LILA - Language Identification LAboratory

Instalacja

System działa z Python3 i wymaga jedynie zainstalowanej biblioteki Unidecode

```
pip install Unidecode
```

Aby korzystać z systemu wystarczy go zaimportować do swojego projektu, np

```
from LILA import *
```

Normalizacja tekstu

System oferuje kilka procedur użytecznych w normalizacji tekstów, w szczególności:

- normalizacja myślników i apostrofów (zamiana na znaki ascii)
- latynizacja znaków diakrytycznych i akcentów (jw)
- eliminacja znaków nieliterowych (włącznie z akcentami lub z ich pominięciem)

```
>>> dediacriticized("¡Feliz Año Nuevo!")
'Feliz Ano Nuevo!'
>>> letters_and_apostrophes_ci(dediacriticized("Übung macht den Meister :)"))
'ubung macht den meister '
>>> only_letters_ci("Don't panic!")
"don t panic "
```

Z dostarczonych procedur i wyrażeń regularnych łatwo jest budować własne formy normalizacji.

Budowa klasyfikatora

Korpus należy dostarczyć w postaci słownika gdzie kluczami są identyfikatory języków, wartościami zaś listy dokumentów (ciągów znaków w kodowaniu utf-8).
Dokumenty należy znormalizować wybraną procedurą przed budową modelu:

```
>>> corpus = {
    "en": json.load(open("korpus_en.json")),
    "de": json.load(open("korpus_en.json")),
    "es": json.load(open("korpus_en.json")),
}

>>> normalized = letters_and_apostrophes_ci
>>> corpus = {lang:[normalized(doc) for doc in corpus[lang]] for lang in corpus}
```

Dla obu implementowanych podejść podstawą jest histogram n-gramów. Do jego budowy można wygenerować procedurę przyjmującą tekst oraz opcjonalnie (do budowy korpusów) histogram zbudowany do tej pory. Procedurę taką generuje procedura wyższego rzędu `mk_histogram_builder` przyjmująca cztery opcjonalne argumenty: minimalną i maksymalną długość n-gramu, procedurę akceptacji n-gramu oraz procedurę preprocesowania n-gramu przed dodaniem do histogramu

```
>>> hb =
mk_histogram_builder(min_n=2,max_n=4,accepted=accept_intoken,processed=stripped)
```

```
>>> hb("policz mi histogram dla tego tekstu")
{'po': 1, 'ol': 1, 'li': 1, 'ic': 1, 'cz': 1, 'mi': 1, 'hi': 1, 'is': 1, 'st':
2, 'to': 1, 'og': 1, 'gr': 1, 'ra': 1, 'am': 1, 'dl': 1, 'la': 1, 'te': 2, 'eg':
1, 'go': 1, 'ek': 1, 'ks': 1, 'pol': 1, 'oli': 1, 'lic': 1, 'icz': 1, 'his': 1,
'ist': 1, 'sto': 1, 'tog': 1, 'ogr': 1, 'gra': 1, 'ram': 1, 'dla': 1, 'teg': 1,
'ego': 1, 'tek': 1, 'eks': 1, 'kst': 1, 'poli': 1, 'olic': 1, 'licz': 1, 'hist':
1, 'isto': 1, 'stog': 1, 'togr': 1, 'ogra': 1, 'gram': 1, 'tego': 1, 'teks': 1,
'ekst': 1}
```

Dostępne procedury akceptacji to `accept_any` (wszystkie), `accept_intoken` (jedynie n-gramy występujące w obrębie jednego leksemu), `accept_suffixes` (jedynie n-gramy obejmujące koniec leksemu), `accept_intoken_suffixes` (jedynie sufiksy leksemów). Dostępne procedury preprocesowania to `unprocessed` (identyczność) i `stripped` (elimijący wiodące/kończące znaki białe).

Aby zbudować klasyfikator w oparciu o korpus `corpus` i procedurę budowy histogramów `hb` wystarczy użyć jednej z dwóch dostępnych procedur `mk_cosine_model` lub `mk_ranking_model`:

```
>>> model1 = mk_cosine_model(corpus,hb)
>>> model1("der schnee ist weiß")
{'en': 0.16159130685823997, 'es': 0.1457812620102102, 'de': 0.3696150834426688}
>>> model1("la nieve es blanca")
{'en': 0.1497045427025336, 'es': 0.2006750733646224, 'de': 0.11740020349012875}
>>> model1("the snow is white")
{'en': 0.2937208192557997, 'es': 0.049568374684066795, 'de':
0.06644535429581588}
```

Tak samo jak korpus, klasyfikowany tekst należy poddać normalizacji:

```
>>> model1(normalized("der Schnee ist weiß!!!"))
{'en': 0.1511548267361471, 'es': 0.13636588396343255, 'de': 0.347015073786404}
```

Jeśli nie interesują nas pełne wyniki, przewidziany przez model język można wyłuskać procedurą `predicted_language`:

```
>>> predicted_language(model1(normalized("Good food & great service!")))
'en'
```

Procedura budująca model rankingowy (out-of-place) przyjmuje opcjonalny argument `top_rank` – ilość n-gramów używanych do profilowania dokumentów. Model zwraca wyniki w postaci dużych liczb całkowitych ujemnych (odwrócona metryka przestawień z (Cavnar & Trenkle, 1994)).

```
>>> model2 = mk_ranking_model(corpus,hb,top_rank=1000)
>>> model2("the snow is white")
{'en': -5694, 'es': -11817, 'de': -10942}
>>> predicted_language(model2(normalized("der Schnee ist weiß")))
'de'
```

Dla korpusów 30k tekstów i n w granicach 1-4 modele budują się 5-50s.

Testowanie klasyfikatora

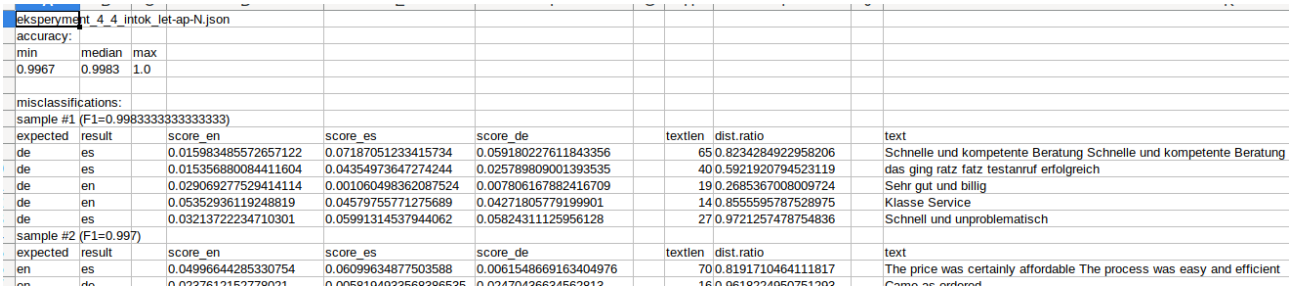
Moduł zawiera kilka użytecznych procedur do testowania klasyfikatorów. Do modułu dołączone są dwa skrypty: `tests.py` przeprowadzający testy klasyfikatorów kosinusowego i rankingowego z

użyciem wskazanego korpusu, oraz `incorrect_classifications.py` generujący plik `.csv` z raportem napotkanych błędów.
Procedura testowa działa w ten sposób że dla ustalonej ilości prób `sample_count` i wielkości próbki testowej `sample_size` dzieli korpus na części testową i treningową (`sample_count` razy), buduje model, ewaluuje próbkę testową a wyniki zapisuje w pliku `.json` o wskazanej nazwie `fname`.
Dla przykładu:

```
>>> sample_size = 1000
>>> sample_count = 9
>>> norm = letters_and_apostrophes
>>> hb = mk_histogram_builder(min_n=2,max_n=2,accepted=accept_intoken)
>>> conduct_cosine_test(corpus,sample_size,sample_count,hb,norm,"test1")
```

Przeprowadzi `9` eksperymentów wycinając próbki testowe wielkości `1000` dokumentów per język używając pozostałych dokumentów jako korpusu treningowego. Wyniki eksperymentu zapisze w pliku `test1.json`. Plik zawiera listę `9` słowników o kluczach `sample_from`, `sample_to` (indeksy początkowy i końcowy dokumentów użytych jako próbka testowa), `acc` (trafność klasyfikacji na ów próbce), `train_s` (czas utwarzania modelu w sekundach), `test_s` (czas testowania w sekundach), oraz `results` pod którym znajduje się lista `1000` wyników częściowych.
Wynik częściowy ma klucze `text` (znormalizowany tekst dokumentu), `expected` (identyfikator języka tekstu z korpusu), `scores` (wyniki modelu dla tego dokumentu).

Plik taki można łatwo przetwarzać własnymi skryptami, w szczególności dołączonym `incorrect_classifications.py` który ze wskazanego pliku generuje plik `.csv` zawierający wyniki klasyfikacji (najgorszy, najlepszy oraz medianę) po którym z każdej próbki wylistowane są teksty niepoprawnie zaklasyfikowane, wraz z ich wynikami, oczekiwanym wynikiem, długością tekstu oraz stosunkiem dwóch najlepszych wyników.



eksperyment_4_4_intok_let-ap-N.json								
accuracy:								
min	median	max						
0.9967	0.9983	1.0						
misclassifications:								
sample #1 (F1=0.9983333333333333)								
expected	result		score_en	score_es	score_de	textlen	dist.ratio	text
de	es		0.015983485572657122	0.07187051233415734	0.059180227611843356	65	0.8234284922958206	Schnelle und kompetente Beratung Schnelle und kompetente Beratung
de	es		0.015356880084411604	0.04354973647274244	0.025789809001393535	40	0.5921920794523119	das ging ratz fatz testanruf erfolgreich
de	en		0.029069277529414114	0.001060498362087524	0.007806167882416709	19	0.2685367008009724	Sehr gut und billig
de	en		0.05352936119248819	0.04579755771275689	0.04271805779199901	14	0.8555595787528975	Klasse Service
de	es		0.03213722234710301	0.05991314537944062	0.05824311125956128	27	0.9721257478754836	Schnell und unproblematisch
sample #2 (F1=0.997)								
expected	result		score_en	score_es	score_de	textlen	dist.ratio	text
en	es		0.04996644285330754	0.06099634877503588	0.0061548669163404976	70	0.8191710464111817	The price was certainly affordable The process was easy and efficient
en	de		0.023761215277902	0.005810493356838653	0.024704366345628	16	0.961823405075129	Came as ordered

---

Tabelki z raportu zostały wygenerowane skryptem `tabela.py`

Wyniki z korpusu SentiOne jako pliki `.json` znajdują się w katalogu `wyniki_senti/` zaś dla korpusu LCC w `wyniki_lcc/`. Oba korpusy załączono w projekcie (`korpus_*.json` i `lcc_news_*.json` odpowiednio).