Pyner Documentation

Wydanie 0.1

Konrad Delong, Antoni Piechnik

February 20 2009

Spis treści

Zawartość:

Wstęp

Projekt został zrealizowany w ramach przedmiotu Inżynieria Oprogramowania. Głównym zadaniem projektu jest utworzenie systemu typu Named Entity Recognition - wyszukiwanie konkretnego typu słów w tekście stron WWW (takich jak nazwiska, imiona, czy pseudonimy).

Projekt docelowo ma być integralną częścią systemu SWAT. Sercem systemu jest algorytm łączący zestawy reguł językowych do utworzenia optymalnych kombinacji rozpoznających encje danego typu w tekście. Dzięki takiemu podejściu nie tylko wykorzystuje podane reguły do stworzenia mozliwie najlepszej kombinacji, ale również pozwoli na łatwe dodanie nowych reguł do aplikacji.

ROZDZIAŁ 2

Struktura systemu

System składa się z następujących części:

2.1 Tagger tekstów do testów

Skrypt mający na celu skanowanie przykładowych tekstów a następnie generowania specyficznych raportów odnośnie ilości wystąpień poszczególnych encji w tekście.

Dzięki skryptowi będzie możliwe przygotowanie tekstów odpornych na konkretne reguły, oraz zwiększenie możliwości testowania aplikacji. Tagger bazować będzie na liście danych encji (np. bazie nazwisk), z której nie będziemy korzystać w pracy Rdzenia aplikacji.

2.2 Generator reguł n-gramowych

W związku z chęcią implementacji reguł bazujących na najpopularniejszych n-gramach utworzony został generator reguł który zbiera najczęściej występujące n {3,4,5} gramy występujące w tekstach przed nazwiskami oraz po, które posłużą za wykładnię do łączenia reguł. Do tworzenia list n-gramów potrzebny jest odpowiedni korpus zawierający nazwiska, jak i również ich baza, dzięki której można je wykrywać w tekście. Generator, w celu uniknięcia zbierania nic nie znaczących n-gramów porównywał będzie nie tylko jego zawartość ale również pozycję w słowie.

2.3 Rdzeń

W tej części aplikacji znajduje się zaimplementowany algorytm który aplikuje przygotowane i wygenerowane reguły do wyszukiwania nazwisk w tekście. Przygotowanych została konkretna liczba reguł, oraz ich przykładowe kombinacje dzięki którym będzie można porównywać ich efektywność na tekstach przykładowych. Rdzeń pobiera również odpowiednie listy n-gramów o różnej długości, wygenerowane uprzednio przez Generatora reguł n-gramowych. Rdzeń został napisany w języku Python, dzięki czemu wyjątkowo proste jest ewentualne dodanie dodatkowych reguł.

Opis działania aplikacji

3.1 Zastosowane reguły (przykłady)

Reguły kontekstowe:

- Słowo ‘doktor’ poprzedza nazwisko
- Słowo ‘mgr’ poprzedza nazwisko
- Słowo ‘mgr inż.’ poprzedza nazwisko
- Reszta tytułów naukowych.
- Słowo ‘lek.’
- Słowo ‘mec.’
- Słowo ‘arch.’
- Reszta tytułów zawodowych.
- Słowo ‘pan’ poprzedza nazwisko
- Słowo ‘pani’ poprzedza nazwisko
- Czasownik w trzeciej osobie liczby pojedynczej następuje po nazwisku (czasownik w formie męskiej czy tez damskiej)
- Imiesłowy takie jak ‘zamieszkały’, ‘urodzony’.

Reguły bazujące na innych encjach:

- Imię poprzedza nazwisko
- Przezwisko znajduje się na 2 gim miejscu w wyrażeniu trójsłownym (np. Antoni ‘Tosiek’ Piechnik)
- Przezwisko znajduje się na 3 cim miejscu w wyrażeniu trójsłownym (np. Antoni Piechnik ‘Tosiek’)

Reguły bazujące na regułach ortograficznych:

- Nazwisko zaczyna się dużą literą

- Nazwisko jest częścią dwu lub trzy wyrazowego wyrażenia w których wszystkie elementy są pisane wielką literą.

Reguły n-gramowe:

- Nazwisko kończy się charakterystycznym sufiksem (np. '-ski', '-icz')
- Pozostałe reguły generowane poprzez system

Dzięki zastosowaniu systemu reguł oraz ich kombinacji, bez problemu będzie można do systemu wstawiać dodatkowe reguły co usprawni jego prace oraz uświetni wyniki osiągane przez algorytm.

Wielką zaletą zastosowanego algorytmu jest fakt, iż reguły tak naprawdę nie muszą dokładnie precyzować wystąpień danych encji (tu nazwisk), ale jedynie sprzyjające temu warunki (które w połączeniu z innymi warunkami mogą definiować reguły).

ROZDZIAŁ 4

Zastosowane technologie

Do implementacji znacznej części aplikacji wykorzystano język Python, ze względu na jego perfekcyjne przystosowanie do prac nad przetwarzaniem języka naturalnego.

W związku z faktem, iż projekt SWAT jest napisany w języku Java, należało wykorzystać swojego rodzaju pomost pomiędzy naszą częścią aplikacji a dotychczasowymi interfejsami wyszukiwania encji w tekstach. W tym celu wykorzystano Jythona, czyli implementację języka Python napisaną w języku Java.

Poza Jythonem korzystano z funkcji wbudowanych w język Python, związanych z przetwarzaniem tekstu oraz konwersją między różnego rodzaju kodowaniem (pozwalająca na komunikację oraz transfer danych z poziomu Javy do Pythona i na odwrót).

ROZDZIAŁ 5

Opis zastosowanych narzędzi

5.1 Jython

Implementacja języka Python w Javie pozwalający na transparentna komunikacje miedzy Pythonem a klasami Javy. Okazał się niezastąpiony przy wiązaniu aplikacji Rdzenia z dotychczasowymi interfejsami projektu SWAT.

Dzięki wykorzystaniu zewnętrznych bibliotek związanych m.in. z kodowaniem udało się bez najmniejszych problemów wywoływać klasy napisane w Rdzeniu (w Pythonie) z poziomu Javy, jak również importować wszelakie pakiety projektu SWAT do kodu aplikacji w Pythonie.

Wiecej informacji na stronie Jythona: http://www.jython.org

5.2 encodings

Moduł Pythona zawierający zbiór najważniejszych i najpopularniejszych kodowań oraz metod z nimi związanych.

Dzięki niemu udało się rozwiązać problem związany z komunikacją (w szczególności przesyłaniem polskich znaków z obiektów Javy do obiektów Pythona)

5.3 egothor

Silnik full-text search z którego korzystano przy tworzeniu systemu.

Wiecej informacji na stronie: http://www.egothor.org/

5.4 Morfologik

Analizator morfologiczny, słownik morfologiczny, korektor gramatyczny.

Wiecej informacji na stronie: http://morfologik.blogspot.com/

Podręcznik użytkownika

6.1 Korzystanie z klas Jythona w Javie

Aby móc skorzystać z detektorów zaimplementowanych w Jythonie na poziomie Javy, należy najpierw je zimportować przy użyciu klasy JythonFactory *jyinterface.factory*:

```
String interfaceName = "org.ppbw.agh.swat.hoover.smith.quantum.detection.IQuantumDetector";
Object obj = JythonFactory.getJythonObject(interfaceName,
                        "pyner/detectors.py", "CapitalDetector");
IQuantumDetector detector = (IQuantumDetector) obj;
```

Następnie można korzystać z detektora tak jak z każdej klasy implementującej interfejs *org.ppbw.agh.swat.hoover.smith.quantum.detection.IQuantumDetector*.

Częścią projektu jest klasa *agh.io.Main* w której zamieszczono przykładowe użycie detektorów zaimplementowanych w Jythonie.

6.2 Implementacja detektorów w Jythonie

Detektory w projekcie pyner to proste funkcje o prostym interfejsie (opisanym w dokumentacji modułu `pyner.rules`). Zgodnośc z interfejsem *org.ppbw.agh.swat.hoover.smith.quantum.detection.IQuantumDetector* uzyskano generując klasy na podstawie funkcji. Zajmuje się tym funkcja `pyner.detectors.gen_detector()`, która otrzymując obiekt funkcji zwraca obiekt metaklasy. Przykładowe użycie funkcji gen_detector:

```
NgramsNeighboursDetector    = gen_detector(rules.ngrams_neighbours)
PrefixesDetector            = gen_detector(rules.prefixes)
SuffixesDetector            = gen_detector(rules.suffixes)
CorpusDetector              = gen_detector(rules.in_name_corpus)
CapitalDetector             = gen_detector(rules.starts_with_capital)
```

pyner.rules – Proste reguły na znajdywanie NE

W tym module znajdują się funkcje implementujące reguły proste. Wszystkie mają taki sam prosty interfejs: otrzymują pojedynczy argument będący listą słów w zdaniu i zwracają listę pozycji, na których znajdują się słowa dopasowane wg. danej reguły.

Dzięki takiej implementacji, łatwo można tworzyć reguły wyższego rzędu, korzystające z reguł prostych.

prefixes(*words*)
: Znajduje słowa poprzedzone którymś z ustalonych (arbitralnie) poprzedników.

suffixes(*words*)
: Znajduje słowa poprzedzające któryś z ustalonych (arbitralnie) następników.

in_name_corpus(*words*)
: Znajduje słowa znajdujące się w korpusie nazwisk (data/names.iso).

starts_with_capital(*words*)
: Znajduje słowa rozpoczynające się wielką literą.

ngrams_neighbours(*words*)
: Znajduje słowa, w których poprzednikach, lub następnikach znajdują się ngramy charakterystyczne dla poprzedników lub następników.

pyner.detectors – Detektory zgodne ze SWAT

gen_detector(*detector*)

Funkcja tworzy obiekt zgodny z interfejsem *org.ppbw.agh.swat.hoover.smith.quantum.detection.IQuantumDetector* na podstawie reguły w formie funkcji.

class NgramsNeighboursDetector()

Detektor utworzony na podstawie reguły pyner.rules.ngrams_neighbours()

class PrefixesDetector()

Detektor utworzony na podstawie reguły pyner.rules.prefixes()

class SuffixesDetector()

Detektor utworzony na podstawie reguły pyner.rules.suffixes()

class CorpusDetector()

Detektor utworzony na podstawie reguły pyner.rules.in_name_corpus()

class CapitalDetector()

Detektor utworzony na podstawie reguły pyner.rules.starts_with_capital()

class CombinedDetector()

Detektor kombinowany, zwracający jedynie te wyniki, które znajdują się w odpowiedziach więcej niż połowy reguł prostych.

class CombinedDetector2()

Detektor kombinowany, podobny do CombinedDetector, lecz obniżający nieco próg wejścia (wymagana zgodność jedynie z 1/3 reguł). Podnosi to pokrycie kosztem dokładności odpowiedzi.

class SmartDetector()

Detektor kombinowany, korzystający z reguł prostych w bardziej wyszukany sposób. Niektóre reguły są traktowane jako pewne dopasowania (np. obecność w korpusie nazwisk nie posiadających znaczenia jako słowa), inne jako pewne ograniczenia (słowa nie wykrytę przez regułę wielkich liter na pewno nie będą nazwiskami). Pozostałe reguły są kombinowane tak jak w CombinedDetector.

Dodatkowe informacje

9.1 Opis struktury folderów/plików w systemie

Pliki projektu umieszczone są w następujących lokalizacjach:

- build.xml - plik build.xml dla ANTa
- data - folder zawiera jący dane na których pracuje system.
- doc - folder zawiera jący tę dokumentację w formacie TeX
- java-lib - folder zawiera jący wykorzystywane biblioteki Javy
- java-src - kody źródłowe Javy
- pyner - źródła systemu (Rdzenia, Taggera oraz Generatora)
- python-lib - biblioteki wykorzystywane przez źródła Pythona

9.2 Repozytorium

Repozytorium publiczne projektu znajduje się na serwerze GitHub pod adresem http://github.com/tph/io-projekt/tree

ROZDZIAŁ 10

Tabele i indeksy

- *Indeks*
- *Indeks modułów*
- *Wyszukiwanie*

Indeks modułów

P

Indeks