Reprezentacja Wiedzy 2010 – Projekt

Jacek Spólnik, Informatyka rok IV, EAIiE

Wstęp

Pierwsza wersja projektu:

- wygenerowałem odpowiednie dane z CLP aby stworzyć własną bazę danych. Jako bazę wybrałem SQL Server Compact Edition 3.5 jako że jest przechowywana w jednym pliku i działa bardzo szybko (możemy zakładać indeks na odpowiednie dane) + projekt w języku C#. Baza danych składała się z dwóch tabel:

  BaseWords:
  * ID (PK)
  * BaseForm
  * Label
  * Prefix (wygenerowany na podstawie base form i forms)

  Forms
  * ID (FK)
  * Form

Dane do bazy danych generowałem skryptem preprocess.py, sprawdzanie czy dane znajdują się w CLP (początkowe testy) znajduje się w pliku checker.py.

Adres projektu – wersja 1 projektu - C# (poza samą bazą > 100 MB !).

Problemy, dlaczego zrezygnowałem - pomimo szybkości rozwiązania (w stosunku do aktualnej wersji w pythonie) pojawił się znaczący problem, mianowicie bazy tego typu mają jedną malutką słabość, gdy ich rozmiar zaczyna przekraczać 100 MB dzieją się z nimi nie za dobre rzeczy. Dodatkowo indeksy nakładane na tę bazę przy jej początkowym rozmiarze 116 MB podczas operowania na niej rosły przekraczając 129 MB co jest maksymalnym rozmiarem domyślnym takiej bazy, później efektywność i niezawodność niestety spada. Zatem postanowiłem przenieść funkcjonalność na pythona i pracować nadal lokalnie (problemy z wifi), ale ściągnąłem lokalnie CLP i wszystko działa:)

Aktualna wersja projektu

Projekt w pythonie.

Składa się z 6 głównych elementów:
'formy.txt' - plik z formami wyrazów występujących w CLP
 (ściągniety z katalogu /usr/local/plp/lab02/ lub podobnego)

- 'application.py' - plik główny aplikacji, przyjmuje nazwę pliku, który ma analizować i nazwę pliku wyjściowego(raport) tutaj należy zaznaczyć, że aktualnie w linijce:

plp.plp_init()
# change encoding - encoding input
--> text_checker.process(sys.argv[1], text_checker.utf_encoding())

należy zmienić drugi argument, jeśli kodowanie jest różne. text_checker defniuje jeszcze jeden typ kodowania -> 'default_encoding()' odpowiadający 'iso-8859-2'

- 'text_checker.py' - zawiera moduł służący do analizy tekstu wejściowego
- 'stemming.py' - zawiera definicję klasy służącej do wykonywania stemmingu na pojedynczym słowie
- 'utility.py' - moduł zawierające pomocne metody
- 'plp.py' - wraper na clp napisany przez Pana ;)

Opis aktualnego rozwiązania

- użytkownik ustawia (aktualnie) kodowanie pliku wejściowego w 'application.py' i podaje nazwy plików wejściowego i wyjściowego (raport)
- program następnie wywołuję metodę process modułu text_checker która przetwarza plik wejściowy - 'text_checker' definiuje trzy wynikowe słowniki, 'results', 'shortcuts', oraz 'rome' (liczby rzymskie)
- Słowa które nie znalazły się w CLP są dalej analizowane
  - Jeśli słowa zawierają tylko duże litery złożone z rzymskich znaków, dodawane są do romes i pomijane w dalszej analizie.
  - Jeśli słowa składają się albo tylko z dużych liter, lub z 1-3 znaków dodawane są do Shortcuts (jak np pl, EURO itd).
  - Pozostałe słowa dodawane są do Results w celu dokonania na nich stemmingu, koncepcje dopiszę do samej dokumentacji:)
- Dokonujemy **stemmingu** - jeśli słowo kończy się na 'x' (w Polskim języku nie odmieniamy słów kończących się na x) lub analizowane słowo z forms.txt ma końcówkę o nie większej liczbie niż 2 podobnych znaków, wówczas słowo dalej nie jest analizowane tylko wrzucane do stałych posiadających tylko formę bazową (wyświetlane na końcu działania programu)
- 'Stemming' - wyszukujemy słowa w 'formy.txt' posiadające jak najdłuższy identyczny postfix z postfix'em naszego słowa. Tutaj warto nadmienić że sam plik 'formy.txt' cache'ujemy w klasie FileCache ('utility.py')
- Następnie tworzymy dla każdego takiego słowa Entity (definicja klasy w 'stemming.py'). Na podstawie takiego entity, tworzymy dla naszego słowa analogiczne entity operując na tożsamych końcówkach (dodając je i ucinając z prefixu)
- aplikacja wypisuje najpierw poszczegolne informacje dla słów posiadających odmiane, a później wypisuje słowa posiadające tylko formę bazową.

Testy

zostały wykonane dwa testy – znajdują się tutaj

a) 'pap-100.not.in' z raportem w 'pap-100.not.output'
b) 'test_rw.in' (testy z Pańskiego pdf) z raportem w 'test_rw.output'