WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

Albert Gawin

Inteligentna wyszukiwarka narzędzi bioinformatycznych, stworzona w oparciu o algorytmy klasy NLP.

Projekt inżynierski

Opiekun pracy:
dr Michał Piętal

Rzeszów, 2024

Spis treści

1. Wstęp 4
1.1 Cel i zakres projektu 5
1.2 Znaczenie NLP w bioinformatyce 5
2. Przygotowanie danych 6
2.1 Proces pobierania danych 7
2.2 Proces wyodrębniania danych 8
2.3 Czyszczenie i przetwarzanie danych 8
2.4 Zapisanie wyczyszczonych danych 9
3. Rozwój modelu uczenia maszynowego 10
3.1 Wybór modelu 10
3.2 Wczytanie danych 12
3.3 Oznaczenie dokumentów 13
3.4 Trenowanie modeli 14
3.5 Walidacja i ocena modelu 16
4. Tworzenie strony internetowej 19
4.1 Projektowanie interfejsu użytkownika 19
4.2 Integracja modelu NLP 20
5. Terminarz prac 23
5.1 Tworzenie strony internetowej 23
5.2 Uczenie modelu 24
6. Analiza wyników 25
6.1 Interpretacja wyników 25
6.2 Ograniczenia i potencjalne ulepszenia 25
7. Podsumowanie i wnioski 27
7.1 Główne osiągnięcia projektu 27
7.2 Wnioski końcowe 28
Literatura 29

1. Wstęp

W dzisiejszych czasach, dostęp do informacji jest ważny dla postępu w dziedzinie biologii i bioinformatyki oraz pozostałych nauk. Istnieje kilka rozwiązań wyszukujących artykuły naukowe z tych dziedzin, w tym jedna z największych tj. PubMed[1]. Jednakże, pomimo że strona ta posiada największą bazę danych artykułów, to istnieje zapotrzebowanie na rozwiązanie pozwalające na wyszukiwanie wyłącznie narzędzi bioinformatycznych, a takiej opcji strona PubMed nie posiada.

Projekt ten został stworzony w celu rozwiązania wyżej opisanego problemu oraz zaliczenia przedmiotu Projekt Inżynierski na studiach Inżynieria i Analiza Danych. Model uczenia maszynowego oparty został na technikach przetwarzania języka naturalnego, inaczej NLP[2] (ang: Natural Language Processing). Dzięki takiej technice uczenia, możliwym było wyodrębnienie sensu oraz korpusu każdego z artykułów.

Następnie, wykorzystując przygotowane rozwiązania oraz obliczoną dokładność każdego z artykułów, wyszukiwarka w formie strony internetowej tworzy ranking, który wyświetlany jest w formie listy, aby użytkownik dostał przejrzyste oraz łatwe w obsłudze rozwiązanie. Forma taka pozwala na przysłowiowe jedno kliknięcie, aby dostać się do interesującego użytkownika artykułu. Dzięki temu poszukiwanie narzędzi bioinformatycznych stało się szybsze, prostsze, a sam projekt może posłużyć jako solidna podstawa do dalszych prac.

Całość wykorzystanego kodu można znaleźć w poniższym linku:

https://github.com/albertgawin-al/projekt_inzynierski

1.1. Cel i zakres projektu

PubMed to największa strona internetowa gromadząca artykuły biotechnologiczne oraz medyczne. Strona ta zawiera wbudowaną wyszukiwarkę, która niestety nie posiada wyszukiwania artykułów tylko i wyłącznie o narzędziach bioinformatycznych. Aktualna wersja pozwala na wyszukiwanie zaawansowane, po autorze, typach artykułów, dacie i tym podobnych, lecz nie posiada opcji wyszukiwania tylko i wyłącznie artykułów o narzędziach bioinformatycznych.

Dlatego też, niniejszy projekt ma na celu ułatwienie wyszukiwania o nich informacji, poprzez specjalnie do tego przygotowanej strony internetowej, wykorzystującej model uczenia maszynowego, wytrenowany za pomocą techniki NLP.

1.2. Znaczenie NLP w bioinformatyce

Wyszukiwanie narzędzi bioinformatycznych może zostać uproszczone poprzez wykorzystanie algorytmów NLP, które wykorzystane na stronie internetowej pozwalają użytkownikowi na łatwe wyszukiwanie narzędzi. Algorytmy NLP wzbogacają funkcjonalność wyszukiwarki poprzez umożliwienie użytkownikom zadawania bardziej złożonych pytań. Za pomocą tej techniki możliwym jest wyodrębnienie znaczenia całego tekstu artykułu oraz znalezienie tylko tych tekstów, które opisują narzędzia bioinformatyczne.

Projekt ten jest rozwinięciem wyszukiwarki PubMed, poprzez dodanie opcji wyszukiwania tylko i wyłącznie narzędzi bioinformatycznych, ale również przykładem efektywnego wykorzystania informatyki w dziedzinie biotechnologii i medycyny.

Stosując tak zaawansowane algorytmy wyciągania kontekstu z dokumentów, wyszukiwarka jest w stanie pokazać użytkownikowi artykuły, które nie posiadają żadnego z wpisanych słów, lecz całość tekstu najlepiej opisuje informację, której badacz poszukiwał. Dlatego też, wyszukiwarka ta pomoże nie tylko, na szukanie informacji o narzędziach ale również artykułów do nich zbliżonych, co może pozytywnie wpłynąć na wyniki badań.

2. Przygotowanie Danych

Przygotowanie danych stanowi fundament w dalszych punktach uczenia maszynowego, dlatego też jego wpływ ma duże znaczenie na całość projektu. Dane wykorzystane w projekcie, pobrane zostały z bazy danych PubMed Annual baseline[3]. Strona ta, zawiera listę ponad tysiąca skompresowanych plików zawierających od kilkunastu do kilkuset tysięcy artykułów bioinformatycznych oraz medycznych.

Niestety pobieranie ręczne było niemożliwe, ponieważ jeden taki folder posiada około 30MB (trzydzieści megabajtów), co przekłada się na długotrwały proces pobierania. Dlatego też, aby rozwiązać ten problem stworzony został skrypt, który ma na celu zautomatyzowanie procesu pobierania i rozpakowywania danych.

Foldery, które nie posiadały ani jednego artykułu z przedziału 2020-2023 zostały automatycznie usunięte. Takie rozwiązanie posłużyło do przygotowania modelu, który działać miał tylko i wyłącznie na najnowszych danych. Na listingu 2.1 przedstawiono proces przygotowywania danych, obejmujący pobieranie folderów, następnie ich rozpakowywania oraz usuwania folderów nie zawierających artykułów ze wcześniej wspomnianego przedziału czasowego.

```
for i in range(start, end):
    print(f"[{i:04d}/1166]")
    downloadFolder(i)
    extractFolder(i)

    if checkIfYear(i, 'DateRevised', 'Year') == False:
        removeFolder(i)

    removeFile(i)
```

Listing 2.1. Fragment kodu przygotowującego dane

2.1. Proces pobierania danych

W celu pobrania tysiąca stu sześćdziesięciu sześciu folderów zawierających artykuły naukowe, stworzono funkcje *downloadFolder* pokazaną na listingu 2.2. Korzystając z faktu, że wszystkie foldery znajdują się na jednej stronie, utworzono zmienną trzymającą dokładny adres dla podanego numeru folderu (index).

Aby uniknąć powtórzeń zastosowano warunek sprawdzający, czy pobierany folder nie istnieje już we wcześniej przygotowanym folderze zbiorczym. Funkcja ta, posiada również wbudowane zabezpieczenia, które pozwalają na pobieranie danych bez zakłóceń, a w razie wystąpienia błędu, na wyświetlenie informacji o jego szczegółach.

```python
def downloadFolder(index):
    url = f"https://ftp.ncbi.nlm.nih.gov/pubmed/baseline/pubmed23n{index:04d}.xml.gz"
    filename = url.split("/")[-1]
    dataPath = "./data/" + filename[:-3]
    compressedPath = "./compressedData/" + filename

    try:
        print("    Downloading folder", filename)
        if os.path.exists(compressedPath) == False and os.path.exists(dataPath) == False:
            with open(compressedPath, "wb") as f:
                r = requests.get(url)
                if (r.status_code == 200):
                    f.write(r.content)
            f.close()
```

Listing. 2.2. Funkcja pobierająca folder z artykułami

2.2. Proces wyodrębniania danych

Wcześniejsza faza polegała na pobieraniu skompresowanych folderów, dlatego też zaistniała potrzeba stworzenia rozwiązania, które wyodrębniałoby te dane oraz zapisywało do specjalnie przygotowanego folderu zbiorczego. Aby sprostać temu problemowi, stworzona została funkcja *extractFolder* przedstawiona na listingu 2.3.

```
def extractFolder(index):
    filename = f"pubmed23n{index:04d}.xml.gz"
    dataPath = "./data/" + filename[:-3]
    compressedPath = "./compressedData/" + filename

    try:
        print("    Extracting folder", filename)
        if os.path.exists(dataPath) == False:
            with gzip.open(compressedPath, 'rb') as f_in:
                with open(dataPath, 'wb') as f_out:
                    shutil.copyfileobj(f_in, f_out)
            f_in.close()
            f_out.close()
```

Listing. 2.3. Funkcja wyodrębniająca folder z artykułami

2.3. Czyszczenie i przetwarzanie danych

Czyszczenie danych stanowi kluczowy etap przygotowania do tworzenia modelu uczenia maszynowego. W celu ułatwienia trenowania modelu, zastosowano procesy czyszczenia, takie jak:

1) Zmniejszanie liter - funkcja, która konwertuje wszystkie duże litery na małe, co ujednolica format tekstu.
2) Usuwanie łamań linii - proces usuwania łamań linii z tekstu w celu poprawy spójności danych.
3) Usuwanie punktacji - jest to usuwanie znaków takich jak: przecinek, kropka, dwukropek itp.

4) Usuwanie „stop words” – redukcja ilości słów poprzez usunięcie tzw. „stop words” – słowa, które nie niosą za sobą istotnej informacji z perspektywy modelu. Mogą to być słowa typu: i, oraz, lub, mp3.
5) Usuwanie cyfr – eliminacja cyfr z tekstu
6) Specjalna lematyzacja – zastosowano lematyzację przy użyciu biblioteki NLP w celu sprowadzenia słów do ich podstawowych form. Na przykład słowa: psu, psom, psy zamieniane jest na jedno słowo tj. pies.

2.4. Zapisanie wyczyszczonych danych

Proces zapisywania i odczytywania wyczyszczonych danych został skonfigurowany w sposób umożliwiający efektywne operowanie nimi w późniejszych etapach projektu. Wczytane dane poddawane są procesowi usuwania pustych wierszy, następnie zapisywane są wyłącznie te artykuły, które posiadają informację o narzędziach bioinformatycznych. Fragment tych operacji pokazano na listingu 2.4. Takie ograniczenie danych pozwoli na wyszkolenie modelu wyłącznie na artykułach dotyczących tematu tego projektu.

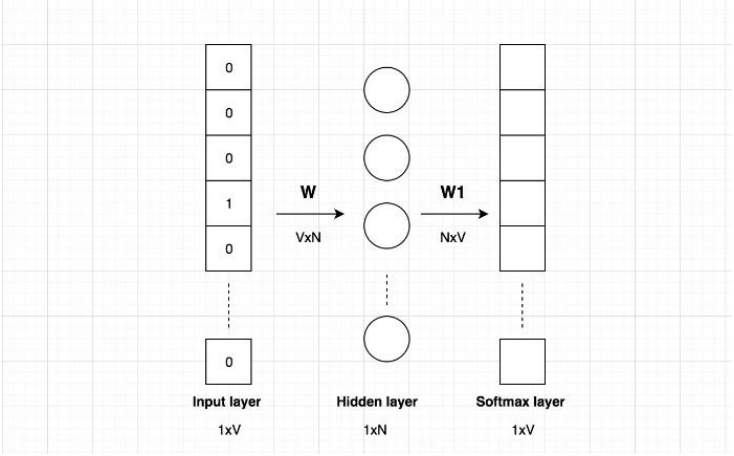
```
toolArticles['abstract'].progress_apply(lambda x: clean_string(x))
```

Listing. 2.4. Czyszczenie każdego dokumentu

3. Rozwój modelu uczenia maszynowego

3.1. Wybór modelu[4]

Word2Vec[5] jest to technika uczenia maszynowego polegająca na zamianie słów na wektory liczbowe. Za pomocą takich wektorów model jest w stanie znaleźć słowa, które najlepiej oddają charakter, są bardzo zbliżone lub tożsame z wejściem użytkownika. Poprzez zamianę słowa na wektor oraz wykorzystanie go w modelu, który na podstawie wcześniejszych słów, stworzył nowy wektor, który lepiej opisuje dane słowo, co pokazano na rysunku 3.1. Proces ten jest powtarzany dla każdego ze słów, aż model nie zostanie wytrenowany.



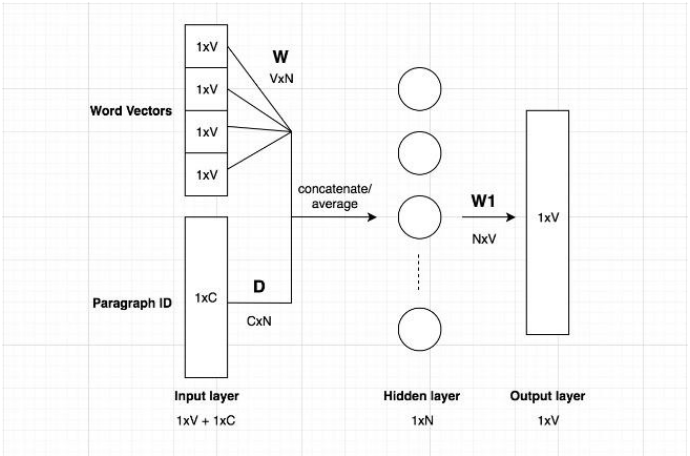
Rys. 3.1. Graficzne przedstawienie Word2Vec

Używając funkcji matematycznej cosine similarity[6] (3.1) (podobieństwo cosinusowe), technika ta oblicza poziom podobieństwa semantycznego. Limitem tej techniki jest fakt, że wyciąga ona znaczenie każdego słowa z osobna, co nie ma dobrego zastosowania dla wyszukiwania artykułów po znaczeniu całego tekstu.

$$cosine\ similarity = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.1}$$

gdzie: A_i, B_i – to i-te składowe wektorów A i B

Doc2Vec[7] jest to technika uczenia maszynowego polegająca na przedstawieniu całego dokumentu jako wektora liczbowego. Dzięki takiemu podejściu technika ta posiada przewagę nad Word2Vec, ponieważ jest w stanie wyciągać podobieństwa semantyczne dla całego tekstu artykułu, a nie dla pojedynczych słów. Poprzez zmianę wszystkich słów w dokumencie na wektory, a następnie połączenie ich za pomocą specjalnego wektora opisującego całość dokumentu, tworzony jest główny wektor opisujący całość tekstu artykułu. Następnie model uczy się na podstawie takiego wektora, co jest powtarzane dla wszystkich dokumentów.



Rys. 3.2. Graficzne przedstawienie Doc2Vec

Dlatego też, do rozwinięcia modelu uczenia maszynowego w projekcie zdecydowano się na zastosowanie algorytmu Doc2Vec. W odróżnieniu od Word2Vec, który reprezentuje poszczególne słowa jako wektory, Doc2Vec umożliwia wydobycie wniosków na podstawie całego wczytanego tekstu. Dla każdego dokumentu tworzony jest wektor liczbowy, który następnie może zostać porównany do wektora tekstu wprowadzonego przez użytkownika. Doc2Vec sprawdza się szczególnie dobrze w przypadku długich tekstów, umożliwiając wyciąganie kontekstu z całego dokumentu.

3.2. Wczytanie danych

Proces wczytywania jest ważnym elementem projektu, ponieważ niepoprawne zapisanie oraz odczytanie danych może doprowadzić do nieprzewidzianych lub niepoprawnych wyników. Dlatego też, używając bibliotek takich jak *os (interakcja z systemem operacyjnym)* oraz *pandas (analiza oraz manipulacja danymi)* wczytano ścieżki wszystkich plików w folderze z wyczyszczonymi danymi. Użycie gotowych narzędzi umożliwiło poprawne odczytanie oraz zapisanie każdego z plików do wcześniej przygotowanej ramki danych. Całą funkcję wykonującą powyżej opisane operacje przedstawiono na listingu 3.3.

```
def getCleanData():
    clean_paths = os.listdir('new_clean_data')
    clean_data = pd.DataFrame()

    for path in tqdm(clean_paths, desc='Reading Data'):
        clean_articles = pd.read_csv(f'new_clean_data/{path}', sep=',')
        clean_data = pd.concat([clean_data, clean_articles], ignore_index=True)

    return clean_data
```

Listing. 3.3. Funkcja wczytująca czyste dane

Użyto również biblioteki tqdm, która to pozwala na zobrazowanie wykonywanego polecenia. Zaobserwować to będzie można w dalszych częściach kodu. Takie rozwiązanie ułatwiło pilnowanie procesu wczytywania danych, co pokazano na rysunku 3.4.

Rys. 3.4. Proces wczytywania danych

3.3. Oznaczenie dokumentów

Jest to proces służący do przygotowania danych wykorzystanych podczas trenowania modelu. Polega on na reprezentacji, każdego z dokumentów jako unikalny tag (identyfikator). Dzięki takiemu zabiegowi, każdy tekst posiada swój unikalny identyfikator co jest wymagane przez technikę Doc2Vec, jest to inaczej mówiąc przygotowanie artykułów do użycia ich w trenowaniu modelu.

Funkcja *TaggedDocument* z biblioteki gensim[8], przedstawiona na listingu 3.5, oznacza każdy dokument, we wcześniej opisany sposób. Wyczyszczone dane zostały podzielone na dwie kolumny, pierwsza z nich „clean_abstract" jest to wyczyszczony tekst artykułu. Natomiast druga kolumna „pmid" jest to specjalny identyfikator strony PubMed stworzony dla każdego artykułu.

```
TaggedDocument(str(article["clean_abstract"]).split(), tags=[str(article["pmid"])])
```

Listing. 3.5. Użycie gotowej funkcji do oznaczania dokumentów

Aby zilustrować wczytywanie tych danych użyto biblioteki tqdm, co pozwoliło na animację wykonywania programu przedstawionego na listingu 3.6. Funkcja *getTaggedDocuments* wczytywała oraz animowała wyżej opisane procesy, na końcu zwracając poprawnie oznaczone dokumenty do dalszej pracy.

```
def getTaggedDocuments(clean_data):
    tqdm.pandas(desc="Tagging documents")

    def process_article(article):

        return TaggedDocument(str(article["clean_abstract"]).split(

    documents = clean_data.progress_apply(process_article, axis=1)

    return documents
```

Listing. 3.6. Użycie gotowej funkcji do oznaczania dokumentów

3.4. Trenowanie modeli

Wyczyszczone oraz oznaczone dokumenty wykorzystano do trenowania modeli, które posłużyły do wybrania najlepszego z nich. Poprzez przygotowanie listy prawdopodobnych ustawień dla każdego z hiperparametrów przedstawioną na listingu 3.5, łączna ilość modeli do wytrenowania wynosiła sześćdziesiąt cztery.

```
vector_size_list = [200, 300]
min_count_list = [20, 25]
window_list = [10, 12]
sample_list = [5e-5, 1e-4]
alpha_list = [0.03, 0.035]
min_alpha_list = [0.00001, 0.000001]
```

Listing. 3.5. Listy hiperparametrów użytych do znalezienia najlepszego modelu

Każdy z hiperparametrów różni się zastosowaniem, dlatego poniżej krótko opisano co powoduje zmiana każdego z nich:

1) *vector_size* - rozmiar wektorów, których model uczy się dla każdego dokumentu. Większa liczba pozwoli modelowi nauczyć się bardziej złożonych tekstów, natomiast grozi to przetrenowaniem.

2) *min_count* – minimalna liczba wystąpień słowa w całym zbiorze danych, aby zostało ono uwzględnione podczas trenowania

3) *window* – maksymalna odległość między głównym słowem a słowami wokół niego. Większe okno uchwyci więcej relacji, ale spowolni trening modelu.

4) *sample* – próg konfigurujący, które słowa o wyższej ilości wystąpień są losowo zaniżane.

5) *alpha* – początkowa szybkość uczenia modelu

6) *min_alpha* – jest to minimalna wartość *alpha*. Jako, że podczas trenowania *alpha* spada liniowo, to jej dolną granicą jest właśnie ta wartość.

Za pomocą kodu z listingu 3.6 rozpoczęto uczenie modeli. Cały proces zajął siedemset osiemdziesiąt cztery minuty, co przełożyło się na sześćdziesiąt cztery modele gotowe do sprawdzenia i oceny. W poniższym kodzie zastosowano technikę uczenia poprzez podzielenie dużego zbioru na mniejsze fragmenty, inaczej nazywaną „batch learning". Dzięki takiemu rozwiązaniu pamięć laptopa nie została przepełniona, a samo trenowanie mogło przebiegać bez problemów.

```
batch_size = 1000
num_batches = len(documents) // batch_size + 1

for i in tqdm(range(num_batches), desc='Training'):
    batch = documents[i*batch_size:(i+1)*batch_size]
    model.train(batch, total_examples=len(batch), epochs=80)
```

Listing. 3.6. Fragment kodu uruchamiający uczenie modelu dla wybranych hiperparametrów

Wytrenowanie tylu modeli pozwoliło na ręczne sprawdzenie każdego z nich oraz ocenę ich skuteczności. Była to faza, która pomogła w znalezieniu najlepszego modelu do wykorzystania w wyszukiwarce narzędzi bioinformatycznych.

3.5. Walidacja i ocena modelu

W celu oceny jakości modelu przeprowadzono eksperymenty z różnymi zestawami hiperparametrów. Wygenerowano sześćdziesiąt cztery modele, z których każdy został wytrenowany na pięćdziesięciu losowych plikach z artykułami. Następnie, oceniono jakość zwracanych artykułów poprzez poszczególne modele oraz sporządzono tabelę porównawczą.

Ocena „1” oznaczała poprawny zwrot artykułów dla każdego z zapytań, „0,5” oznaczało, że model zwracał odpowiednie artykuły lecz nie dla wszystkich zapytań, natomiast ocena „0” oznaczała brak jakichkolwiek poprawnych odpowiedzi udzielonych przez model. Fragment pliku z ocenami przedstawiono na rysunku 3.7.

L.n.	vector_size	mincount	window	sample	alpha	min_alpha	best
1	200	20	10	5,00E-05	0.03	0.00001	1
2	200	20	10	5,00E-05	0.03	0.000001	1
3	200	20	10	5,00E-05	0.035	0.00001	0,5
4	200	20	10	5,00E-05	0.035	0.000001	0,5
5	200	20	10	1,00E-04	0.03	0.00001	1
6	200	20	10	1,00E-04	0.03	0.000001	0

Rys. 3.7. Ocena poprawności modeli

W oparciu o oceny modeli stworzono ranking każdego z hiperparametrów. Celem takiego rozwiązania było znalezienie najbardziej wpływowych hiperparametrów podczas uczenia modelu. Poprzez zliczanie ilości ocen dla każdej z opcji, opracowano działanie matematyczne (3.2).

$$punkty_{hiper\ parametr} = \sum ocena * ilość\ ocen_{ocena} \tag{3.2}$$

gdzie: $ocena$ – jedna z trzech ocen modelu (0, 0.5, 1), $ilośc\ ocen_{ocena}$– ilość uzyskanych ocen przez model dla podanej oceny

Na podstawie obliczonych punktów utworzony został ostateczny ranking pokazany na rysunku 3.8, pomagający w identyfikacji najbardziej wpływowego hiperparametru w kontekście tworzenia modeli wyszukiwania artykułów bioinformatycznych.

		0	0,5	1	punkty	miejsce
vector_size	**200**	12	14	6	13	2
	300	18	10	4	9	11
mincount	**20**	13	13	6	12,5	3
	25	17	11	4	9,5	10
window	**10**	16	9	7	11,5	4
	12	14	15	3	10,5	8
sample	**5,00E-05**	14	13	5	11,5	4
	1,00E-04	16	11	5	10,5	8
alpha	**0.03**	12	12	8	14	1
	0.035	18	12	2	8	12
min_alpha	**0.00001**	17	8	7	11	6
	0.000001	13	16	3	11	6

Rys. 3.8. Ranking hiperparametrów

Używając wcześniej obliczonych punktów utworzono działanie (3.3), na podstawie którego wyliczono ranking każdego z modeli. Działanie to sumowało punkty użytych hiperparametrów modelu, a następnie przemnożyło otrzymany wynik przez ocenę modelu. Dzięki takiego zabiegowi możliwym było stworzenie rankingu, które nakierowało dalszą analizę w poszukiwaniu najlepszego modelu wyszukiwania narzędzi bioinformatycznych.

$$ranga_{model} = ocena_{model} * \sum punkty_{hP} \tag{3.3}$$

gdzie: $ocena_{model}$ – ocena dla danego modelu (0, 0.5, 1), $punkty_{hP}$ – punkty dla danego hiperparametru

Na rysunku 3.9 przedstawiono omówiony ranking, który wspomógł wyodrębnienie najlepszych hiperparametrów.

MODEL SCORE	MODEL RANK
73,5	1
73,5	1
33,75	19
33,75	19
72,5	3
0	35

Rys. 3.9. Ranking modeli

Dzięki wcześniej opisanym procesom znaleziono najlepszy model do wykorzystania w projekcie, przedstawiony na listingu 3.10. Ocenę jego skuteczności przeprowadzono w sposób ręczny, dokładnie analizując wiele zapytań, jakich użytkownik mógłby dokonać podczas używania tej wyszukiwarki.

```
model = Doc2Vec(vector_size=200, min_count=20, window=4, sample=1e-4,
                alpha=0.03, min_alpha=0.00001, workers=cores-1)
```

Listing. 3.10. Finalny model

4. Tworzenie strony internetowej

Aby ułatwić użytkownikom szukanie interesujących ich narzędzi, zbudowano stronę internetową opartą na interfejsie Flask[9]. Za pomocą takiego rozwiązania użytkownik może w łatwy sposób wyszukać artykułów o narzędziach bioinformatycznych. Jest to przyspieszenie, ale również ułatwienie procesu wyszukiwania, co pomoże badaczom na sprawniejsze znalezienie interesujących ich artykułów.

4.1. Projektowanie interfejsu użytkownika

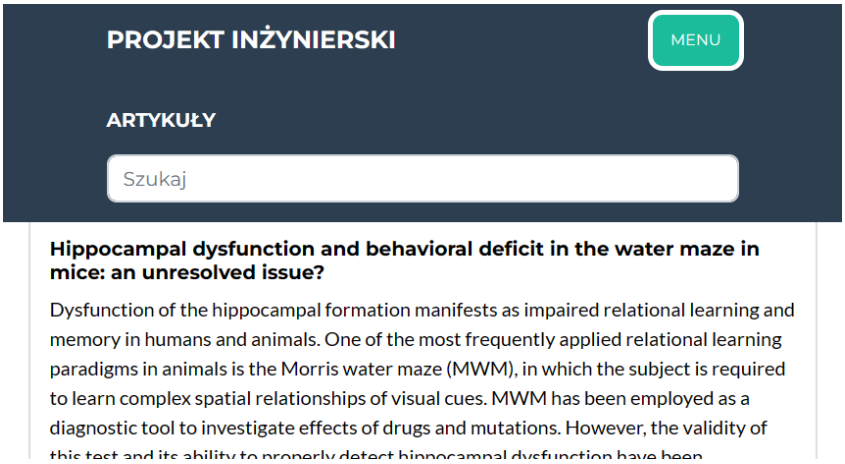
W celu stworzenia interaktywnego środowiska, użyto interfejsu Flask przeznaczonego do tworzenia stron internetowych. Użytkownik posiada możliwość wprowadzenia interesującego go zapytania w celu użycia wytrenowanego modelu do znalezienia odpowiednich artykułów. Dzięki użyciu uczenia maszynowego użytkownik po kilku sekundach otrzymuje listę artykułów posortowanych według oceny porównawczej wektorów opisanej w rozdziale 3.1.

Stworzenie wyglądu strony opierało się głównie na wcześniej wspomnianym interfejsie Flask oraz językach tworzenia stron internetowych takich jak: HTML, CSS oraz JavaScript. Cała strona dzieli się na tak zwane szablony za pomocą, których możliwe było używanie stworzonych stron w wielu miejscach bez powtarzania kodu. Na listingu 4.1 pokazano fragment kodu tworzącego bloki dla każdego z artykułów.

```
{% block content %}
    <div class="articles_container">
        {% for article in articles %}
        <a target="_blank" href="https://pubmed.ncbi.nlm.nih.gov/{{ article.id }}">
            <div class="article card">
                <div class="card-body">
                    <h5 class="article_title card-title">{{ article.title }}</h5>
                    <p class="card-text" style="color: black">{{ article.abstract }}</p>
                </div>
            </div>
        </a>
        {% endfor %}
    </div>
{% endblock %}
```

Listing. 4.1. Fragment kodu tworzący bloki z informacją o artykułach

Strona podzielona jest na część górną tj. fragment z opcją wprowadzenia zapytania oraz rozwijane menu. Dolna część strony zawiera listę bloków z informacjami o artykułach. Każdy z tych bloków posiada tytuł oraz opis danego artykułu, co zostało przedstawione na rysunku 4.2. Użytkownik posiada opcją naciśnięcia danego artykułu, które otwiera nową stronę internetową z oryginalnym źródłem artykułu tj. PubMed.



Rys. 4.2. Interfejs użytkownika

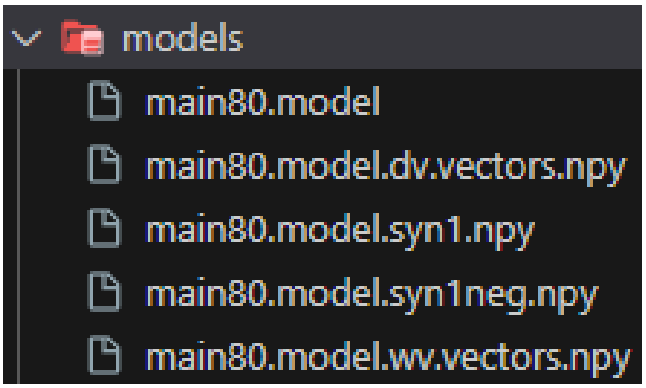
4.2. Integracja modelu NLP

Wczytywanie przygotowanego modelu odbywa się poprzez użycie wbudowanej funkcji *Doc2Vec.load* z biblioteki gensim co zostało pokazane na listingu 4.3.

```
model = Doc2Vec.load("models/main80.model")
```

Listing. 4.3. Wczytanie modelu w celu użycia go na stronie internetowej

Przygotowane modele zapisywane są w folderze „models” pokazanym na rysunku 4.4, dzięki czemu strona posiada dostęp do kilku modeli naraz. Jest to przygotowane w taki sposób, aby ułatwić przyszły rozwój tego projektu. Jeżeli pojawi się opcja rozbudowania danej strony internetowej, to dodanie nowych modeli nie będzie stanowiło problemu.



Rys. 4.4. Folder z modelem

Cały proces przygotowania opiera się na pobieraniu oraz czyszczeniu danych, a następnie wykorzystaniu tych danych do utworzenia modelu. Taki model, a dokładniej jego wektor porównywany jest do wygenerowanego wektora na podstawie wejścia użytkownika, który przeszedł taki sam proces czyszczenia co wszystkie artykuły. Miało to na celu ujednolicenie informacji, aby porównywane wektory operowały na tak samo wyczyszczonych danych, co zostało pokazane na listingu 4.5.

```
userQuery = request.form['search']
query = clean_string(userQuery).split()
```

Listing. 4.5. Czyszczenie wejścia użytkownika

Po utworzeniu wektora wejścia użytkownika, rozpoczęto proces porównywania, który polegał na odczytaniu wyniku oraz zapisania go do wcześniej przygotowanej klasy *PubMedArticle*, fragment tego procesu pokazany został na listingu 4.6.

```
data = eapi.efetch(articleIds)
i = 0
for article_data in data:
    article = PubmedArticle(article_data)
    if article.abstract != "":
        article.weight = articleWeights[i]
        articles.append(article)
    i += 1
```

Listing. 4.6. Porównywanie wektorów

Na jego podstawie został utworzony ranking, który następnie jest wyświetlany na wcześniej przygotowanej i omówionej stronie internetowej. Ranking ten określał jak najlepsze dopasowanie wejścia użytkownika do artykułów, co pokazano na rysunku 4.7.

```
new_vector = model.infer_vector(query)
sims = model.dv.most_similar(positive=[new_vector], topn=100)
```

Listing. 4.7. Porównywanie wektorów

5. Terminarz prac

Rozdział ten podsumowuje tworzenie projektu z podziałem na terminy, aby można było zrozumieć postępujące prace, oraz okres w jakim były one realizowane.

5.1. Tworzenie strony internetowej

1) 03.02.2023, Rozpoczęcie prac – była to pierwsza faza mająca na celu zbadanie wielu środowisk jak Flask, gensim, Python oraz wybór najlepszego rozwiązania w celu dalszych prac. To w tym czasie przeprowadzono pierwsze pisanie kodu, oraz opracowywanie pomysłów.

2) 27.02.2023, Pierwsza strona – moment dodanie pierwszego ekranu na stronie internetowej, cały proces zajął kilka dni, ponieważ nauka a następnie wprowadzenie nabytej wiedzy z zakresu tworzenia stron za pomocą interfejsu Flask wymagała czasu.

3) 07.03.2023, Dodanie klasy PubmedArticle – jest to rodzaj instrukcji, za pomocą której w łatwy sposób zapisywano informacje o artykułach w programie.

4) 24.03.2023, Zliczanie wag artykułów – proces, który obliczał podobieństwo artykułów na podstawie wyuczonego modelu. W tej fazie model był bardzo prosty, głównym celem było utworzenie działającego systemy rankingu danych.

5) 27.03.2023, Ostatnie poprawki – jest to ostatnia faza przygotowywania strony internetowej do dalszych prac. Na tym etapie interfejs użytkownika jak i cała logika wczytywania modeli została zakończona.

5.2. Uczenie modelu

1) 17.03.2023 Pierwsze próby nauki modelu – w tym momencie tworzono najprostszy możliwy model, w celu sprawdzenia jego działania oraz integracji ze stroną internetową.

2) 02.05.2023 Rozpoznanie problemów – okres ten, a dokładniej cały miesiąc poświęcony został na lepsze zrozumienie problemu jakim jest wytrenowanie modelu rozpoznawania tekstu.

3) 25.06.2023 Wdrożenie nabytej wiedzy – faza pobierania oraz czyszczenia wszystkich danych w celu poprawnego trenowania modelu.

4) 08.10.2023 Poprawa projektu – miesiące październik oraz listopad zostały poświęcone na poprawie skryptów odpowiedzialnych za pobieranie oraz czyszczenie danych, ponieważ zauważono wiele błędów, zwłaszcza w czyszczeniu danych.

5) 12.12.2023 Trenowanie wielu modeli – rozpoczęto fazę trenowania sześćdziesięciu czterech modeli w celu znalezienia najlepszego z nich. Dzięki wcześniejszym przygotowaniom, okres ten nie był skomplikowany. Jedynym utrudnieniem był czas jaki został poświęcony na naukę modeli.

6) 09.01.2024 – Dodanie finalnego modelu – po znalezieniu najlepszego modelu, dodano go do wcześniej przygotowanej strony internetowej co zakończyło prace nad projektem inżynierskim.

6. Analiza wyników

6.1. Interpretacja wyników

Wyniki analizy stanowią istotną część projektu, pozwalając na zrozumienie skuteczności modelu oraz jego przydatności w kontekście wyszukiwania artykułów o narzędziach bioinformatycznych. Interpretacja wyników obejmuje analizę ocen jakości artykułów zwracanych przez model wytrenowany techniką Doc2Vec, ocenę trafności wyników w stosunku do oczekiwań użytkowników, oraz ewentualne wnioski dotyczące dalszych działań.

6.2. Ograniczenia i potencjalne ulepszenia

W trakcie analizy wyników istotne jest uwzględnienie ograniczeń modelu oraz możliwości jego ulepszenia. Ograniczenia mogą obejmować niską trafność w przypadku specyficznych zapytań, których nie sprawdzono podczas analizy modeli. Dużą uwagę trzeba poświęcić również na nadmierną czułość na pewne frazy, tematy czy też zależność skuteczności od wielkości zbioru treningowego.

Głównym ograniczeniem, jest fakt, że modele zostały wytrenowane na bazie danych z przedziału lat 2020-2023, co skutecznie pomogło w fazie testowania jak i trenowania, ale stanowczo ograniczyło potencjał finalnego modelu. Ważną sprawą był okres trenowania modelu, ponieważ czas grał główną rolę. Ograniczeniem był również, sam sprzęt do trenowania modeli, czyli prywatny laptop. Eliminacja wszystkich wyżej opisanych utrudnień spowodowałoby, że projekt miałby lepiej wytrenowany model, a użytkownicy mieliby dostęp do artykułów, które dokładniej odpowiadałyby ich zapytaniom.

Potencjalne ulepszenia mogą obejmować dalsze dostosowywanie hiperparametrów modelu, ponieważ proces poszukiwania modelu nie został jeszcze zakończony, a kontynuacja poszukiwań z pewnością mogłaby prowadzić do znalezienia jeszcze lepszego modelu. Większy i bardziej zróżnicowany zbiór treningowy pozwoliłby na projekt, który

uwzględniałby wszystkie dostępne artykuły na stronie PubMed. Wykup dostępu do super komputerów lub wzmocnienie obecnie używanej maszyny do szybszych obliczeń, co znacznie przyczyniłoby się do rozwoju uczenia modelu wyszukiwania narzędzi bioinformatycznych.

Ponadto, możliwe jest uwzględnienie opinii użytkowników w procesie oceny skuteczności wyników, co drastycznie przyczyniłoby się do znalezienia modelu idealnego na potrzeby badaczy. Ponieważ należy pamiętać, że dana realizacja tworzona jest właśnie dla takich ludzi, dlatego ich opinia o działaniu strony internetowej, ale również samego modelu pozwoliłaby na poprawę całego projektu.

7. Podsumowanie i wnioski

7.1. Główne osiągnięcia projektu

Projekt osiągnął kilka kluczowych celów, które mam nadzieję, przyczynią się do rozwoju dziedziny biotechnologii, medycyny oraz przetwarzania języka naturalnego (NLP). Główne osiągnięcia projektu obejmują:

1) Stworzenie wyszukiwarki artykułów o narzędziach bioinformatycznych - zaimplementowana wyszukiwarka, oparta na technice uczenia modelu Doc2Vec, umożliwia skuteczne wyszukiwanie artykułów pobranych z bazy danych PubMed. To narzędzie może być wykorzystane przez badaczy, naukowców oraz specjalistów z dziedzin biotechnologii jak i medycyny w celu szybkiego dostępu do istotnych informacji.
2) Integracja modelu NLP z interfejsem użytkownika - przy użyciu biblioteki Flask, stworzono prostą, intuicyjną stronę internetową, umożliwiającą użytkownikom łatwe korzystanie z wyszukiwarki artykułów bioinformatycznych. Integracja modelu NLP pozwala na dynamiczne generowanie wyników wyszukiwania na podstawie wprowadzonych zapytań.
3) Ocena jakości modelu i opracowanie rankingu - przeprowadzono analizę wyników, oceniając jakość artykułów zwracanych przez model. Opracowany ranking hiperparametrów pozwalający na identyfikację najbardziej efektywnego modelu w kontekście zadania wyszukiwania artykułów o narzędziach bioinformatycznych.
4) Utworzenie modelu uczenia maszynowego NLP za pomocą techniki Doc2Vec – przy użyciu biblioteki gensim, stworzono model, który rozpoznaje sens tekstu dla każdego z artykułów, co pomogło w wyszukiwaniu interesujących użytkownika artykułów.

7.2. Wnioski końcowe

Projekt potwierdza potencjał zastosowania modeli NLP, konkretnie używających techniki Doc2Vec, w dziedzinie analizy artykułów o narzędziach bioinformatycznych. Integracja tego modelu z interfejsem użytkownika pozwala na efektywne wykorzystanie zaawansowanych technik przetwarzania języka naturalnego do celów naukowych.

Pomimo osiągniętych sukcesów, istnieje potencjał do dalszego doskonalenia modelu, zwłaszcza poprzez uwzględnienie większego i bardziej zróżnicowanego zbioru danych treningowych. Analiza wyników pozwala na zrozumienie skuteczności modelu i identyfikację obszarów do poprawy. Skrupulatne podejście do interpretacji wyników stanowi kluczowy element rozwoju projektu. Otwiera on perspektywę dalszych badań w zakresie biotechnologii oraz medycyny. Możliwe jest rozwinięcie projektu, uwzględniając różne techniki, inne hiperparametry oraz nowe dane.

Podsumowując, projekt stanowi istotny krok w kierunku wykorzystania zaawansowanych technik NLP do ułatwienia dostępu do wiedzy oraz tekstów z zakresu biotechnologii oraz medycyny. Wyszukiwarka ta stanowczo przyczyni się do łatwiejszego oraz szybszego odnajdywania artykułów o narzędziach bioinformatycznych.

Literatura

[1] https://pubmed.ncbi.nlm.nih.gov/. Dostęp 15.03.2023.

[2] https://en.wikipedia.org/wiki/Natural_language_processing. Dostęp 18.01.2024.

[3] https://ftp.ncbi.nlm.nih.gov/pubmed/baseline/. Dostęp 18.03.2023.

[4] https://shuzhanfan.github.io/2018/08/understanding-word2vec-and-doc2vec/. Dostęp 25.06.2023.

[5] https://en.wikipedia.org/wiki/Word2vec. Dostęp 02.11.2023.

[6] https://en.wikipedia.org/wiki/Cosine_similarity. Dostęp 4.11.2023.

[7] https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html. Dostęp 02.11.2023.

[8] https://radimrehurek.com/gensim/index.html. Dostęp 17.03.2023.

[9] https://flask.palletsprojects.com/en/3.0.x/. Dostęp 03.02.2023.

Sygnatura:

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza Rzeszów, 2024
Wydział Matematyki i Fizyki Stosowanej

STRESZCZENIE PROJEKTU INŻYNIERSKIEGO

Inteligentna wyszukiwarka narzędzi bioinformatycznych, stworzona w oparciu o algorytmy klasy NLP.

Autor: Albert Gawin, nr albumu: FS-DI - 166648 Opiekun: dr Michał Piętal

Słowa kluczowe: NLP, bioinformatyka, narzędzia, artykuły, wyszukiwarka

Głównym celem projektu było stworzenie wyszukiwarki narzędzi bioinformatycznych w oparciu o technikę NLP. Wykorzystano algorytm Doc2Vec, za pomocą którego wyciągnięto kontekst artykułów, w celu znalezienia najlepszego dopasowania do wejścia użytkownika. Poprzez wytrenowanie sześćdziesięciu czterech modeli, zdefiniowano najlepsze hiperparametry, które przyczyniły się do wyboru finalnego modelu. Zaprojektowana strona internetowa, pozwoliła na stworzenie rankingu najlepiej dopasowanych artykułów. Użytkownik posiada możliwość wyszukania artykułów na bazie wpisanego wyrażenia. Podsumowując, projekt stanowi istotny krok w kierunku wykorzystania zaawansowanych technik NLP do ułatwienia dostępu do wiedzy oraz tekstów z zakresu biotechnologii oraz medycyny.

RZESZOW UNIVERSITY OF TECHNOLOGY Rzeszow, 2024
The Faculty of Mathematics and Applied Physics

DIPLOMA THESIS (BS) ABSTRACT

Intelligent search engine for bioinformatics tools, developed on the basis of NLP-class algorithms

Author: Albert Gawin, code: FS-DI - 166648 Supervisor: Michał Piętal, PhD

Key words: NLP, bioinformatics, tools, articles, search engine

The main goal of the project was to create a search engine for bioinformatics tools based on the NLP technique. The Doc2Vec algorithm was used, using which the context of articles was extracted to find the best match for user input. By training sixty-four models, the best hyperparameters were defined, which contributed to the selection of the final model. The website designed, allowed for the ranking of the best-fit articles. The user has the ability to search for articles based on the phrase entered. All in all, the project represents an important step towards using advanced NLP techniques to facilitate access to knowledge and texts in the fields of biotechnology and medicine.