Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Sztuczna Inteligencja

Projekt aplikacji do monitorowania DNA w środowisku

Patryk Filip Gryz
Numer albumu 318657

promotor
dr hab. inż. Robert Nowak

WARSZAWA 2024

Projekt aplikacji do monitorowania DNA w środowisku

Streszczenie.

Test

Słowa kluczowe:

Keyword1, Keyword2, Keyword3

Design of an application for monitoring DNA in the environment

Abstract.
Abstract

Keywords:
Keyword1, Keyword2, Keyword3

Spis treści

1. Wprowadzenie

Odkrycie DNA zapoczątkowało nową erę w biologii oraz medycynie, umożliwiając badania nad mechanizmami dziedziczenia [1] i molekularnymi podstawami życia oraz precyzyjną diagnostykę wielu chorób **Louie:2000**. Pierwsze kroki na tej drodze uczynił w 1869 roku Friedrich Miescher, który po raz pierwszy wyizolował z jądra komórkowego substancję nazwaną przez niego ńukleiną"**Dahm:2005**, później zidentyfikowaną jako kwas deoksyrybonukleinowy. W latach 1895 - 1901 Albrecht Kossel wyizolował oraz nazwał cztery podstawowe zasady azotowe budujące kwas deoksyrybonukleinowy - adeninę, tyminę, cytozynę oraz guaninę **Kossel:1893**. Przełomowe znaczenie DNA w przenoszeniu informacji genetycznej wykazano jednak dopiero w 1944 roku dzięki eksperymentowi Avery'ego-MacLeoda-McCarty'ego **Avery:1944**, w którym udowodnili, że to DNA, a nie białka są nośnikiem informacji. W 1951 roku Erwin Chargaff odkrył prawidłowość, że ilość adeniny w DNA jest porównywalna ilości tyminy, a ilość guaniny ilości cytozyny **Chargaff:1952**. Prawidłowość ta, nazwana później zasadami Chargaffa, stała się jedną z kluczowych przesłanek do odkrycia struktury podwójnej helisy DNA przez Watsona i Cricka w 1953 roku **Watson:1953**.

W 1970 Francis Crick sformułował centralny dogmat biologii molekularnej **Crick:1970**, który głosi, że informacja genetyczna przepływa z DNA do RNA, a następnie do białek, tworząc fundament współczesnej biologii molekularnej. Dalsze przełomowe odkrycia metod sekwencjonowania, opracowane w 1976 roku przez Allana Maxima oraz Waltera Gilberta **Maxam:1977**, a także w 1977 roku przez Fredericka Sangera **Sanger:1977**, pozwoliły na odczytywanie sekwencji DNA dowolnych organizmów, otwierając nowe możliwości badań nad DNA oraz przepływem informacji genetycznej. W tym samym czasie zespołowi Fredericka Sangera za pomocą nowej metody udało się po raz pierwszy zsekwencjonować w całości materiał genetyczny wirusa DNA - bakteriofaga X174 **Sanger:1977_2**. Odkrycie te uwypukliło ograniczenia tradycyjnych metod analizy i wykazało konieczność zastosowania komputerów do przetwarzania sekwencji DNA. **Staden:1979**

W miarę postępu badań nad sekwencjami DNA, zespoły badawcze zaczęły gromadzić sekwencje w celu dalszych analiz i porównań. Pod koniec lat 70. XX wieku pojawiła się potrzeba stworzenia bazy danych, która umożliwiłaby gromadzenie sekwencji w celu ich późniejszego wykorzystania oraz redukcji kosztów badań. Odpowiedzią na to zapotrzebowanie było sfinansowanie bazy danych GenBank w 1982 w Stanach Zjednoczonych[13] oraz europejskiej bazy danych EBML Data Library w 1980 roku [14]. W 1983 roku Johnowi Wilburowi oraz Davidowi Lipmanowi udało się opracować algorytm wyszukiwania podobnych sekwencji w bazach danych [15], co pozwoliło na szybkie porównywanie sekwencji genetycznych. W 1990 roku wprowadzono narzędzie BLAST [16], które przyśpieszyło ten proces.

Pierwszy pomysł na analizę materiału genetycznego organizmów obecnych w glebie bez konieczności ich hodowli został zaproponowany w 1998 roku przez Jo Handlesmana i jego zespół [17]. Wykorzystanie informacji o materiale genetycznym znalezionym w środowisku, w połączeniu z bazami danych sekwencji, umożliwia identyfikację organizmów obecnych

w badanej próbce [??]. Proces analizy dużej liczby sekwencji DNA oraz ich porównywania z bazami danych wymaga jednak znacznych zasobów obliczeniowych.

Wraz z rozwojem metod sekwencjonowania nowej generacji [18], koszty sekwencjonowania znacząco spadły, a liczba analizowanych sekwencji DNA znacznie wzrosła [19]. Dalsze gromadzenie danych sekwencyjnych oraz wzrost przepustowości technologii sekwencjonowania prowadzą do zwiększonego zapotrzebowania na zasoby obliczeniowe.

—- REFERENCES 1. XYZ 2. https://pmc.ncbi.nlm.nih.gov/articles/PMC80298/ 3. https://www.sciencedire
Kossel, A. and Neumann, A.: Ueber das Thymin ein Spaltungsprodukt der Nucleinsaure.
Ber. Deut. chem. Ges., 1893, 26, 2753. 5. https://pubmed.ncbi.nlm.nih.gov/19871359/c 6.
https://www.sciencedirect.com/science/article/pii/S0021925819508845?via7. https://www.nature.com/arti
8. https://pubmed.ncbi.nlm.nih.gov/4913914/ 9. https://pubmed.ncbi.nlm.nih.gov/265521/
10. https://pubmed.ncbi.nlm.nih.gov/271968/ 11. https://pubmed.ncbi.nlm.nih.gov/870828/
12. https://pmc.ncbi.nlm.nih.gov/articles/PMC327874/
13. https://doi.org/10.1093/nar/14.1.1 14. https://pmc.ncbi.nlm.nih.gov/articles/PMC333983/
15. https://pubmed.ncbi.nlm.nih.gov/6572363/ 16. https://pubmed.ncbi.nlm.nih.gov/2231712/
17. https://www.sciencedirect.com/science/article/pii/S1074552198901089?via18. https://pubmed.ncbi.
19. https://pmc.ncbi.nlm.nih.gov/articles/PMC4806511/

1.1. Cel pracy dyplomowej

Celem niniejszje pracy jest stworzenie projektu oraz implementacja aplikacji umożliwiającej użytkownikowi zlecanie analiz materiału genetycznego DNA pobranego z środowiska. Aplikacja ma usprawnić proces przeprowadzania analiz oraz proces agregacji końcowych wyników do postaci czytelnej przez człowieka.

1.2. Zakres pracy

Zakres pracy obejmuje:

- Projekt i implementację aplikacji do analizy materiału genetycznego.
- Klienta przeglądarkowego pozwalającego na zlecanie nowych analiz.
- Funkcjonalności pozwalające na wczytanie sekwencji DNA w formacie FASTA oraz FASTQ oraz wyszukiwanie w bazie danych NCBI BLAST
- Implementację własnej metody porównywania sekwencji
- Eksperymenty wydajnościowe oraz jakościowe klasycznych metod oraz metody własnej

Praca nie obejmuje:

- Wdrożenia aplikacji w środowisku produkcyjnym
- Wykorzystania innych baz sekwencji
- Wersji mobilnej klienta przeglądarkowego

1.3. Układ pracy

Niniejsza praca składa się z następujących rozdziałów:

- Rozdział 2. skupia ię na przeglądzie literatury, analizie istniejących badań oraz przedstawieniu obecnych rozwiązań.
- Rozdział 3. zawiera wstęp teoretyczny, objaśniający definicję oraz operacje na sekwencjach oraz wykorzystywane algorytmy.

TODO [ok. 3 strony]

1. Krótki opis problemu [akapit]
2. Cel pracy (co zamierzam zrobić) [1 akapit]
3. Zakres pracy (jak zamierzam zrobić) [1 akapit]
4. Układ pracy (co w którym rozdziale) [1 akapit]

2. Wstęp teoretyczny

2.1. Podstawowe definicje

2.1.1. Sekwencja DNA

Sekwencja DNA stanowi zapis genetyczny, który w sposób symboliczny odwzorowuje strukturę cząsteczki DNA, stosując alfabet złożony z czterech symboli: A, T, C, G. Każdy z symboli odnosi się do jednej z zasad azotowych zawartych w nukleotydach tworzących cząsteczkę DNA odpowiednio: adeniny, tyminy, cytozyny oraz guaniny.

2.2. KMer

K—mery są to podsłowa sekwencji genetycznej o długości k. Dla danego alfabetu G składającego się z n symboli istnieje n^k różnych k-merów o długości k. Sekwencja genetyczna o długości n zawiera dokładnie $n-k+1$ k-merów o długości k.

2.2.1. Dopasowanie sekwencji

Proces dopasowywania sekwencji polega na wyrównywaniu ich symboli w celu maksymalizacji ich wzajemnego podobieństwa, co realizowane jest poprzez wstawianie przerw (ang. *gaps*).

2.2.2. Algorytm Needlema-Wunscha

Algorytm Needlemana-Wunscha stanowi metodę wykorzystywaną do ustalania globalnego dopasowania pomiędzy dwiema sekwencjami **NeedlemanWunsch1970**. Metoda ta polega na skonstruowaniu macierzy podobieństwa pomiędzy sekwencjami zgodnie z ustalonymi regułami:

$$\begin{aligned} D_{i,0} &= i \cdot g, && \text{dla } i \in [0,n] \\ D_{0,j} &= j \cdot g, && \text{dla } j \in [1,m] \\ D_{i,j} &= \max \begin{cases} D_{i-1,j} + g \\ D_{i,j-1} + g \\ D_{i-1,j-1} + s(A_i, B_j) \end{cases}, && \text{dla } i \in (0,n] \text{ oraz } j \in (0,m] \end{aligned} \tag{2.1}$$

gdzie,

$g, s(A_i, B_j) \in \mathbb{R}$

A, B – porównywane sekwencje,

n, m – długości sekwencji A oraz B,

D – macierz podobieństwa o rozmiarach n x m,

g – kara za przerwę,

$s(A_i, B_j)$ – podobieństwo między i-tym elementem w sekwencji A, a j-tym elementem w sekwencji B.

Wartość znajdująca się w $D_{n,m}$ określa liczbowo jakość globalnego dopasowania sekwencji.

2.2.3. Uczenie kontrastowe

Uczenie kontrastowe (ang. *contrastive learning*) **Bromley1993** jest metodą polegającą na nauce reprezentacji danych poprzez porównywanie i różnicowanie podobnych oraz różnych przykładów. Dzięki zastosowaniu tej techniki, reprezentacje danych zachowują właściwości podobieństwa i różnicy między danymi, które reprezentują.

2.2.4. Podobieństwo i niepodobieństwo kosinusowe

Podobieństwo i niepodobieństwo kosinusowe są miarami, które mogą być wykorzystywane do porównywania wektorów liczbowych, zdefiniowane one są wzorami:

$$similarity_{cosine} = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.2}$$

$$dissimilarity_{cosine}(A, B) = 1 - similarity_{cosine}(A, B) \tag{2.3}$$

gdzie,

A, B – porównywane wektory,

θ – kąt między wektorami A i B,

A_j, B_j – j-ty element wektora odpowiednio A oraz B.

2.3. Metody

W pracy zaimplementowano trzy metody: nową metodę opartą na sieciach neuronowych oraz dwie klasyczne metody wykorzystywane do porównania. Nowa metoda oparta na sieciach neuronowych została opracowana w celu połączenia zalet obu klasycznych metod oraz eliminacji ich niedoskonałości. W szczególności skoncentrowano się na przyspieszeniu procesu określania niepodobieństwa w porównaniu do algorytmu Needlemana-Wunscha oraz na zwiększeniu jakości mierzonych niepodobieństw między

sekwencjami DNA w porównaniu z metodą wykorzystującą k-mery, która nie bierze pod uwagę w pełni struktury porównywanych sekwencji.

2.3.1. Zmodyfikowany algorytm Needlemana-Wunscha

Pierwszą metodą klasyczną, która została wykorzystana do określania niepodobieństwa między sekwencjami DNA jest zmodyfikowany algorytm Needlama-Wunscha. Algorytm pozwala na dokładne określanie niepodobieństwa z uwzględnieniem przerw oraz zmiany danych zasad. Modyfikacja algorytmu polega na zmianie budowy macierzy podobieństwa oraz wprowadzeniu dodatkowych ograniczeń w celu zapewnienia, że niepodobieństwo będzie mieściło się w przedziale: $[0,\infty)$:

$$\begin{aligned} D_{i,j} &= \min \begin{cases} D_{i-1,j} + g \\ D_{i,j-1} + g \\ D_{i-1,j-1} + s(A_i, B_j) \end{cases}, \quad \text{dla } i \in (0,n] \text{ oraz } j \in (0,m] \\ g, s(A_i, B_j) &\in \mathbb{R}^+ \end{aligned} \tag{2.4}$$

Wartość znajdująca się w $D_{n,m}$ jest miarą niepodobieństwa między sekwencjami.

2.3.2. KMer

Druga zaimplementowana metoda wykorzystuje k-mery do określania niepodobieństwa między sekwencjami DNA, poprzez wyznaczenie odległości euklidesowej w przestrzeni k-merów, gdzie osie tej przestrzeni odpowiadają różnym k-merom występującym w sekwencjach, a wektory reprezentują ich częstości wystąpień. Niepodobieństwo wyliczone jest według wzoru:

$$dissimilarity_{kmer}(A,B,k) = \sqrt{\sum_{m \in M_k} (A_m - B_m)^2} \tag{2.5}$$

gdzie,

k – długość k-merów,

A, B – porównywane sekwencje,

M – zbiór wszystkich możliwych sekwencji genetycznych o długości k,

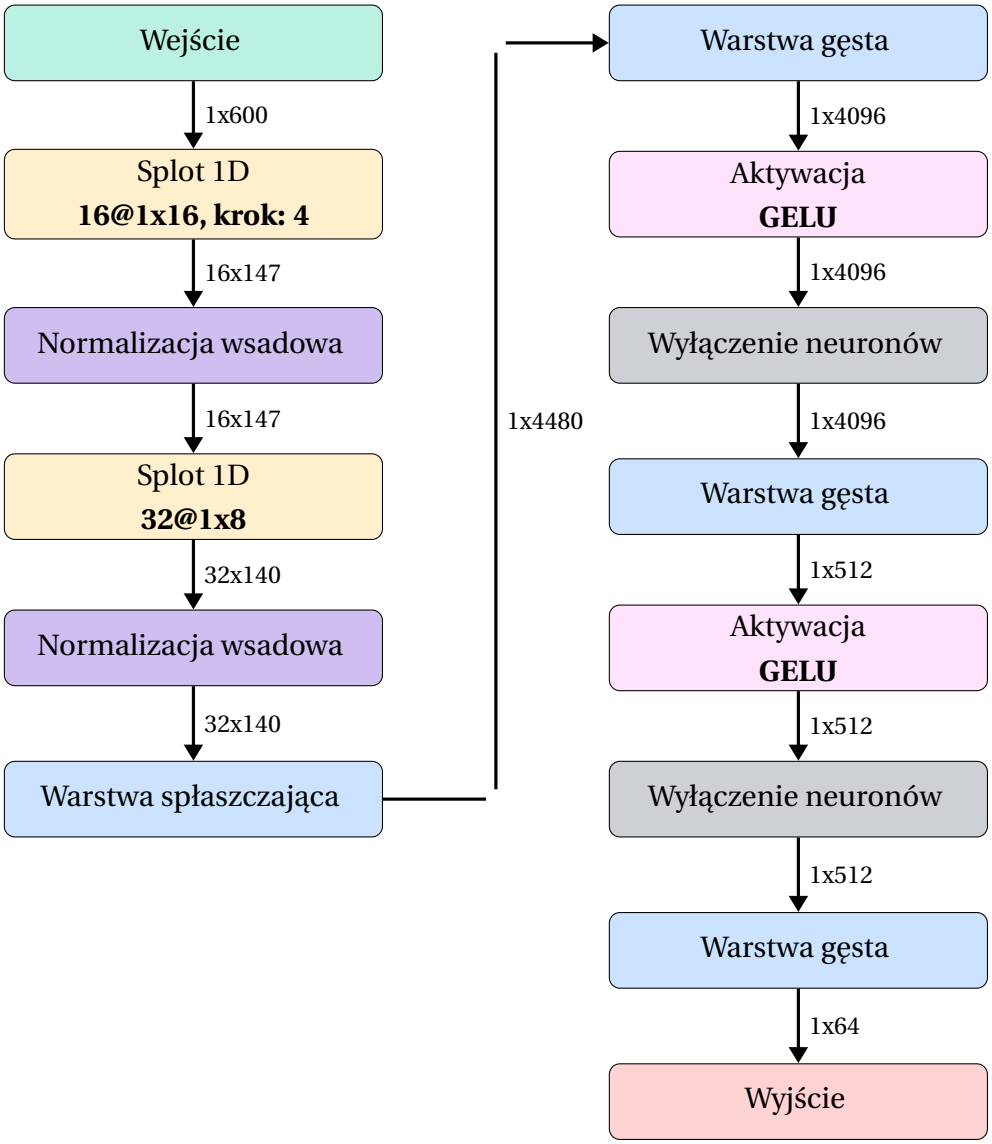
A_j, B_j – liczba wystąpień sekwencji j odpowiednio w sekwencjach A i B.

2.3.3. Sztuczna sieć neuronowa

Zaproponowana metoda wykorzystuje sztuczne sieci neuronowe wraz z uczeniem kontrastowym do stworzenia modelu, który będzie pozwalał na tworzenie reprezentacji sekwencji wejściowych, które zachowają właściwości niepodobieństwa między sekwencjami. Na podstawie stworzonych reprezentacji sekwencji będzie wyliczone niepodobieństwo z wykorzystaniem niepodobieństwa kosinusowego.

Architektura

Architektura sieci neuronowej składa się z 2 bloków wykorzystujących warstwy splotowe, które odpowiadają za ekstrakcję niskopoziomowych cech sekwencji, warstwy spłaszczającej oraz trzech warstw perceptronów wielowarstwowych, które odpowiadają za stworzenie reprezentacji na podstawie wyekstrahowanych cech. Wyjściem jest wektor zanurzeń o rozmiarze 64. Schematycznie architektura została przedstawiona na rysunku 2.1).



Rysunek 2.1. Schemat architektury sieci neuronowej.

Przykłady uczące Przykłady uczące składają się z kotwicy (ang. *anchor*) oraz dwóch sekwencji: pierwsza sekwencja podobna do kotwicy, druga nie.

Zbiór danych Do stworzenia zbioru danych: treningowego oraz walidacyjnego została wykorzystana pierwsza próbka sekwencji genetycznych ze zbioru *CAMI II Toy Human Microbiome Project* **Fritz2019**. Zbiory zostały uzyskane poprzez losowy wybór sekwencji genetycznych z próbki, które następnie zostały uznane za kotwice. Przykłady podobne oraz niepodobne uzyskano poprzez modyfikację kotwicy odpowiednio w zakresie $[0;0.2]$, $[0.2;0.8]$.

Proces uczenia

Przeprowadzono proces uczenia na zbiorze 10^6 przykładów treningowych oraz 10^4 walidacyjnych.

Funkcja straty Wykorzystano funkcję straty zdefiniowaną jako:

$$\text{Strata kontrastowa} = [m_{pos} - s_{pos}]_+ + [s_{neg} - m_{neg}]_+ \tag{2.6}$$

gdzie,

m_{pos}, m_{neg} – margines podobieństwa między przykładami pozytywnymi a kotwicą, oraz między przykładami negatywnymi a kotwicą,

s_{pos}, s_{neg} – podobieństwo kosinusowe przykładu pozytywnego do kotwicy, oraz negatywnego do kotwicy.

Optymalizator W procesie uczenia wykorzystano optymalizator *AdamW* **Loshchilov2017DecoupledWD** z wykładniczym spadkiem współczynnika uczenia oraz zanikiem wag (ang. *weight decay*).

Parametry Parametry optymalizatora: $\lambda = 10^{-7}$, $\gamma = 0.99999$, zanik wag = 10^{-7}. Parametry funkcji straty: $m_{pos} = 1.0$, $m_{neg} = 0.25$. Parametr warstw wyłączenia (ang. *dropout*) = 0.5 dla obu warstw. Liczbę epok ustawiono na 5, ze względu na ryzyko przeuczenia.

Miara jakości Jako miarę jakości modelu wykorzystano stratę kontrastową modelu obliczoną na zbiorze walidacyjnym.

2.4. Wykorzystane narzędzia

- Python - Rust - HTML, JS - + biblioteki

3. Analiza literatury

KONIECZNIE DODAĆ: https://cami-challenge.org/ https://cami-challenge.org/datasets/Toycami-chal
sample 1 - skin sample

TODO [ok. 15 stron]

1. Opis problemu
2. Opis rozwiązań problemu
3. Opis algorytmów rozwiązujących problem
4. Opis narzędzi (tutaj: Rust, Python itd.) + używane biblioteki

Wprowadzenie do zagadnienia

- Duża liczba organizmów
- Nowe metody "masowegośekwencjowania

Opis problemu, przeglądówki

- https://arxiv.org/pdf/1510.06621 - Przeglądówka różnych narzędzi stosowanych do np. klasyfikacji czy klastrowania
- https://arxiv.org/pdf/1808.01038 - Wyszukiwanie sekwencji - przeglądówka po historii
- https://arxiv.org/html/2408.12751v1 - Storage clustering
- https://arxiv.org/pdf/1006.4114 - DNA search engine

Rozwiazania:

- https://arxiv.org/abs/1306.1569 - znajdowanie kilku representatów na podstawie losowo wybranych odczytów. Porównanie pozostałych sekwencji z wybranymi reprezentatami
- https://arxiv.org/abs/2407.02538 - zamiana DNA w mapę 2D, wykonanie algorytmów DL na mapach 2D w celu obliczania odległości bez alignmentu oraz klastrowania
- https://arxiv.org/abs/2111.09656 - Contrastive learning binning

Opis algorytmów

- k-mer clustering -> k-medoids clustering
- jeden z algorytmów do alignmentu
- Contrastive learning
- Indeks Jaccarda
- Baza danych Blast
- Needleman-Wunsch Algorithm - https://pubmed.ncbi.nlm.nih.gov/5420325/

Opis narzędzi

Python - krótki opis pythona

Biblioteki - psutil - do monitorowania zużycia procesora oraz pamięci przez eksperymenty - typing (wbudowana) - opis - subprocess (wbudowana) - do uruchamiania eksperymentów w nowych procesach - w celu możliwości monitorowania zużycie przez nie zasobów

Rust

Biblioteki

Frontend - askama - (Jinja template engine) - wykorzystywana do eegenerowania zawartości (SSR - Server side rendering) stron internetowych - tower - - sqlx - wykorzystywana do łączenia się z bazą danych - dotenv - wczytywanie zmiennych środowiskowych z pliku - tracing - śledzenie i instrumentowanie aplikacji - wyświetlanie informacji na temat stanu aplikacji (np. requestów czy błędów) - tokio - biblioteka implementująca operacje asynchroniczne na plikach oraz pozwalająca na współbieżność realizacji zadań - axum - framework tworzenia aplikacji webowych

Backend - serde - wykorzystywane do serializacji oraz deserializacji danych w celu zapisania do pliku - $serde_json - serializacja do JSON - tempfile - tworzenie plików tymczasowych, wyk$
$float_cmp - wykorzystywane do testów, pozwala na porównywanie liczb miennoprzecinkowych$
$num_traits - wykorzystanie w celu opisania generycznych funkcji do obliczania np. dystansu euk$

HTML, CSS, JS - realizacja strony internetowej - (brak bibliotek zewnętrznych)

Historia

Needleman and Wunsch (1970):

- Klastrowanie sekwencji z wykorzystaniem alignmentu https://pmc.ncbi.nlm.nih.gov/articles/PMC33894
- Klastrowanie sekwencji z wykorzystaniem k-merów

- CD-HIT - Mothur

1. Algorytm Needleman-Wunscha 2. Odległość k-mer

- definicja problemu

RZECZYWISTY PRZEGLĄD LITERATURY

———————————— - powiększanie się baz danych sekwencji - zwiększenie się ilości i przepustowośc sekwencjowania sekwencji

- metagenomika - wprowadzenie - intensowne wyszukiwanie w bazach danych - wzrost czasochłonności wyszukiwania

- definicja problemu

———————————————

- metody - wady i zalety tych metody - poszukiwania rozwiązania

———————————————

Opisać moje rozwiązanie i pokazac różnice

WSTEP TEORETYCZNY - wprowadzenie podstawowych pojęc - wprowadzenie algorytmów - narzędzia np. Python, Rust - biblioteki - wykorzystane algorytmy w potoku - (wspomnieć i z grubsza opisać działanie)

4. Projekt i Implementacja

4.1. Wymagania funkcjonalne i niefunkcjonalne

4.1.1. Wymagania funkcjonalne

- Aplikacja powinna umożliwiać przeprowadzanie klasyfikacji taksonomicznej dla wprowadzonych sekwencji
- Aplikacja powinna umożliwiać wybór metody grupowania sekwencji genetycznych
- Aplikacja powinna zawierać zaimplementowane trzy metody grupowania sekwencji genetycznych
- Aplikacja powinna umożliwiać porównanie jakości klasyfikacji taksonomicznej wykonanej przy użyciu różnych metod grupowania
- Aplikacja powinna umożliwiać zapis wyników w formacie JSON

4.1.2. Wymagania niefunkcjonalne

- Aplikacja powinna być wykonana przy wykorzystaniu kompilowanego wysoko wydajnego języka programowania
- Implementacja powinna zawierać testy jednostkowe poszczególnych modułów i funkcji

4.2. Model architektury rozwiązania

Architektura rozwiązania została przedstawiona w postaci modelu C4**C4**. Model ten opisuje architekturę na czterech diagramach: kontekstowych, kontenerów, komponentów oraz kodu, przechodząc od najbardziej abstrakcyjnego spojrzenia na architekturę rozwiązania do kodu implementacji.

4.3. Opis rozwiązania

Centralnym elementem rozwiązania jest potok przetwarzania, zaprojektowany z myślą o elastycznej konfiguracji oraz możliwości zastosowania różnych metod grupowania sekwencji DNA. Elementy potoku zostały zaimplementowane w bibliotece *exquisitor-core*, w postaci funkcji oraz rekordów. Realizacja potoku została umieszczona w aplikacji konsolowej, która wykorzystuje tę bibliotekę. Aplikacja umożliwia uruchamianie i konfigurację potoku, oferując dostęp do dostępnych metod. Zawiera również polecenia pomocnicze, które zostały przygotowane w celu przeprowadzenia eksperymentów.

Dodatkowo stworzono aplikację przeglądarkową, oferującą prosty interfejs w postaci strony internetowej. Umożliwia ona użytkownikowi zlecanie klasyfikacji taksonomicznej dla przesłanych sekwencji DNA. Do realizacji zleconych zadań aplikacja przeglądarkowa korzysta z aplikacji konsolowej.

Ostatnim komponentem rozwiązania jest sieć neuronowa, której elementy zostały zawarte w bibliotece *exquisitor-core*.

Biblioteka, aplikacja konsolowa oraz aplikacja przeglądarkowa zostały umieszczone w oddzielnych skrzynkach (ang. *crate*) w ramach jednego rozwiązania. Dzięki temu moż-

liwe jest rozdzielenie odpowiedzialności między poszczególnymi elementami, niezależny rozwój kodu oraz umożliwienie wykorzystania biblioteki w innych aplikacjach.

4.3.1. Potok przetwarzania

Potok przetwarzania składa się z szeregu kroków, które są realizowane kolejno, jeden po drugim, w celu przeprowadzenia pełnej klasyfikacji taksonomicznej DNA. Każdy krok potoku jest odpowiedzialny za określoną część procesu i może być niezależnie modyfikowany lub wymieniany, co umożliwia elastyczność w doborze metod. Do kluczowych kroków potoku należą:

1. **Wstępne przetwarzanie sekwencji**
   Wejście: *zbiór sekwencji DNA*
   Wyjście: *zbiór sekwencji DNA*
   Krok ten odpowiada za weryfikację poprawności dostarczonych sekwencji DNA oraz ich wyrównanie do wymaganej długości, niezbędnej w kolejnych krokach przetwarzania. Może on zostać pominięty, jeśli zastosowana metoda w następnych krokach nie jest wrażliwa na długość sekwencji wejściowych.
2. **Grupowanie sekwencji**
   Wejście: *zbiór sekwencji DNA.*
   Wyjście: *Grupy (reprezentant oraz elementy).*
   Głównym celem tego kroku jest wybór najlepszych reprezentantów dla każdej grupy sekwencji, która składa się z elementów oraz reprezentanta. Dzięki temu możliwa jest redukcja liczby sekwencji do przetworzenia w kolejnych etapach. Krok ten może zostać zrealizowany przy użyciu jednej z trzech dostępnych metod.
3. **Wyszukiwanie w bazie danych**
   Wejście: *reprezentanci grup (sekwencje DNA).*
   Wyjście: *wykryte organizmy.*
   Wyszukiwanie w bazie danych sekwencji pozwala znaleźć sekwencje podobne do wyszukiwanych sekwencji DNA. Pozwala to na określenie do jakich organizmów mogą należeć wyszukiwane sekwencje. W tym kroku przetwarzani są wyłącznie reprezentanci grup. W wyniku wyszukiwania dla każdego reprezentanta uzyskuje się listę odpowiadających organizmów.
4. **Przetwarzanie wyników wyszukiwania**
   Wejście: *wykryte organizmy, grupy.*
   Wyjście: *wykryte organizmy.*
   W tym kroku następuje połączenie informacji o grupach z listą wykrytych organizmów. Obliczane są jakość oraz pewność identyfikacji organizmów na podstawie wyników wyszukiwania uzyskanych w poprzednim kroku oraz liczby elementów w grupach.
5. **Generowanie raportów dla użytkownika**
   Wyjście: *wykryte organizmy.*
   Wyjście: *wyniki.*
   Na podstawie danych z poprzedniego kroku filtrowane są istotne informacje, a następnie generowane są wyniki w formie przystępnej dla użytkownika.

Schematyczne przedstawienie potoku zaprezentowano na rysunku 4.1. W kółkach umieszczono wejście oraz wyjście z potoku przetwarzania, natomiast w prostokątach – kolejne kroki. Na strzałkach opisano rodzaj przekazywanych danych. Niebieskim kolorem obramowania oznaczono kroki, których wyjściem są sekwencje DNA, czerwonym – kroki, których wyjściem są wykryte organizmy, a zielonym – kroki generujące wyniki w formie czytelnej dla człowieka.

4.3.2. Biblioteka *exquisitor-core*

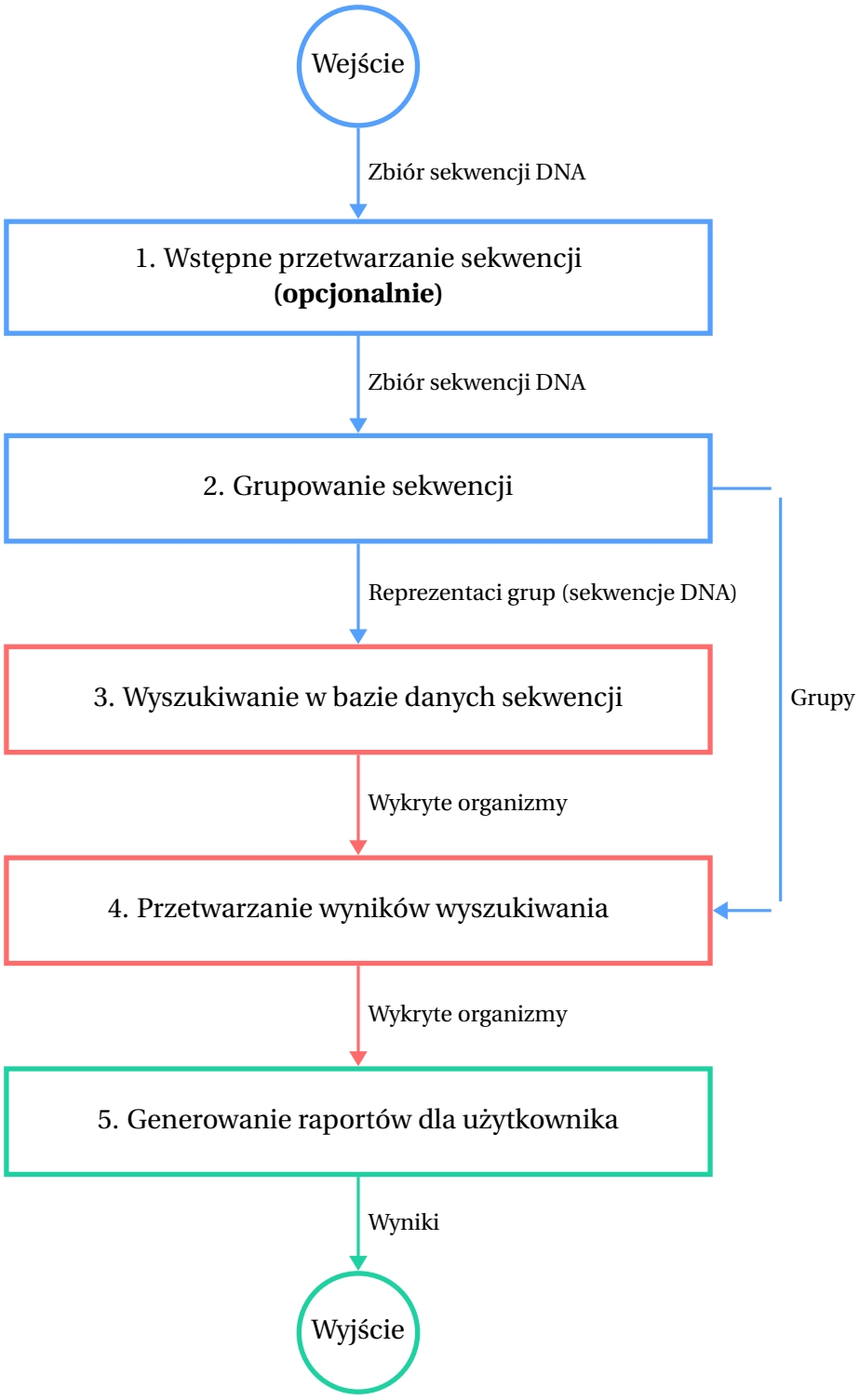
Podstawę rozwiązania stanowi biblioteka *exqusitor-core* zawarta w skrzynce (ang. *crate*) *exquisitor-core*, stworzona w języku Rust. Zawiera ona implementację wszystkich elementów potoku, które wymagane są przez aplikacje oraz dodatkowo opakowywuje implementację algorytmów k-medoidów. Biblioteka podzielona jest na kilka modułów ze względu na odpowiedzialność:

- Moduł *io* odpowiadający za operacje wejścia/wyjścia dla sekwencji DNA zapisanych w formatach FASTA oraz FASTQ. Zawiera on definicje struktur przechowujących sekwencje, metody do ich odczytu i zapisu oraz cechy umożliwiające opis funkcjonalności związanych z obsługą tych formatów.
- Moduł *clustering* realizujący proces grupowania sekwencji DNA na podstawie ich niepodobieństwa. Zawiera metody do obliczania niepodobieństwa między sekwencjami DNA oraz wektorami liczbowymi, a także algorytmy grupowania obiektów. Dodatkowo, moduł zawiera strukturę reprezentującą grupę, składającą się z reprezentanta i elementów tej grupy. Zawiera także funkcję do budowy macierzy niepodobieństwa oraz cechy, które definiują interfejs, z którego korzystają inne elementy biblioteki, umożliwia to implementację własnych struktur i funkcji, pozwalając na elastyczne rozszerzanie systemu o nowe metody.
- Moduł *neural*, który zawiera definicję modelu sieci neuronowej, implementację funkcji straty oraz funkcjonalności odpowiedzialnych za wczytanie oraz przygotowanie danych dla sieci neuronowej, w tym danych uczących. Ponadto zawiera on funkcję trenującą sieć.
- Moduł *searching* zawierający implementację struktur opisujących wykryte organizmy oraz integrację z algorytmem *blastn*.

W skrzynce *exquisitor-core* znajdują się dwa dodatkowe programy – pierwszy służy do przygotowania danych uczących dla sieci neuronowej i eksperymentalnych poprzez losowanie próbek z większego zbioru danych, natomiast drugi odpowiada za uczenie sieci neuronowej.

4.3.3. Aplikacja konsolowa *exquisitor-cli*

Główną aplikacją, która implementuje pełny potok przetwarzania i realizuje klasyfikację taksonomiczną, jest aplikacja konsolowa *exquisitor-cli* zawarta w skrzynce o tej samej nazwie. Aplikacja oferuje standardowy interfejs użytkownika typowy dla aplikacji konsolowych i wykonuje komendy wywołane przez użytkownika. Interfejs tekstowy umożliwia uruchamianie aplikacji w środowiskach bez dostępu do środowiska graficznego.



Rysunek 4.1. Schemat potoku przetwarzania.

Aplikacja implementuje nastepujące komendy:

- *experiment* — umożliwia uruchomienie wskazanej komendy w ramach eksperymentu. Pozwala na monitorowanie wykorzystania pamięci RAM oraz procesora oraz określenie maksymalnego czasu trwania eksperymentu, po których zostanie przerwany. Wykorzystywana jest do przeprowadzania eksperymentów. W wyniku działania generuje plik w formacie CSV zawierający monitorowane dane z zadaną rozdzielczością.
- *compare* — pozwala na porównanie dwóch zestawów wyników klasyfikacji taksonomicznej wyliczając jakość względną jednego zestawu do drugiego. Wyniki są wyświetlane na ekranie lub, w przypadku podania ścieżki, zapisywane do pliku.
- *compare-clusters* — pozwala na porównanie jakości dwóch zestawów grup, które zostały otrzymane w wyniku zastosowania algorytmu grupowania.
- *search* — pozwala na wyszukanie sekwencji w bazie danych bez wykonywania całego potoku przetwarzania. Wyniki działania komendy zapisywane są w wskazanym pliku.
- *run* — jest główną komendą, która realizuje potok przetwarzania. Uruchamia potok przetwarzania dla sekwencji DNA zawartych w podanym pliku. Pozwala na konfigurację wykorzystywanej metody grupowania sekwencji, algorytmu grupowania sekwencji oraz ich parametrów. Dodatkowo umożliwia zapisanie grup otrzymanych w wyniku grupowania. Wymaga podania ścieżki do programu *blastn* oraz ścieżki do jego bazy danych. Wyniki zapisuje do pliku, a w przypadku braku podania ścieżki na ekranie.

4.3.4. Aplikacja przeglądarkowa *exquisitor-app*

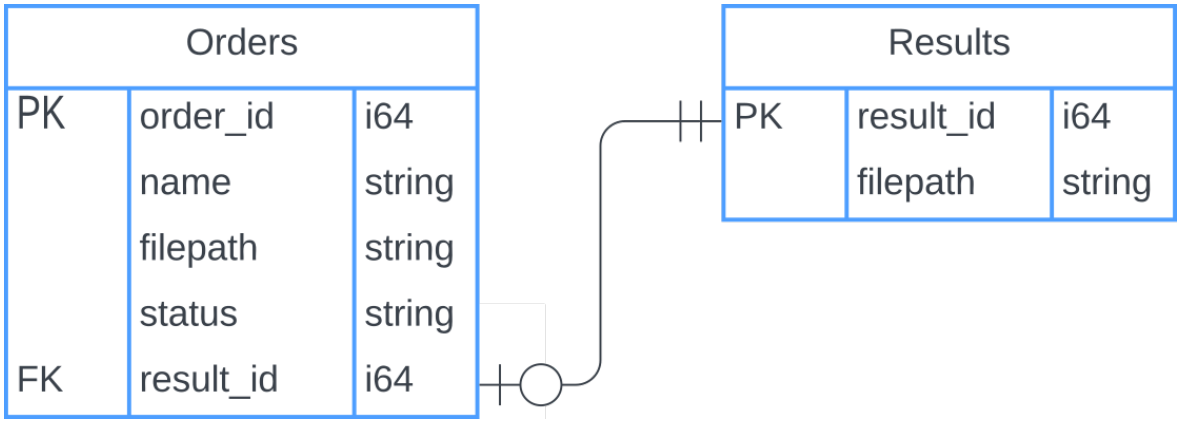
Aplikacja przeglądarkowa została zaprojektowana z myślą o umożliwieniu użytkownikom końcowym zlecania zadań klasyfikacji taksonomicznej oraz przeglądania wyników po zakończeniu procesu. Wybór formy aplikacji przeglądarkowej wynika z jej dostępności na różnych urządzeniach posiadających dostęp do internetu i przeglądarki, co zapewnia łatwy, uniwersalny dostęp do systemu. Aplikacja składa się z serwera oraz stron internetowych generowanych za pomocą szablonów.

Serwer udostępnia stronę główną, na której użytkownik może zapoznać się z ostatnimi dziesięcioma zadaniami, zlecić wykonanie nowego zadania lub wyszukać zadanie po identyfikatorze lub nazwie. Do przechowywania informacji o zleconych zadaniach wykorzystywana jest baza danych *SQLite*.

Do głównych funkcji serwera należą:

- Generowanie stron internetowych na podstawie szablonów.
- Serwowanie statycznych plików CSS oraz JavaScript.
- Obsługa błędów i wyświetlanie stosownych komunikatów dla użytkownika.

Schemat bazy danych znajduje się na rysunku 4.2.



Rysunek 4.2. Schemat bazy danych aplikacji przeglądarkowej.

Rysunek 4.3. Needleman-Wunsch metoda.

4.4. Metody grupowania sekwencji

4.4.1. Needleman-Wunsch

4.4.2. KMer

4.4.3. Sieć neuronowa

4.5. Wykorzystane technologie, narzędzia oraz biblioteki

4.5.1. Języki programowania

W pracy wykorzystano języki programowania Rust**Rust** oraz Python**Python**.

Język Python był wykorzystywany w początkowych fazach rozwoju pracy jako narzędzie do prototypowania rozwiązania oraz w ostatecznej wersji pracy do stworzenia skryptów automatyzyjących niektóre czynności związane z nauką sieci neuronowej oraz do generowania wykresów. Został on wybrany ze względu na bogatą bibliotekę standardową, dostępność wielu bibliotek zewnętrznych oraz wieloplatformowość.

Język Rust został użyty do stworzenia wszystkich aplikacji oraz programów. Wybrany został ze względu na wysokość wydajność, bezpieczne zarządzanie pamięcią oraz dużą dostępność bibliotek programistycznych, które można zainstalować za pomocą menedżera pakietów *cargo***Rust:cargo** dołączonego wraz ze środowiskiem języka Rust. Dodatkowymi atutami, które przyczyniły się do wyboru języka, jest bogaty system typów oraz kompilacja do kodu maszynowego.

4.5.2. Biblioteki programistyczne

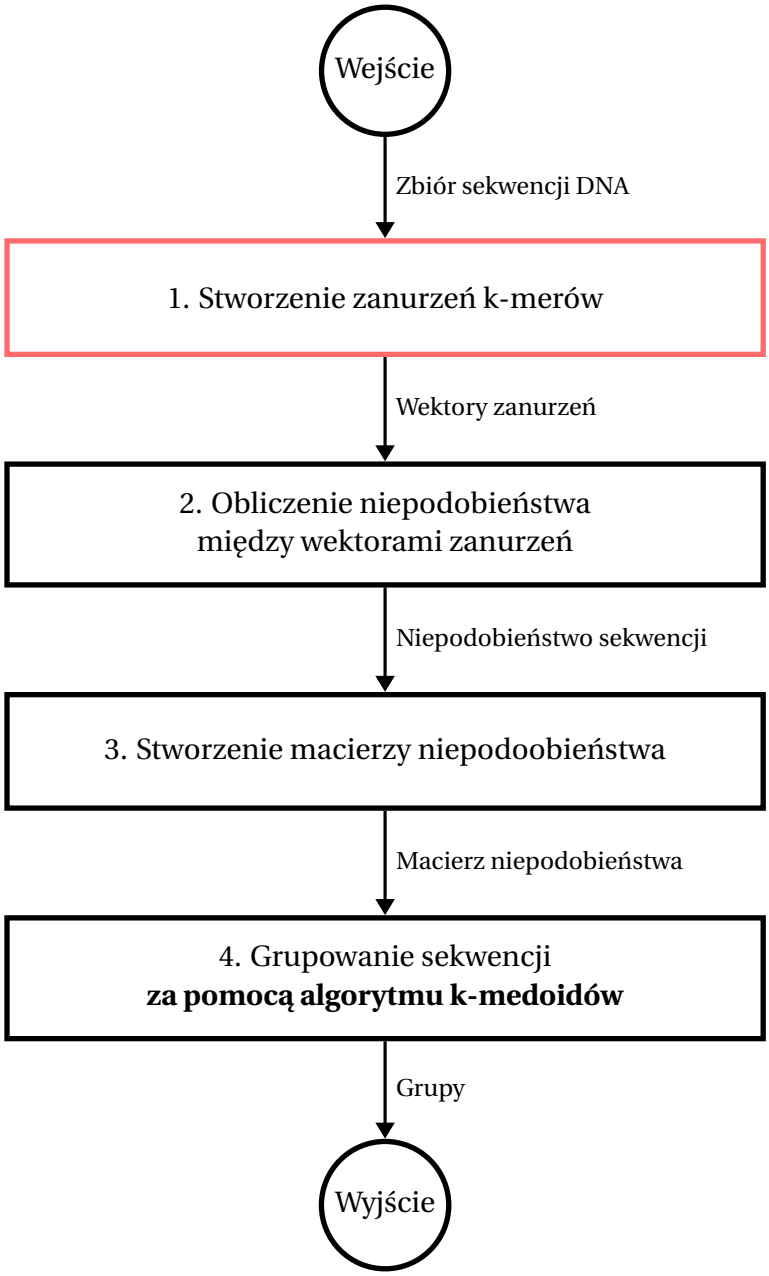
Aplikację przeglądarkową zrealizowano z wykorzystaniem biblioteki *axum***Rust:axum** opartej na asynchronicznym środowisku wykonawczym *tokio***Rust:tokio** języka Rust. Do generowania zawartości stron w formacie HTML wykorzystano silnik szablonów *askama***Rust:askama**. Komunikację z bazą danych zapewniła biblioteka *sqlx***Rust:sqlx**. Użyto dodatkowo biblioteki *dotenv***Rust:dotenv** w celu załadowania zmiennych środowiskowych z pliku, które niezbędne są do prawidłowego działania aplikacji.

Aplikacja konsolowa została oparta na bibliotece *clap***Rust:clap**, która pozwoliła na zdefiniowanie interfejsu użytkownika, w postaci dostępnych poleceń wraz z parametrami.

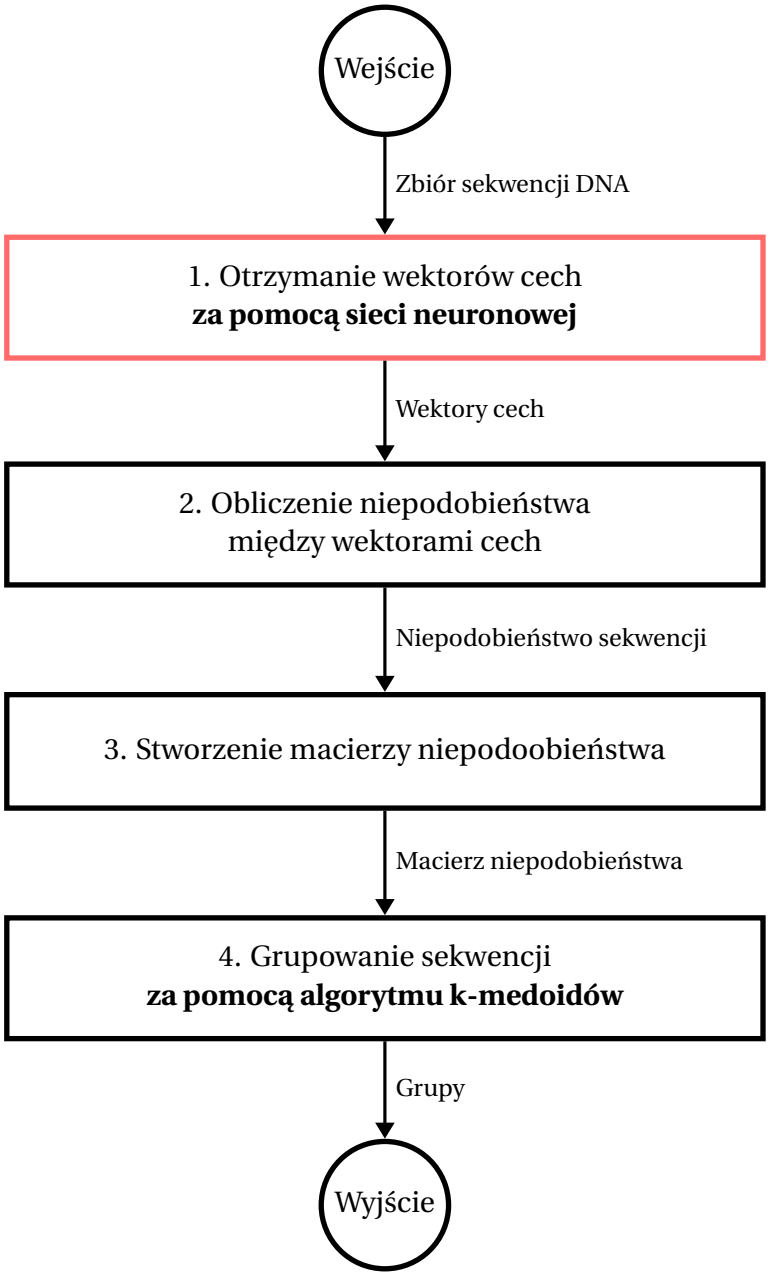
Bibioteka *exquisitor-core* korzysta z biblioteki *kmedoids***Schubert:2022**, która implementuje grupowanie k-medoidów oraz bibliotek pomocniczych *num-traits*, *tempfiles* oraz *float-cmp*, które wykorzystywane są w testach jednostkowych.

Model sieci neuronowej został zbudowany przy użyciu biblioteki *burn***Rust:burn** oraz silnika obliczeniowego *wgpu*.

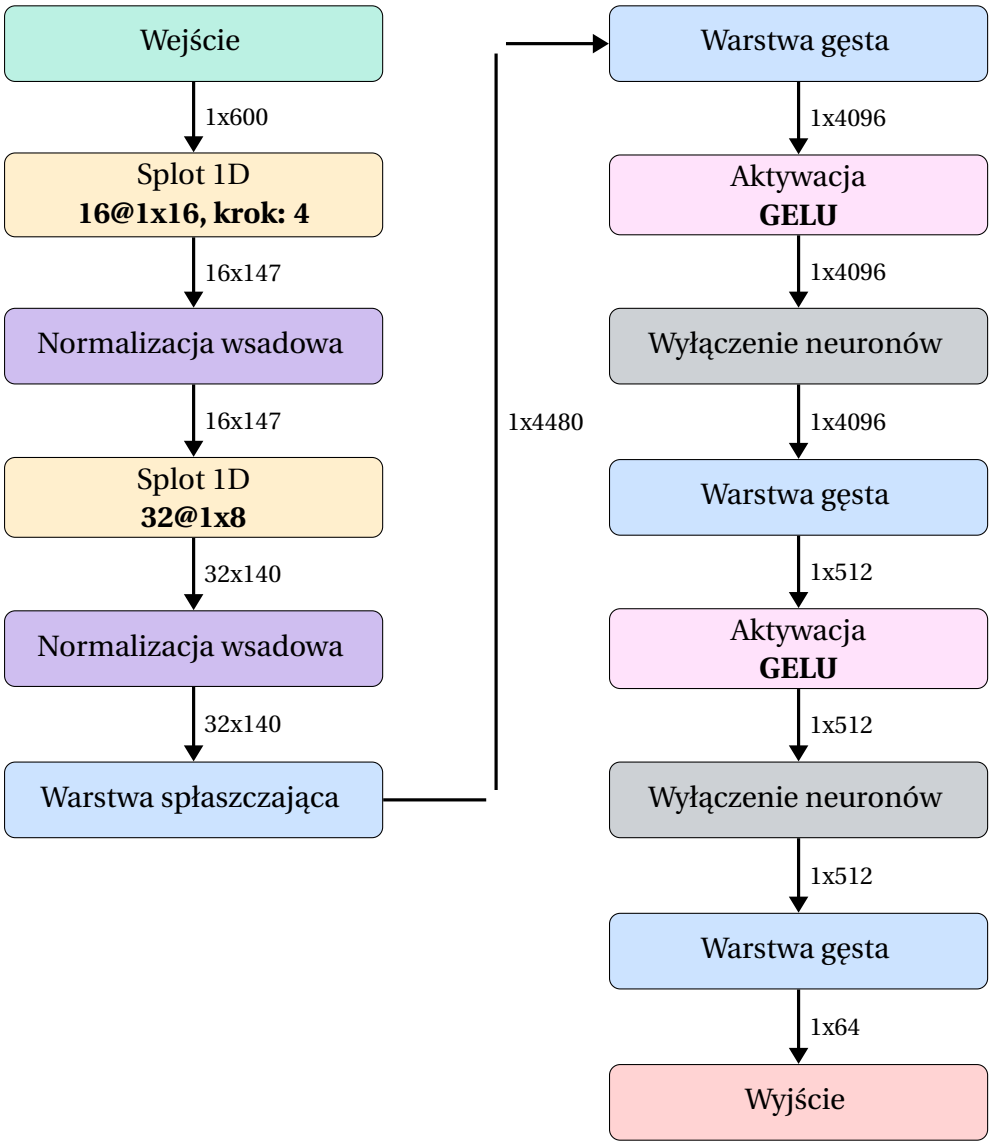
Ponadto w obu aplikacjach oraz bibliotece wykorzystywana jest biblioteka *serde***Rust:serde** umożliwiającą serializację i deserializację danych do różnych formatów oraz biblioteka *rand***Rust:rand** zapewniająca generator liczb pseudolosowych.



Rysunek 4.4. KMer



Rysunek 4.5. NN



Rysunek 4.6. NNe Model

4.5.3. Narzędzia

W pracy zostały wykorzystane następujące narzędzia:

- *cargo* jako menedżer pakietów i system budowania w Rust,
- *rustup* do automatycznego zarządzania wersjami Rust,
- *clippy* do statycznej analizy kodu w Rust,
- *rustfmt* do automatycznego formatowania kodu źródłowego w Rust,
- *cargo test* do przeprowadzania testów jednostkowych,
- *git* jako system kontroli wersji, umożliwiający śledzenie zmian oraz zarządzanie historią kodu.

4.6. Interfejs użytkownika

4.6.1. Opis interfejsu użytkownika

4.6.2. Instrukcja użytkownika

4.7. Testy

4.7.1. Testy3 jednostkowe

5. Eksperymenty

Przeprowadzono eksperymenty jakościowe oraz wydajnościowe dla trzech metod oraz dodatkowo dla klasyfikacji taksonomicznej.

5.1. Zbiory danych

Do przeprowadzenia eksperymentów wykorzystano zbiór *CAMI II Toy Human Microbiome Project*, z którego wylosowano zbiór 2047 sekwencji genetycznych, które nie były wykorzystane w procesie uczenia sieci neuronowej. Zbiór następnie został podzielony na podzbiory o wielkościach n^k dla $k \in [0,10]$.

5.2. Miara jakości

Do oceny jakości wyników pełnej klasyfikacji taksonomicznej został wykorzystany zmodyfikowany indeks Jaccarda, wyrażony wzorem:

$$\text{Miara jakosci} = \frac{1}{||R \cup E||} \sum_{o \in (R \cup E)} \frac{min(R_o, E_o)}{max(R_o, E_o)} \tag{5.1}$$

gdzie

R, E – zbiory organizmów referencyjnych i otrzymanych,

E_o, R_o – jakość organizmów w zbiorach E i R.

Wykorzystano również dodatkowe miary jakości, które oceniają tworzone klastry w wyniku wykorzystania różnych metod określania niepodobieństwa. Pierwszą z nich jest znormalizowana informacja wzajemna (ang. Normalized mutual information, NMI) i służy do oceny klastrów, drugą jest czułość wykorzystywaną do oceny reprezentantów (ang. *sensitivity*). Miary zdefiniowane są jako:

$$sensitivity = \frac{\text{liczba wyników prawdziwie dodatnych}}{\text{liczba wyników prawdziwie dodatnych} + \text{liczba wyników fałszywie ujemnych}} \tag{5.2}$$

$$NMI = \frac{I(X;Y)}{\sqrt{H(X) \cdot H(Y)}} \tag{5.3}$$

gdzie

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)},$$

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x),$$

$I(X;Y)$ – informacja wzajemna między zbiorami X oraz Y,

$H(X)$ – entropia zbioru X,

$H(Y)$ – entropia zbioru Y,

$p(x)$ – prawdopobieństwo zajścia zdarzenia x,

$p(x,y)$ – wspólny rozkład prawdopodobieństwa X oraz Y .

5.3. Procedura

Eksperymenty zostały przeprowadzone przy wykorzystaniu zbudowanego narzędzia, które monitorowało wykorzystanie procesora oraz pamięci przez eksperyment. Maksymalny czas eksperymentu ustawiono na 4 godziny. Liczbę grup ustawiono na $\sqrt{n}$, gdzie n to liczba sekwencji w danym eksperymencie.

1. Badania jakościowe
    - opis bazy / referencji
    - miara jakości
    - 2-3 eksperymenty
2. Badania wydajnościowe
    - pamięć, CPU
    - 2-3 eksperymenty (w zależności od parametrów)
3. Wnioski

6. Podsumowanie

TODO [ok. 10 stron]

1. Najważniejsze cechy wytworzonego rozwiązania
2. Plan rozbudowy

Bibliografia **knuth1984texbook**.

1. Spis literatury [30 prac]
2. Spis rysunków