PK4: Platforma NLP
Sprawozdanie

Michał Czyż

10 czerwca 2024

1 Wstęp

Projektem przewidzianym do wykonania w tym semestrze była platforma przetwarzania języka naturalnego *(eng. natural langugage processing)*. W ramach projektu zaimplementowane zostało tłumaczenie tekstu, wykrywanie języka tekstu, anonimizacja danych wrażliwych, oraz badanie sentymentu.

Przetwarzanie języka naturalnego to dziedzina nauki łącząca językoznawstwo oraz dziedzinę sztucznej inteligencji. Zajmuje się ona analizą i przetwarzaniem języka ludzkiego przez komputer. Po stworzeniu i wyszkoleniu modelu językowego, program jest w stanie z dużą dokładnością przewidywać oraz wykryć logiczny ciąg zdań języka. Na tej podstawie może być on dalej analizowany i przetwarzany.

Tłumaczenie maszynowe może być realizowane na wiele różnych sposobów. Skupiono się na dwóch rozwiązaniach jakimi są statystyczne tłumaczenie maszynowe oraz duże modele językowe opartych na transformerach. Do tłumaczenia statystycznego wykorzystano pierwszy model opracowany przez IBM w latach 90. Skupia się on w głównej mierze na tłumaczeniu leksykalnym, zatem nie będzie bardzo dokładny przy wychwytwaniu dokładnego kontekstu, ale jest w stanie osiągnąć dostateczne rezulataty. Dodatkowym atutem tego modelu jest jego wysoka wydajność. Model oparty jest o prawdopodobieństwo warunkowe.

2 Analiza tematu

Platforma NLP składa się z trzech usług:

- Core - główny program napisany w języku C++. Jest to główny program skupiający się na komunikacji z warstwą API oraz na przetwarzaniu danych.
- Middleware - serwer API, napisany w Pythonie. Łączączy komunikację pomiędzy użytkownikiem, a programem głównym. Middleware udostępnia warstwę programistyczną, którą wykorzystuje frontend.
- Frontend - strona internetowa napisana przy pomocy Next.JS oraz Reacta. Korzysta z API i udostępnia wartswę graficzną użytkownikowi końcowemu.

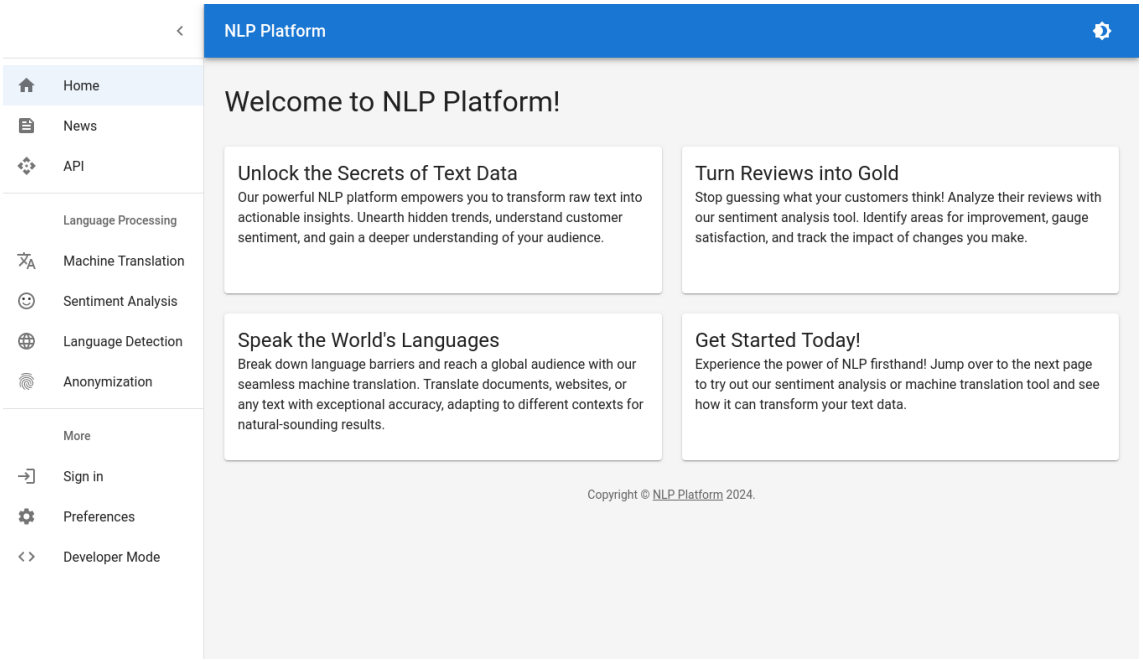
Język C++ wybrano przede wszystkim ze względu na wydajność oraz dostępność wielu elementów programowania obiektowego. Język Python został wybrany, ponieważ ma bardzo rozbudowane biblioteki do nauczania maszynowego oraz posiada bardzo łatwe i wygodną bibliotekę - *Flask*.

Jednym z głównych założeń projektu jest bycie modularnym i łatwym w rozbudowie. W szczególności, iż projekt jest złożony i wymaga wykorzystania wielu języków i bibliotek. Główna struktura programu w C++ wygląda następująco:

- Communication - zajmuje się komunikacją pomiędzy pythonem i programem.
- Engine - zbiór wielu klas obsługujących działanie programu i przetwarzających dane.
- Translate - zbiór klas odpowiedzialnych za obsługę tłumaczenia.
- Sentiment - zbiór klas odpowiedzialnych za obsługę analizy sentymentu.
- AsyncLogger - klasa odpowiedzialna za prawidłowe logowanie stanu programu.

Wykorzystanie biblioteki podczas tworzenia głównego programu w C++ to zeromq, cppzmq oraz boost. ZeroMQ jest wykorzystywany do komunikacji pomiędzy programem a interfejsem sie-

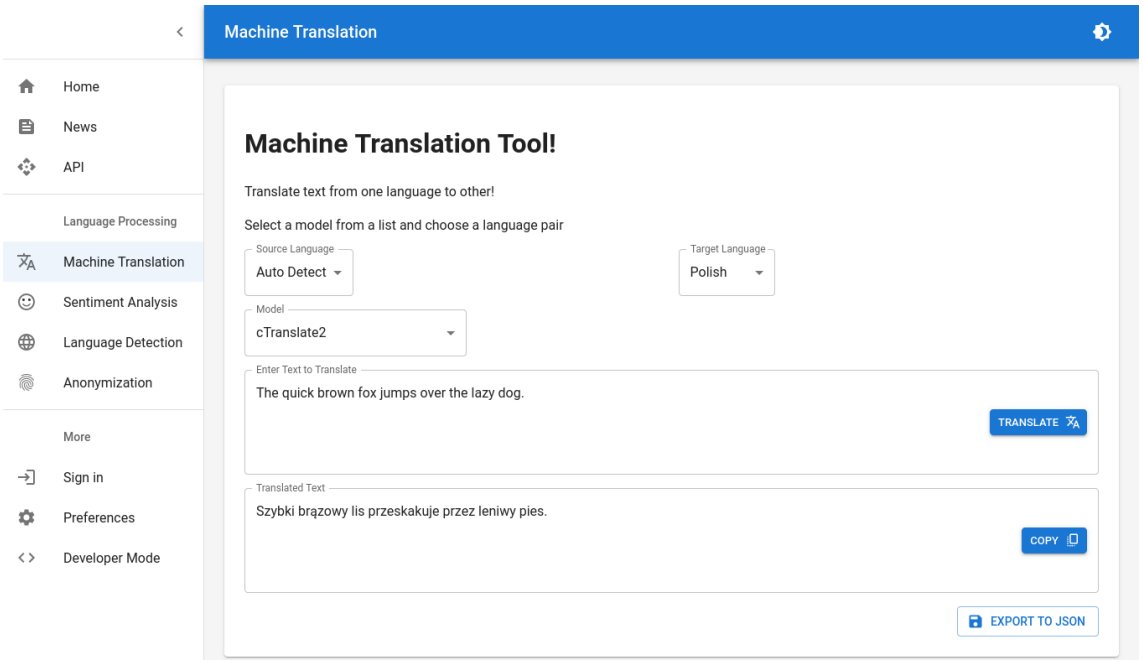
ciowym. Biblioteka cppzmq udostępnia wygodny sposób do inicjalizacji i zarządzania komunikacją. Biblioteka Boost była używana przy łączeniu C++ z Pythonem oraz do komunikacji.



Rysunek 1: Wygląd srony głównej

3 Specyfikacja zewnętrzna

Program posiada warstwę API oraz graficzną co pozwala na interakcję z platformą w wygodny sposób dla użytkownika końcowego jak i także dla programisty, który chciałby wykorzystać funkcjonalność platformy NLP w swoim programie. Serwer API posiada podstawową autoryzację poprzez token API.



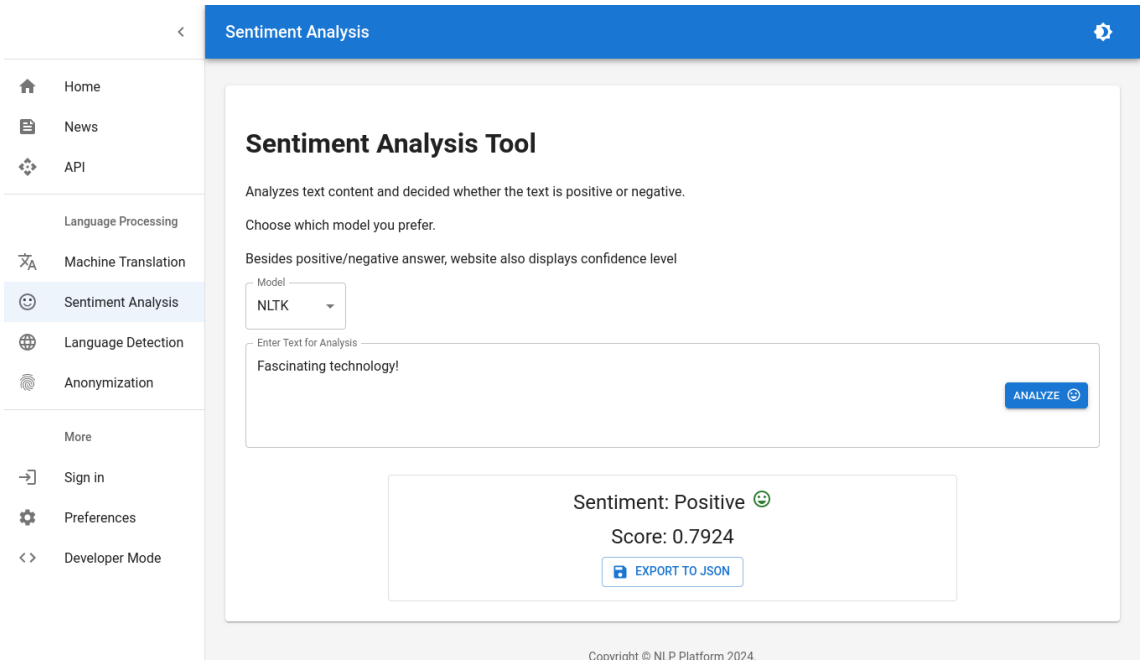
Rysunek 2: Interfejs tłumaczenia maszynowego

3.1 Instrukcja dla użytkownika

Jeżeli usługa jest uruchomiona na komputerze lub port interfejsu webowego jest udostępniony do internetu wystarczy, że użytkownik ma dostęp do internetu oraz posiada zainstalowaną przeglądarkę internetową. Po odwiedzeniu odpowiedniego adresu ukaże się strona internetowa po której może nawigować i wybierać interesujące go funkcje.

3.2 Przykłady działania

Przykłady działania aplikacji zostały przedstawione na rysunkach widocznych w dokumencie. Jest możlwie swobodne nawigowanie po interfejsie webowym, wybranie modelu i pary językowej, która nas interesuje oraz odczytanie i pobranie wyniku w postaci pliku JSON.



Rysunek 3: Interfejs analizy sentymentu

Interfejs Programowania Aplikacji wygląda następująco. Należy wysłać zapytanie na następującą scieżkę na serwerze:

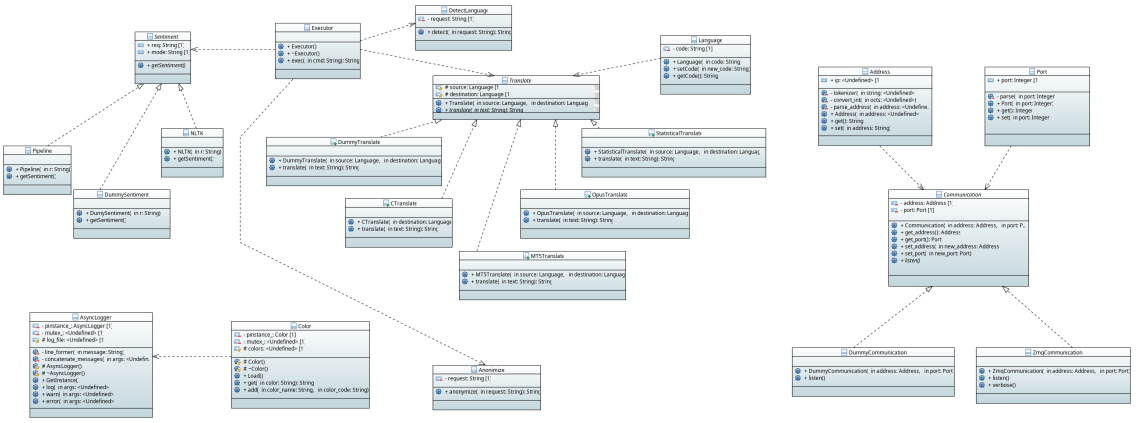
```
GET http://localhost:8080/api/ping

POST http://localhost:8080/api/translate
POST http://localhost:8080/api/sentiment
POST http://localhost:8080/api/detect
POST http://localhost:8080/api/anonymize

POST http://localhost:8080/api/auth
POST http://localhost:8080/api/signin
```

Struktura dla przykładowego tłumaczenie przy wykorzystaniu cTranslate2 wygląda następująco:

```
{
  "srcLanguage": "en",
  "dstLanguage": "pl",
  "mode": "ct2",
  "data": "Natural Language Processing"
}
```

4 Specyfikacja wewnętrzna

Diagram klas został przedstawiony na Rysunku 4. Główne klasy to Translate, Communication oraz Sentiment, które odpowiadają za główną funkcjonalność programu. Reszta klas to m.in. Async-



Rysunek 4: Diagram Klas

Logger - odopowiedzialny za logowanie danych do pliku i do konsoli. Klasy DetectLanguage oraz Anonymize, które są odpowiedzialne za detekcję i anonimizację danych.

Zostały także utworzone klasy Dummy, które symulują tylko swe działanie. Jest zwracana tylko stała odpowiedź. Jest to przydatne przy debuggowaniu i implmentacji konkretnych części aplikacji.

Z technik obiektowych wykorzystano m.in. polimorfizm. Kluczowe było to, aby program był bardzo modularny i aby była możliwość modyfikacji każdej funkcjonalności programu. Drugim ważnym czynnikiem była także skalowalność programu.

Wykorzystano wzorzec projektowy Singleton, który zapewnia tylko jedną instancję danej klasy. Klasa AsyncLogger jest zaawansowaną klasą zajmującą się logowaniem statusu i działania programu. Został napisany jako Singleton aby zapewnić bezpieczeństwo zapisu do pliku i do standardowego wyjścia przy działaniu wielu wątków równolegle.

Wykorzystane elementy z laboratorium to:

- Wyrażenia regularne do weryfikacji przepływu danych oraz poprawności przekazywanych danych.
- Ranges do łatwego przechodzenia przez pętle.
- Asynchroniczność do poprawnego zapisu do pliku i konsoli.
- Wielowątkowość do prawidłowej obsługi workerów i brokera do komunikacji z API.
- Koncepty do weryfikacji danych i zmiennych jakie są przekazywane do funkcji i template'ów.

Niewykorzystane zostały moduły z tego powodu iż program wykorzystuje kompilator G++, który nie wspiera ich dobrze. Drugim niewykorzystanem elementem laboratorium była biblioteka filesystem, ponieważ program nie posiada żadnej zaawansowej obsługi systemu plików.

Działanie programu wygląda następująco. Po odebraniu zapytania, serwer API sprawdza czy request jest poprawny. Jeżeli request jest poprawny, jest on parsowany i przekazywany do C++. Kiedy odbierze to C++, request jest dokładnie analizowany i wybierane są konkretne parametry. Po analizie wyborów, następuje przekazanie do konkretnej metody z konkretnymi parametrami. Metoda w konkretnej klasie wywołuje instancję Executora, który poprawnie wysyła i odbiera wynik z częsci Pythonowej. Nastepnie wynik jest zwracany do API. API, pakuje wynik do JSONa i zwraca go użytkownikowi końcowemu.

Wykorzystane modele i datasety do trenowania:

- Madlad400 3b CT2
- Langdetect Library
- Google MT5 Base
- Opus MT
- NLTK IBM Model 1
- Presidio Analyzer

- AnkiWeb Polish-English Vocabulary
- European Parliament Languages Dataset

5 Testowanie

Program był testowany w kilku etapach. Na początku testowano interfejs webowy aby sprawdzić czy wszystkie możliwości i kombinacje ustawień modeli są prawidłowe. Sprawdzana była także komunikacja warstwy webowej z API oraz czy wszystkie requesty są poprawnie obsługiwane lub odrzucane.

Później sprawdzano poprawność działania warstwy API z warstwą C++. Komunikacja ta była kluczowa do poprawnego działania programu. Sprawdzono także czy wszystkie przekazywane dane są przekazywane prawidłowo, czy nie można poprzez wysłanie błędnych danych dokonać *remote code execution*.

Ostatecznie sprawdzano warstwę C++, pod kątem bezpieczeństwa wykonywania modeli. A także testowano czy wszstkie klasy zachowują się zgodnie z założeniem.

6 Uruchamianie

Zalecana kolejność uruchamiania usług to warstwa webowa, następnie API i na końcu program w C++.

6.1 Uruchomienie serwera WWW

Wymagane pakiety: `nodejs`, `npm`

Aby uruchomić usługę należy wykonać następujące kroki:

```
$ cd web
$ npm install
$ npm run build
$ npm run start
```

Aplikacja uruchomi się na porcie *3000*.

W przypadku chęci debuggowania aplikacji lub wprowadzania zmian w kodzie można uruchomić serwer deweloperski:

```
$ npm run dev
```

6.2 Uruchomienie serwera API

Wymagane pakiety: `python`, `python-pip`

```
$ cd api
```

Przed uruchomieniem serwera należy aktywować wirtualne środowisko.

```
$ source activate.sh
```

Potem można wywołać serwer API.

```
$ python main.py
```

Aby opuścić wirtualne środowisko, można wykonać polecenie:

```
$ deactivate
```

6.3 Uruchomienie Core Platform

Wymagane pakiety: `base-devel`, `boost`, `cmake`, `python`, `python-pip`, `mold`, `zeromq`, `cppzmq`

Na początku należy aktywować wirtualne środowisko, aby móc wykorzystywać modele.

```
$ cd nlp/models
```

Umożliwia to skrypt activate.sh.

```
$ source activate.sh
$ cd ..
```

Ostatecznie wystarczy skompilować i uruchomić program wykonując:

```
$ make
```

Jeżeli nie chcemy ponownie kompilować programu, można wykorzystać:

```
$ make run
```

Aby opuścić wirtualne środowisko, można wykonać polecenie:

```
$ deactivate
```

6.4 Parametry programu

W C++ można także przekazać parametr –verbose, który włącza dodatkowe logowanie do konsoli informacji o działaniu ZeroMQ i dokładnego przebiegu pakietów.

7 Uwagi i wnioski

Podsumowując, platforma NLP okazała się fascynującym i bardzo rozwojowym projektem. Poznałem jak działają modele językowe, jak tworzyć i trenować własne. Nauczyłem się także jak łączyć warstwę sieciową z programami niższego poziomu, jak C++. Mogłem także dobrze rozwinąć swoje umiejętności programowania obiektowego oraz poznać i lepiej zgłębić zagadnienia poznane w ramach laboratorium.