Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki

Projekt dyplomowy

Biblioteka Datasets dla Elixira

Datasets library for Elixir

Autorzy: Radosław Rolka, Weronika Wojtas
Kierunek studiów: Informatyka
Opiekun pracy: dr inż. Aleksander Smywiński-Pohl

Kraków, 2025

Spis treści

Rozdział 1

Cel prac i wizja produktu

Celem pracy jest stworzenie biblioteki w języku Elixir, która umożliwi łatwe pobieranie, przetwarzanie i zarządzanie zbiorami danych, które są powszechnie wykorzystane w uczeniu maszynowym. Biblioteka powinna oferować szeroki wybór gotowych zbiorów danych, a także możliwość dodawania własnych.

1.1. Opis dziedziny problemu

Praca z modelami uczenia maszynowego jest ściśle związana z wykorzystaniem danych, które stanowią fundament dla procesów trenowania i walidacji algorytmów. Dostęp do dobrze przygotowanych i różnorodnych zbiorów danych jest kluczowy dla efektywnego rozwoju i nauki modeli. Zbiory danych muszą być nie tylko obszerne i reprezentatywne, ale również odpowiednio przetworzone i znormalizowane, co często stanowi wyzwanie ze względu na duży nakład czasu i zasobów wymaganych w procesie przygotowania. Nieodłącznym elementem pracy z danymi jest ich czyszczenie, skalowanie, oraz odpowiednie formatowanie, które umożliwia integrację danych wejściowych z modelami. Ponadto, same dane często pochodzą z różnorodnych źródeł i są zapisane w różnych formatach, co dodatkowo komplikuje ich użyteczność bezpośrednio po pozyskaniu.

1.2. Motywacja

Podczas studiów zainteresowaliśmy się językami programowania funkcyjnego, szczególnie Elixirem. Choć na początku jego podejście może wydawać się nietypowe, szybko dostrzegliśmy, jak prosty i elegancki jest ten język. Podczas pracy z Elixirem zauważyliśmy, że brakuje w nim biblioteki do zarządzania zbiorami danych, która w innych językach jest szeroko stosowana, szczególnie w sztucznej inteligencji.

Postanowiliśmy, że to będzie temat naszej pracy inżynierskiej. Chcemy stworzyć bibliotekę w Elixirze, inspirowaną Hugging Face Datasets [4], która pozwoli programistom łatwiej pracować ze zbiorami danych i ułatwi dostęp do nich.

1.3. Rola produktu

Głównym celem biblioteki jest uproszczenie i automatyzacja zarządzania, przetwarzania oraz optymalizacja zbiorów danych. Umożliwienie łatwego dostępu do różnorodnych zbiorów danych pozwoli na szybsze rozpoczęcie pracy nad projektami, eliminując konieczność manualnego zbierania i konfigurowania danych. Integracja funkcji automatycznego czyszczenia, normalizacji, skalowania i augmentacji danych znacząco zredukuje czasochłonne procesy przygotowywania danych, co jest zazwyczaj barierą w szybkim prototypowaniu i testowaniu modeli uczenia maszynowego. Użytkownikami końcowymi projektowanej biblioteki są przede wszystkim analitycy, specjaliści od uczenia maszynowego oraz studenci zajmujący się analizą danych i sztuczną inteligencją, którym to narzędzie ma za zadanie zwiększyć produktywność poprzez automatyzację rutynowych zadań.

1.4. Obszary funkcjonalne

1. **Pobieranie i zarządzanie zbiorami danych**

    - Pobieranie gotowych zbiorów danych z różnych źródeł: Implementacja mechanizmów umożliwiających pobieranie danych z platform takich jak Hugging Face Hub [1], Kaggle [3] czy innych repozytoriów.
    - Dodawanie własnych zbiorów danych: Możliwość integracji i zarządzania własnymi zestawami danych w systemie.

2. **Przeglądanie zbiorów danych**

    - Łatwe przeglądanie dostępnych zbiorów danych: Interfejsy umożliwiające szybki podgląd i analizę dostępnych danych.
    - Filtrowanie zbiorów danych: Narzędzia do selekcji danych na podstawie określonych kryteriów.

3. **Przetwarzanie i transformacja danych**

    - Czyszczenie danych: Funkcje do usuwania błędów i niekompletnych rekordów.
    - Normalizacja danych: Metody standaryzacji wartości w zbiorach danych.
    - Tokenizacja: Proces dzielenia tekstu na mniejsze jednostki, takie jak słowa czy zdania.
    - Tworzenie podzbiorów danych: Możliwość dzielenia większych zbiorów na mniejsze, bardziej zarządzalne części.

4. **Przykłady rozwiązań**

    - Przykłady użycia: Praktyczne scenariusze i case studies demonstrujące zastosowanie poszczególnych funkcji.

1.5. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne odgrywają kluczową rolę w zapewnieniu, że stworzona biblioteka nie tylko spełni swoje zadania funkcjonalne, ale również będzie przyjazna dla użytkownika. Poniżej przedstawiono główne wymagania niefunkcjonalne dla projektu:

- Wydajność - Biblioteka powinna efektywnie zarządzać i przetwarzać duże zbiory danych z minimalnym opóźnieniem.
- Kompatybilność - Interfejs powinien być kompatybilny z różnymi systemami operacyjnymi i integrować się z istniejącymi popularnymi narzędziami i bibliotekami w ekosystemie Elixira.
- Dokumentacja - Kompletna i zrozumiała dokumentacja techniczna jest niezbędna, by użytkownicy mogli efektywnie wykorzystywać wszystkie funkcje biblioteki.

1.6. Przegląd dostępnych rozwiązań

Jednym z głównych narzędzi w tej dziedzinie rozwiązań jest biblioteka datasets od Hugging Face [4], która jest szeroko stosowana w społeczności uczenia maszynowego. Biblioteka datasets oferuje łatwy dostęp do szerokiej gamy zbiorów danych w różnych językach programowania. Oferuje ona również różnorodne narzędzia do przetwarzania i transformacji danych.

W kontekście Elixira, który nadal jest dynamicznie rozwijającym się językiem, nie istnieje jeszcze takie narzędzie, które w pełni odpowiadałoby potrzebom użytkowników w zakresie zarządzania i przetwarzania danych dla uczenia maszynowego, co tworzy przestrzeń na rynku dla nowego rozwiązania, które może lepiej odpowiadać na unikalne potrzeby społeczności Elixira, zwiększając efektywność ich pracy dzięki specjalizowanym narzędziom dostosowanym do ich środowiska i metod pracy.

1.7. Analiza technologiczna

Stos technologiczny został zaprojektowany z myślą o maksymalnym wykorzystaniu możliwości języka Elixir oraz jego ekosystemu. Do obliczeń numerycznych oraz operacji na tensorach wykorzystamy bibliotekę NX [5]. Biblioteka zapewnia wydajność w przeprowadzaniu operacji matematycznych, szczególnie w kontekście obliczeń związanych z dużymi zbiorami danych i sztuczną inteligencją. NX oferuje wsparcie dla operacji na tensorach, które są kluczowe w procesach uczenia maszynowego oraz analizy danych.

Zintegrowany z NX jest także Explorer [2], który będziemy wykorzystywać w naszej pracy do efektywnego zarządzania i analizy danych. Explorer to biblioteka, która umożliwia pracę z dwoma głównymi typami struktur danych: seriami, oraz dataframe'ami. Te struktury pozwalają na wygodne i szybkie eksplorowanie danych, co jest szczególnie istotne podczas analizy informacji. Explorer, jako backend, korzysta z Polars, biblioteki napisanej w języku Rust co przekłada się na znaczną poprawę wydajności w obliczeniach z dużymi zbiorami.

Do współpracy z modelami głębokiego uczenia maszynowego w naszym projekcie zastosujemy bibliotekę Bumblebee [8], która pozwala na łatwą integrację z pretrenowanymi modelami sieci neuronowych. Bumblebee umożliwia dostęp do popularnych modeli, które zostały udostępnione przez platformy sztucznej inteligencji, takie jak Hugging Face Transformers [9]. Ta

biblioteka umożliwi łatwą implementację i wykorzystanie zaawansowanych modeli AI w naszej pracy, co pozwoli na efektywne wdrożenie algorytmów uczenia maszynowego i głębokiego uczenia w środowisku Elixira.

1.8. Analiza ryzyka

W procesie projektowania i rozwijania nowej biblioteki istnieje wiele potencjalnych ryzyk, których zidentyfikowanie pozwala na lepsze przygotowanie, co z kolei zwiększa szanse na pomyślne zakończenie projektu. Są to między innymi:

- Adaptacja przez społeczność - Jako że Elixir jest stosunkowo mniej popularny niż inne języki wykorzystywane w dziedzinie uczenia maszynowego, takie jak Python, istnieje ryzyko, że biblioteka nie zyska szerokiego grona użytkowników. Promocja biblioteki i demonstrowanie jej wartości w rzeczywistych projektach będzie kluczowe.
- Integracja z istniejącymi narzędziami - Problemy z integracją nowej biblioteki z już istniejącymi ekosystemami i narzędziami, których niekompatybilność może powstrzymać potencjalnych użytkowników przed korzystaniem z biblioteki.
- Obsługą dużych zbiorów danych - Możliwe, że biblioteka nie będzie w stanie efektywnie procesować dużych zbiorów danych lub że wystąpią problemy z wydajnością.
- Niedostateczne testowanie - Niewystarczające testowanie w różnych środowiskach i scenariuszach użytkowania może prowadzić do niezauważonych błędów, które ujawnią się dopiero po wdrożeniu biblioteki.

1.9. Podsumowanie

Projekt ma na celu stworzenie biblioteki w języku Elixir, która będzie odpowiadała funkcjonalności biblioteki Hugging Face Datasets [4], umożliwiając łatwe pobieranie, przetwarzanie i zarządzanie zbiorami danych używanymi w uczeniu maszynowym. Biblioteka ta oferować będzie funkcje takie jak pobieranie gotowych zbiorów danych z różnych źródeł, możliwość dodawania własnych zbiorów danych, filtrowanie i przeglądanie dostępnych zbiorów, a także przetwarzanie danych (czyszczenie, normalizacja, tokenizacja). Użytkownicy będą mogli tworzyć podzbiory danych oraz integrować bibliotekę z innymi narzędziami w Elixirze, takimi jak Nx [5], Explorer [2] i Bumblebee [8]. Projekt zakłada również dostarczenie pełnej dokumentacji oraz przykładów użycia.

Rozdział 2

Zakres funkcjonalności

Celem niniejszego rozdziału jest przedstawienie specyfikacji funkcjonalnej projektowanej biblioteki. Specyfikacja została opracowana na podstawie analizy potrzeb użytkowników i rozmów konsultacyjnych z zainteresowanymi stronami.

2.1. Charakterystyka użytkownika

System zakłada istnienie jednego głównego rodzaju użytkownika – **programisty pracującego z językiem Elixir**, który zajmuje się tworzeniem, trenowaniem lub walidacją modeli uczenia maszynowego. Użytkownicy ci mogą być częścią większych zespołów badawczo-rozwojowych lub niezależnymi deweloperami.

Zakłada się, że użytkownik:

- posiada podstawową lub zaawansowaną znajomość języka Elixir,
- zna podstawy uczenia maszynowego oraz pracy z danymi,
- wymaga efektywnego i prostego dostępu do przygotowanych zbiorów danych,
- oczekuje narzędzi ułatwiających wstępne przetwarzanie danych, takich jak czyszczenie, normalizacja, tokenizacja.

2.2. Charakterystyka systemów współpracujących

Tworzona biblioteka do zarządzania zbiorami danych w języku Elixir zakłada ścisłą współpracę z wybranym zestawem zewnętrznych narzędzi i bibliotek, które pełnią kluczową rolę w zapewnieniu pełnej funkcjonalności systemu. Integracja tych komponentów pozwala na maksymalne wykorzystanie potencjału języka Elixir w kontekście przetwarzania danych na potrzeby uczenia maszynowego. Poniżej przedstawiono charakterystykę najważniejszych współpracujących systemów:

- **Źródła zbiorów danych** – Biblioteka umożliwia pobieranie popularnych zbiorów danych wykorzystywanych w uczeniu maszynowym z różnych publicznych repozytoriów, takich jak Hugging Face Datasets [4]. W związku z tym wymagana jest integracja z usługami HTTP i obsługą różnorodnych formatów danych.

- **Nx** [5] – Biblioteka opiera się na integracji z narzędziem Nx, które zapewnia funkcje numeryczne i przetwarzanie tensorów. Współpraca z Nx pozwala na bezproblemowe przygotowanie danych do trenowania modeli uczenia maszynowego w Elixirze.
- **Explorer** [2] – Do eksploracji, filtrowania i transformacji danych tabularycznych wykorzystywana jest biblioteka Explorer , która umożliwia przetwarzanie danych w sposób zbliżony do narzędzi takich jak Pandas [7] w Pythonie. Dzięki temu użytkownik może łatwo analizować i przygotowywać dane w ramach jednego ekosystemu.
- **Bumblebee** [8] – W celu dalszego wykorzystania przetworzonych danych, możliwa jest integracja z biblioteką Bumblebee, która dostarcza gotowe modele i narzędzia do pracy z NLP i uczeniem głębokim.
- **Lokalna pamięć masowa** – System wspiera lokalne przechowywanie danych oraz cache'owanie zbiorów, aby zminimalizować potrzebę wielokrotnego pobierania tych samych danych i zwiększyć wydajność pracy z dużymi zestawami.

2.3. Wymagania funkcjonalne

W poniższym rozdziale szczegółowo opisano wymagania funkcjonalne dla projektowanej biblioteki w języku Elixir, klasyfikując je zgodnie z metodologią MoSCoW [6] ukierunkowaną na najbardziej krytyczne aspekty funkcjonalności systemu.

Must Have

- Możliwość pobierania datasetów z zewnętrznych źródeł – biblioteka musi umożliwić użytkownikowi łatwy dostęp do zasobów danych oferowanych przez różne repozytoria.
- Integracja z narzędziami Elixir takimi jak Nx i Explorer – niezbędne jest zapewnienie kompatybilności i efektywnej współpracy narzędziowej.
- Pełna dokumentacja funkcji biblioteki – użytkownicy muszą mieć dostęp do jasnych i zrozumiałych instrukcji korzystania z biblioteki.

Should Have

- Możliwość dodawania własnych datasetów do biblioteki – funkcja ta pozwala użytkownikom na personalizację i rozszerzenie bazy danych.
- Narzędzia do czyszczenia i normalizacji danych – choć nie krytyczne, znacząco podnoszą wartość użytkową biblioteki.

Could Have

- Rozbudowane funkcje tokenizacji danych – ułatwiłyby przetwarzanie tekstu, zwiększając potencjalne obszary zastosowań biblioteki.
- Wtyczki wspierające nowsze frameworki i biblioteki w ekosystemie Elixir – mogą zwiększyć atrakcyjność biblioteki dla szerokiej grupy użytkowników.

Won't Have

- Automatyczne tłumaczenia dokumentacji na różne języki – choć przydatne w przyszłości, nie będą dostępne w pierwszej wersji produktu.
- Zaawansowane algorytmy sztucznej inteligencji do analizy danych – nie są planowane w aktualnym zakresie projektu.

2.4. Wymagania jakościowe

Poniżej przedstawione zostały wymagania niefunkcjonalne, które mają na celu zapewnienie wysokiej jakości tworzonej biblioteki oraz komfortu jej użytkowania. Odpowiednie spełnienie tych wymagań wpłynie pozytywnie na łatwość integracji, rozwój i utrzymanie projektu w dłuższym okresie.

- **Czytelny i spójny interfejs API** – Interfejs biblioteki powinien być intuicyjny i dobrze zaprojektowany, umożliwiając użytkownikowi szybkie rozpoczęcie pracy bez konieczności zapoznawania się z nadmiernie rozbudowaną dokumentacją. Nazewnictwo funkcji, struktur i modułów powinno być spójne i zgodne z konwencjami języka Elixir.
- **Wydajność** – Biblioteka powinna umożliwiać efektywne operacje na dużych zbiorach danych, takich jak filtrowanie, czyszczenie czy transformacja. Szczególny nacisk powinien zostać położony na optymalizację operacji na dużych zbiorach dancyh oraz minimalizację zużycia pamięci i czasu przetwarzania.
- **Skalowalność** – Projekt powinien być skalowalny zarówno pod względem wielkości obsługiwanych danych. Powinien umożliwiać bezproblemowe dodawanie nowych źródeł danych, funkcjonalności i formatów danych bez konieczności istotnej przebudowy istniejącej architektury.
- **Bezpieczeństwo danych** – W przypadku integracji z zewnętrznymi źródłami danych, komunikacja powinna być realizowana z użyciem bezpiecznych protokołów (np. HTTPS). System powinien być odporny na wstrzykiwanie niepoprawnych danych oraz błędne formaty plików.
- **Odporność na błędy** – Biblioteka powinna zawierać mechanizmy wykrywania i obsługi błędów, umożliwiające użytkownikowi uzyskanie jasnych komunikatów w przypadku problemów z danymi, połączeniem lub działaniem funkcji.
- **Kompatybilność z ekosystemem Elixira** – Projekt musi zapewniać pełną kompatybilność z innymi bibliotekami wykorzystywanymi w uczeniu maszynowym w języku Elixir, w szczególności Nx [5], Explorer [2] i Bumblebee [8]. Powinien też działać na różnych platformach systemowych wspierających środowisko Elixira.
- **Dokumentacja** – Biblioteka powinna być opatrzona pełną dokumentacją techniczną, zawierającą opisy funkcji, struktur danych, przykłady użycia oraz wskazówki dotyczące integracji z innymi narzędziami. Dokumentacja powinna być dostępna zarówno w kodzie (np. jako modułowe `@doc`), jak i w formie zewnętrznej (np. README, przewodniki).
- **Łatwość w utrzymaniu i rozwoju** – Kod źródłowy powinien być przejrzysty, modularny, zgodny z dobrymi praktykami programistycznymi i łatwy do testowania oraz rozszerzania. Projekt powinien uwzględniać przyszłą rozbudowę, np. o nowe formaty danych, dodatkowe transformacje lub integracje.

- **Testowalność** – System powinien być w pełni testowalny. Moduły powinny być projektowane w sposób umożliwiający tworzenie testów jednostkowych oraz testów integracyjnych, co ułatwi utrzymanie wysokiej jakości kodu i szybką detekcję błędów w przyszłości.

2.5. Scenariusze użytkowania

Poniżej przedstawiono przykładowe scenariusze użytkowania systemu, które ilustrują typowe sytuacje, w jakich programista może korzystać z biblioteki. Scenariusze te odzwierciedlają główne funkcjonalności systemu i obrazują sposób interakcji użytkownika z interfejsem programistycznym (API) biblioteki.

- **Pobranie gotowego zbioru danych**

  Użytkownik chce szybko pobrać popularny zbiór danych w celu przetestowania modelu klasyfikacji tekstu. W tym celu korzysta z funkcji udostępnianych przez bibliotekę, które automatycznie pobierają dane z repozytorium, zapisują je lokalnie i przygotowują do dalszego przetwarzania.

- **Dodanie własnego zbioru danych**

  Użytkownik posiada własny zbiór danych zapisany w formacie CSV. Chce go załadować, wstępnie przefiltrować oraz znormalizować. Korzystając z biblioteki, użytkownik definiuje strukturę zbioru danych, wskazuje lokalizację pliku, a następnie stosuje dostępne funkcje do oczyszczenia danych i konwersji ich do odpowiedniego formatu.

- **Filtrowanie danych na podstawie kryteriów**

  Użytkownik analizuje zbiór danych zawierający opinie klientów i chce utworzyć podzbiór, który zawiera wyłącznie opinie pozytywne. Biblioteka umożliwia zastosowanie funkcji filtrowania na podstawie warunków logicznych, co pozwala szybko uzyskać interesujący użytkownika fragment danych.

- **Integracja danych z modelem uczenia maszynowego**

  Po przygotowaniu danych, użytkownik chce przesłać je jako tensory do modelu zaimplementowanego w bibliotece Bumblebee [8]. Biblioteka wspiera konwersję danych do struktury kompatybilnej z Nx[5] oraz umożliwia bezpośrednie przekazanie ich do dalszego przetwarzania przez model.

- **Przegląd dostępnych zbiorów danych**

  Użytkownik nie jest jeszcze zdecydowany, z jakim zbiorem chce pracować. Korzysta z funkcji przeglądania dostępnych zbiorów, które zawierają metadane, takie jak: źródło, liczba rekordów, typ danych (tekst, obraz, liczby), wymagania wstępne itp. Po zapoznaniu się z informacjami, wybiera odpowiedni zbiór i rozpoczyna pracę.

2.6. Podsumowanie

Ten rozdział dostarcza wszechstronnej analizy projektowanego systemu, obejmującej szczegółowy opis funkcji z uwzględnieniem priorytetów, opis procesów biznesowych oraz scenariuszy użytkowania, co pozwala lepiej zrozumieć i wizualizować funkcjonalności projektu. Ta

całościowa prezentacja ułatwia zarządzanie oczekiwaniami i planowanie dalszych etapów rozwoju systemu, uwzględniając ustalony zakres prac i zasoby, co jest kluczowe dla skutecznego planowania przyszłych etapów wdrożenia.

Rozdział 3

Wybrane aspekty realizacji

Niniejszy rozdział poświęcony jest praktycznym aspektom implementacji biblioteki. Celem tego rozdziału jest przedstawienie kluczowych decyzji projektowych, które miały wpływ na strukturę i funkcjonalności finalnego produktu. Rozdział ten stanowi podstawę do głębszego zrozumienia technicznego podejścia przyjętego podczas tworzenia biblioteki oraz wyjaśnia, jakie specyficzne problemy zostały rozwiązane w trakcie pracy nad projektem.

3.1. Architektura systemu

System opracowany w ramach tej pracy inżynierskiej został skonstruowany tak, aby zapewnić użytkownikowi łatwy dostęp do popularnych zestawów danych wykorzystywanych w uczeniu maszynowym oraz umożliwić dodawanie i zarządzanie własnymi zbiorami danych. Aby osiągnąć te cele, architektura systemu została podzielona na trzy główne moduły.

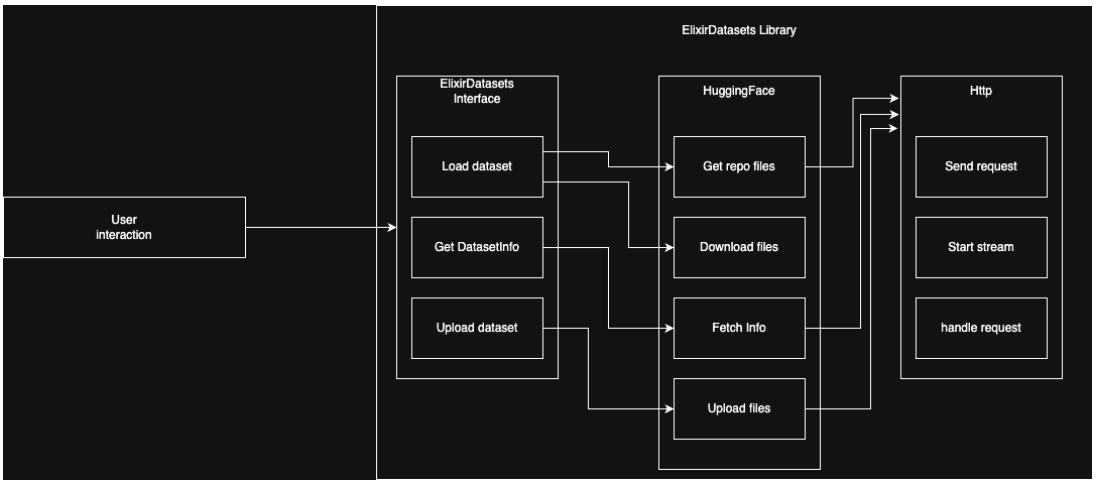
Moduł networkingowy jest odpowiedzialny za wszystkie operacje wymagające komunikacji z zewnętrznymi serwerami za pomocą protokołu HTTP. Jego głównymi zadaniami są: pobieranie danych z zewnętrznych źródeł, przesyłanie żądań o dostęp do zbiorów danych i odbieranie odpowiedzi.

Moduł HuggingFace odpowiada za integrację z API Hugging Face, co pozwala na wyszukiwanie, pobieranie i zarządzanie zbiorami danych dostępnymi na platformie oraz autoryzację do materiałów z ograniczonym dostępem.

Interfejs biblioteki ElixirDatasets to komponent, z którym bezpośrednio wchodzi w interakcję użytkownik końcowy. Stanowi on "fasadę"dla wszystkich operacji dostępnych w bibliotece, ukrywając za sobą złożoność modułów niższego poziomu. Każdy z tych komponentów pełni specyficzne funkcje, które razem tworzą spójny i efektywny system.

3.2. Stos technologiczny

Niniejszy rozdział opisuje technologie wybrane do budowy i wsparcia rozwoju naszej biblioteki w Elixirze. Przedstawione zostaną zarówno główne składniki stosu technologicznego, jak i przyczyny ich wyboru, co umożliwi zrozumienie, dlaczego stanowią one optymalne rozwiązanie.



Rysunek 3.1: Schemat systemu

3.2.1. Elixir i biblioteka standardowa

Język programowania Elixir został wybrany jako główny język programowania ze względu na jego skalowalność, wydajność oraz wyjątkowe wsparcie dla programowania współbieżnego. Biblioteka standardowa Eliksira dostarcza szeroką gamę modułów i funkcji, które pomagają w efektywnym budowaniu aplikacji, w tym w obsłudze sieciowej, przetwarzaniu danych i konstrukcjach współbieżnych.

3.2.2. Explorer

Współpraca z danymi wymaga używania narzędzi, które umożliwiają łatwą manipulację i transformację zbiorów danych. Biblioteka Explorer służy do łatwego zarządzania danymi w Elixirze, oferując funkcje podobne do tych z pakietu pandas w Pythonie. Umożliwia ona efektywne przetwarzanie dużych zbiorów danych.

3.2.3. Mix

Mix jest narzędziem służącym do zarządzania projektami w Elixirze, które umożliwia kompilację kodu, jego testowanie oraz zarządzanie zależnościami. Mix jest integralną częścią ekosystemu Elixir i stanowi fundament pod względem konstrukcji i administracji projektami.

3.2.4. ExDoc

Dokumentacja jest niezmiernie ważna dla utrzymania i rozwijania projektów programistycznych, zwłaszcza tych otwartych, gdzie inni programiści mogą wnosić wkład. ExDoc to narzędzie do generowania dokumentacji w Elixirze, które pozwala na tworzenie przejrzystej i łatwo przeszukiwalnej dokumentacji dla projektów.

3.2.5. ExUnit i ExCoveralls

Zapewnienie jakości kodu poprzez testy jest fundamentem stabilnego oprogramowania. ExUnit to framework testowy dostarczany razem z Elixirem, który umożliwia pisanie czytelnych i efektywnych testów jednostkowych. ExCoveralls z kolei to narzędzie, które pozwala na

mierzenie pokrycia kodu testami, co jest ważnym wskaźnikiem jakości projektu programistycznego.

3.3. Przegląd poszczególnych komponentów

W tej sekcji dokonujemy szczegółowego przeglądu kluczowych komponentów systemu, które mają zasadnicze znaczenie dla działania i efektywności naszej biblioteki. Każdy z komponentów odpowiada za specyficzne funkcje i jest istotny zarówno z punktu widzenia realizacji podstawowych zadań, jak i możliwości rozbudowy czy utrzymania całego systemu.

3.3.1. Moduł Interfejsu

Moduł interfejsu odgrywa kluczową rolę w naszej bibliotece, zapewniając interaktywny dostęp do funkcjonalności platformy HuggingFace/datasets. Ten komponent ma za zadanie odtworzenie interfejsu tej platformy, co umożliwia użytkownikom łatwe pobieranie zbiorów danych, ich udostępnianie oraz przeglądanie szczegółowych informacji o danych przed ich pobraniem.

Interaktywność modułu interfejsu znacząco ułatwia korzystanie z bogatych zasobów dostępnych na platformie Hugging Face, pozwalając na bezpośrednie i intuicyjne wykorzystanie dostępnych zbiorów w procesach analitycznych i badawczych. Dzięki temu użytkownicy mogą nie tylko efektywnie zarządzać danymi, ale również optymalizować czas potrzebny na przygotowanie i przetwarzanie informacji niezbędnych do analiz lub treningów modeli uczenia maszynowego.

Podsumowując, moduł interfejsu jest niezbędnym elementem biblioteki, umożliwiającym integrację z zewnętrznymi źródłami danych i zapewniającym użytkownikowi dostęp do funkcjonalności kluczowych dla efektywnego wykorzystania dostępnych zbiorów danych.

3.3.2. Moduł HuggingFace

Moduł HuggingFace w bibliotece ElixirDatasets pełni kluczową rolę w zapewnianiu dostępu do szerokiej gamy zbiorów danych dostępnych na platformie Hugging Face. Ten komponent modułu jest odpowiedzialny za interakcję z API Hugging Face, co ułatwia pobieranie danych, ich udostępnianie, a także podgląd informacji o zbiorach przed ich pobraniem.

Dzięki wykorzystaniu funkcji ‘file_url‘ i ‘file_listing_url‘, moduł umożliwia uzyskanie URL-i do konkretnych plików i listowania zawartości repozytorium na platformie Hugging Face. Obejmuje to zarówno publiczne zbiory danych, jak i te prywatne, dostępne tylko dla autoryzowanych użytkowników. Proces ten jest wspomagany przez mechanizm cache’owania, gdzie każde pobrane pliki są zapisywane lokalnie wraz z ich metadanymi, co pozwala na optymalizację kolejnych zapytań.

Dodatkowo, rozszerzona funkcja ‘cached_download‘ pozwala na pobieranie plików z zastosowaniem cache bazującego na ETagach (Entity Tags). Jeśli zasób nie uległ zmianie, dane mogą być serwowane bezpośrednio z lokalnego cache, co znacznie przyspiesza dostęp i redukuje zużycie zasobów sieciowych.

Użytkownik może również korzystać z opcjonalnych ustawień, takich jak tryb offline, który zapewnia dostęp do zasobów nawet w przypadku braku połączenia z Internetem, pod warunkiem że dane zostały wcześniej pobrane i zapisane w cache'u.

3.3.3. Moduł networkingowy

Moduł networkingowy w projekcie ElixirDatasets pełni fundamentalną rolę w umożliwianiu komunikacji sieciowej, stosując się do wiodących standardów praktyk internetowych. Jest odpowiedzialny za mechaniczne aspekty pobierania danych, obsługę protokołów HTTP/HTTPS oraz zarządzanie połączeniami sieciowymi. Moduł ten, zaimplementowany w 'ElixirDatasets.Utils.HTTP', służy jako centralny punkt dla wszystkich operacji sieciowych wykonanych przez bibliotekę.
Fundamentalne funkcje modułu networkingowego obejmują:

- **Pobieranie danych:** Moduł potrafi wykonywać bezpieczne żądania do zdalnych serwerów, pobierając dane bezpośrednio na lokalne dyski użytkownika. To mechanizm, który stoi za funkcją 'download', pozwala na efektywne zarządzanie przepływem danych.
- **Zarządzanie nagłówkami HTTP:** Za pomocą funkcji 'get_header', moduł umożliwia manipulację i odczyt nagłówków HTTP, co jest kluczowe do prawidłowego rozumienia i kontrolowania odpowiedzi serwera.
- **Opcje konfiguracyjne:** Użytkownicy mogą dostosować wiele aspektów żądań HTTP, takich jak nagłówki, ciało żądania, czy timeout, co daje elastyczność potrzebną do obsługi różnorodnych scenariuszy użytecznych.

Jedną z istotnych cech modułu jest jego zdolność do obsługi przekierowań i autoryzacji, co czyni go potężnym narzędziem do integracji z różnymi API oferującymi dane. Ponadto, moduł zapewnia mechanizmy bezpieczeństwa, takie jak weryfikacja certyfikatów SSL czy obsługa etykiet ETag, które minimalizują ryzyko przechwycenia danych i umożliwiają inteligentne zarządzanie cache'owaniem.

3.4. Ciekawsze algorytmy i mechanizmy systemu

Poniższa sekcja koncentruje się na przedstawieniu najbardziej istotnych komponentów opracowywanego systemu. Zostaną one omówione nie tylko pod kątem szczegółowego działania, ale także w kontekście ich wpływu na całość tworzonego systemu.

3.4.1. Cacheowanie danych

Cacheowanie danych jest krytycznym aspektem naszego systemu, zwiększającym jego efektywność poprzez redukcję liczby koniecznych operacji wejścia/wyjścia, zewnętrznych zapytań API oraz pobrań danych, które już wcześniej zostały załadowane. Aby zminimalizować niepotrzebne operacje, szczególnie w przypadku wielokrotnych uruchomień kodu przez użytkownika, implementujemy mechanizm etag (entity tag) oraz OId (Object Identifier). Etag i OId to unikalne identyfikatory przypisane do każdej wersji zasobu, które umożliwiają jednoznaczne stwierdzenie, czy przechowywana wersja zasobu nadal jest aktualna.

Przykładem wykorzystania etagu oraz oidu może być kontrola stanu zbiorów danych na zdalnym repozytorium. Poniżej przedstawiono przykłady odpowiedzi serwera, ilustrującej zastosowanie etagu i oidu:

```
{
  "etag": "W/\"129-cBr/GjmAu235zSmcAE8hRzjbA90\"",
  "url": "https://huggingface.co/api/datasets/fka/awesome-chatgpt-prompts"
}%
```

Listing 3.1: Podgląd odpowiedzi serwera z etagiem

```
[
    {
        "type": "file",
        "oid": "f4f3945bd7150d3e12988485c42da1f8c29c59f8",
        "size": 2265,
        "path": ".gitattributes"
    },
    {
        "type": "file",
        "oid": "d3dc1fd0061ff5ebb4c34451bd6c17d4547b6612",
        "size": 339,
        "path": "README.md"
    },{
        "type": "file",
        "oid": "6df098dd5d2ff6d9fedd3aa052e6fd49c3389b77",
        "size": 104186,
        "path": "prompts.csv"
    }
]%
```

Listing 3.2: Podgląd odpowiedzi serwera z oid

3.4.2. Pasek Postępu (ProgressBar)

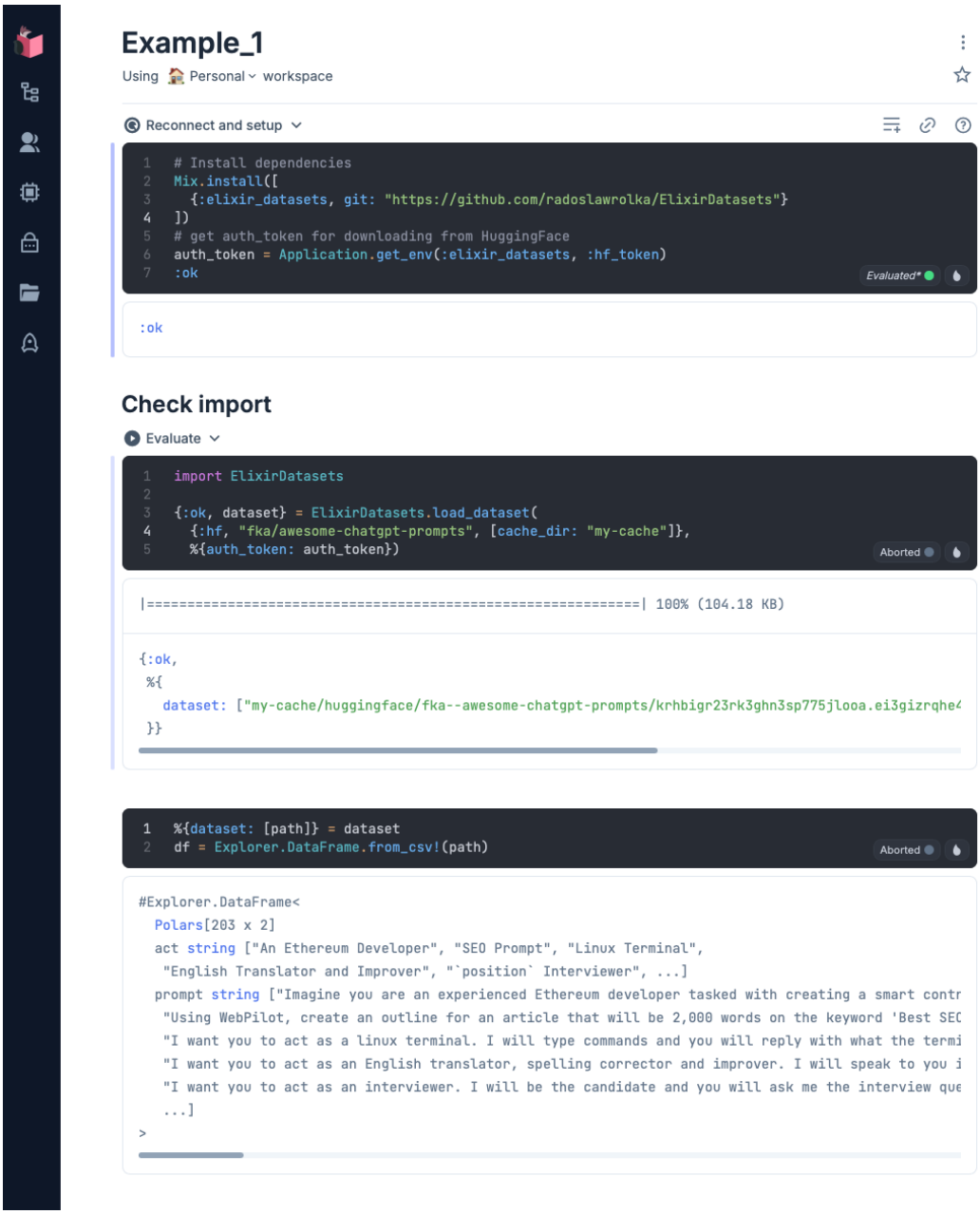
Aby ułatwić użytkownikom wizualizację postępu pobierania dużych zbiorów danych, w bibliotece został zaimplementowany pasek postępu (ProgressBar). Jest to narzędzie graficzne, które dynamicznie aktualizuje się w trakcie pobierania danych, pokazując użytkownikowi w sposób bezpośredni ile danych już zostało pobrane, a ile pozostało do końca procesu. Taka wizualizacja jest szczególnie przydatna w przypadku pobierania dużych zbiorów danych, gdzie proces może trwać znaczną ilość czasu.

3.4.3. Przykłady LiveBook

LiveBook to narzędzie interaktywne, zaprojektowane do tworzenia dokumentów, które mogą zawierać zarówno treść edukacyjną jak i wykonywalny kod. W ramach naszej biblioteki w języku Elixir, zintegrowaliśmy przykłady LiveBook, które umożliwiają użytkownikom eksperymentowanie z kodem na żywo, bezpośrednio w przeglądarce.

Rysunek 3.2: Przykład wizualizacji paska postępu

Te interaktywne dokumenty są niezwykle przydatne w edukacji i prezentacji możliwości biblioteki, ponieważ pozwalają na natychmiastowe obserwowanie wyników działania kodu. Użytkownicy mogą modyfikować lub rozwijać przykłady, co zwiększa ich zrozumienie działania biblioteki i zachęca do głębszego eksplorowania jej funkcji.



Rysunek 3.3: Przykład wykorzystania LiveBook

3.5. Zapewnienie jakości

W projektowaniu oprogramowania, zapewnienie jakości odgrywa kluczową rolę w sprawdzaniu, czy produkt spełnia oczekiwania użytkowników i wewnętrzne standardy jakości przed jego wydaniem. W kontekście rozwijania biblioteki przedstawione zostaną techniki testowania automatycznego oraz utrzymywania jakości oprogramowania.

3.5.1. Testy jednostkowe

Testy jednostkowe koncentrują się na pojedynczych, izolowanych fragmentach kodu, takich jak funkcje lub metody, sprawdzając, czy zachowują się one zgodnie z oczekiwaniami na podstawie zdefiniowanych przypadków testowych. W projektowanej bibliotece w języku Elixir, testy te są realizowane z wykorzystaniem frameworka ExUnit, który oferuje wsparcie dla asercji, testowania równoległego, a także umożliwia łatwą integrację z narzędziami do pomiaru pokrycia kodu testami, takimi jak ExCoveralls, którego pokrycie testami zostało ustawione na 90%. Efektywne wykorzystanie testów jednostkowych pozwala na szybką identyfikację i izolację błędów oraz zapewnia stabilność kodu poprzez ciągłą weryfikację jego poprawności w miarę jego rozwoju.

3.5.2. Analiza statyczna kodu

W napisanej bibliotece zostały wykorzystane narzędzia analizy statycznej kodu, które przeprowadzało rygorystyczne sprawdzanie typów oraz wykrywanie nieosiągalnych fragmentów kodu. Z uwagi na dynamiczną naturę Elixira, narzędzie to stanowi kluczowy element zapewniający dodatkowe bezpieczeństwo typów, które nie jest domyślnie ścisłe w tym języku. Ponadto, w procesie tworzenia biblioteki wykorzystano narzędzia zapewniające przestrzeganie ujednoliconego stylu kodowania, co znacząco przyczyniło się do poprawy czytelności kodu. Te narzędzia automatycznie egzekwowały zasady dotyczące formatowania i struktury kodu, co ułatwiło współpracę w zespole i zwiększyło efektywność w utrzymaniu oraz rozwijaniu projektu. Implementacja analizy statycznej pozwala na wcześniejsze zidentyfikowanie problemów oraz ułatwia utrzymanie wysokiej jakości kodu.

3.5.3. Ciągła integracja (CI)

Kluczowym narzędziem usprawniającym proces ciągłej integracji jest GitHub Actions. Ta platforma CI/CD pozwala na automatyczne wykonywanie różnorodnych operacji w arbitralnie skonfigurowanym środowisku wirtualnym. W naszym przypadku, każde zgłoszenie do repozytorium inicjuje serię zadań w GitHub Actions, które obejmują kompilację kodu w trybie ścisłym, uruchomienie testów, zgodność z ustanowionymi standardami formatowania, a także analizę pokrycia testami.

3.5.4. Code review

Code review, czyli przegląd kodu prez innego członka zespołu, jest niezastąpionym procesem w cyklu rozwoju oprogramowania. Proces ten polegał na ocenie kodu przez osobę niezwiązaną bezpośrednio z implementacją przy utworzeniu żądania dodania nowych funkcjonalności do oficjalnej gałęzi repozytorium. Dopiero po pozytywnym rozpatrzeniu takiego wniosku możliwe było połączenie dwóch wersji.

3.6. Podsumowanie

Biblioteka została zaprojektowana zgodnie z zasadami modularności, co klarownie definiuje rolę i odpowiedzialności poszczególnych segmentów kodu. Narzędzia oraz technologie użyte do jej rozwoju są powszechnie akceptowane i standardowe w społeczności deweloperów Elixira. Dla zapewnienia niezawodności systemu stosowane są różnorodne formy testowania, a także regularne recenzje kodu wewnątrz zespołu. Jakość kodu jest dodatkowo monitorowana przez narzędzia do statycznej analizy, co potwierdza wysokie standardy jego utrzymania.

Rozdział 4

Organizacja pracy

todo - wstep

4.1. 4thchapter-todo

todo

Rozdział 5

Wyniki projektu

todo

5.1. 5thchapter-todo

todo

Bibliografia

[1] H. Face. *Hugging Face Hub*. 2025. URL: https://huggingface.co/.

[2] C. Grainger i J. Valim. *Explorer: Fast and Elegant Data Exploration in Elixir*. 2025. URL: https://github.com/elixir-explorer/explorer.

[3] Kaggle. *Kaggle: Your Machine Learning and Data Science Community*. 2025. URL: https://www.kaggle.com.

[4] Q. Lhoest i in. „Datasets: A Community Library for Natural Language Processing". W: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online i Punta Cana, Dominican Republic: Association for Computational Linguistics, list. 2021, s. 175–184. arXiv: 2109.02846 [cs.CL]. URL: https://aclanthology.org/2021.emnlp-demo.21.

[5] S. Moriarity, J. Valim i P. O. L. Valente. *Nx - Numerical Elixir*. 2022. URL: https://github.com/elixir-nx/nx.

[6] P. Podgórni. *Metoda MoSCoW – co to jest?* 2024. URL: https://www.itvision.pl/experts/model-moscow-czym-jest-jak-korzystac/.

[7] T. pandas development team. *pandas-dev/pandas: Pandas*. Wer. latest. Lut. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[8] J. Valim i contributors. *Bumblebee: Pre-trained Neural Network Models in Elixir*. 2025. URL: https://github.com/elixir-nx/bumblebee.

[9] T. Wolf i in. „Transformers: State-of-the-Art Natural Language Processing". W: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, paź. 2020, s. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Spis rysunków

Spis tabel

Spis algorytmów

Spis listingów