka, gdzie model próbuje przewidzieć tokeny zakryte poprzez specjalny token maskujący, podobnie jak w BERT. Model PROP jest wytrenowany na danych pochodzących z angielskiej Wikipedii oraz na MS MARCO, a także testowany, w dosyć skomplikowany sposób, na aż pięciu różnych zbiorach danych, w porównaniu z kilkoma różnymi metodami wyszukiwania informacji z grupy porównawczej. Model ten osiągnął bardzo silne wyniki, pobijając SOTA w czterech z pięciu zadań. Może on być dostrojony do specyficznego zadania z dziedziny IR i sprawdzać się w tym kontekście lepiej od innych modeli językowych (Ma et al., 2021). W 2021 roku PROP 400 tysięcy iteracji w wersji ensemble wraz z doc2query osiągnął pierwsze miejsce w rankingu MS MARCO (Xinyu Ma, 2021).

3.1.6. Condenser i coCondenser

Wykorzystanie pretrenowanych modeli językowych, jakkolwiek popularne w zastosowaniach wyszukiwania informacji, wiąże się z jeszcze inną niedogodnością.

Modele takie jak BERT, czy T5 są modelami przetwarzania tekstu, zwracającymi rozproszoną reprezentację tokenów, którą można interpretować jako tekst. Model BERT posiada co prawda token klasyfikacji "[CLS]", dla którego wielu autorów zmieniało przeznaczenie na token reprezentacji, lecz nie jest jasnym czy taka formulacja jest lepsza niż, na przykład średnia ze wszystkich tokenów, a ponadto takie zastosowanie nie było zamysłem architektury. Na podstawie analizy aktywacji mechanizmu uwagi (ang. 'attention') autorzy zauważają, że token klasyfikacji nie jest aktywnie połączony z innymi tokenami w środkowych warstwach modeli językowych, a uaktywnia się dopiero w ostatnich warstwach. Według autorów, nie jest to idealna sytuacja, gdy celem jest utworzenie bienkodera (ang. 'bi-encoder'), który będzie zwracał całkowitą informację ze wszystkich warstw jak reprezentację do porównania. Jako rozwiązanie zaproponowany jest model Condenser zawierający dodatkowe krótkie połączenie bezpośrednio z wczesnych warstw, które połączone z późną reprezentacją jest przetwarzane przez głowę (ang. 'head'). Nagrodą jest suma funkcji cross-entropy na zadaniu maskowanego modelowania językowego dla głowy Condensera i ostatniej warstwy modelu, co jest motywowane późniejszym odrzuceniem głowy po wczesnym treningu. Model wytrenowano na podobnych danych, a także w podobnym rozwiązaniu jak BERT. Dla zadania wyszukiwania informacji model jest dotrenowywany na zbiorze MS MARCO z nagrodą kontrastową. Architektura daje dobre wyniki, znacząco przebijając zwykły model BERT. Ta praca naukowa pokazuje, że bazowy model BERT będący w szerokim zastosowaniu w dziedzinie wyszukiwania informacji, nie posiada idealnej struktury do generowania reprezentacji, a w szczególności do wykorzystania jako bi enkoder (Gao & Callan, 2021a). Używając Condensera pozwalającego na efektywną reprezentację tekstu w tokenie klasyfikacyjnym, Gao i Callan rozbudowali swoje rozwiązanie o coCondenser. System jest trenowany na nagrodzie mającej być odzwierciedleniem semantycznej relacji odległości między dokumentami. Dokładniej nagroda jest kontrastowym błędem, obliczanym podobnie jak to opisano w poprzednich paragrafach, wynikającą z odległości między reprezentacjami wycinków z pasujących dokumentów i tych z grupy porównawczej. Dodatkowo do nagrody dodano błąd z maskowanego modelowania języka i obliczono średnią z obu. Trening inicjowany jest wagami z modelu BERT base i trenowany jak to zostało opisane w publikacji dot. Condensera. Następnie w fazie drugiej treningu, powtarzany jest trening wstępny na MS MARCO i Wikipedii, tym razem z nagrodą dla coCondensera. Użyto optymalizatora AdamW. Po wstępnym treningu odrzuca się głowę pozostawiając sam enkoder z architekturą identyczną do BERT'a. Model jest dostrajany na MS MARCO (Gao & Callan, 2021b). Wersja tego algorytmu osiągnął pierwsze miejsce w rankingu MS MARCO dla rankingu dokumentów.

3.1.7. HLATR

HLATR jest rozwiązaniem służącym jako trzeci stopień wyszukiwania. Nowoczesne systemy wyszukiwania są zbudowane jako dwustopniowe architektry. W pierwszym stopniu opartym na budowaniu oddzielnej reprezentacji dla dokumentu i zapytania a później ich połączeniu znajduje się dokumenty kandydujące. Takimi rozwiązaniami są klasyczny BM25, ale również w modelach dualnych kodujących oddzielnie dokument i zapytanie, przykładowo może to być ColBERT. Drugi stopień jest często interakcyjny, a więc oblicza dopasowanie dla każdej pełnej pary dokumentu i zapytania. Jako drugi stopień autorzy wykorzystują coCondenser i ANCE. Motywacją do stworzenia trzeciego stopnia jest obserwacja pokazująca, że kombinacja wyników z pierwszego i drugiego stopnia może dać lepszy rezultat niż używanie bezpośrednio wyników końcowych ze stopnia drugiego. Ponadto autorzy proponują użycie nagrody uczenia opartej o uczenie kontrastowe, gdzie nagroda jest uwarunkowana jako ujemna wartość logarytmu z ułamka, gdzie licznik jest dopasowaniem dla dokumentu pasującego, natomiast mianownik jest sumą dopasowań dla dokumentów z grupy porównawczej. Wejściem do HLATR jest reprezentacja wynikowa otrzymana dla każdej pary dokument, zapytanie, połączona z kodowaniem pozycyjnym według otrzymanego rankingu. Modelem wyboru jest Transformer, ale sieć osiąga gorsze, choć porównywalne wyniki przy użyciu innych modeli. Jako wyjście otrzymujemy nowy ranking. Okazuje się, że po uprzednim wytrenowaniu otrzymujemy wyniki lepszej jakości niż w dwustopniowym układzie. Przesądzające o użyteczności HLATR jest jego złożoność czasowa, osiągnięta dzięki małemu modelowi, wynosząca 2ms dla 1000 zapytań w porównaniu z kilkuset milisekundowymi kosztami uruchomienia BERT'a dla identycznej liczby zapytań (Zhang et al., 2022).

3.2. Modele QA - odpowiadania na pytania

Dziedziną, która może przyczynić się do zmiany sposobu korzystania z informacji dostępnych w internecie jest MRC ('Machine Reading Comprehension'), czyli rozwiązania pozwalające na rozumienie, a także wykonywanie zadań, na podstawie zdobytych informacji, przez maszyny. Modele językowe są przeznaczone przede wszystkim do generowania tekstu podobnego do tego, który stworzyłby człowiek. Mimo, iż w procesie uczenia na gigantycznej części tekstów z internetu są w stanie zdobyć pewną generalizację swoich umiejętności, taką jak np. dodawanie liczb do pewnego ograniczonego rozmiaru, to równie widoczne są ich niedoskonałości. Jedną z najczęściej opisywanych są tak zwane 'halucynacje' czyli sytuacje gdy model językowy wymyśla lub zgaduje pewne informacje, które nie były dla niego dostępne

ani podczas treningu, ani w fazie ewaluacji. Zostało udowodnione, że rozwiązania te są w stanie zapamiętać część faktów o świecie w wagach sieci neuronowej, jednak nie posiadają zazwyczaj żadnej zewnętrznej bazy wiedzy lub pamięci długotrwałego zapisu z której mogłyby nauczyć się korzystać. Można więc przypuszczać, że problemy 'halucynacji' mogłyby zostać rozwiązane gdyby model został dostosowany do używania pewnego rodzaju zewnętrznej pamięci posiadającej poprawne informacje o świecie.

Bardziej generalnym podejściem do odpowiadania na postawione pytania jest OpenQA, czyli otwartodziedzinowe odpowiadanie bez posiadanego kontekstu. Przy rozwiązaniach typu MRC wymagany jest generalnie, pewien tekst określany kontekstem, na podstawie którego system generuje odpowiedź. Natomiast w zadaniu typu QA, generator nie ma dostępu do informacji lub korzysta z ogólnie dostępnych danych znajdujących się np. w internecie. Wyszukiwanie informacji dostarcza gotowego rozwiązania, które może generować kontekst dla generatora odpowiedzi. Przy użyciu odpowiednich procedur treningowych możliwym staje się wykorzystanie bazy dostępnych dokumentów do poprawy efektywności działania systemu odpowiadania na pytania. Typową obecnie architekturą dla zadań OpenQA jest połączenie 'Wyszukiwarka-Czytnik' (ang. 'Retriver-Reader'), gdzie pierwsza część wyszukuje dokumenty, a druga generuje odpowiedź. Architektura może być złożeniem dwóch oddzielnych systemów lub być trenowana wspólnie w reżimie *end-to-end*. Pierwszy moduł jest rozwiązaniem wyszukiwania informacji, drugi może być neuronalnym systemem MRC (Zhu et al., 2021). W porównaniu do zwykłego zadania rozumienia tekstu, mamy w tym wypadku do czynienia z wieloma odnalezionymi paragrafami zamiast tylko jednego, co niewątpliwie komplikuje problem. Możliwa jest budowa czytnika jako rozwiązania odnajdującego w dokumentach odpowiedni cytat odpowiadający na postawione pytanie lub rozwiązanie syntezujące odpowiedź. Tworzenie odpowiedzi będzie zazwyczaj oparte o dostarczone paragrafy/dokumenty, chociaż może działać również bez kontekstu. Mielibyśmy wtedy do czynienia z architekturą samego czytnika, tak jak ma to miejsce w przypadku chociażby GPT-3 (Zhu et al., 2021). W tej sekcji skupiamy się jednak na architekturze dwuwarstwowej.

3.2.1. Wykorzystanie wyszukiwania paragrafów wraz z modelami generatywnymi do zadań OpenQA

Metody wyszukiwania informacji i modele generatywne wydają się być świetnie dopasowanym połączeniem, które pozwala na zniwelowanie wad i uwydatnienie mocnych stron obu podejść. Z jednej strony, wyszukiwarki posiadają nieporównywalne

zdolności przywoływania informacji w danym kontekście, natomiast sposób ich przedstawienia jako listy linków, pozostawia wiele do życzenia. Z drugiej perspektywy, językowe modele generatywne posiadają wysokie zdolności tworzenia i przedstawiania informacji w zachęcający dla człowieka sposób, lecz brakuje im często wiedzy o świecie co skutkuje generowaniem treści, które w najlepszym razie można interpretować jako wytwory literackie.

Prace, takie jak ta zatytułowana 'Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering' (Izacard & Grave, 2020), podejmują próbę połączenia tych podejść w celu osiągnięcia przewyższających inne systemy rezultatów w dziedzinie OpenQA. Paragrafy są zwracane przy użyciu BM25 lub gęstego wyszukiwania neuronalnego (DPR) w zależności od wykorzystywanego zbioru danych. Generatorem odpowiedzi jest przystosowany model T5, który został dotrenowany przy użyciu zbiorów odpowiadania na pytania: NaturalQuestions, TriviaQA i SQuAD. Wejściem dla generatora jest konkatenacja uzyskanych paragrafów w fazie wyszukiwania. Rozwiązanie ma przewagę nad podejściami ekstrakcyjnymi i czysto generatywnymi na zbiorze NQ i TriviaQA. Dodatkowo, zostało zauważone, że efektywność systemu skaluje się z ilością dostarczonych do generatora danych, nawet do 100 paragrafów, co zdaje się potwierdzać ich przydatność w generowaniu ostatecznej odpowiedzi (Izacard & Grave, 2020).

3.2.2. REALM

Wcześniejszym podejściem do ekstraktywnego generowania odpowiedzi przy pomocy wspólnie uczonego systemu wyszukiwania i odpowiadania był REALM. Rozważany problem został rozbity na dwa kroki. Zaczynając od kroku wyszukiwania, system zwraca dokumenty związane z zapytaniem. Następnie bazując na dokumentach i zapytaniu generowana jest odpowiedź. Określenie pasujących dokumentów polega na zwróceniu rozkładu prawdopodobieństwa nad dokumentami. Zrealizowane zostało to przez obliczenie reprezentacji przy pomocy kodowania przez sieć transformera, tak że otrzymujemy wektor. Między wektorami dokumentów i zapytania obliczany jest produkt wewnętrzny interpretowany jako dopasowanie. Następnie wyniki dla wszystkich dokumentów są normalizowane do rozkładu prawdopodobieństwa przy wykorzystaniu funkcji *softmax*. Dla fazy zwracania odpowiedzi przekazywana jest pewna liczba najbardziej dopasowanych dokumentów wraz z zapytaniem. Model jest wstępnie trenowany używając nagrody maskowanego modelu językowego (MLM). Zadanie dostrajania polega na przewidzeniu początku i końca odpowiedzi znajdującej się w przekazanym przez krok wyszukiwania korpusie. System osiągał w czasie

publikacji najlepsze dostępne wyniki na zbiorach otwartego odpowiadania na pytania: NaturalQuestions, WebQuestions (Guu et al., 2020).

3.3. Nawigowanie Wikipedii

Podejściem do wyszukiwania, które nie było wcześniej eksplorowane jest problem nawigacji po grafie dokumentów połączonych hiperłączami. Graf taki oddaje w dobry sposób organizację, przynajmniej części zasobów internetowych. Autorzy tego rozwiązania wybrali Wikipedię jako badany zbiór danych. Celem modelu jest nawigowanie od wybranego węzła grafu do innego węzła, który został wybrany z rządnej dystrybucji prawdopodobieństwa. Rozważanymi rozkładami są: rozkład jednostajny, rozkład o najkrótszych ścieżkach i rozkład odwrotnego przechodzenia grafu. Zadanie jest określone w ten sposób, że odpowiednio podzielone na paragrafy dokumenty są kodowane do reprezentacji przy użyciu sieci transformera. Następnie w każdym kroku nawigacji tzw. sieć 'policy' określa który z możliwych linków wybrać aby dostać się do węzła celu. Po wytrenowaniu system potrafi nawigować z sukcesem ścieżki na grafie Wikipedii w 90% przypadków. Co ciekawe, autorzy proponują możliwość wykorzystania tego systemu nawigacji do poprawy wyników na zadaniu zwracania właściwych paragrafów. Rozwiązanie działa na zasadzie połączenia z innymi technikami wyszukiwania informacji. Początkowe linki zwracane są za pomocą BM25, kolejno używany jest wytrenowany system do poszerzenia puli dokumentów o te do których nawiguje RFBC (zaproponowane rozwiązanie), a ostatecznie system ponownego obliczania rankingu BigBird jest używany do otrzymania ostatecznej listy linków. Wynik są porównywane dla Recall@1 do Recall@5. Badanie ablacyjne porównujące działanie systemu z oraz bez wyszukiwania ścieżek wskazuje wyraźnie na przydatność jego użycia. Ogólnie wyniki osiągane na zbiorze przy wykorzystaniu tej metody są porównywalne do nowoczesnego systemu RocketQA (Zaheer et al., 2022). Warto zauważyć, że zwrócone ścieżki na grafie są łatwo interpretowalne, system działa podobnie jak człowiek korzystający z Wikipedii. Ponadto użyte metody odwołują się w jasny sposób do algorytmów uczenia się gier typu AlphaGo, które zostały już skutecznie wykorzystane, a więc istnieje potencjał do ich ponownego wykorzystania.

Rozdział 4

Implementacja modelu wyszukiwania informacji

4.1. Wykorzystywane rozwiązania

Zamierzonym celem tego rozdziału jest przedstawienie modelu wyszukiwania informacji w celu pokazania możliwości stworzenia, w stosunkowo prosty sposób, nowoczesnej wyszukiwarki informacji bazując na otwartych i ogólnodostępnych komponentach. Badamy także jakość uzyskiwanych wyników oraz szybkość działania. Podstawą tego systemu jest są przede wszystkim udostępnione dzięki platformie HuggingFace modele językowe: enkoder, reranker i model odpowiadania na pytania. Rozwiązania takie są dostępne w bardzo prosty sposób dzięki wykorzystaniu udostępnionej biblioteki programistycznej języka Python. Należy zaznaczyć, że wykorzystanie takich rozwiązań jest nieodpłatne (w zależności od celu wykorzystania i udzielanej przez twórców licencji), oraz wymaga jedynie zwykłej karty graficznej dla mniejszych modeli. Dodatkowo możemy uruchomić te modele językowe z własnego komputera lub wykorzystując zasoby chmurowe. Drugim z kluczowych wykorzystanych rozwiązań jest biblioteka programistyczna Faiss pozwalająca na wyszukiwanie i klastrowanie gęstych reprezentacji wektorowych. Rozwiązanie to jest udostępniane na licencji wolnego oprogramowania MIT, więc można go używać do dowolnych celów. Według twórców tej biblioteki, skaluje się ona nawet do miliardów wektorów. W tej pracy wykorzystujemy Faiss jako indeks wyszukiwarki, który umożliwia różne rodzaje kodowania, szybkie wyszukiwanie, a także płynne przejście na zasoby GPU. Trzecim z elementów jest zbiór danych MS MARCO. Jest to, według naszej wiedzy, największy i jednocześnie jedyny tak kompleksowy zbiór danych pozyskany z jednej z najpopularniejszych wyszukiwarek. Pozwala on, na w miarę obiektywne, porównywanie do tej pory niemożliwej do badania jakości różnych

wyszukiwarek i sposobów wyszukiwania.

Dokumentacja wykorzystanych rozwiązań:

– SentenceTransformers: https://www.sbert.net/

– MS MARCO: https://microsoft.github.io/msmarco/

– Faiss: https://github.com/facebookresearch/faiss/wiki/

Tekst tej pracy licencjackiej wraz z wykorzystywanym kodem znajduje się w repozytorium GitHub dostępnym pod linkiem https://github.com/MateuszPuto/praca_licencjacka. Nie dołączono do niego zbioru danych 'collectionandqueries' dostępnego do pobrania z oficjalnej strony MS MARCO oraz wektorów kodowania ze względu na duży rozmiar przestrzeni dyskowej, który zajmują.

4.2. Sposób działania systemu

Należy mieć na uwadze, że rozwiązanie powstało, aby przetestować możliwości prezentowanego sposobu wyszukiwania. System nie jest w żaden sposób przystosowany, czy też zoptymalizowany jako rozwiązanie wydajne zasobowo do zastosowań produkcyjnych. Struktura kodu zorganizowana została przede wszystkim dla łatwości wprowadzania zmian oraz tak, aby umożliwiała przeprowadzanie różnych eksperymentów. Kod zawiera plik *helper_funcs.py* z najprzydatniejszymi i często wykorzystywanymi funkcjami. Badania były prowadzone w oddzielnych notatnikach Jupyter Notebook.

Pierwszym z kroków budowy wyszukiwarki było pobranie danych potrzebnych do wypełnienia bazy danych. W naszym przypadku skorzystaliśmy z gotowego zbioru danych MS MARCO zawierającego paragrafy. Pobrane zostały pliki zawierające zapytania, dokumenty oraz dopasowania z oficjalnej strony: https://microsoft.github.io/msmarco/Datasets.html. Zbiór ten zawiera 8.8 miliona dokumentów przeznaczonych do testowania rozwiązań IR. Następnie każdy z paragrafów został zakodowany do wektorów w gęstej reprezentacji, w wyniku czego uzyskano około 20 GB danych zapisanych w 9 plikach zawierających macierze numpy. Do kodowania użyto model **all-MiniLM-L6-v2**, który koduje tekst jako 384 wymiarowy wektor. Model ten jest dostępny do

pobrania z repozytoriów HuggingFace i dystrybuowany z licencją Apache-2.0 (https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2). Założenie rzeczonego systemu jest proste: nauczyć się najlepszego modelu reprezentowana tekstu przy użyciu praw skalowania. Przeznaczeniem tego modelu jest kodowanie zdań i paragrafów we wspólnej reprezentacji, którą jest przystosowana m. in. do wyszukiwania informacji. Jeśli chcielibyśmy kodować dłuższe teksty, to trzeba mieć świadomość, że są one skracane do 256 pierwszych wyrazów. Jako podstawowy blok budulcowy tego modelu wykorzystano przyciętą wersję większego modelu MiniLM-L12, która wykorzystuje co drugą warstwę większego modelu i zawiera zaledwie 6 warstw. Dostrajanie tego modelu wykorzystywało ponad 1 miliard punktów danych z różnych zbiorów, w tym także MS MARCO. Funkcją straty była nagroda kontrastowa. Wybór tego bardzo małego modelu językowego był oczywiście nieprzypadkowy i wynikał z chęci efektywnego wykorzystania ograniczonych zasobów obliczeniowych.

Mając tak przygotowane dane, zakładając, że dysponujemy odpowiednimi zasobami RAM-u, możemy je z łatwością wyszukiwać po dodaniu ich do indeksu Faiss. Indeks Faiss obsługuje wyszukiwanie za pomocą procesora centralnego (CPU), jak również procesora graficznego (GPU). W tej pracy korzystałem przede wszystkim z indeksu typu 'Flat' uruchamianym na GPU. Faiss oferuje wiele różnych niedokładnych indeksów, jak i dodatkowe opcje. Wśród nich znajdują się: IVF - wykorzystujący indeks oparty o kwantyzator z wybraną liczbą centroid, HNSW - oparty o strukturę grafową, kodowanie wektorów przy użyciu np. kodów PQ. Dodatkowo na zbiorze wektorów można wykonać krok wstępnego przetworzenia taki jak np. zmniejszenie przez PCA lub krok dokładnego wyliczenia odległości, pozwalający na dokładne wyszukanie wektorów znalezionych wcześniej przy pomocy przybliżonego indeksu. Rozwiązania takie nie są wykorzystane w tej pracy. Przeprowadziłem kilka małych eksperymentów dla różnych indeksów wykorzystujących te funkcje. Część z nich wydaje się działać dobrze dla tego problemu. Natomiast, ponieważ szybkość wyszukiwania w indeksie Faiss nie była wąskim gardłem, a dodatkowo chmura pozwala dowolnie skalować ilość pamięci operacyjnej, nie było powodu używać większości z tych udogodnień. Zwykły indeks typu 'Flat' jest równie dobry dla naszego zastosowania.

Jeśli wykonujemy krok utworzenia ponownego rankingu to wykorzystywanym do tego modelem jest cross-enkoder **ms-marco-MiniLM-L-6-v2** (https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-6-v2) udostępniany przez bibliotekę programistyczną Sentence-Transformers z repozytorium HuggingFace, podobnie jak ma to miejsce przy użyciu modelu kodującego. Według dokumentacji

model ten uzyskuje efektywność na poziomem 0.39 dla miary MRR@10 na MS MARCO. Do generowania odpowiedzi skorzystałem z modelu **tinyroberta-squad2** (https://huggingface.co/deepset/tinyroberta-squad2). Jest to wydestylowany model RoBERT'a przeznaczony do odpowiadania na pytania ze zbioru SQuAD 2.0, który działa dwa razy szybciej od modelu bazowego przy porównywalnej jakości działania. Do modelu przekazujemy tekst wszystkich pozyskanych paragrafów rozdzielonych tokenem separującym. Możliwe jest, że lepsze rezultaty można uzyskać, przekazując tylko pierwszy lub tylko kilka pierwszych paragrafów albo wykorzystując inne lepsze rozwiązanie. Jednak nawet wykorzystany w bardzo rudymentarnej formie ten zaawansowany model językowy pozwala w bardzo prosty sposób dodać znaczącą funkcjonalność do systemu wyszukiwania. Model jest dystrybuowany na licencji Creative Commons 4.0.

W końcu, stworzone zostało kilka pipeline'ów pozwalających na wyszukiwanie oraz mierzenie eketywności wyszukiwania. W pliku *search_evaluation.ipynb* zawarto sposób mierzenia MRR@k dla wyszukiwarki. Plik *simple_search.ipynb* umożliwia proste wyszukanie danego zapytania. Ostatecznie *search_tests.ipynb* w sposób deterministyczny generuje zestaw wcześniej wylosowanych zapytań, co umożliwia podejrzenie wyników działania i porównanie ich z tymi, które uzyskałem w tej pracy. W *redis_test.ipynb* pokazano, jak dokonano obliczeń wydajnościowych. Listingi pokazujące najważniejsze elementy rozwiązania znajdują się w dodatku.

4.3. Możliwości wykorzystania

System pozwala na wykonanie pełnego wyszukiwania z możliwością wykorzystania drugiego stopnia ponownego rankingu, oraz 'trzeciego stopnia' uzyskiwania odpowiedzi. Funkcjonalności takie znane są z powszechnie używanych wyszukiwarek internetowych takich jak Google. Przykładowo dla pierwszego zapytania ze zbioru treningowego MS MARCO możemy wygenerować odpowiedź, tak jak to pokazano w Tabeli 4.1. Dodatkowo z modelu TinyRoBERT'a uzyskujemy stopień przekonania modelu o poprawności udzielonej odpowiedzi oraz zaznaczenie miejsca w kontekście, w którym to według modelu znajduje się odpowiedź. Ostatnia z prezentowanych wartości oddaje przekonanie autora co do udzielonej odpowiedzi. Znakiem minus został oznaczony brak obiekcji, natomiast znak plus symbolizuje to, że, znalazłem informacje, które w mojej opinii wydawały się bardziej wiarygodne, które przeczą uzyskanej odpowiedzi. W dodatku pokazano w tej konwencji dziesięć zapytań pozyskanych w sposób losowy ze zbioru ewaluacyjnego (dev.small), dla których kontekstem były paragrafy uzyskane

z dwustopniowego wyszukiwania w pełnym zbiorze paragrafów MS MARCO.

Tabela 4.1: Zapytanie nr. 1 ze zbioru treningowego[1]

Zapytanie	")what was the immediate impact of the success of the manhattan project?"
Wygenerowana odp.	'hundreds of thousands of innocent lives obliterated'
Pewność dopasowania	0.18403230607509613
Czy błędna?	-

Co więcej, aby zobrazować działanie wyszukiwarki, dla kilku z wylosowanych zapytań stworzone zostały tabele tak jak w przykładzie dla zapytania pierwszego z Tabeli 4.2. Zaprezentowane zostało w ten sposób pierwsze dziesięć wyników wyszukiwania, podobnie jak w popularnych wyszukiwarkach internetowych. Dodatkowo wszystkie dziesięć z zapytań, które zostały wybrane, posiada prezentowane dane zapisane w pliku tekstowym, który znajduje się w publicznie udostępnionym repozytorium.

Tabela 4.2: Zwrócone wyniki dla zapytania nr. 1 ze zbioru treningowego[1]

Ranking	Początek paragrafu	ID dokumentu	Dopasowanie
1	'The Manhattan Project and its atomic [...]'	1	4.1254525
2	'The presence of communication amid scientific [...]'	0	3.8581057
3	'Essay on The Manhattan Project - [...]'	2	3.2710392
4	'Manhattan Project. The Manhattan Project was [...]'	7	1.7574501
5	'The Manhattan Project was the name [...]'	3	1.0350964
6	'Inspired by scientists who feared atomic [...]'	749027	-0.7804665
7	'Next. Inspired by scientists who feared [...]'	749029	-1.1856941
8	'In June 1942, the United States [...]'	8	-1.4733196
9	'Nor will it attempt to substitute [...]'	6	-2.9964163
10	'versions of each volume as well [...]'	4	-5.106743

4.4. Badanie efektywności

Pomiarów dokonano w Google Cloud, wykorzystując instancję z kartą graficzną Nvidia V100. W tabeli 4.3 zaprezentowane są uzyskane wyniki w mierze MRR@10 dla zbioru testowego MS MARCO oznaczonego *dev.small*. Pierwszy wiersz prezentuje wynik uzyskany dla całego zbioru testowego, natomiast kolejne dla pierwszych **n** przykładów z tego zbioru. Jak widać w tabeli, ze względu na kosztowne obliczanie pełnego rankingu, wykonano pomiar dla jedynie pierwszego tysiąca przykładów

ze zbioru. Jeśli chodzi o wyniki uzyskiwane za pomocą samego rankingu bez późniejszego ponownego rangowania, to MRR@10 wynoszący około 0.1727 jest porównywalny z wynikami raportowanymi na stronie MS MARCO dla klasycznego sposobu wyszukiwania BM25. Jest to zadowalający wynik, biorąc pod uwagę fakt, że wykorzystany sposób kodowania posiada raczej niewielkie specjalne dostosowanie do zbioru MS MARCO. Jednak fakt nieprzewyższenia klasycznej, stosunkowo prostej techniki, pozostawia pewien niedosyt. Dla pełnego rankingu na pierwszym tysiącu przykładów z deweloperskiego zbioru testowego uzyskano poważany wynik 0.4401. Według oficjalnych danych używany reranker powinien uzyskiwać wynik nieco poniżej 0.4000 na zbiorze testowym. Zakładając, że rozwiązanie nie jest bezpośrednio konkurencyjne z najlepszymi systemami, które osiągnęły obecnie maksymalnie 0.463 MRR, biorąc pod uwagę prostotę zastosowania, jest to, wydaje się, zadowalający kompromis jakości dla niektórych zastosowań.

Tabela 4.3: Wartość MRR@10 osiągnięta dla zbioru testowego MS MARCO[1]

MRR@10	Ranking	Pełen ranking
dev. small	0.1727	-
100	-	0.3694
500	0.2660	0.4415
1K	0.2260	0.4401
5K	0.1768	-

Jedną uwagą, jeśli chodzi o porównywanie prezentowanych wyników z tymi dostępnymi na stronie MS MARCO, jest fakt, że oficjalne wynik podają MRR@1000. Jednak znając sposób obliczania miary MRR, można dojść do wniosku, że obliczenie MRR dla mniejszej wartości jest przybliżeniem MRR dla większej wartości, ponieważ im niżej w wynikach pojawia się rezultat, tym mniejszy wpływ na MRR. W przybliżeniu MRR@10 nie bierze pod uwagę maksymalnie mniej niż 10% wartości MRR@1000. W praktyce te różnice są prawdopodobnie dużo niższe.

W czasie eksperymentów biblioteka Dask została zidentyfikowana jako wąskie gardło działania systemu. Przykładowo, gdy chcemy wyszukać, a następnie wykonać ponowny ranking paragrafów, konieczne jest, wyszukanie pełnych tekstów paragrafów znając ich identyfikatory. Początkowo zrealizowane zostało to przy pomocy biblioteki Dask, jednak okazało się, że skutkuje to wyraźnym spowolnieniem działania systemu. Z tego powodu zaimplementowałem inną wersję wyszukiwania z rerankingiem, która korzysta z bazy danych zapisywanej w pamięci operacyjnej — Redis. Tabela 4.4

pokazuje eksperyment przeprowadzony na komputerze osobistym z wykorzystaniem indeksu CPU i miliona zindeksowanych paragrafów. Wyszukujemy i obliczamy ponowny ranking dla dziesięciu wyników, wykorzystując dwa alternatywne rozwiązania. Tabela przedstawia ilość sekund, które zajęło wykonanie pełnego rankingu dla pięciu iteracji zapytań. Okazuje się, że rozwiązanie wykorzystujące Redis jest szybsze o prawie dwa rzędy wielkości.

Tabela 4.4: Porównanie szybkości działania biblioteki Dask i bazy danych Redis[1]

	Dask	Redis
1	110.79	3.46
2	96.40	1.86
3	96.12	2.19
4	94.79	1.86
5	94.73	1.85

4.5. Wnioski

Początkowo do mapowania z reprezentacji wektorowej do tekstu użyto biblioteki Dask, która działa podobnie jak popularna biblioteka Pandas. Jak pokazano, baza danych typu *klucz: wartość* może znacząco przyspieszyć wyszukiwanie. Oznacza to oczywiście dodatkowe zapotrzebowanie na i tak eksploatowane przez indeks zasoby pamięci operacyjnej. Inną z potencjalnych wad jest konieczność korzystania z akceleratorów sprzętowych, co może być bardziej kosztowne. Dodatkowo wykorzystanie GPU wiąże się z koniecznością odpowiedniego rozkładania obciążenia obliczeniowego, co wymaga dodatkowych kompetencji programistycznych. Ostatecznie modele deep learningowe mogą nie być bardziej efektywne niż tańsze obliczeniowo klasyczne algorytmy.

Mając to na uwadze, należy zaznaczyć również mocne strony tego podejścia. Przede wszystkim osiągnięta jakość nie odstaje znacząco od najlepszych systemów wyszukiwania. Wyszukiwanie w indeksie jest właściwie przyspieszone w porównaniu ze zwykłym algorytmem, ze względu na prostą reprezentację i korzystanie z akceleratorów sprzętowych, które są szczególnie opłacalne jeśli będziemy chcieli przetworzyć wiele zapytań jednocześnie, tak jak ma to miejsce np. w obciążonej wyszukiwarce internetowej. Co więcej, każde ulepszenie modelu językowego, którym się dysponuje,

[1] Opracowanie własne

powinno prowadzić do lepszego kodowania, a co za tym idzie lepszych uzyskiwanych wyników. Podejście to zapewnia też jednolitość w całym stosie technologicznym, gdyż praktycznie wszystkie moduły obliczeniowe korzystają z identycznych zasobów.

Ciekawym rozszerzeniem, które wymaga jedynie zmiany modelu kodowania, jest wykorzystanie modeli potrafiących kodować różne modalności, takie jak tekst i obraz do wspólnej przestrzeni wektorowej. Umożliwia to bardzo ciekawe interakcje takie jak: wyszukiwanie tekstu przy pomocy obrazu, czy wspólne wyszukiwanie różnych modalności takich jak zdjęcia i tekst. Ważną decyzją podczas kodowania reprezentacji dłuższego dokumentu jest wybór odpowiedniego poziomu granularności (cały tekst, paragrafy, wybrane fragmenty). Problem ten wydaje się w dużej mierze nierozwiązany. Kluczowym dla możliwości działania podobnego systemu jest zoptymalizowana baza danych oraz małe, a więc wydajne model neuronalne. Co więcej sam reranking mógłby potencjalnie odbywać się w nauczonej reprezentacji wektorowej. Sprawiłoby to, że reranking byłby potencjalnie wystarczająco szybki do masowego zastosowania. Połączenie modelu językowego z wyszukiwarką pozwala na stworzenie pętli zwrotnej, gdzie model współpracuje z człowiekiem, aby uzyskać jak najlepsze rezultaty. Sygnały w stylu 'relevance feedback' mogą być wykorzystane jako dane treningowe dla rerankera.

Systemy wyszukiwania neuronalnego mogą być konkurencyjną propozycją dla wyszukiwarek na stronach internetowych. Mimo istnienia bibliotek i modeli udostępnionych na licencjach wolego oprogramowania brak jest kompleksowego rozwiązania, które można by wdrożyć bezpośrednio do serwisu internetowego. Dla witryn poszukujących wyszukiwarki wysokiej jakości rozwiązania te mogą stanowić czasami konkurencyjną opcję. Wdrożenie takiego systemu wymaga obecnie jednak ekspertyzy, aby zachować odpowiednią ekonomiczność. Wykorzystanie modeli DL do wyszukiwania stwarza nowe pole do konkurencji z gigantami dla start-up'ów takich jak 'Perplexity AI'. Jednocześnie liderzy rynku, tacy jak: Google i Microsoft inkrementalnie wdrażają rozwiązania oparte na sieciach neuronowych do swoich silników. Aby oferować konkurencyjne rezultaty względem klasycznych technik, wyszukiwanie neuronalne musi nie tylko generować lepsze rezultaty, ale również być konkurencyjne pod względem kosztów obliczeniowych. W końcu, moim zdaniem, największymi usprawnieniami, które przyniesie wdrożenie uczenia głębokiego do wyszukiwarek, jest interoperacyjność między różnymi modalnościami, kooperacja z innymi modelami uczenia maszynowego oraz wyposażenie modeli, takich jak modele językowe w wydajny moduł pamięci.

Załączniki

4.6. Słownik pojęć

.

Dokument - plik w wyszukiwaniu informacji

Term - wyraz, podstawowa część tekstu

Stemming - proces wydobycia ze słowa tematu wyrazu

BM25 - dokładniej Okapi BM25, funkcja rankingowa używana w wyszukiwaniu informacji

P(Q|d) - wykorzystywany w tekście skrót, określający prawdopodobieństwo warunkowe wystąpienia zapytania pod warunkiem istnienia dokumentu

MRC - ang. Machine Reading Comprehension

IR - ang. Information Retrieval (Wyszukiwanie Informacji)

Token - podstawowa jednostka kodowania tekstu dla przetwarzania przez modele językowe, najczęściej odpowiada części wyrazu

Trening wstępny - (ang. pretraining) podejście do trenowania sieci neuronowych, w których model jest najpierw uczony generalnych zależności na całym dostępnym zbiorze danych

Dostrajanie modelu - (ang. fine-tuning) polega na dostosowywaniu wag sieci, które zostały wstępnie ustalone w fazie treningu wstępnego, do potrzeb specyficznego zadania

Mechanizm uwagi - (ang. attention mechanism) technika pozwalająca na ważeniu różnych części danych wejściowych (np. tokenów), najczęściej stosowaną wersją jest uwaga typu QKV (zapytanie, klucz, wartość)

Głowa modelu - (ang. model head) końcowe warstwy sieci neuronowej, często można ją odciąć i dodać inną "głowę", w zależności od przeznaczonego zadania

Bienkoder - (ang. bi-encoder) spolszczona nazwa, określająca w tym wypadku sieć modelu, która tworzy oddzielne reprezentacje tekstu, które mogą zostać później porównane np. wykorzystując podobieństwo kosinusowe

4.7. Przykładowe wyniki wyszukiwań

Tabela 4.5: Zapytanie nr. 342115 ("how therapist refer to dr for evaluation")[2]

Ranking	Początek paragrafu	ID dokumentu	Dopasowanie
1	'Evaluation. OT can be initiated when [...]'	6877831	2.8268788
2	'Traditionally, to receive an evaluation and/or [...]'	5408137	2.588758
3	'Definition of therapist for Students. [...]'	51075	-4.2144966
4	'When choosing a professional to work [...]'	6956788	-9.4832325
5	'When choosing a professional to work [...]'	6956785	-9.625549
6	'Most physical therapist assistants and aides [...]'	527164	-10.133472
7	'John T. Philipsborn, in private practice [...]'	2586713	-10.978702
8	'Although the text in the document [...]'	220928	-11.360448
9	'Fitting regular exercise into your daily [...]'	85197	-11.366676
10	'If this Article is silent or [...]'	4182815	-11.438606

Tabela 4.6: Zapytanie nr. 1069981 ("what is a mustelidae")[2]

Ranking	Początek paragrafu	ID dokumentu	Dopasowanie
1	'The Mustelidae (from Latin mustela, weasel) [...]'	5450693	10.599756
2	'Mustelid: Mustelid, (family Mustelidae), any [...]'	7158641	7.1848607
3	'1. any small carnivore of the genus [...]'	6666355	6.8733773
4	'weasel. 1. any small carnivore of the [...]'	6666356	6.52673
5	'Weasels and all their variants make [...]'	7460419	6.051278
6	'[...] Minks and weasels [...]'	5098133	5.03244
7	'LarivÃ¨re and Jennings (2009) recognized 57 [...]'	7158646	3.1976109
8	'1 Ferrets come from the same family [...]'	1795078	0.275114
9	'By the little garden pergola open [...]'	260407	-11.308208
10	'Buy Ty Ice Age Beanie Babies [...]'	204680	-11.419775

[2]Opracowanie własne, Pełne wyniki wyszukiwania dla 10 przedstawionych zapytań są dostępne pod adresem: https://github.com/MateuszPuto/praca_licencjacka/tree/main/exmpl_queries.

Tabela 4.7: Zapytanie nr. 671692 ("what involves problems sleeping")[2]

Ranking	Początek paragrafu	ID dokumentu	Dopasowanie
1	'Sleeping problems are common in many [...]'	6743826	5.9614067
2	'Sleeping problems are common in many [...]'	7707677	5.7937694
3	'Although causes may differ, the end [...]'	6466291	4.1647916
4	'Sleep Apnea Related Diseases & Conditions. [...]'	691303	-0.09731629
5	'Most people go to bed at [...]'	3339195	-1.271653
6	'Find out how to tell if [...]'	3239019	-1.4049392
7	'1 Before you have surgery, tell your [...]'	240471	-1.6717714
8	'Causes of Childhood Insomnia. One [...]'	3339199	-3.1157022
9	'Most adults need 7 to 9 hours [...]'	1301637	-8.087761
10	'For patients with a co-pay of 20 percent [...]'	3696268	-11.304798

Tabela 4.8: Zapytanie nr. 342115[2]

Zapytanie	"how therapist refer to dr for evaluation"
Wygenerowana odp.	'physical therapist'
Pewność dopasowania	2.258343101857463e-06
Czy błędna?	+

Tabela 4.9: Zapytanie nr. 1069981[2]

Zapytanie	"what is a mustelidae"
Wygenerowana odp.	'a family of carnivorous mammals'
Pewność dopasowania	0.5254051089286804
Czy błędna?	-

Tabela 4.10: Zapytanie nr. 671692[2]

Zapytanie	"what involves problems sleeping"
Wygenerowana odp.	'changes in the brain regions and neurotransmitters that control sleep'
Pewność dopasowania	0.332465797662735
Czy błędna?	-

Tabela 4.11: Zapytanie nr. 987237[2]

Zapytanie	"who sings the wedding song on walker texas ranger"
Wygenerowana odp.	'Bree Anna Hutchinson'
Pewność dopasowania	0.16306354105472565
Czy błędna?	+

Tabela 4.12: Zapytanie nr. 927553[2]

Zapytanie	"what year was america founded"
Wygenerowana odp.	'July 4,1776'
Pewność dopasowania	0.7525272965431213
Czy błędna?	-

Tabela 4.13: Zapytanie nr. 68095[2]

Zapytanie	"can hives be a sign of pregnancy"
Wygenerowana odp.	'Pregnancy hives are common in women of all stages of pregnancy'
Pewność dopasowania	0.33129480481147766
Czy błędna?	-

Tabela 4.14: Zapytanie nr. 447551[2]

Zapytanie	"definition of culture in an organization"
Wygenerowana odp.	"CARLAâx80x99s"
Pewność dopasowania	0.144457146525383
Czy błędna?	+

Tabela 4.15: Zapytanie nr. 992840[2]

Zapytanie	"meaning of current ratio"
Wygenerowana odp.	'a measure of a company's ability to meet its short-term liabilities'
Pewność dopasowania	0.11757927387952805
Czy błędna?	-

Tabela 4.16: Zapytanie nr. 272605[2]

Zapytanie	"how long to wait to swim after tattoo"
Wygenerowana odp.	'two weeks'
Pewność dopasowania	0.5983613133430481
Czy błędna?	-

Tabela 4.17: Zapytanie nr. 989894[2]

Zapytanie	"who is mark davis"
Wygenerowana odp.	'bassist and founding member for the band Emmure'
Pewność dopasowania	0.524512767791748
Czy błędna?	-

4.8. Listingi najważniejszych części kodu

```
def create_index(d: int, description: str, gpu: bool):
    '''Creates FAISS search index'''
    index = faiss.index_factory(d, description, faiss.METRIC_INNER_PRODUCT)

    if gpu == True:
        res = faiss.StandardGpuResources()
        index = faiss.index_cpu_to_gpu(res, 0, index)

    return index
```

Listing 4.1:]Tworzenie indeksu[3]

```
def text_to_tensor(sentence: str, model):
    '''Converts string to representation using provided encoder model'''

    return model.encode(sentence, convert_to_tensor=False)

def get_entries(filename: str, size: int, encoder):
    '''Creates matrix of encoded MS MARCO paragraphs'''
    x = []

    with open(filename, 'r') as file:
        for i in range(size):
            paragraph = file.readline().split('\t')[1]
            encoding = text_to_tensor(paragraph, encoder)
            x.append(encoding)

    return np.squeeze(np.stack(x))
```

Listing 4.2:]Przetwarzanie zbioru danych[3]

```
def mrr_at(k: int, result: list, match_id: int):
    '''Calculates reciprocal rank; MRR@k = sum(reciprocal_ranks[:])'''
    if match_id in result[:k]:
        reciprocal_rank = 1 / (result[:k].index(match_id) + 1)
    else:
        reciprocal_rank = 0

    return reciprocal_rank
```

Listing 4.3:]Sposób obliczania MRR[3]

```
def populate_index(xb, index, to_cpu):
    '''Trains and adds vectors to FAISS index'''
    if to_cpu:
        xb = xb.cpu().numpy()

    faiss.normalize_L2(xb)

    index.train(xb)
    index.add(xb)

d = 384
description = "Flat"

index = helper_funcs.create_index(d, description, gpu=True)
xb = torch.load('msmarco-vectors-1.pt')
populate_index(xb, index, to_cpu=True)
```

Listing 4.4:]Dodawanie rekordów do indeksu[3]

```
def search(query_val: str, num_results: int, encoder, index, normalize=True):
    '''Search query string in FAISS index, returns results and distances'''
    vec = text_to_tensor(query_val, encoder).unsqueeze(0).detach()
            .cpu().numpy().astype('float32')

    if normalize:
        faiss.normalize_L2(vec)

    d, i = index.search(vec, num_results)

    return (i, d)
```

Listing 4.5:]Wyszukiwanie zasobów[3]

```
def rerank(query_val: str, documents: list, indices: list, cross_encoder):
    '''Reranks documents with respect to query_val'''
    query_document = zip([query_val for i in range(len(documents))], documents)
    q_d = list(query_document)
    scores = cross_encoder.predict(q_d)

    reranked_results = sorted(zip(zip(indices[0], documents), scores),
            key=itemgetter(1), reverse=True)

    return reranked_results
```

Listing 4.6:]Ponowny ranking[3]

[3]Opracowanie własne, źródło: https://github.com/MateuszPuto/praca_licencjacka/tree/main/code

Wykaz tabel

Bibliografia

Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., et al. (2016). Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Berners-Lee, T. (1995). *Hypertext markup language - 2.0 memo*. https://www.ietf.org/rfc/rfc1866.txt

Broadley, C. (2023). Altavista search engine history lessons for internet nerds. https://digital.com/altavista/

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, *26*(2), 1–26.

Chang, P. (2022a). Product quantization for similarity search. https://towardsdatascience.com/product-quantization-for-similarity-search-2f1f67c5fddd

Chang, P. (2022b). Similarity search with ivfpq. https://towardsdatascience.com/similarity-search-with-ivfpq-9c6348fd4db3

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dutton, S. (2018). How to add full text search to your website. https://medium.com/dev-channel/how-to-add-full-text-search-to-your-website-4e9c80ce2bf4

Gao, L., & Callan, J. (2021a). Condenser: A pre-training architecture for dense retrieval. *arXiv preprint arXiv:2104.08253*.

Gao, L., & Callan, J. (2021b). Unsupervised corpus aware language model pre-training for dense passage retrieval. *arXiv preprint arXiv:2108.05540*.

Google. (2023a). Bigtable overview. https://cloud.google.com/bigtable/docs/overview

Google. (2023b). Googlebot. https://developers.google.com/search/docs/crawling-indexing/googlebot

Google. (2023c). Jak google interpretuje specyfikację pliku robots.txt. https://developers.google.com/search/docs/crawling-indexing/robots/robots_txt

Google. (2023d). Szczegółowy przewodnik po działaniu wyszukiwarki google. https://developers.google.com/search/docs/fundamentals/how-search-works

Google Developers. (2023). Szczegółowy przewodnik po działaniu wyszukiwarki google. https://developers.google.com/search/docs/fundamentals/how-search-works

Guu, K., Lee, K., Tung, Z., Pasupat, P., & Chang, M. (2020). Retrieval augmented language model pre-training. *International conference on machine learning*, 3929–3938.

Hu, K. (2023). Chatgpt sets record for fastest-growing user base - analyst note. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/

Izacard, G., & Grave, E. (2020). Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*.

Joachims, T. (2002). Optimizing search engines using clickthrough data. https://www.cs.cornell.edu/~tj/publications/joachims_02c.pdf

Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, *7*(3), 535–547.

Khattab, O., & Zaharia, M. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over bert. *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 39–48.

Kinsta. (2022). Search engine market share: Who's leading the race in 2022. https://kinsta.com/search-engine-%20market-share/

Lamoine, J. (2016a). Inside the algolia engine series. https://www.algolia.com/blog/engineering/inside-the-algolia-engine-part-1-indexing-vs-search/

Lamoine, J. (2016b). Inside the algolia engine series: Part 2. https://www.algolia.com/blog/engineering/inside-the-algolia-engine-part-2-the-indexing-challenge-%20of-instant-search/

Lamoine, J. (2016c). Inside the algolia engine series: Part 3. https://www.algolia.com/blog/engineering/inside-the-algolia-engine-part-3-query-processing/

Lamoine, J. (2016d). Inside the algolia engine series: Part 4. https://www.algolia.com/blog/engineering/inside-the-algolia-enginepart-4-textual-relevance/

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. *International conference on machine learning*, 1188–1196.

Ma, X., Guo, J., Zhang, R., Fan, Y., Ji, X., & Cheng, X. (2021). Prop: Pre-training with representative words prediction for ad-hoc retrieval. *Proceedings of the 14th ACM international conference on web search and data mining*, 283–291.

Mallia, A., Khattab, O., Suel, T., & Tonellotto, N. (2021). Learning passage impacts for inverted indexes. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1723–1727.

Manning, C. D. (2009). *An introduction to information retrieval*. Cambridge university press.

MarketMeChina. (2022). Baidu search engine market share in china mar 2022. https://www.marketmechina.com/baidu-search-engine-market-share-in-china-mar-2022/

Mehdi, Y. (2023). Reinventing search with a new ai-powered microsoft bing and edge, your copilot for the web. https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/

Meta AI Research. (2023). Faiss library. https://github.com/facebookresearch/faiss/wiki

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mirtaheri, S. M., Dinçktürk, M. E., Hooshmand, S., Bochmann, G. V., Jourdan, G.-V., & Onut, I. V. (2014). A brief history of web crawlers. *arXiv preprint arXiv:1405.0749*.

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., et al. (2021). Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Nayak, P. (2022). How ai powers great search results. https://blog.google/products/search/how-ai-powers-great-search-results/

Nogueira, L. (2019). From doc2query to doctttttquery. *Online preprint*, *6*.

Nogueira, R., & Cho, K. (2019). Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

Nogueira, R., Yang, W., Lin, J., & Cho, K. (2019). Document expansion by query prediction. *arXiv preprint arXiv:1904.08375*.

Page, L. (1998). *The pagerank citation ranking: Bringing order to the web* (tech. rep.) [http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf]. Stanford.

Phillips, H. A. (2010). Great library of alexandria. *Library Philosophy and Practice*.

Pichai, S. (2023). An important next step on our ai journey. https://blog.google/technology/ai/bard-google-ai-search-updates/

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, *21*(1), 5485–5551.

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Robertson, S. (1977). The probability ranking principle in ir. *Journal of Documentation*, *33*, 294–304. https://doi.org/10.1108/eb026647

Sanderson, M., & Croft, W. B. (2012). The history of information retrieval research. *Proceedings of the IEEE*, *100*.

Statcounter. (2022). Search engine market share in japan - november 2022. https://gs.statcounter.com/search-engine-market-share/all/japan

Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., et al. (2022). Lamda: L anguage models for dialog applications. *arXiv preprint arXiv:2201.08239*.

Tonellotto, N. (2022). Lecture notes on neural information retrieval. *arXiv preprint arXiv:2207.13443*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Xinyu Ma, J. C. (2021). Prop github repository. https://github.com/Albert-Ma/PROP/

Zaheer, M., Marino, K., Grathwohl, W., Schultz, J., Shang, W., Babayan, S., Ahuja, A., Dasgupta, I., Kaeser-Chen, C., & Fergus, R. (2022). Learning to navigate wikipedia by taking random walks. *arXiv preprint arXiv:2211.00177*.

Zhang, Y., Long, D., Xu, G., & Xie, P. (2022). Hlatr: Enhance multi-stage text retrieval with hybrid list aware transformer reranking. *arXiv preprint arXiv:2205.10569*.

Zhu, F., Lei, W., Wang, C., Zheng, J., Poria, S., & Chua, T.-S. (2021). Retrieving and reading: A comprehensive survey on open-domain question answering. *arXiv preprint arXiv:2101.00774*.