Przetwarzanie języka naturalnego

Informatyka, Studia stacjonarne II stopnia

Temat projektu: *Wyszukiwarka dokumentów*

Autorzy: Kamil Kapliński, Mateusz Surynowicz

1. Wstęp

Celem projektu było stworzenie aplikacji umożliwiającej wyszukiwanie dokumentów. Utworzony system jest aplikacją webową, z którą użytkownik komunikuje się za pośrednictwem przeglądarki internetowej. Wyszukiwanie odbywa się w oparciu o obliczanie podobieństwa między zapytaniem, a dokumentami istniejącymi w stworzonej bazie danych. Jako miarę podobieństwa zastosowano podobieństwo kosinusowe. Wektory dokumentów zostały stworzone przy użyciu technik takich jak algorytm *TF-IDF* oraz *Latent Semantic Analysis* (LSA). W związku z powyższym, oprócz poszukiwania dokumentów, które zawierają frazy występujące w zapytaniu (*TF-IDF*), program znajduje dokumenty, których reprezentacje wektorowe znajdują się blisko zapytania w przestrzeni semantycznej. W ramach funkcjonalności dodatkowej zaimplementowano autokorektę zapytania wpisanego przez użytkownika korzystając z odległości Levenshtein'a.

2. Użyte korpusy dokumentów

Baza danych stworzonej aplikacji zawiera dokumenty tekstowe pochodzące z dwóch źródeł: *BBC News* (Insight - BBC Datasets (ucd.ie)) oraz *The 20 Newsgroups* (Home Page for 20 Newsgroups Data Set (qwone.com)). *BBC News* to zbiór artykułów prasowych pochodzących ze strony Home - BBC News. Zawiera on 2225 dokumentów z pięciu różnych kategorii: *business*, *entertainment*, *politics*, *sport*, *tech*. Zbiór *The 20 Newsgroups* to kolekcja 20 000 dokumentów należących do jednej z 20 kategorii widocznych na rysunku 2.1. Kategorie zostały podzielone na grupy o zbliżonej tematyce. Szczegółowe źródła, z których pochodzą dokumenty z korpusu 20 *Newsgroups* nie są znane. Treść dokumentów stanowią nieduże artykuły prasowe, posty użytkowników stron internetowych i związane z nimi komentarze innych użytkowników.

W przypadku obu korpusów możliwe jest pobranie dokumentów w postaci plików .txt, które należy samodzielnie przetworzyć w zależności od potrzeb. Dostępne są również pliki zawierające wyodrębnione informacje z korpusów, takie jak np. słownik wyrazów. Autorzy niniejszego projektu zdecydowali się sami przetworzyć korpusy korzystając z plików .txt.

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Rys. 2.1. Kategorie dokumentów korpusu *The 20 Newsgroups*

3. Zastosowane technologie

Kolejność opisywania poszczególnych narzędzi została oparta na warstwowej architekturze systemu począwszy od warstwy prezentacji. Szczegóły dotyczące architektury zostały opisane w punkcie 4 dokumentacji. Do stworzenia systemu użyto następujących narzędzi

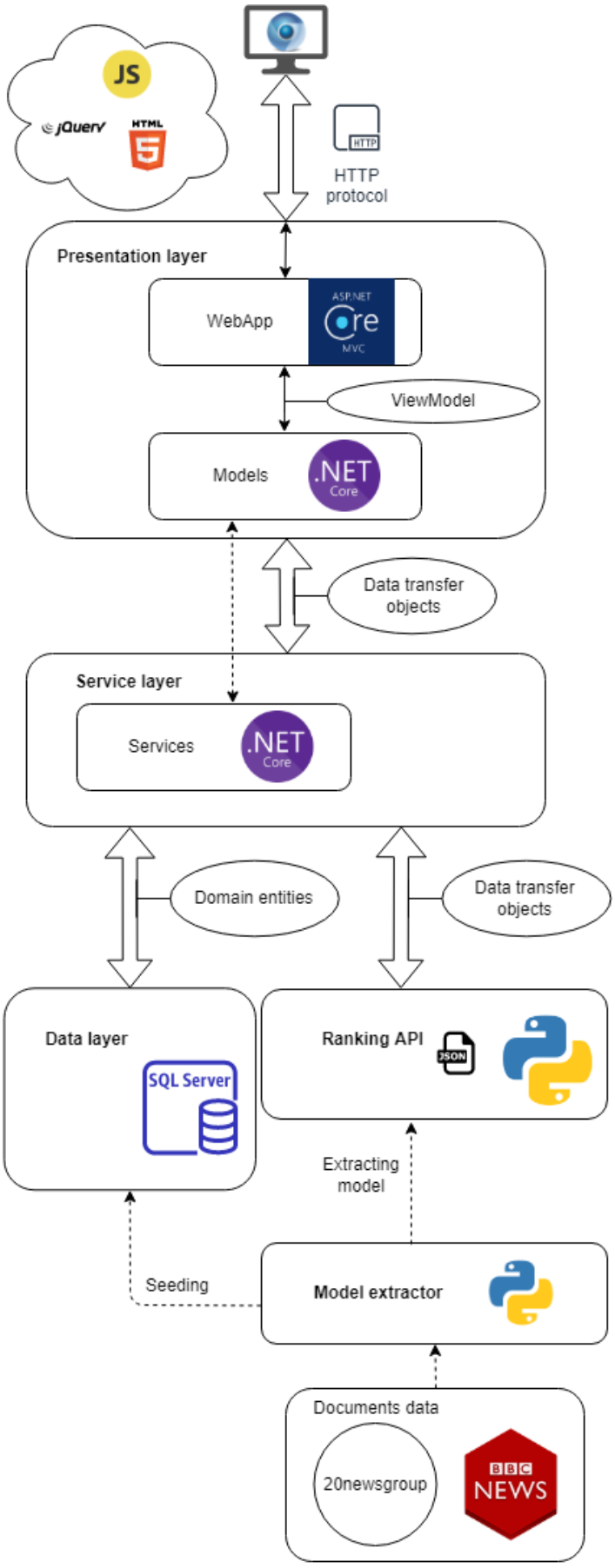
- ASP.NET Core MVC – szablon projektu zgodny ze wzorcem MVC w pakiecie z serwerem webowym IIS renderującym strony internetowe w formacie HTML
- JavaScript, jQuery – języka JavaScript i biblioteki jQuery użyto do implementacji zdarzeń interakcji użytkownika z komponentami wyrenderowanymi na stronie internetowej
- RestSharp – biblioteka języka C# umożliwiająca wysyłanie żądań HTTP, użyta do komunikacji z API zwracającym ranking dokumentów do wyświetlenia
- Linq2Db – biblioteka języka C# mapująca encje bazodanowe na obiekty .NET, służąca do tworzenia obiektów-dokumentów ze zwróconych z bazy danych rekordów
- Microsoft SQL Server 2017 – serwer bazodanowy użyty do przechowywania dokumentów
- Python – język Python został użyty do ekstrakcji istotnych danych z korpusów dokumentów, służących do utworzenia reprezentacji wektorowej dokumentów oraz tworzenia rankingu dokumentów w odpowiedzi na zapytanie użytkownika
- Flask – biblioteka języka Python użyta do utworzenia API zwracającego ranking dokumentów w formacie JSON, w odpowiedzi na zapytanie użytkownika
- Sklearn – biblioteka języka Python zbudowana w oparciu o biblioteki NumPy, SciPy i matplotlib posłużyła do obliczania wektorów semantycznych i *TF-IDF* dokumentów
- Spacy – biblioteka języka Python służąca do przetwarzania języka naturalnego użyta do tokenizacji i lematyzacji korpusów i zapytań użytkownika.
- Pandas – biblioteka języka Python ułatwiająca pracę z danymi tabelarycznymi
- SymSpell – biblioteka języka Python użyta do obliczania odległości Levenshtein’a

4. Architektura aplikacji

Rysunek 4.1. przedstawia architekturę aplikacji. Użytkownik końcowy komunikuje się z serwerem webowym przy użyciu przeglądarki internetowej. Przygotowywanie stron internetowych do wyświetlenia odbywa się w warstwie prezentacji, gdzie serwer deleguje żądanie protokołu http do biblioteki klas nazwanych *Models*. Klasy tej biblioteki odpowiadają za przygotowanie danych z warstwy serwisów. Przygotowane dane to obiekty nazywane *ViewModel*. Zwrócone do projektu serwera (*WebApp*), zostają wyrenderowane, a następnie wysłane użytkownikowi w postaci strony HTML.

Warstwa serwisów odpowiada za pobranie dokumentów z bazy danych i przekazanie ich do klas *Models*. Kontrakt pomiędzy warstwą serwisów, a warstwą prezentacji stanowią obiekty *Data transfer*, których zawartość jest wykorzystywana do tworzenia obiektów *ViewModel*. Otrzymując zapytanie z interfejsu użytkownika, serwisy przekazują je do warstwy obliczeniowej zwracającej najbardziej pasujące do zapytania identyfikatory dokumentów, które są następnie użyte do pobrania treści dokumentów z warstwy danych.

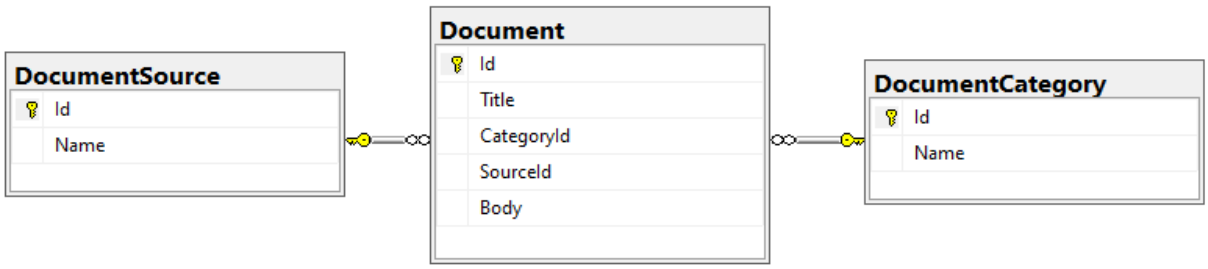
Warstwa obliczeniowa to programistyczny interfejs aplikacji (API), który zwraca ranking dokumentów utworzony w oparciu o zapytanie użytkownika. Do obliczenia rankingu wykorzystywany jest model powstały w wyniku przetworzenia korpusów danych. Model został stworzony przez odrębną aplikację w języku Python – tzw. „Ekstraktor”. Podczas przetwarzania korpusów ekstraktor wypełnia bazę danych SQL (warstwa danych) treścią dokumentów. Szczegółowy model bazy został przedstawiony w punkcie 5. Dokumentacji. W trakcie przetwarzania obliczane są również wektory semantyczne i wektory *TF-IDF* dokumentów. Dzięki informacjom zawartym w modelu, API jest w stanie dokonać odpowiednich obliczeń i zwrócić serwisom ranking.



Rys. 4.1. Architektura systemu

5. Model bazy danych

Baza danych składa się z 3 tabel (rys. 5.1.). Tabela *Document* zawiera informacje o tytule, identyfikatorze kategorii, identyfikatorze źródła i treści dokumentu. Identyfikatory kategorii i źródła to klucze obce wskazujące na tabele *DocumentCategory* i *DocumentSource*. Tabela *DocumentCategory* przechowuje nazwę kategorii, a tabela *DocumentSource* – nazwę źródła (*BBC News*, bądź *The 20 Newsgroups*).



Rys. 5.1. Model bazy danych SQL

6. Ekstrakcja modelu dokumentów i jego użycie

W niniejszym punkcie zostaną przedstawione szczegóły tworzenia modelu, który wykorzystywany przez API, służy do zwracania rankingu dokumentów najbardziej pasujących do zapytania użytkownika.

Przygotowanie korpusów dokumentów

Zbiór *BBC News* i *The 20 Newsgroups* wymagają innej techniki czyszczenia i wydobywania treści na inny format. Korpus *Newsgroups* jest silnie „zanieczyszczony” wieloma frazami, które nie reprezentują żadnej wartości semantycznej. Takie frazy to adresy e-mail, numery telefonów, adres url stron internetowych, a także odpowiedzi użytkowników na pytania innych użytkowników, które zaczynają się w charakterystyczny sposób od nazwy autora poprzedzonej znakiem małpy „@”, bądź frazą „Re:”. W związku z tym utworzono oddzielne funkcje czyszczące dokumenty. Wspólną część przetwarzania dokumentów stanowi funkcja tokenizująca i lematyzująca tekst. Do takiej funkcji zostaną podane wyczyszczone treści dokumentów z obu korpusów.

Po przetworzeniu tekstu dokumenty zapisywane są w obiekcie *DataFrame* biblioteki pandas. Na rysunku 6.1. umieszczono pierwszy element tej struktury. Struktura składa się z następujących pól:

- Tytuł dokumentu
- Wyczyszczony tytuł (tytuł pozbawiony znaków specjalnych)
- Wyczyszczony i zlematyzowany tytuł

- Wyczyszczony i zlematyzowany tytuł pozbawiony stop-słów
- Ciało dokumentu
- Wyczyszczone ciało
- Wyczyszczone i zlematyzowane ciało
- Wyczyszczone i zlematyzowane ciało pozbawione stop-słów

W następnym etapie *DataFrame* zostanie wykorzystany do budowy wektorów *TF-IDF* i wektorów semantycznych.

Title	Cleaned title	Cleaned title lemmatized	Cleaned title lemmatized no stopwords	Body	Cleaned body	Cleaned body lemmatized	Cleaned body lemmatized no stopwords
Ad sales boost Time Warner profit	ad sales boost time warner profit	ad sale boost time warner profit	sale boost time warner profit	Quarterly profits at US media giant TimeWarner...	quarterly profits at us media giant timewarner...	quarterly profit at us medium giant timewarner...	quarterly profit medium giant timewarner jump ...

Rys. 6.1. Fragment zawartości obiektu *DataFrame* z przetworzonym dokumentem

Utworzenie reprezentacji wektorowej dokumentów

Każdy dokument jest reprezentowany przez 3 wektory: wektor tytułu, treści oraz semantyki. Wektory tytułu i treści to wektory *TF-IDF* powstałe w oparciu o kolumny „*Cleaned title*”, „*Cleaned title lemmatized*”, „*Clenaed body*” i „*Cleaned body lemmatized*”. Wektory *TF-IDF* służą do sprawdzenia dosłownego podobieństwa wyrazowego między zapytaniem użytkownika a treścią dokumentu. W związku z użytymi do utworzenia wektorów *TF-IDF* kolumnami, wektory te oprócz słów zlematyzowanych, posiadają ich odmiany. Dzięki temu wyszukiwarka będzie mogła reagować na wyszukiwanie odmian słów w formie podstawowej. Wyszukując słowo „running”, system powinien umieścić dokumenty posiadające owe słowo wyżej w rankingu, niż dokumenty ze słowami „run” (pomijając aspekt semantyczny). Wektory zawierają również stop słowa, dzięki czemu możliwe jest wyszukiwanie po słowach takich jak „you’ll”, „where”, itd. Tego typu wyrazy również pojawiają się w tytułach wielu tekstów kultury i mogą zwiększyć trafność wyszukiwania. Użycie *TF-IDF* to szczegół implementacyjny i autorzy wiedzą, że w celu sprawdzenia podobieństwa wyrazowego równie dobra mogłaby być zwykła reprezentacja *Bag of words*. Listing 6.1. przedstawia fragment kodu odpowiedzialnego za utworzenie wektorów tytułów i treści dokumentów. Obiekt *TfidfVectorizer* pochodzi z biblioteki *sklearn*, która zawiera gotową implementację algorytmu *TF-IDF*. Zmienna *articles_df* to obiekt *DataFrame* przechowujący przetworzone dokumenty.

```
vectorizer = TfidfVectorizer(tokenizer=identity_tokenizer, lowercase=False)
tfidf_fit = vectorizer.fit(articles_df['Cleaned title'] + ' ' +
                           articles_df['Cleaned title lemmatized'] + ' ' +
                           articles_df['Cleaned body'] + ' ' +
                           articles_df['Cleaned body lemmatized'])

tfidf_transform_body = tfidf_fit.transform(articles_df['Cleaned body'] + ' ' +
                                           articles_df['Cleaned body lemmatized'])
tfidf_transform_title = tfidf_fit.transform(articles_df['Cleaned title'] + ' ' +
                                            articles_df['Cleaned title lemmatized'])
```

List. 6.1. Utworzenie reprezentacji wektorowej *TF-IDF* tytułu i treści dokumentu

Po wektoryzacji *TF-IDF* słownik wyrazów składa się z 151442 słów (list. 6.2.).

```
len(tfidf_fit.vocabulary_)
```

```
151442
```

List. 6.2. Utworzenie reprezentacji wektorowej *TF-IDF* tytułu i treści dokumentu

Kolejnym krokiem jest stworzenie wektorów opisujących semantykę dokumentów. Dokumenty o podobnej tematyce powinny być umieszczone blisko siebie w przestrzeni semantycznej. Reprezentacja semantyczna jest tworzona przy wykorzystaniu techniki Singular Value Decomposition (SVD). Jest to algorytm rozkładania macierzy na iloczyn trzech prostszych macierzy. SVD rozkłada macierz A o wymiarach $m\ x\ n$ na trzy macierze: U o wymiarach $m\ x\ k$, S o wymiarach $k\ x\ k$ i V o wymiarach $n\ x\ k$:

$$A = U \cdot S \cdot V^T$$

Gdzie:

A – macierz oryginalna,

U – tzw. „left singular vectors”,

S – tzw. „singular values”,

R – tzw. „right singular vectors”.

SVD nie musi odtwarzać macierzy pierwotnej w sposób jednoznaczny. Macierz oryginalną można aproksymować poprzez SVD niższego rzędu (*lower-order SVD*), gdzie macierz wynikowa ma znacznie mniejszy wymiar niż macierz oryginalna. Zastosowanie aproksymacji może odkryć interesujące relacje pomiędzy wierszami i kolumnami powstałych w wyniku rozkładu macierzy. Wykorzystanie tej techniki w przetwarzaniu języka naturalnego jest znane jako Latent Semantic Analysis (LSA) i służy do wykrywania ukrytych znaczeń w tekstach. W niniejszym systemie użyto techniki Truncated SVD, zgodnie z którą wybrano n największych wartości z macierzy S (singular values), a w miejsce reszty wartości macierzy S, wstawiono zera.

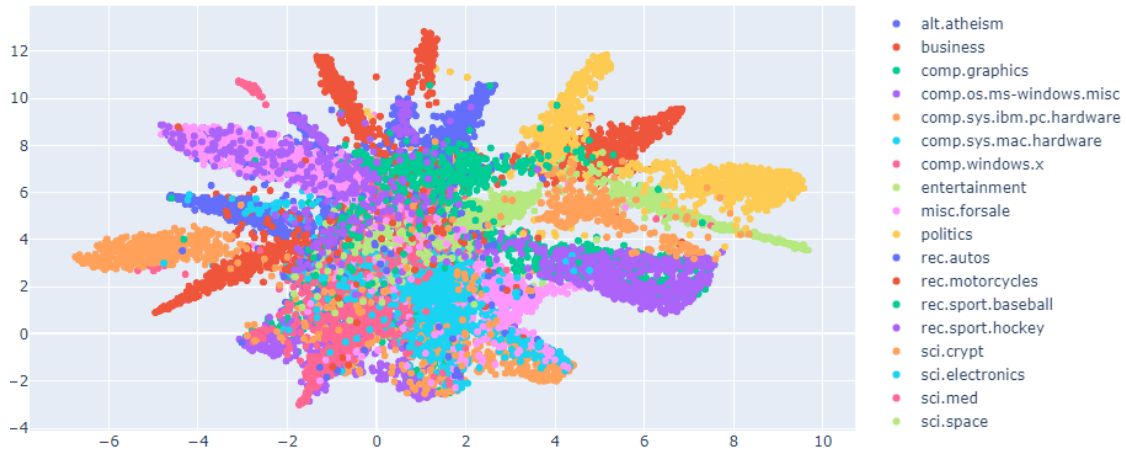
Do utworzenia wektorów semantycznych użyto kolumn „*Cleaned title lemmatized*” i „*Cleaned body lemmatized*”, pozbywając się odmian fleksyjnych i stop słów, gdyż takie frazy nie posiadają żadnej

wartości semantycznej. Używając tych kolumn utworzono długie i rzadkie wektory *TF-IDF* dokumentów, które następnie zaproksymowano techniką Truncated SVD. W związku z tym, że w korpusie danych znajdują się teksty należące do jednej z 25 kategorii, zmienna k w wymiarach macierzy U, S, V powstałych z rozkładu, wynosi 25. W ten sposób można stwierdzić w jakim stopniu dany dokument należy do jednej z 25 kategorii. Skorzystano z biblioteki *sklearn*, która posiada implementację SVD. Na listingu 6.3. zamieszczono fragment kodu zawierający wywołanie funkcji *svd.fit* oraz *svd.transform* biblioteki *sklearn*. Klasa *TruncatedSVD* posiada implementację algorytmu Truncated SVD. Z macierzy S (singular values) wybrano 25 największych wartości (parametr *n_components*). Zmienna *tfidf_transform_lsa* to wektory *TF-IDF* dokumentów poddane zabiegowi SVD. Zmienna *lsa_fit* zawiera transponowaną macierz V. Jest to macierz, w której wiersze reprezentują kategorie dokumentów, a kolumny – wyrazy występujące w korpusie. Dzięki temu możemy określić semantyczność wyrazów słownika. Zmienna *lsa* to macierz, w której wiersze reprezentują dokumenty, a kolumny – kategorie korpusu. Podsumowując, dzięki technice SVD udało się odkryć semantykę dokumentów, a także pojedynczych wyrazów korpusu.

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=25, random_state=42)
lsa_fit = svd.fit(tfidf_transform_lsa)
lsa = svd.transform(tfidf_transform_lsa)
```

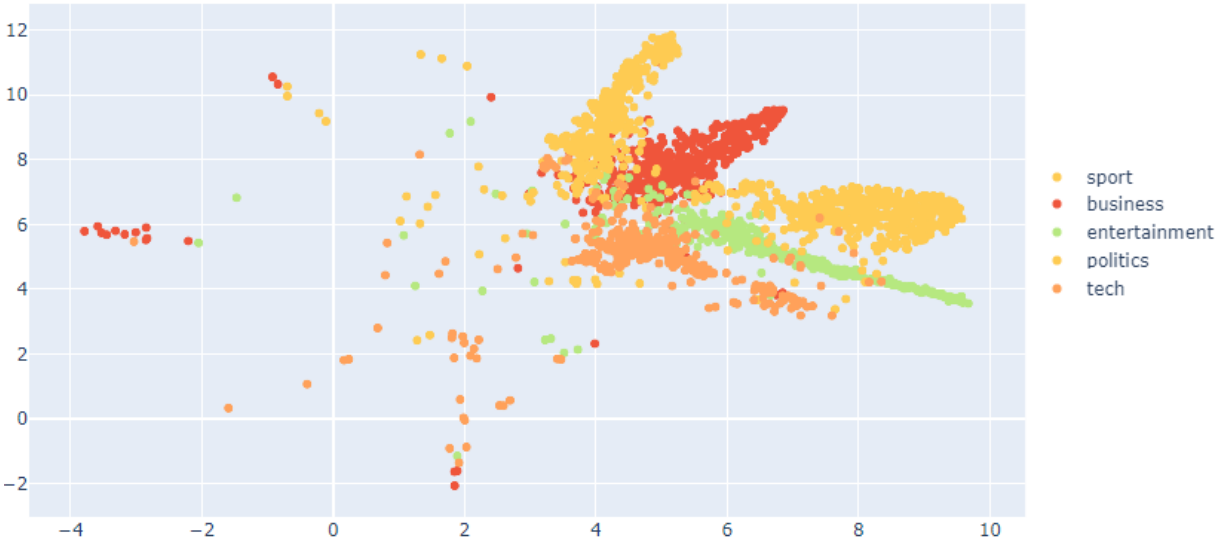
List. 6.3. Faktoryzacja macierzy *TF-IDF* dokumentów metodą Truncated SVD

W niniejszym akapicie zostaną przytoczone wykresy przedstawiające rozkład dokumentów w przestrzeni semantycznej po zastosowaniu techniki Truncated SVD. Wymiary wektorów zostały zredukowane z 25 do 2 techniką *UMAP* w celu przedstawienia rozkładu na płaszczyźnie dwuwymiarowej. Rysunek 6.2. przedstawia rozkład wszystkich dokumentów w przestrzeni semantycznej. Kolory oznaczają kategorię, do której należą dokumenty. Legenda opisująca kategorie zawiera ich fragment w celu zwiększenia czytelności.



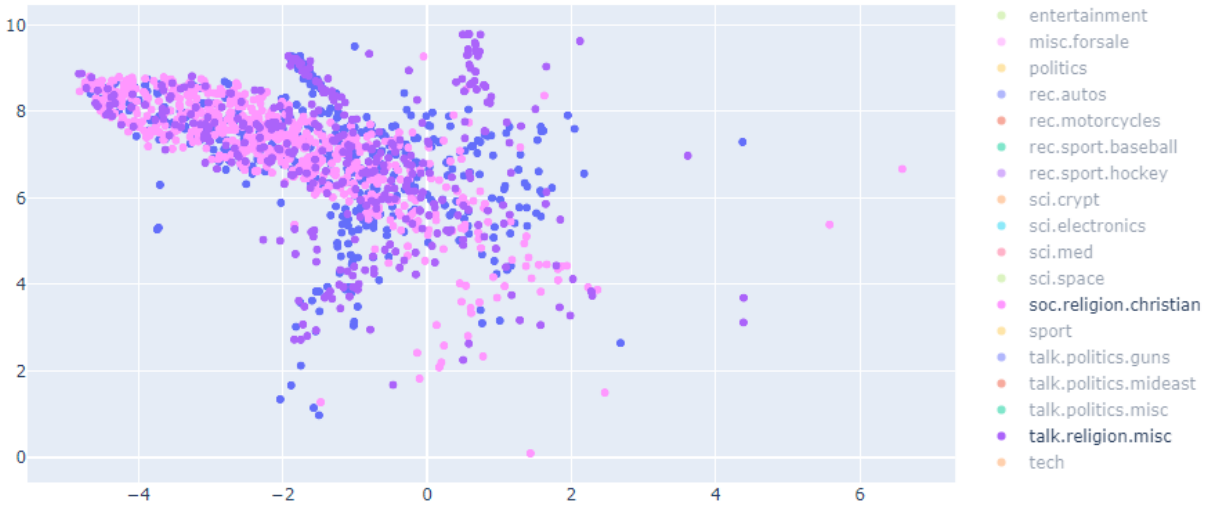
Rys. 6.2. Wizualizacja rozmieszczenia wektorów dokumentów w przestrzeni semantycznej

Rysunek 6.3. przedstawia rozmieszczenie artykułów korpusu *BBC News* w przestrzeni semantycznej. Rozproszenie artykułów jest nieduże. W związku z tym, że teksty pochodzą z tego samego korpusu, towarzyszy im ten sam charakter pisma i używanego słownictwa, a zatem poszczególne kategorie leżą stosunkowo blisko siebie.



Rys. 6.3. Wizualizacja rozmieszczenia wektorów dokumentów korpusu *BBC News* w przestrzeni semantycznej

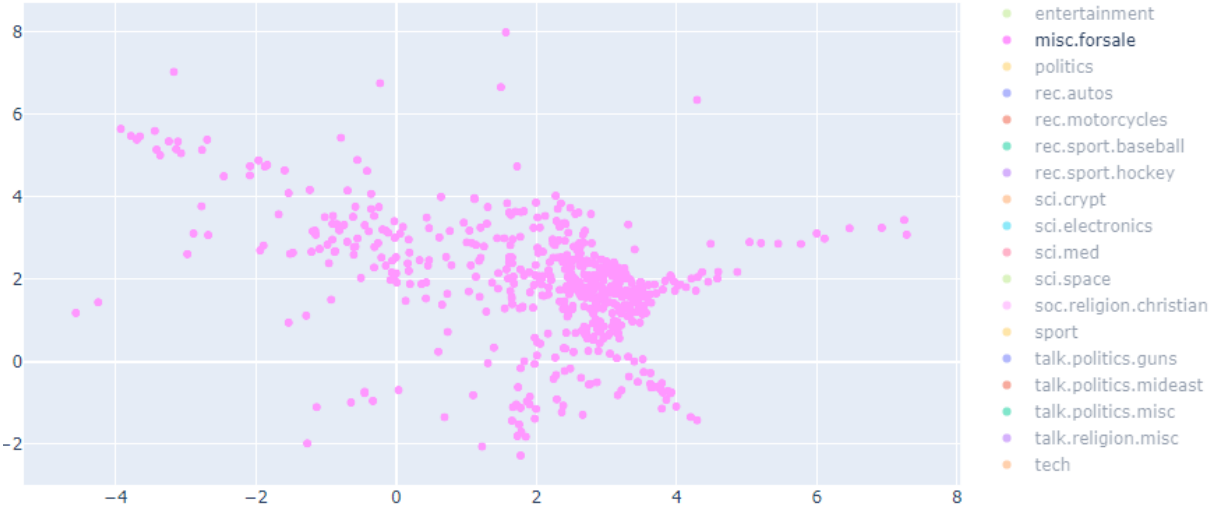
Na rysunkach 6.4. i 6.5. wybrano dokumenty z podobnych kategorii. Rysunek 6.4. przedstawia dokumenty z kategorii religijnych: *soc.religion.christian*, *talk.religion.misc*, *alt.atheism*. Rysunek 6.5. przedstawia dokumenty z kategorii dotyczących sprzętu komputerowego: *comp.sys.ibm.pc.hardware*, *comp.sys.mac.hardware*. Na obu rysunkach można zauważyć, że wybrane kategorie znajdują się bardzo blisko siebie, a wręcz na siebie zachodzą.



Rys. 6.4. Wizualizacja rozmieszczenia wektorów dokumentów tematyki religijnej w przestrzeni semantycznej

Rys. 6.5. Wizualizacja rozmieszczenia wektorów dokumentów kategorii sprzętu komputerowego w przestrzeni semantycznej

Istnieje też kategoria, której dokumenty są silnie rozproszone w przestrzeni semantycznej (rys. 6.6.). Tą kategorią jest *misc.forsale*. Dokumenty, które z niej pochodzą, przedstawiają ogłoszenia dotyczące sprzedaży różnych przedmiotów, m.in: aut, motocykli, sprzętu komputerowego.

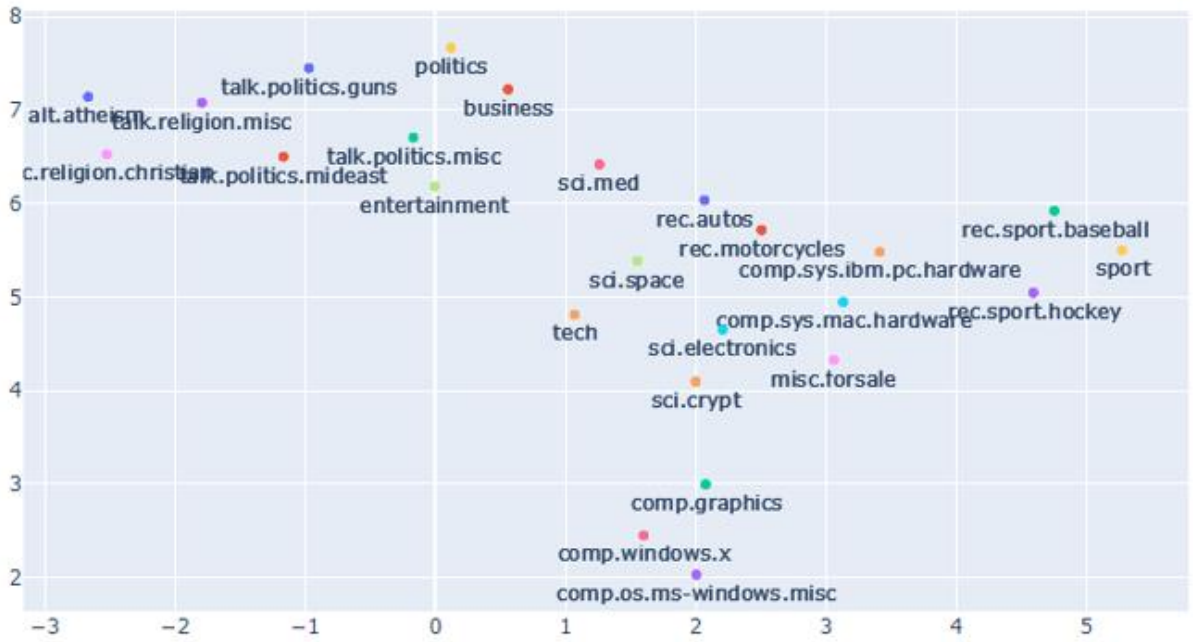


Rys. 6.6. Wizualizacja rozmieszczenia wektorów dokumentów o treści zawierającej ogłoszenia dotyczące sprzedaży w przestrzeni semantycznej

Obliczanie semantycznych wektorów kategorii

Model zawiera również wektory kategorii, które są centroidami (średnią arytmetyczną) wektorów semantycznych dokumentów każdej kategorii. Powodem obliczenia wektorów kategorii było zwiększenie semantycznej trafności podczas wyszukiwania. Wyszukując frazę *„manchester united”* oczekujemy dokumentów o tematyce sportowej. Jednakże stosunkowo blisko wektorów o tematyce związanej z klubem piłkarskim Manchesteru, pojawiają się teksty o tematyce politycznej

związanej z tym miastem. Wektor kategorii można wykorzystać w obliczeniach podobieństwa semantycznego zapytania użytkownika i dokumentów. W ostatniej sekcji niniejszego punktu dokumentacji zostanie opisane szczegółowe użycie wektorów kategorii, w celu zwiększenia trafności wyszukiwania znaczeniowego. Na rysunku 6.7. zamieszczono wykres z rozmieszczeniem wektorów kategorii w przestrzeni semantycznej. Kategorie o podobnej tematyce, np. *rec.sport.baseball*, *sport*, *rec.sport.hockey* znajdują się blisko siebie.



Rys. 6.7. Wizualizacja rozmieszczenia wektorów kategorii w przestrzeni semantycznej

Autokorekta zapytań

W przypadku błędów w pisowni zapytania, system spróbuje poprawić błędy, znajdując wyrazy najbliższe wyrazom z błędami w kontekście odległości Damerau-Levenshteina. Do stworzenia słownika, z którego pobierane będą wyrazy do autokorekty, użyto tych samych kolumn, których użyto w trakcie wektoryzacji *TF-IDF* tytułów i treści (list 6.4.). Klasa *CountVectorizer* oblicza częstotliwość występowania wyrazów w korpusie.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vectorizer = CountVectorizer(tokenizer=identity_tokenizer, lowercase=False)
count_vectorizer_transform = count_vectorizer.fit(articles_df['Cleaned title'] + ' ' +
                                                  articles_df['Cleaned title lemmatized'] + ' ' +
                                                  articles_df['Cleaned body'] + ' ' +
                                                  articles_df['Cleaned body lemmatized'])
```

List. 6.4. Utworzenie słownika częstotliwości wyrazów w korpusie

Słownik częstotliwości wyrazów został zapisany do pliku tekstowego, w którym w każdej linii znajduje się wyraz wraz z liczbą jego wystąpień w korpusie (list. 6.5.). Fragment zawartości pliku widnieje na rysunku 6.8.

```
f = open("vocab.txt", "w")
for key in count_vectorizer_transform.vocabulary_:
    try:
        string = str(key) + ' ' + str(count_vectorizer_transform.vocabulary_[key]) + '\n'
        f.write(string)
    except UnicodeEncodeError:
        continue
f.close()
```

List. 6.5. Zapisanie słownika częstotliwości wyrazów do pliku

```
ad 16476
sales 85990
boost 24312
time 96203
warner 103254
```

Rys. 6.8. Fragment pliku słownika

Do poprawiania błędów wykorzystano bibliotekę *SymSpell*. Biblioteka wykorzystuje stworzony słownik częstotliwości wyrazów. Po podaniu zdania z błędami, biblioteka oblicza odległość Levensthtein'a pomiędzy wyrazami zapytania, a wyrazami ze słownika. Wybierana jest para wyrazów o najmniejszej odległości. W ten sposób system proponuje użytkownikowi nowe, poprawione zapytanie. Przykładowo, dla zapytania „manheszter junaited" system poprawia wyrazy otrzymując nowe zapytanie – „manchester united" – listingi 6.6., 6.7., 6.8.

```
input_term = ("manheszter junaited")
suggestions = sym_spell.lookup_compound(input_term, max_edit_distance=3,
                                        transfer_casing=True)
for suggestion in suggestions:
    print(suggestion)
```

```
manchester united, 4, 0
```

List. 6.6. Korekcja zapytania „manheszter junaited"

```
suggestions = sym_spell.lookup('manheszter', Verbosity.CLOSEST,
                               max_edit_distance=2)
for suggestion in suggestions:
    print(suggestion)
```

```
manchester, 2, 89075
```

List. 6.7. Korekcja słowa „manheszter"

```
suggestions = sym_spell.lookup('junaited', Verbosity.CLOSEST,
                               max_edit_distance=2)
for suggestion in suggestions:
    print(suggestion)
```

```
united, 2, 139266
unaided, 2, 138512
```

List. 6.8. Korekcja słowa „junaited"

Tworzenie rankingu dokumentów

Posiadając model dokumentów, API jest w stanie obliczyć ranking dokumentów w odpowiedzi na zapytanie użytkownika. Zapytanie jest poddawane takiemu samemu procesowi przetwarzania jak dokumenty. Po wyczyszczeniu zapytania ze znaków specjalnych, jest ono rozszerzane o formę zlematyzowaną. W takiej postaci zapytanie jest transformowane do wektora *TF-IDF* i porównywane z wektorami tytułów i treści dokumentów. W ten sposób z zapytania „I was trying to catch a bus!!!!" powstanie zapytanie „I was trying to catch a bus try". Dzięki rozszerzaniu o formę niezlematyzowaną, wyszukiwarka jest w stanie umieszczać dokumenty ze słowem „trying" wyżej od dokumentów ze słowem „try".

Następnym etapem jest przygotowanie wektora semantycznego. Wektory semantyczne poszczególnych wyrazów zapytania zostają zsumowane (korzystając z macierzy „right singular vectors") tworząc wektor semantyczny całego zapytania. Taki wektor jest używany do obliczenia podobieństwa kosinusowego między zapytaniem, a dokumentami. Końcowe podobieństwo semantyczne między dokumentem, a zapytaniem, to średnia arytmetyczna podobieństw kosinusowych zapytania i dokumentu oraz zapytania i wektora semantycznego kategorii tego dokumentu (semantycznego centroidu tejże kategorii):

$$S = \frac{cosine_{similarity(Q,D)} + cosine_{similarity(Q,D_C)}}{2}$$

Gdzie:

Q – wektor semantyczny zapytania,

D – wektor semantyczny dokumentu,

D_C – wektor semantyczny kategorii dokumentu.

Ostateczne podobieństwo zapytania i dokumentu jest obliczanie na podstawie następującego wzoru:

$$Score = \frac{S + cosine_{similarity(Q_{TFIDF},D_{S-TFIDF})} + cosine_{similarity(Q_{TFIDF},D_{B-TFIDF})} \cdot S}{3}$$

Gdzie:

S – podobieństwo semantyczne zapytania i dokumentu,

Q_{TFIDF} – wektor *TF-IDF* zapytania

$D_{S-TFIDF}$ – wektor *TF-IDF* tytułu dokumentu

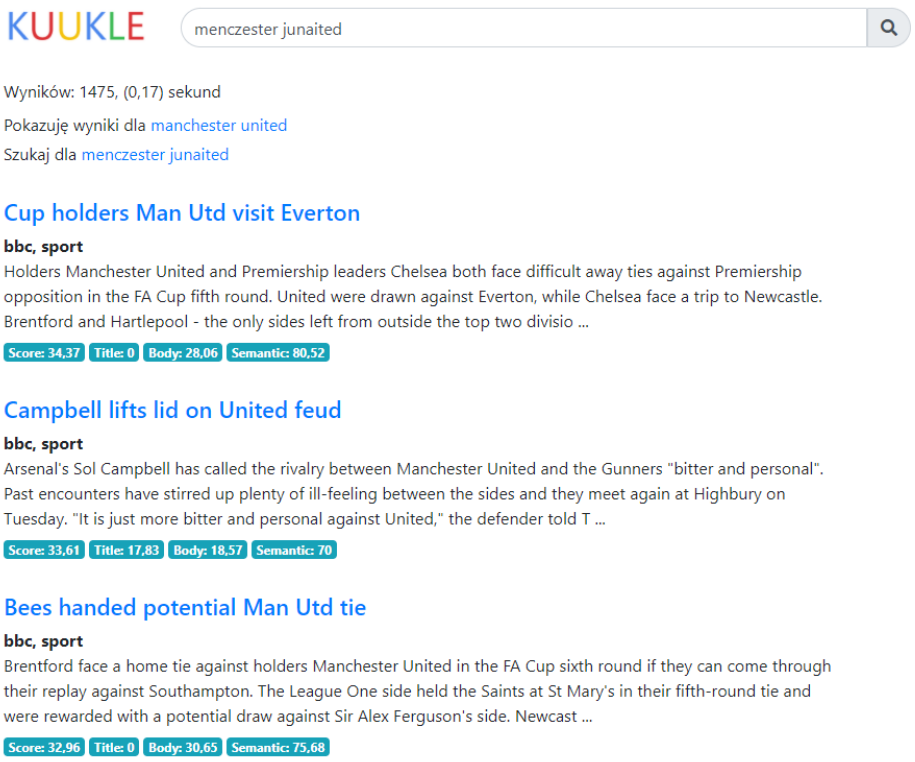
$D_{B-TFIDF}$ – wektor *TF-IDF* treści dokumentu

Podobieństwo kosinusowe zapytania a wektora treści jest ważone podobieństwem semantycznym. Treść dokumentu powinna mieć mniejsze znaczenie w przypadku odległych tematyk zapytania i dokumentu.

7. Opis działania aplikacji

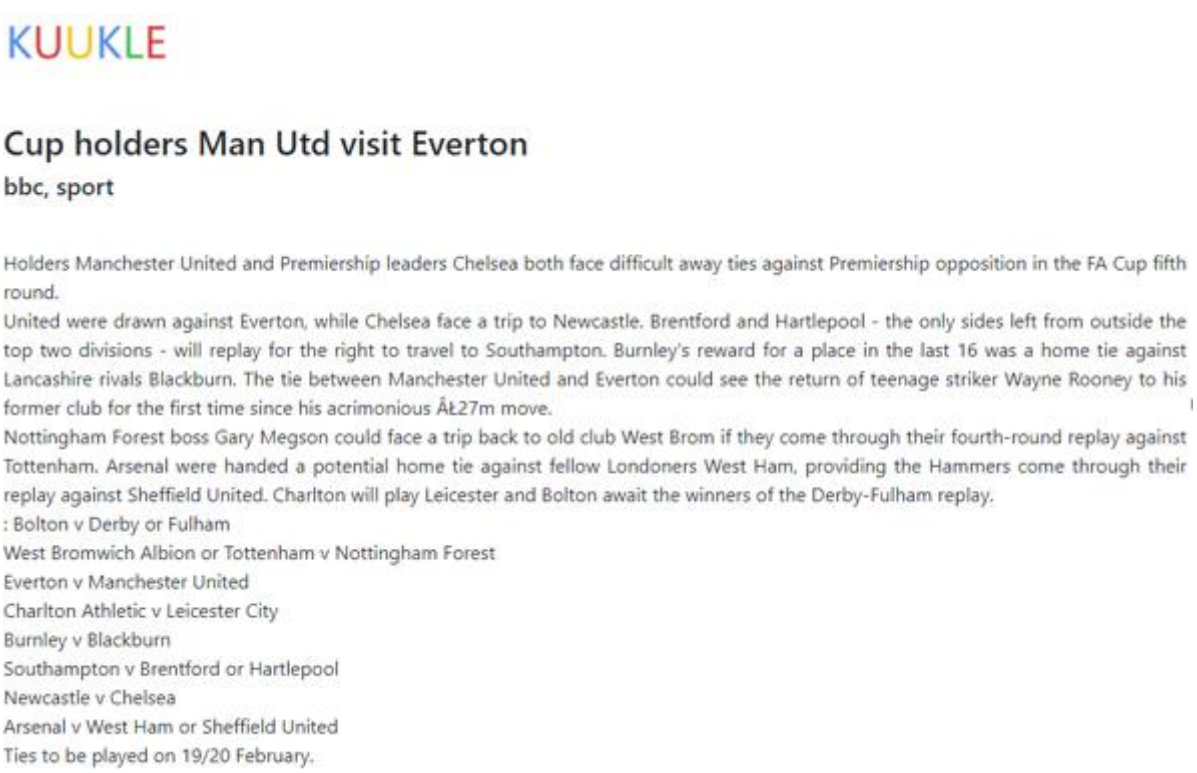
Stronę główną aplikacji stanowi komponent tekstowy umożliwiający podanie frazy do wyszukania w systemie (rys. 7.1.). Po naciśnięciu przycisku enter, bądź kliknięciu myszką przycisku „Szukaj" system zwróci wynik wyszukiwania, które dla frazy „menczester junaited" są widoczne na rysunku 7.2. W wynikach widoczne są tytuły, kategorie, źródła i fragmenty treści dokumentów, pod którymi znajdują się podobieństwa między zapytaniem, a tytułem, treścią i semantyką dokumentu, wyrażone w procentach. Zapytanie „menczester junaited" nie występuje w słowniku systemu, dlatego system pokazuje wyniki wyszukiwania dla frazy „manchester united", która według niego jest poprawna.

Rys. 7.1. Strona główna aplikacji – pole tekstowe umożliwiające wprowadzenia zapytania



Rys. 7.2. Wynik wyszukiwania frazy „menczester junaited"

Po kliknięciu w tytuł dokumentu użytkownik zostanie przekierowany do strony przedstawiającej treść dokumentu (rys. 7.3.).



Rys. 7.3. Strona przedstawiająca treść dokumentu