Gabriela Lewandowska
Paweł Prusałowicz

07.06.2019

Prowadzący: mgr Łukasz Neumann

Dokumentacja końcowa

Heurystyczne naprawianie atrybutów tekstowych
Zaawansowane Programowanie w C++

1. Zadanie projektowe

Zadanie projektowe polega na stworzeniu biblioteki z interfejsem w języku Python, która pozwoli na czyszczenie danych rzeczywistych na podstawie słownika. Podstawowymi funkcjonalnościami są ładowanie słownika oraz wykonywanie czyszczenia danych przekazanych jako pythonowa lista.

2. Dokumentacja użytkownika

Biblioteka text_correction jest biblioteką pozwalającą na poprawianie danych tekstowych na podstawie danego słownika.

2.1. Kompilacja biblioteki

Przed rozpoczęciem korzystania z biblioteki należy zapewnić, aby na komputerze była zainstalowana biblioteka C++ Boost oraz narzędzie scons. Po instalacji, w pliku SConstruct należy zmodyfikować ścieżki dostępu do biblioteki Boost i Pythona, następnie włączyć terminal i przejść w nim do folderu projektu oraz wywołać polecenie kompilacji *scons*. Zbudowana w ten sposób biblioteka znajduje się w folderze build.

2.2. Interfejs biblioteki

Text_Correction Text_Correction(string dictionary_path, string pop_words_path) Tworzy obiekt klasy Text_Correction. Atrybuty: dictionary_path: ścieżka do pliku z danymi słownika pop_words_path: ścieżka do pliku z danymi najpopularniejszych słów w języku angielskim
void text_correction.loadDictionary(string dictionary_path) Ładuje słownik z użyciem danych znajdujących się pod wskazaną ścieżką. Atrybuty: dictionary_path: ścieżka do pliku z danymi słownika

list text_correction.correctData(string str_algorithm, Bool english_language, list ns) Wykonuje czyszczenie danych. Atrybuty: str_algorithm: Algorytm poprawiania danych. Może przyjmować wartości: -„norvig”: dla algorytmu Norviga -„trie”: dla algorytmu opartego na drzewie trie -„symspell”: dla algorytmu SymSpell Domyślnym parametrem w przypadku podania nieprawidłowego argumentu jest algorytm Symspell. english_language: Zmienna logiczna wskazująca czy algorytm czyszczenia danych ma być wykonany dla języka angielskiego. ns: Dane poddawane czyszczeniu, jedno słowo stanowi jedną pozycję na liście.
list text_correction.getDictionary() Zwraca słownik obiektu text_correction. Atrybuty: Funkcja nie przyjmuje żadnych atrybutów.
string text_correction.info() Zwraca informacje na temat obiektu. Atrybuty: Funkcja nie przyjmuje żadnych atrybutów.

3. Implementacja

Silnik i główną część aplikacji stanowi moduł napisany w C++. Interfejs do języka c++ stanowi klasa Text_Correction. Zawiera ona funkcje pozwalające wykonywać podstawowe operacje na słowniku oraz czyszczenie danych. Słownik został zaimplementowany jako samodzielna klasa, wykorzystująca wzorzec Singletona. Proces czyszczenia danych jest wykonywany dwuetapowo.

3.1. Wyszukiwanie zbioru prawdopodobnych rozwiązań

Zależnie od wybranej metody czyszczenia tworzona jest instancja klasy Norvig, Trie_Algorithm lub SymSpell_Adapter. Norvig oraz Trie_Algorithm zawierają definicje poszczególnych algorytmów, szukających słów o małej odległości Levenshteina od danego słowa. Klasa Symspell_Adapter stanowi adapter do klasy Symspell, który ze względu na swoją złożoność, został zaczerpnięty z serwisu github [1] i traktowany przez nas jak zewnętrzna biblioteka.

3.2. Tworzenie rankingu słów i wybór najlepszego dopasowania

Do tworzenia rankingu słów stworzono klasę Word_Rating. Wykorzystuje ona prawdopodobieństwa poszczególnych typów pomyłek, bazując na badaniach uniwersytetów Cambridge oraz Aalto [2]. Zgodnie z badaniami, najczęściej występują zamiany liter miejscami (1.65%), rzadziej pominięcia (0.8%), a jeszcze rzadziej wstawienia (0.67%). Na wyliczane przez nas prawdopodobieństwo wpływa

także odległość levenshteina oraz częstotliwość występowania wyrazów w języku angielskim na podstawie danych zgromadzonych przez Adama Kilgarriffa [3].

4. Testy

Program zawiera około 1800 linijek kodu. Osiągnięto ok. 85 % pokrycia kodu testami jednostkowymi. Dokumentację kodu wygenerowano z użyciem programu Doxygen.

W czasie pracy nad projektem wykonano 3 różne testy:
1) Testy jednostkowe z wykorzystaniem Boost biblioteki unittest - **unit_test.cpp**, uruchamiany w terminalu z głównego katalogu projektu poprzez *bash run_unit_test.sh*
Testy podzielone zostały na moduły testujące klasy. Sprawdzały one działanie pojedynczych klas i ich metod.
Wyniki pokrycia testami prezentują się następująco:
File 'src/algorithms/symspell_adapter.cpp' Lines executed: 100.00% of 21
File 'src/algorithms/trie_algorithm.cpp' Lines executed: 53.49% of 43
File 'src/algorithms/trie.cpp' Lines executed:100.00% of 12
File 'src/english_word_popularity.cpp' Lines executed: 96.97% of 33
File 'src/algorithms/norvig.cpp' Lines executed: 87.67% of 73
File 'src/dictionary.cpp' Lines executed: 62.50% of 16
File 'src/word_rating.cpp' Lines executed: 26.83% of 41

2) Test jednostkowy **test_text_correction.py** w Pythonie, wykonany przy użyciu biblioteki unittest. Uruchamiany był z głównego katalogu projektu w terminalu, poprzez *python3 -m test.test_text_correction.* Przetestowany został w ten sposób interfejs biblioteki, komunikacja użytkownika z nią. Komunikacja przebiega sprawnie, a wyniki są zgodne z oczekiwanymi.

3) Test związany z założeniem naszego projektu - porównaniem działania różnych algorytmów służących do wyszukiwania wyrazów o bliskiej odległości Levenshteina - **run_speed_test.sh** uruchamiany z głównego katalogu poprzez *bash run_speed_test.sh*, polegający na znajdowaniu proponowanych rozwiązań dla tekstu z błędami (około 240 wyrazów).

Wyniki:

NORVIG SEARCH TIME: **18960289** MICROSECONDS, MATCHES: 1793 SETUP TIME: 212226
TRIE SEARCH TIME: **910991** MICROSECONDS, MATCHES: 2128 SETUP TIME: 848444
SYMSPELL ADAPTER SEARCH TIME: **388351** MICROSECONDS, MATCHES: 26883 SETUP TIME: 8365623

Obserwujemy przewagę szybkości algorytmu SymSpell ponad dwukrotną nad trie i prawie pięciokrotną nad algorytmem Norviga. Wynika to ze złożonego początkowego procesu przetwarzania danych (setup time) -10 razy dłuższego, niż drzewa trie - który umożliwia późniejsze szybsze wyszukiwanie rozwiązań. Oferuje ich także więcej w tym samym czasie.

4. Podsumowanie

Wynikiem prac jest biblioteka realizująca założone funkcjonalności. Udało nam się wykorzystać wszystkie z wymienionych w dokumentacji wstępnej algorytmów korekcji danych oraz porównać ich działanie.

Problemem okazało się jednak stworzenie i dodanie do biblioteki modułu wstępnie przetwarzającego dane (oczyszczanie ich z niepotrzebnych znaków, np. interpunkcyjnych). Początkowo planowaliśmy zaimplementować ten moduł w języku Python, jednak ostatecznie zabrakło nam czasu. Kolejną funkcją, którą warto byłoby dodać to dane umożliwiające sprawdzanie częstotliwości występowania wyrazów także w innych językach, nie tylko w angielskim.

Wśród problemów, które znacząco wpłynęły na jakość wykonywanych prac należy wymienić instalację biblioteki boost, a następnie kompilację programu w środowisku Windows. Pomimo wielu prób, a także dostosowania pliku SConstruct do działania zarówno na Linuxie jak i na Windowsie nie udało się poprawnie skompilować biblioteki na drugim systemie. Żałujemy tutaj głównie bezowocnej utraty czasu (kilkadziesiąt godzin na osobę), którą moglibyśmy włożyć w rozwój projektu.

Technicznym ulepszeniem obecnej wersji biblioteki mogłoby być przekazywanie funkcjom słownika przy pomocy wskaźnika. W planach autorów była także implementacja funkcji interfejsu użytkownika z parametrami domyślnymi.

Stworzoną przez nas bibliotekę można uznać za udaną, ponieważ w szybki i bardzo precyzyjny sposób podaje rozwiązanie zadanego problemu, a dodatkowo pozwala na spostrzeżenie różnic w działaniu różnorodnych algorytmów.

Autorzy projektu deklarują poświęcenie na projekt 80 i 90 godzin.

[1] https://github.com/erhanbaris/SymSpellPlusPlus
[2] https://userinterfaces.aalto.fi/136Mkeystrokes/resources/chi-18-analysis.pdf
[3] https://www.kilgarriff.co.uk/bnc-readme.html