

Narzędzie do rozpoznawania przy pomocy biblioteki FastText

Filip Żołyniak

Problem

Rozpoznawanie języka tekstu jest ważną częścią współczesnego biznesu. Dzięki niemu Google i inne produkty mają możliwość personalizowania prezentowanych treści oraz reklam. Każdy język ma swoje unikalne cechy. Potrafią się różnić też składnią lub nawet znakami. Niezbędne jest dopasowanie metod uczenia maszynowego do odpowiednich języków.

Przykładem takiego podejścia jest implementacja kodu wykorzystującego bibliotekę FastText, stworzoną przez firmę Meta. Przy odpowiednich danych wejściowych skuteczność rozpoznawania języka może wynieść nawet 100%.

FastText

Biblioteka FastText opracowana przez Meta służy do szybkiego uczenia modeli oraz klasyfikacji tekstu. Wykorzystywana jest do przetwarzania tekstu oraz do budowania modeli, które potrafią rozpoznawać i klasyfikować tekst na podstawie jego zawartości. Może być wykorzystywana do różnych zastosowań, takich jak klasyfikacja spamu, detekcja języka, klasyfikacja sentymentu czy klasyfikacja tematów.

FastText umożliwia szybkie trenowanie modeli na dużych zbiorach danych oraz dokonywanie predykcji na nowych danych. Pozwala również na 2 przetwarzanie tekstu bez potrzeby jego tokenizacji oraz umożliwia pracę z tekstem w językach, które nie posiadają spacji między słowami. Biblioteka FastText jest wykorzystywana w różnych dziedzinach, takich jak przetwarzanie języka naturalnego, analityka internetowa, reklama czy edukacja. Może być również używana do tworzenia chatbotów oraz do automatyzacji procesów biznesowych.

Implementacja

Ze względu na specyfikę dostarczonych danych najpierw musimy przygotować dane do poprawnej interpretacji przez bibliotekę. W tym celu zastosujemy kod który sformatuje nam linie do formatu {__label__(country_iso2)} {content} .

Prepare our data from csv

```
In [9]: csv_file_name = 'europarl.csv'
        txt_file_name = 'train.txt'

        with open(csv_file_name, mode='r', encoding='utf-8') as csv_file:
            reader = csv.reader(csv_file)
            with open(txt_file_name, mode='w', encoding='utf-8') as txt_file:
                for row in reader:
                    if row:
                        label = f'__label__{row[0]}'
                        content = row[1]
                        txt_file.write(f'{label} {content}\n')
```

Przygotowany plik do uczenia modelu "train.txt" wprowadzamy do parametru input w funkcji biblioteki fasttext "train_supervised" z możliwymi parametrami:

dim - wymiar wektora słowa - wektor numeryczny reprezentujący słowo w przestrzeni cech. Domyślna wartość to 100. Im większy rozmiar wektora słowa tym więcej informacji może być zawarte w wektorze (skutkiem tego jest większe zużycie pamięci i dłuższy czas trenowania).

minn - minimalna długość char ngram, domyślna to 0

maxn - maksymalna długość char ngram, domyślna to 0

loss - określa funkcje straty do użycia podczas trenowania modelu. Mierzy ona jakość modelu podczas trenowania modelu i używana jest do aktualizowania wag modelu.

Domyślną wartością parametru jest "softmax", możemy skorzystać również z "hs", "ns", lub "ova".

epoch - określa nam liczbę epok trenowania modelu. Epoka jest jednym przejściem przez wszystkie próbki danych uczących. Domyślnie liczba epok wynosi 5. Im więcej epok tym dłuższy czas trenowania modelu, natomiast może to prowadzić do lepszych rezultatów.

First model

```
In [21]: model1 = fasttext.train_supervised(input="train.txt", dim=16, minn=2, maxn=5)
model1.save_model("train1.bin")
```

Second model

```
In [23]: model2 = fasttext.train_supervised(input="train.txt", dim=16, minn=2, maxn=5, loss="ova")
model2.save_model("train2.bin")
```

Third model

```
In [12]: model3 = fasttext.train_supervised(input="train.txt", dim=100, epoch=25, lr=0.5, minn=2, maxn=5, loss="ova")
model3.save_model("train3.bin")
```

Wytrenowane zostały 3 modele. Każdy z modelu został wyuczony na różnych parametrach. Model pierwszy i drugi po wytrenowaniu ważył ok 300 MB przy czym plik danych wejściowych ważył 400MB.

Trzeci model ze względu na 5 razy większą ilość epok waży 1,5 GB.

Jako zdanie testowe wybrałem sentencje “Witam i o zdrowie pytam”. Model został wyuczony na wykrywanie łącznej ilości 19 języków.

```
In [14]: sentences = [
    "Μένω σε μια μεγάλη πόλη και είμαι ηλεκτρολόγος μηχανικός.",
    "Hej, jeg spørger om dit helbred",
    "Dobrý den, ptám se na Váš zdravotní stav",
    "Hello, I'm asking about your health",
    "Witam i o zdrowie pytam",
    "Hallo, ich frage nach Ihrem Gesundheitszustand",
    "Hola te pregunto por tu salud",
    "Tere, küsin teie tervise kohta",
    "Hei, kysyn terveydestäsi",
    "Bonjour, je vous pose des questions sur votre santé",
    "Sziasztok, egészségi állapotodról kérdezem",
    "Ciao, ti chiedo della tua salute",
    "Labdien, es jautāju par jūsu veselību",
    "Hallo, ik vraag naar uw gezondheid",
    "Olá, estou perguntando sobre sua saúde",
    "Buna ziua, intreb despre sanatatea ta",
    "Dobrý deň, pýtam sa na Váš zdravotný stav",
    "Pozdravljeni, sprašujem za vaše zdravje",
    "Hej, jag frågar om din hälsa",
]
```

Rezultaty

First trained model

```
In [24]: language, probability = model1.predict(sentences)
         for index, sentence in enumerate(sentences):
             print("{} , probability: {:.2%}".format( language[index][0], probability[index][0] ))

__label__el, probability: 99.95%
__label__da, probability: 97.60%
__label__cs, probability: 96.28%
__label__en, probability: 63.98%
__label__pl, probability: 96.96%
__label__de, probability: 91.26%
__label__es, probability: 68.60%
__label__et, probability: 55.26%
__label__fi, probability: 98.42%
__label__fr, probability: 99.69%
__label__hu, probability: 99.76%
__label__it, probability: 62.18%
__label__lv, probability: 99.90%
__label__nl, probability: 99.69%
__label__pt, probability: 88.65%
__label__ro, probability: 76.16%
__label__sl, probability: 92.16%
__label__sl, probability: 99.95%
__label__sv, probability: 99.93%
```

Second trained model

```
In [25]: language, probability = model2.predict(sentences)
         for index, sentence in enumerate(sentences):
             print("{} , probability: {:.2%}".format( language[index][0], probability[index][0] ))

__label__el, probability: 99.86%
__label__da, probability: 41.49%
__label__cs, probability: 96.27%
__label__en, probability: 53.12%
__label__pl, probability: 98.41%
__label__de, probability: 23.94%
__label__sl, probability: 8.76%
__label__et, probability: 34.87%
__label__fi, probability: 92.19%
__label__fr, probability: 91.49%
__label__hu, probability: 99.95%
__label__it, probability: 98.20%
__label__lv, probability: 99.43%
__label__nl, probability: 100.00%
__label__pt, probability: 87.06%
__label__sl, probability: 39.98%
__label__sl, probability: 79.82%
__label__sl, probability: 99.86%
__label__sv, probability: 96.27%
```

Third trained model

```
In [26]: language, probability = model3.predict(sentences)
         for index, sentence in enumerate(sentences):
             print("{} , probability: {:.2%}".format( language[index][0], probability[index][0] ))

__label__el, probability: 100.00%
__label__da, probability: 99.57%
__label__cs, probability: 99.90%
__label__en, probability: 99.00%
__label__pl, probability: 100.00%
__label__de, probability: 100.00%
__label__es, probability: 97.90%
__label__et, probability: 88.40%
__label__fi, probability: 99.65%
__label__fr, probability: 100.00%
__label__hu, probability: 100.00%
__label__it, probability: 85.20%
__label__lv, probability: 100.00%
__label__nl, probability: 100.00%
__label__pt, probability: 99.46%
__label__sl, probability: 0.38%
__label__sk, probability: 86.70%
__label__sl, probability: 100.00%
__label__sv, probability: 99.96%
```

Pierwszy i drugi model cechują się wysoką skutecznością rozpoznawanych języków. Możemy zauważyć problemy z rozpoznawaniem języka Rumuńskiego oraz Słowackiego.

Podobnie sytuacja wygląda w modelu trzecim gdzie możemy zauważyć jeszcze wyższą skuteczności problem z rozpoznaniem języka Rumuńskiego.

Pretrained model by meta

```
In [27]: language, probability = model4.predict(sentences)
for index, sentence in enumerate(sentences):
    print("{}, probability: {:.2%}".format( language[index][0], probability[index][0] ))

__label__el, probability: 99.93%
__label__da, probability: 99.12%
__label__cs, probability: 97.96%
__label__en, probability: 97.61%
__label__pl, probability: 86.43%
__label__de, probability: 99.51%
__label__es, probability: 96.83%
__label__et, probability: 89.36%
__label__fi, probability: 94.43%
__label__fr, probability: 99.71%
__label__hu, probability: 99.98%
__label__it, probability: 99.25%
__label__lv, probability: 98.32%
__label__nl, probability: 99.94%
__label__pt, probability: 99.97%
__label__ro, probability: 50.22%
__label__cs, probability: 58.11%
__label__sl, probability: 83.08%
__label__sv, probability: 99.78%
```

Jako próbę badawczą wprowadziłem pre-trenowany model przez firmę Metą który obsługuje 176 języków . Model waży ok. 130Mb i jak możemy zauważyć charakteryzuje się również wysoką skutecznością rozpoznawania poza językiem Rumuńskim.

Biorąc pod uwagę rozmiar modelu, dużo większą ilość obsługiwanych języków oraz jego skuteczność model wydaje się atrakcyjną propozycją do wykorzystywania.

Cały projekt dostępny na:

https://github.com/Zoltw/NLP-language-detection/blob/master/NLP_Languages_Detection.ipynb

Bibliografia

<https://www.statmt.org/europarl/>

<https://docs.python.org/3/>

<https://fasttext.cc/docs/en/supervised-tutorial.html>