Scientific Data Extraction App

Cel aplikacji

Scientific Data Extraction App to narzędzie przeznaczone do przeglądania i analizy danych biologicznych zawierających komórki raka piersi. Aplikacja umożliwia wyszukanie związku na wykresie i podgląd jego szczegółów. Każdy punkt na wykresie przedstawia parę: związek i jego skoncentrowanie, który jest pokolorowany w zależności od poziomu skoncentrowania lub wartości MOA. Po wybraniu punktu wyświetlają się szczegółowe dane takie jak: nazwa związku, wartość SMILES, wartość MOA, stężenie.

Funkcjonalności aplikacji

1. **Interaktywna wizualizacja danych**
   Użytkownicy mogą przeglądać dane w formie interaktywnych wykresów w przeglądarce, gdzie punkty są kolorowane w zależności od poziomu stężenia lub wartości MOA.
2. **Szczegółowe informacje o związkach**
   Po wybraniu punktu na wykresie wyświetlane są szczegółowe dane dotyczące wybranego związku, takie jak: nazwa, wartość SMILES, wartość MOA oraz stężenie.

Uogólniona struktura projektu

Strukturę projektu można podzielić na **frontend** i **backend**. Głównym zadaniem backendu jest analiza plików .csv oraz zdjęć w formacie .tif zgromadzonych na stronie: [BBBC021](). Zdjęcia przed analizą programu należy przetworzyć przez aplikację **Cell Profiller** z użyciem pipeline'u dostępnego na wyżej wymienionej stronie.

Główne elementy oprogramowania

Klasa Program

Główna klasa odpowiedzialna za:

- utworzenie bazy danych wraz z tabelami,
- modyfikowanie plików .csv i przystosowanie ich do analizy,
- wypełnienie bazy początkowymi danymi,
- wyliczenie wektorów rozmieszczenia punktów na podstawie danych zebranych z plików powstałych po analizie przez program **Cell Profiller**.

Wyliczone dane są przetwarzane i przekazywane do modułu aplikacji (app), który, dzięki wykorzystaniu technologii **FastAPI**, udostępnia je w warstwie front-endowej.

Opis klas zajmujących się przetwarzaniem danych

1. **Klasa DatabaseCreator**
   Zarządza połączeniem z bazą danych oraz tworzeniem tabel. Obsługuje cztery rodzaje tabel (**compounds**, **images**, **color_by_concentration**, **color_by_moa**) i zatwierdza zmiany po każdej operacji. Automatycznie zamyka połączenie z bazą danych po zakończeniu pracy.

2. **Klasa DatabaseFiller**
   Wypełnia bazę danych informacjami o związkach chemicznych, ich stężeniach oraz wartościach MOA na podstawie danych z plików CSV. Tworzy unikalne kolory dla stężeń i MOA, przypisując je do odpowiednich rekordów w bazie. Działa w oparciu o interfejs bazy danych, zapewniając spójność i integralność danych.

3. **Klasa DatabaseInterface**
   Interfejs definiujący zestaw abstrakcyjnych metod do operacji na bazie danych, takich jak:
   - łączenie się z nią,
   - zamykanie połączenia,
   - dodawanie danych do tabel,
   - aktualizowanie rekordów,
   - pobieranie danych.
     Wszystkie te operacje muszą być zaimplementowane w klasie dziedziczącej. Celem jest zapewnienie jednolitego sposobu obsługi bazy danych w aplikacji.

4. **Klasa SQLiteDatabase**
   implementuje wymieniony wyżej interfejs

5. **Klasa Paths**
   Zawiera stałe przechowujące ścieżki do różnych zasobów w projekcie. Używa modułu **Path** do tworzenia dynamicznych ścieżek, zależnych od bieżącego katalogu roboczego. Przechowywane są tu ścieżki do folderów bazy danych, zasobów oraz plików CSV.

6. **Klasa CsvFormatter**
   Służy do przetwarzania plików CSV, usuwając określoną liczbę kolumn i zapisując przetworzone dane w nowym folderze. Umożliwia dynamiczne tworzenie folderów wyjściowych oraz modyfikowanie ścieżek plików, dodając do nich "_formatted". Pliki CSV są przetwarzane w częściach, co pozwala na oszczędność pamięci przy pracy z dużymi danymi. Główna metoda **run_formatter** wykonuje całą operację, analizując pliki z katalogu wejściowego i zapisując przetworzone wersje w katalogu wyjściowym.

7. **Klasa CalculateVectors**
   Wykonuje obliczenia wektorów średnich na podstawie plików CSV, a następnie przekształca je do przestrzeni 2D za pomocą algorytmu **UMAP**. Umożliwia iterację po folderach, przetwarzanie danych z plików oraz powiązanie obliczonych wektorów z obrazami zapisanymi w bazie danych. Obliczone współrzędne (x, y) są zapisywane w bazie danych, a obrazy związane z danymi są przypisywane do odpowiednich związków chemicznych. Proces obejmuje wczytywanie danych z CSV, tworzenie wektorów, konwersję do 2D oraz zapis wyników do bazy danych.

API do zarządzania związkami

Sekcja kodu zajmująca się wystawianiem endpointów w aplikacji **FastAPI** definiuje cztery różne trasy:

1. **/compounds**
   Pobiera wszystkie związki chemiczne z bazy danych za pomocą funkcji **get_all_compounds()** z klasy **Repository**.
2. **/compounds/colored_by_concentration**
   Pobiera wszystkie związki chemiczne z bazy danych, uwzględniając kolorowanie według stężenia, dzięki funkcji **get_all_compounds_colored_by_concentration()**.
3. **/compounds/colored_by_moa**
   Pobiera związki chemiczne z kolorowaniem według wartości MOA za pomocą **get_all_compounds_colored_by_moa()**.
4. **/compound/details/{compound_name}/{compound_concentration}**
   Pobiera szczegóły dotyczące konkretnego związku na podstawie nazwy i stężenia, używając **get_compound_details()**.

Klasa Repository odpowiada za komunikację z bazą danych, pobieranie danych o związkach oraz ich szczegółów, zwracając je w odpowiedniej strukturze.

Opis metod

DatabaseInterface, SQLiteDatabase

1. **`connect`**:
   Służy do nawiązania połączenia z bazą danych SQLite. W implementacji powinna otwierać połączenie z określonym plikiem bazy danych.
2. **`close`**:
   Zamyka istniejące połączenie z bazą danych, uwalniając zasoby.
3. **`commit`**:
   Zatwierdza bieżącą transakcję w bazie danych, zapisując zmiany (np. wstawienia, aktualizacje).
4. **`create_table_compounds`**:
   Tworzy tabelę w bazie danych o nazwie `compounds`, w której będą przechowywane informacje o związkach chemicznych (np. nazwa, stężenie, SMILES, aktywność).
5. **`insert_into_table_compounds`**:
   Wstawia dane do tabeli `compounds`. Przyjmuje parametry takie jak:
   - `compound_name`: nazwa związku chemicznego,
   - `compound_concentration`: stężenie związku,
   - `smiles`: reprezentacja SMILES,
   - `is_active`: informacja o tym czy związek został już przetworzony.

Operacje na tabeli `compounds`:

1. **`update_compounds_moa(compound_name, moa_id)`**
   Aktualizuje przypisanie MOA dla związku o podanej nazwie.
2. **`updata_compounds_empty_moa(moa_id)`**
   Przypisuje MOA do wszystkich związków, które aktualnie nie mają przypisanego MOA.
3. **`update_compound_coordinates(compound_id, new_x, new_y, is_active)`**
   Aktualizuje współrzędne (`new_x`, `new_y`) i stan aktywności (`is_active`) związku o podanym identyfikatorze.
4. **`fetch_compound_by_name_and_concentration(compound_name, concentration)`**
   Pobiera szczegóły związku na podstawie jego nazwy i stężenia.
5. **`fetch_all_compounds()`**
   Pobiera dane wszystkich związków zapisanych w tabeli `compounds`.

6. **`fetch_all_compounds_colored_by_concentration()`**
   Pobiera dane wszystkich związków z kolorami przypisanymi na podstawie ich stężenia.
7. **`fetch_all_compounds_colored_by_moa()`**
   Pobiera dane wszystkich związków z kolorami przypisanymi na podstawie ich MOA.
8. **`fetch_compound_details(compound_name, compound_concentration)`**
   Pobiera szczegółowe informacje o związku na podstawie jego nazwy i stężenia.

Operacje na tabeli `images`:

1. **`create_table_images()`**
   Tworzy tabelę `images` do przechowywania danych o obrazach związanych z konkretnymi związkami chemicznymi.
2. **`insert_into_table_images(compound_id, folder_path, dapi, tubulin, actin)`**
   Wstawia dane do tabeli `images`, zawierające ID związku, ścieżkę folderu oraz ścieżkę do obrazów (DAPI, Tubulin, Actin).

Operacje na tabelach kolorów:

1. **`create_table_color_by_concentration()`**
   Tworzy tabelę do przechowywania kolorów przypisanych na podstawie stężenia.
2. **`insert_into_color_by_concentration(r, g, b)`**
   Wstawia kolor (RGB) do tabeli kolorów przypisanych na podstawie stężenia.
3. **`create_table_color_by_moa()`**
   Tworzy tabelę do przechowywania kolorów przypisanych na podstawie mechanizmu działania (MOA).
4. **`update_compounds_color_concentration(concentration, color_id)`**
   Aktualizuje związek przypisując kolor na podstawie `color_id`.
5. **`insert_into_color_table_by_moa(moa, concentration, r, g, b)`**
   Wstawia kolor (RGB) do tabeli kolorów, przypisując go określonemu MOA i stężeniu.

DatabaseCreator

1. Metoda `create_table` wybiera odpowiednią metodę do tworzenia tabeli w bazie danych na podstawie przekazanej nazwy tabeli (`table_name`). Korzysta z metod do tworzenia tabel opisanych powyżej

DatabaseFiller

1. **`set_compounds`**

   Wczytuje dane obrazów z pliku CSV (ścieżka w `Paths.IMAGES_CSV_PATH`).

   Wyciąga unikalne kombinacje związku chemicznego i stężenia, a następnie zapisuje je jako atrybut `self.compounds`.

2. **`fill_initial_data`**

   Wywołuje metody do wypełnienia tabel w bazie danych danymi startowymi:

   - `fill_compounds_table` – dane o związkach chemicznych.
   - `fill_color_concentration_tables` – kolory przypisane do stężeń.
   - `fill_color_moa_tables` – kolory przypisane do MOA.

3. **`fill_compounds_table`**

   Wczytuje dane o związkach i ich SMILES z pliku CSV (ścieżka w `Paths.COMPOUND_CSV_PATH`).

   Tworzy mapowanie `compound → smiles`.

   Iteruje przez unikalne związki i wstawia je do tabeli `compounds` w bazie danych:

   - Nazwa związku.
   - Stężenie.
   - SMILES (jeśli istnieje w mapowaniu).
   - Domyślna wartość aktywności `0`.

   Zatwierdza transakcję w bazie (`commit`).

Zarządzanie kolorami w oparciu o stężenia i MOA:

1. **`generate_unique_colors`**
   - Generuje unikalne kolory (RGB) dla każdej wartości stężenia.
   - Używa zbioru `used_colors`, aby zapewnić, że każdy kolor jest unikalny.
   - Zwraca mapowanie `concentration → (r, g, b)`.
2. **`fill_color_concentration_tables`**
   - Wyciąga unikalne stężenia z `self.compounds`.
   - Generuje dla nich unikalne kolory (`generate_unique_colors`).

    - Wstawia kolory do tabeli `color_by_concentration` i zapisuje ich identyfikatory (`color_id`).
    - Przypisuje identyfikatory kolorów do odpowiednich stężeń w tabeli `compounds`.

FormatCsvFiles

`__delete_some_colls_in_csv(self, file, output_file, columns_to_skip)`

- **Wejście**:
    - `file`: Ścieżka do wejściowego pliku CSV.
    - `output_file`: Ścieżka do pliku wynikowego.
    - `columns_to_skip`: Lista indeksów kolumn do usunięcia.
- **Działanie**:
    - Wczytuje plik CSV w częściach (`chunksize=1000`), co pozwala na przetwarzanie dużych plików.
    - Usuwa określone kolumny (`columns_to_skip`) z każdej części.
    - Wypełnia puste wartości (`NaN`) zerami.
    - Zapisuje przetworzony fragment do pliku wynikowego:
        - Pierwsza część zapisuje nagłówki (`header=True`), kolejne już nie.
- **Cel**: Umożliwia efektywne przetwarzanie dużych plików CSV bez ładowania ich w całości do pamięci.

`modify_file_path(self, file_path)`

- **Wejście**:
    - `file_path`: Ścieżka do oryginalnego pliku CSV.
- **Działanie**:
    - Sprawdza, czy plik należy do folderu wejściowego (`self.input_folder_path`), w przeciwnym razie zgłasza wyjątek `FileNotFoundError`.
    - Zmienia ścieżkę wejściową na ścieżkę wyjściową (`self.output_folder_path`).
    - Dodaje sufiks `_formatted` do nazwy pliku przed rozszerzeniem.
- **Cel**: Generuje ścieżkę dla sformatowanego pliku, zachowując logiczną strukturę nazw.

`create_folder(self, new_folder_name)`

- **Wejście**:
    - `new_folder_name`: Nazwa nowego folderu.
- **Działanie**:
    - Tworzy nowy folder w ścieżce wyjściowej (`self.output_folder_path`), jeśli nie istnieje.
- **Cel**: Replikuje strukturę katalogów wejściowych w folderze wyjściowym.

`get_csv_files(self)`

- **Działanie**:
    - Iteruje przez pliki w `self.input_folder_path` i wyszukuje wszystkie pliki z rozszerzeniem `.csv`.
    - Przy pierwszej iteracji replikowana jest struktura katalogów wejściowych w folderze wyjściowym (`self.output_folder_path`).
    - Zwraca listę ścieżek do plików CSV.
- **Cel**: Zbiera pliki CSV i przygotowuje strukturę katalogów dla wyników.

`run_formatter(self)`

- **Działanie**:
    1. Tworzy główny folder wyjściowy (`self.output_folder_path`), jeśli nie istnieje.
    2. Pobiera listę plików CSV do przetwarzania za pomocą `get_csv_files`.
    3. Określa indeksy kolumn do usunięcia (`cols_to_delete`).
    4. Iteruje przez pliki CSV i:
        - Sprawdza, czy nazwa pliku nie zawiera `"Image"`.
        - Usuwa wybrane kolumny i zapisuje wynik do nowego pliku, używając metod:
            - `__delete_some_colls_in_csv` do przetwarzania pliku.
            - `modify_file_path` do generowania ścieżki pliku wynikowego.
- **Cel**: Automatyzuje proces formatowania wielu plików CSV, tworząc wyniki w osobnym folderze.

CalculateVectors

1.`calcualte_averange_from_file(self, folder_name)`

- **Cel**: Oblicza średnią wartość danych numerycznych z pliku CSV.
- **Działanie**:
    - Wczytuje dane z pliku wiersz po wierszu.
    - Używa bufora (`buffer`) do przetwarzania danych w blokach (rozmiar `chunk_size`).
    - Oblicza sumę i liczbę elementów dla każdego bloku.
    - Po zakończeniu iteracji zwraca średnią (`total_sum / total_count`).

2. `create_vector(self, file_paths)`

- **Cel**: Tworzy wektor na podstawie średnich wartości z trzech plików CSV.
- **Działanie**:
    - Iteruje po liście ścieżek do plików CSV.
    - Oblicza średnie wartości dla każdego pliku za pomocą `calcualte_averange_from_file`.
    - Zwraca średnią z trzech plików jako wektor.

3. `iterate_formatted_folder(self)`

- **Cel**: Iteruje przez sformatowany folder, tworzy wektory dla każdego podfolderu i łączy je z danymi obrazów.
- **Działanie**:
    - Iteruje przez wszystkie podfoldery w folderze `self.formatted_folder_path`.
    - Dla każdego podfolderu:
        - Generuje wektor na podstawie plików CSV za pomocą `create_vector`.
        - Pobiera listę obrazów (`find_images`) i dodaje dane do `self.data`.

4. `find_images(self, folder_name)`

- **Cel**: Znajduje listy nazw plików obrazów dla danego folderu.
- **Działanie**:
    - Sprawdza, czy folder i plik CSV (`bbbc021_Image.csv`) istnieją.
    - Wczytuje dane z pliku CSV i zwraca listy nazw plików obrazów (`DAPI`, `Tubulin`, `Actin`).
- **Uwagi**:
    - Zgłasza wyjątek, jeśli folder lub plik nie istnieje.

5. `convert_vectors_to_2D(self)`

- **Cel**: Konwertuje wektory na reprezentację 2D za pomocą UMAP.
- **Działanie**:
    - Tworzy listę wektorów z danych (`self.data`).
    - Stosuje algorytm UMAP, aby zmniejszyć wymiarowość wektorów do 2D.
    - Dodaje współrzędne (`x`, `y`) do odpowiednich wpisów w `self.data`.

6. `save_converted_data_to_database(self)`

- **Cel**: Zapisuje przetworzone dane do bazy danych.
- **Działanie**:
    - Iteruje przez dane w `self.data`.
    - Pobiera informacje o związku chemicznym (compound) na podstawie pierwszego obrazu `DAPI`.
    - Aktualizuje współrzędne w bazie danych, łącząc stare i nowe wartości:
        - Jeśli związek jest aktywny (`is_active == 1`), współrzędne są uśredniane.
        - Jeśli nie, ustawiane są nowe współrzędne.
    - Iteruje przez listy obrazów (`DAPI`, `Tubulin`, `Actin`) i dodaje dane o obrazach do bazy.
    - Wykonuje `commit` po każdej iteracji.

Repository

1. `get_all_compounds(self)`

- **Cel**: Pobiera wszystkie związki chemiczne z bazy danych i zwraca je w formacie JSON.
- **Działanie**:
    - Wywołuje metodę `fetch_all_compounds` bazy danych.
    - Iteruje przez wyniki i mapuje każde wiersze do obiektu zawierającego:
        - `name`: Nazwa związku.
        - `concentration`: Stężenie.
        - `x`, `y`: Współrzędne w przestrzeni 2D.

2. `get_all_compounds_colored_by_concentration(self)`

- **Cel**: Pobiera wszystkie związki chemiczne wraz z przypisanymi kolorami w zależności od ich stężenia.
- **Działanie**:
    - Wywołuje metodę `fetch_all_compounds_colored_by_concentration` bazy danych.
    - Mapuje wyniki do obiektów zawierających:
        - `name`, `concentration`, `x`, `y` – analogicznie jak w `get_all_compounds`.
        - `color`: Słownik z wartościami składowymi koloru RGB.

3. `get_all_compounds_colored_by_moa(self)`

- **Cel**: Pobiera wszystkie związki chemiczne wraz z przypisanymi kolorami w zależności od ich MOA.
- **Działanie**:
    - Wywołuje metodę `fetch_all_compounds_colored_by_moa` bazy danych.
    - Analogicznie jak w funkcji wyżej, mapuje wyniki do obiektów z dodatkowymi danymi o kolorze RGB.

4. `get_compound_details(self, compound_name, compound_concentration)`

- **Cel**: Pobiera szczegółowe informacje o związku chemicznym na podstawie nazwy i stężenia.
- **Działanie**:
    - Wywołuje metodę `fetch_compound_details` bazy danych z podanymi argumentami.
    - Zwraca pojedynczy obiekt zawierający:
        - `smiles`: Reprezentację chemiczną w formacie SMILES.
        - `moa`: Mechanizm działania.
        - `moa_concentration`: Stężenie związane z mechanizmem działania.