



PROGRAM DO ROZPOZNAWANIA JĘZYKA Z WYKORZYSTANIEM BIBLIOTEKI FASTTEXT

Raport



KRZYSZTOF LISZKA, KACPER KROK, KAJETAN MIKRUT

1. Abstrakt

Projekt ten koncentruje się na bibliotece FastText - narzędziu opracowanemu przez Facebook służącego do klasyfikacji tekstów.

2. Wstęp

Nasze badanie koncentruje się na bibliotece FastText i opracowanym przez nas alternatywnym modelu w dziedzinie przetwarzania języka naturalnego (NLP).

2.1 Cel

Celem jest porównanie efektywności obu modeli danych - zawartego w bibliotece FastText oraz utworzonego przez nas.

2.2. Zakres

- a) Implementacja rozwiązania
- b) Analiza wyników
- c) Stworzenie raportu
- d) Wnioski

2.3. Metodyka

Do napisania naszej aplikacji korzystaliśmy przede wszystkim z biblioteki Fasttext. Jest to open-source'owa biblioteka stworzona przez FacebookResearch. Zawiera ona implementację algorytmu dla wydajnego uczenia się reprezentacji wektorowych słów oraz klasyfikacji tekstów. Główną cechą wyróżniającą FastText spośród innych narzędzi do przetwarzania języka naturalnego jest to, że oprócz generowania reprezentacji wektorowych słów, umożliwia on również wykonanie zadania klasyfikacji tekstów na podstawie tych reprezentacji. Biblioteka pierwotnie została zaimplementowana w C++, my oczywiście używamy wersji dla Pythona.

W naszej aplikacji użyliśmy Pythona w wersji 3.8, spowodowane jest to tym, że owa biblioteka nie została zaktualizowana dla nowszych wersji (aktualna 3.13).

W skład projektu wchodzi również GUI czyli interfejs graficzny. Został on zaimplementowany za pomocą biblioteki PySimpleGUI. Dzięki niej, użytkownik może wprowadzić tekst do analizy, wybrać metodę rozpoznawania języka, a wyniki są wyświetlane na ekranie.

Podczas tworzenia systemu, bardzo przydatny okazał się system kontroli wersji GIT. Cały projekt został umieszczony na Githubie i jest dostępny pod linkiem:

https://github.com/kajti0/language_detector

3. Część teoretyczna

W rozwiązaniu zadania bardzo pomogła dokumentacja biblioteki Fasttext, z której to dowiedzieliśmy się o metodach *train_supervised* i *train_unsupervised* oraz *predict*. W naszym przypadku sprawdziła się metoda *train_supervised*, która pozwala na uczenie modelu z etykietami, po czym po wygenerowaniu modelu metoda *predict* do przewidzenia danego języka. Metoda ta zwraca również prawdopodobieństwo konkretnego „trafu”. Kolejnym elementem jest gotowy model, dostępny na stronie internetowej biblioteki Fasttext. Jest on w stanie rozróżnić 176 języków, wyuczony na podstawie Wikipedii, Tatoeba i SETimes. Model ten jest gotowy do użytku, wystarczy go pobrać i stosować za pomocą metody *predict*. Dostępna jest także uproszczona biblioteka do pobrania – *fasttext-langdetect* korzystająca z tego modelu, dostarczająca metodę *detect* również pozwalającą na rozpoznanie języka, jednak w ostatecznej wersji naszego projektu zdecydowaliśmy się na korzystanie z pobranego modelu.

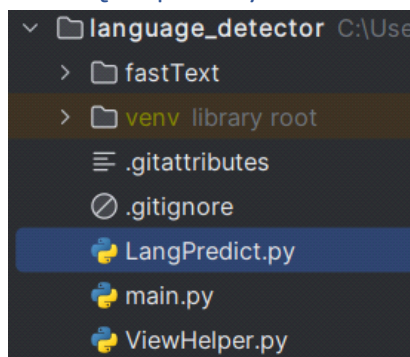
Ostatecznie korzystaliśmy z dwóch metod – jedna na bazie stworzonego przez nas modelu, druga na bazie gotowego modelu ze strony Fasttext. Nasz model pozwala na rozróżnienie dwóch języków i bazuje na książce „1984” autorstwa George Orwella. Dane do modelu obrobiliśmy w następujący sposób – z racji tego że język można rozpoznać nie tylko po samych słowach, ale również po kontekście, interpunkcji, częstości występowania różnych słów, to podzieliliśmy tekst na zdania, które obrobiliśmy za pomocą skryptu w Pythonie aby wyglądały w następujący sposób:

```
_label_pl Zwlekając mógł tylko pogorszyć sytuację
_label_pl Serce waliło mu jak młotem, lecz jego twarz, na skutek wieloletniego przyzwyczajenia, była pozbawiona wszelkiego wyrazu
_label_pl Podniósł się i ciężkim krokiem ruszył w stronę drzwi
_label_eng It was a bright cold day in April, and the clocks were striking thirteen
_label_eng Winston Smith, his chin nuzzled into his breast in an effort to escape the vile wind, slipped quickly through the glass doors of Victory Mansi...
```

Jak widać, przed każdym zdaniem jest *label* określający język, zdań w każdym języku jest około 300. Jest to więc stosunkowo mały model, przez co narzędzie nie działa idealnie. Zapewne, gdyby dostarczyć więcej bardziej zróżnicowanych zdań, to model zadziałałby lepiej.

Model dostarczony przez Fasttext działa dużo lepiej, natomiast warto zauważyć, że czym większy kontekst, tym łatwiej modelowi rozpoznać język. Dzieje się tak m. in. dlatego, że wiele słów powtarza się w różnych językach, wobec czego model musi patrzeć również na połączenia między słowami, tj. kontekst, całe zdania, lub nawet cały blok tekstu.

4. Część praktyczna



Struktura plików w projekcie. Jako środowisko musieliśmy użyć Pythona w wersji 3.8, z racji tego, że Fasttext nie współpracuje z wyższymi wersjami. W folderze venv mamy nasze środowisko, w folderze Fasttext – bibliotekę, zaś w pliku LangPredict.py używamy jej metod, a w pliku ViewHelper.py generujemy interfejs użytkownika.

Proces tworzenia modelu:

Najpierw pobraliśmy fragmenty tekstu (cały blok tekstu) do trenowania. Następnie przeprocesowaliśmy go do formatu, który odpowiada metodzie *train_supervised*:

```
def preprocess_data(self, input_file: str, output_file: str, language_label: str = "__label__pl"):
    with open(input_file, 'r', encoding='utf-8') as input_file:
        data = input_file.read()

    data = data.replace(__old: '\n', __new: ' ')
    sentences = [sentence.strip() for sentence in data.split('.') if sentence.strip()]
    labeled_sentences = [f"{language_label} {sentence}" for sentence in sentences]
    with open(output_file, 'w', encoding='utf-8') as output_file:
        output_file.write('\n'.join(labeled_sentences))
```

Ta metoda dzieli tekst na zdania, przed każdym zdaniem ustawia *label*, zachowuje interpunkcję, gdyż w niektórych językach jest ona ważna i charakterystyczna do jego rozpoznania, zaś usuwa znaki nowej linii.

Następnie pliki łączyliśmy w jeden, aby za jego pomocą wygenerować model do rozpoznawania języka:

```
def train(self, train_data_path: str, output_path: str):
    model = fasttext.train_supervised(*kargs: train_data_path, label="__label__")
    model.save_model(output_path + '.bin')
```

Ta funkcja przygotowuje model i zapisuje w podanej lokalizacji, od tego momentu model jest gotowy do użytku:

```
def predict_language(self, text: str, model_path: str):
    model = fasttext.load_model(model_path)
    prediction = model.predict(text)
    label = language_codes.get(prediction[0][0], 'Unknown language')
    probability = prediction[1][0]
    return label, probability
```

Powyższa funkcja korzysta z modelu, z racji tego że mamy 2 modele to w jej argumentach podajemy tekst do sprawdzenia oraz ścieżkę do modelu, z którego aktualnie korzystamy. Używamy funkcji *predict* do sprawdzenia języka podanego tekstu, funkcja ta zwraca tablicę której elementami są *label* języka i prawdopodobieństwo „trafu”. Z uwagi na to, że *label* jest zwracany jako string w formacie *__label__kod_języka* przygotowaliśmy słownik, który przyporządkowuje mu konkretną nazwę języka, a jeśli danego skrótu nie ma w słowniku, wyświetli się *Unknown language*. Oto fragment tego słownika:

```
language_codes = {
    '__label__af': 'Afrikaans',
    '__label__als': 'Alemannic',
    '__label__am': 'Amharic',
    '__label__an': 'Aragonese',
    '__label__ar': 'Arabic',
```

Do stworzenia widoku wykorzystaliśmy bibliotekę PySimpleGUI, jego tworzenie wygląda następująco:

```

layout = [
    [sg.Text('Wprowadź tekst do rozpoznania języka:'),
    [sg.InputText(key='-INPUT-', size=(30, 1))],
    [sg.Button('Metoda wbudowana'), sg.Button('Metoda autorska'), sg.Button('Wyjście')],
    [sg.Text(size=(50, 1), key='-OUTPUT-')],
]

```

Przygotowujemy *layout* składający się z tekstów pomocniczych, pola na wprowadzenie tekstu do rozpoznania, przycisków do wybrania metody rozpoznawania języka oraz pola na tekst do zwrócenia wyniku.

```

def view(self):
    window = sg.Window(title='Rozpoznawanie Języka', self.layout)

    while True:
        event, values = window.read()

        if event in (sg.WIN_CLOSED, 'Wyjście'):
            break

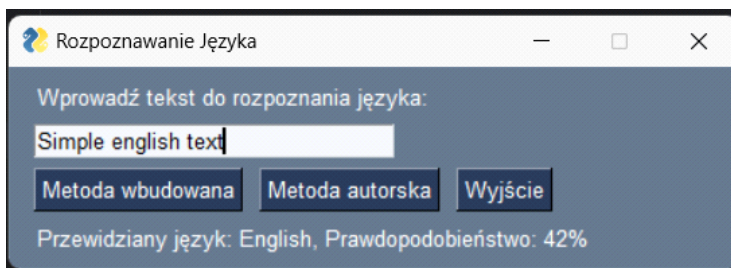
        if event == 'Metoda wbudowana':
            input_text = values['-INPUT-']
            label, probability = self.lang.predict_language(input_text, model_path: "../lid.176.bin")
            window['-OUTPUT-'].update(
                f'Przewidziany język: {label}, Prawdopodobieństwo: "{:.0%}".format(probability)')

        if event == 'Metoda autorska':
            input_text = values['-INPUT-']
            label, probability = self.lang.predict_language(input_text, model_path: "../model.bin")
            window['-OUTPUT-'].update(
                f'Przewidziany język: {label}, Prawdopodobieństwo: "{:.0%}".format(probability)')

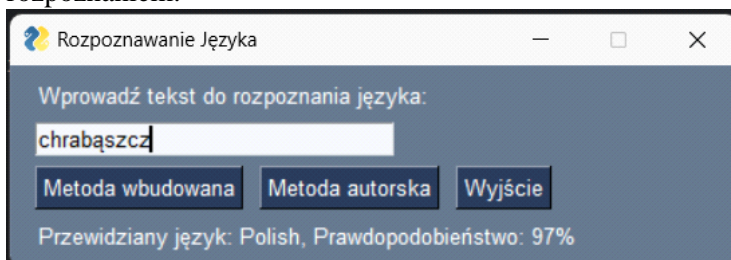
    window.close()

```

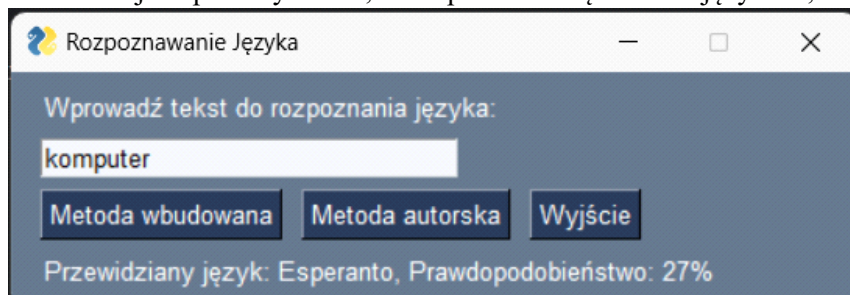
W funkcji *view* tworzymy okno z wykorzystaniem *layoutu*, nadajemy mu tytuł, a następnie w niekończącej się pętli *while* przetwarzamy zapytania, okno zamyka się po kliknięciu przycisku zamknięcia okna, bądź przycisku *Wyjście*. Gotowy widok wygląda następująco:



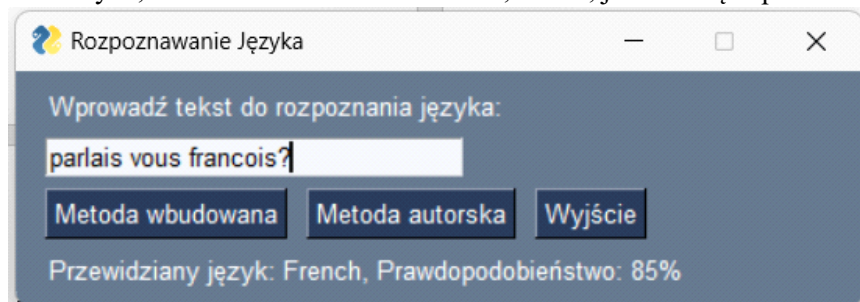
i działa zgodnie z zasadami opisanymi wcześniej. Możemy zauważyć, że jeśli podamy jedno słowo, ale jest ono bardzo charakterystyczne dla danego języka, model nie będzie miał problemów z jego rozpoznaniem:



Natomiast jeśli podamy słowo, które powtarza się w wielu językach, model może się pomylić:



Widać również po prawdopodobieństwie tego trafu, że model nie do końca „wie”, do jakiego języka przypisać dane słowo. Z większymi zdaniami zazwyczaj nie ma problemu, jest w nich dużo więcej zmiennych, które model może analizować, nawet, jeśli nie są napisane w pełni poprawnie:



5. Podsumowanie

Pomyślnie udało się stworzyć narzędzie do wykrywania języka. Program działa poprawnie, zwraca przewidziany język i prawdopodobieństwo poprawności wyniku. Skrypt działa w prawidłowy sposób dla różnych przesłanych języków.

6. Bibliografia

- George Orwell – „1984”
- <https://fasttext.cc/docs/en/supervised-tutorial.html>
- Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of Tricks for Efficient Text Classification
- Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, T. Mikolov, FastText.zip: Compressing text classification models