Wydział Elektroniki
i Technik Informacyjnych

POLITECHNIKA WARSZAWSKA

Systemy agentowe
Wstęp do eksploracji danych tekstowych w sieci WWW

System przeszukiwania projektów open source na podstawie danych z repozytoriów kodu

Sprawozdanie z projektu

Paweł Karwacki *259820*
Maciej Krasowski *259831*
Łukasz Kilaszewski *259822*

28 maja 2018

Wstęp

Podstawowym założeniem projektu było przygotowanie systemu agentowego, który będzie w stanie przeszukiwać projekty open source. Po podaniu wyszukiwanej frazy, system miał zwracać listę najbardziej dopasowanych projektów wraz z odnośnikami prowadzącymi do repozytoriów.

Całość projektu została napisana w języku Python 3. Do zaprojektowania systemu agentowego wykorzystano bibliotekę Pulsar, natomiast do zadań związanych z przetwarzaniem tekstu użyty zostały biblioteki Nltk (tokenizacja, stemming, itp.) oraz Gensim (budowa słownika oraz modeli i indeksów). Kod źródłowy przygotowanego rozwiązania dostępny jest pod adresem `https://github.com/luktor99/SAG-WEDT`.

Dane do przeszukiwania

Jako źródło danych do przeszukiwania wybraliśmy platformę GitHub, głównie ze względu na ilość zgromadzonych tam publicznych repozytoriów oraz dostępność darmowego interfejsu API. Niestety interfejs ten posiada ograniczenie w postaci limitu 5000 zapytań na godzinę. Ponadto, pobieranie repozytoriów w czasie rzeczywistym wiąże się z dużym spowolnieniem działania systemu ze względu na opóźnienia generowane przez liczne zapytania HTTP.

Biorąc pod uwagę powyższe niedogodności, zdecydowaliśmy się na wcześniejsze pobranie pewnej puli repozytoriów. Dzięki temu właściwa aplikacja może uzyskiwać dostęp do wielu repozytoriów z minimalnymi opóźnieniami. Aby wybrać najbardziej interesujące repozytoria, pobieranie obejmuje te o największej ilości "gwiazdek", czyli najbardziej wartościowe z punktu widzenia społeczności serwisu GitHub. Dane składowane są w folderze projektu w folderze `gh-database`. Za pobieranie oraz dostęp do zapisanych repozytoriów odpowiada klasa `GHInterface`, której implementacja znajduje się w pliku `src/ghinterface.py`.

Ilość pobranych repozytoriów może być ustawiana za pomocą parametru `top_repos_to_download` w pliku `src/config.py`. Domyślnie wynosi ona 10000, co daje 100 plików z danymi po 100 repozytoriów w każdym.

Dostosowanie biblioteki Pulsar do wymogów projektu

Analiza zadań koniecznych do wykonania w kolejnych etapach pozwoliła zauważyć, że są one podobne do siebie i wpisują się w jeden schemat – istnieje zbiór niezależnych od siebie zadań, które mogą być wykonywane równolegle. Na podstawie tego spostrzeżenia przygotowana została klasa `Agency` (znajdująca się w pliku `src/Agency.py`), która dostarcza uniwersalny interfejs dla inicjalizacji i finalizacji agentów oraz rozdzielania zadań pomiędzy nich. Zbudowany na bazie tej klasy system pozwala na równoległe wykonywanie wielu zadań oraz jest odporny na uszkodzenie jednego z nich (przy czym może prowadzić do wybrakowania wyników).
Po utworzeniu instancji klasy `Agency` arbiter (w bibliotece Pulsar jest to główny agent) tworzy określoną liczbę agentów potomnych i oczekuje na zakończenie ich inicjalizacji, zazwyczaj jest to ustawienie zmiennych i skopiowanie ich do atrybutu (słownika) klasy `Actor` - `extra`.

Dzięki temu każdy agent ma swoje podręczne dane, gdzie zależnie od przypadku może przechowywać swoją kopię słownika oraz wyniki pracy. W momencie zakończenia inicjalizacji arbiter przydziela agentom zadania z listy zadań do wykonania. Jeśli jakiś agent zakończy swoją pracę wysyła wiadomość do arbitra i oczekuje na nowe zadanie. W przypadku, gdy arbiter nie ma już zadań do rozdysponowania agent wykonuje swoje ostatnie zadanie (podane jako parametr do klasy - zazwyczaj wiąże się to z przesłaniem wyników pracy danego agenta do arbitra). Ostatnim etapem przetwarzania agentowego jest podsumowanie wyników od poszczególnych agentów przez arbitra.

Przygotowanie modeli i plików do przeszukiwania

Przygotowanie ściągniętych wcześniej informacji o repozytoriach do przeszukiwania wymagało przeprowadzenie kilkuetapowego przetwarzania. Każdy etap wykonywany jest przez osobny skrypt, z których każdy tworzony był w oparciu o klasę Agency. Były to kolejno:

- Tokenizacja plików readme, stemming powstałych tokenów oraz utworzenie słownika na bazie ztokenizowanych dokumentów.
- Konwersja ztokenizowanych dokumentów do wektorów bag-of-words na bazie słownika oraz utworzenie modelu TFIDF (term frequency, inverse document frequency), który ma na celu uwzględnienie wartości informacyjnej poszczególnych słów w tekście.
- Konwersja wektorów bag-of-words do przestrzeni TFIDF na bazie modelu TFIDF oraz utworzenie modelu LSI (Latent Semantic Indexing), który ogranicza wielkość rozważanej przestrzeni (w tym przypadku do 400) i kojarzy słowa semantycznie.
- Konwersja wektorów TFIDF do przestrzeni LSI oraz zaindeksowanie powstałych wektorów w celu znacznego przyspieszenia wyszukiwania.

Szczególnie interesujący był punkt pierwszy, bowiem właściwa tokenizacja okazała się bardziej problematyczna, niż można było się tego spodziewać. Po pierwsze należało przekonwertować format markdown (typowy dla plików readme) do czystego tekstu. W pierwotnej wersji później następowała tokenizacja, filtracja, stemming i usunięcie *stop words*. Pierwsze wyniki wyszukiwania pokazały jednak, że chińskie repozytoria psują jakość wyszukiwania. Aby poradzić sobie z tym problemem usunięto wszystkie te repozytoria, dla których liczba słów zawierających znaki nietypowe przekracza 5%.

Dopasowywanie dokumentów do zapytania

Przykładowe wyniki działania systemu

Po uruchomieniu aplikacji użytkownik proszony jest o wpisanie szukanej frazy. W trakcie działania system wyświetla w postaci logów informacje diagnostyczne informujące co aktualnie jest wykonywane i przez którego agenta. W przypadku, gdy jakiś agent zostanie awaryjnie zatrzymany zostanie wyświetlona odpowiednia informacja. W takim przypadku (zgodnie

z założeniami) część pracy wykonanej przez tego agenta zostanie stracona. Ze względu jednak na prawdopodobieństwo takiego zdarzenia oraz znaczne rozproszenie pracy w trakcie eksperymentów nie zaobserwowano drastycznego pogorszenia się wyników działania systemu. Rezultat wyszukiwania pokazywany jest także w postaci logów, które są uporządkowane malejąco względem dopasowania frazy do pliku readme.

Wnioski