Systemy Agentowe

System agentowy do lokalizowania wypadków na podstawie wpisów z Twittera

Dokumentacja projektu

Michał Breiter
Michał Dobrzański
Maciej Janusz Krajsman

Politechnika Warszawska,
Wydział Elektroniki i Technik Informacyjnych.

15 maja 2017

Spis treści

1 Założenia projektowe

1.1 Zadanie projektowe

W ramach projektu należało opracować oraz zaimplementować system agentowy do wykrywania wypadków, katastrof oraz klęsk na podstawie wpisów z Twittera.

1.2 Analiza problemu

Zadaniem systemu agentowego będzie **analiza tekstowa treści tweetów** pobieranych ze znanego serwisu społecznościowego Twitter. Tweet to krótka wiadomość (maksymalnie 140 znaków — na dzień dzisiejszy, wkrótce ma się to zmienić [1]), wysyłana przez użytkowników na serwery systemu. Następnie jest wyświetlana innym użytkownikom.

Projektowany system musi być w stanie pobrać tweety, przeanalizować je pod kątem znaczenia treści oraz przypisać odpowiednią wyjściową kategorię (rodzaj wypadku) wraz z lokalizacją dotyczącą tego wypadku.

W projekcie rozważono dwie kategorie wypadków oraz katastrof. Uwzględniono **pożary** (różnego rodzaju — domów, lasów) oraz **stłuczki pojazdów** (wypadki, kolizje na drogach).

W niniejszym projekcie jako wypadki i katastrofy traktowane są pożary, stłuczki drogowe oraz podobne sytuacje.

1.3 Użyte narzędzia programistyczne

Projekt napisany został w języku Java, przy wykorzystaniu środowiska programistycznego *IntelliJ IDEA 2016*.

W celu pobierania tweetów z Twittera zastosowano bibliotekę **Twitter4J [2]**. Jest to nieoficjalna biblioteka napisana w języku Java służąca do komunikacji z API udostępnionym przez Twittera [3] pozwalającym na pobieranie treści. Aplikacja została zarejestrowana na serwerze Twittera w celu autoryzacji i w ten sposób umożliwiono pobieranie treści w projektowanym systemie.

W ramach przetwarzania językowego (**NLP**) pobieranych treści tekstowych zastosowano bibliotekę **Apache OpenNLP [4]**. Jest to biblioteka udostępniana przez firmę Apache

na licencji Apache. Bazuje na metodach uczenia maszynowego wspomagających przetwarzanie języka naturalnego.

W celu stworzenia środowiska, w którym będą funkcjonowali agenci użyto **framework JADE [5]**. Jest on udostępniany na licencji open-source. Umożliwia tworzenie kontenera, w którym będą funkcjonowali agencji oraz ich dynamicznego dodawania agentów. Framework pozwala na określanie poszczególnych zachowań agentów oraz na implementację sposobu komunikacji pomiędzy nimi.

W części dotyczącej systemu agentowego wykorzystano również system wspomagania zarządzania projektami **Apache Maven [6]**. Podobnie, jak *OpenNLP*, jest udostępniany przez firmę Apache na licencji Apache. Służy do automatyzacji budowy oprogramowania w języku Java.

2 Projekt rozwiązania

Poniżej opisano projekt systemu agentowego od początku do końca.

Proces tworzenia systemu podzielono na trzy etapy:

1. Przygotowanie modelu — Konstrukcja modelu do klasyfikacji treści
2. Implementacja systemu agentowego
3. Wykrywanie bytów w treści (ang. *named entity recigniotion, NER*)

2.1 Przygotowanie modelu

Poniżej znajduje się opis procesu tworzenia modelu do klasyfikacji danych, z którego, w dalszym kroku, korzystać będzie system agentowy.

2.1.1 Dobór narzędzi programistycznych

Na początku wyszukano narzędzia (biblioteki programistyczne) pozwalające klasyfikować treść wiadomości z tweetów. Zastosowano pakiet ***Document Categorizer*** z biblioteki *OpenNLP*. Pozwala on na klasyfikację wszelkiej treści tekstowej do z góry określonych kategorii. Sposób konstruowania klasyfikatora sprowadza się do stworzenia modelu na

podstawie wykrytej treści z wiadomości tweetów. Następnie taki model jest testowany poprzez podanie mu treści wiadomości. Odpowiedzą modelu jest najbardziej prawdopodobna kategoria.

Proces konstrukcji modelu wymagał na stworzenia osobnego projektu w środowisku programistycznym *IntelliJ IDEA*. Projekt ten korzystał z biblioteki *Twitter4J* oraz *OpenNLP*.

Pozostałe użyte później modele są ogólne dla języka angielskiego (sentencer, tokenizer, NER) i pochodzą z oficjalnych źródeł *OpenNLP* [7].

2.1.2 Ograniczenia pobierania treści

Zbadano możliwości biblioteki umożliwiającej pobieranie tweetów. Okazało się, że istnieją pewne ograniczenia dotyczące parametrów pobieranych tweetów.

a) Przede wszystkim nałożone jest **ograniczenie na datę powstania tweetów**. Oficjalne API Twittera umożliwia pobranie wiadomości pochodzących maksymalnie sprzed dwóch tygodni od aktualnej daty [8]. Uniemożliwia to wyszukanie reprezentatywnych treści z przeszłości do konstrukcji modelu. W projekcie posługiwano się relatywnie wąskimi przedziałami czasowymi — między innymi wyszukiwano tweety w przeciągu jednego dnia.

b) Kolejną sprawą wartą uwzględnienia jest fakt, że domyślnie pobierane są tweety standardowe jak i te określone mianem **‚retweeted'** [8]. Oznacza to, że inny użytkownik serwisu Twitter udostępnił tą samą treść wiadomości. W związku z tym pobierane zostaną wiadomości o tej samej treści, ale pochodzące od różnych użytkowników.

   - Przy projektowaniu modelu nie uwzględniano wiadomości typu ‚retweeted'.
   - W trakcie pobierania tweetów przez system agentowy, uwzględniane są wiadomości typu ‚retweeted'. Ilość ponownych wystąpień danego tweetu podkreśla wagę zdarzenia, którego on dotyczy.

c) Następnie uwzględniono język pobieranych wiadomości. Ustawiono go na **angielski**. Tylko tweety pisane w tym języku były pobierane. Wybór języka podyktowany był:

   - Jego popularnością w obecnej kulturze, co bezpośrednio przekłada się na:

  - ilość i jakość dostępnych w sieci modeli i korpusów z nim związanych.
  - ilość materiałów w sieci w nim tworzonych (m.in. tweetów [9]).
- Prostotą jego gramatyki, w porównaniu z np. gramatyką języka polskiego.

d) Oficjalne API posiada ograniczenie na ilość pobieranych wiadomości za pomocą jednego zapytania HTTP. W związku z tym w dla jednego zapytania określano maksymalną możliwą ilość wiadomości do pobrania, czyli **100** [8].

e) Niektóre tweety posiadają parametr **geolocation**. Określa on miejsce, skąd ta wiadomość została wysłana. Mogą to być zarówno współrzędne geograficzne jak nazwa miejsca, miejscowości, miasta. Tworzony on jest przez użytkownika serwisu — nie jest on obligatoryjny. Podczas projektowania oceniono, że wartość tego parametru może być przydatna do ogólnego działania systemu.

   - Okazało się jednak, że średnio **3%** pobieranych tweetów ma określony ten parametr.
   - Ponadto, fakt wysłania tweeta z konkretnej lokalizacji nie musi wcale oznaczać, że tam właśnie miało miejsce opisywane zdarzenie.

   W związku z powyższymi, zaniechano prób konstruowania systemu z jego uwzględnieniem. Zastosowano inne rozwiązanie, opisane dalej.

f) Zapytanie HTTP umożliwia pobieranie treści za pomocą sprecyzowanych słów kluczowych (ang. *keywords*) występujących w wiadomości lub za pomocą **hasztagów** (ang. *hashtags*). Hasztagi to pojedyncze słowa, rozpoczynające się od znaku #. Nadają one szerszy kontekst wypowiedzi i ułatwiają wyszukiwanie treści w sieci. W projekcie oceniono, że są one nieprzydatne jako parametry wyszukiwania z racji tego, że **uogólniają treść**. W systemie w celu stworzenia modelu istotne są szczegółowe wiadomości (pod względem tego, czego one faktycznie dotyczą). Dlatego wyszukiwanie wiadomości odbywa się z pomocą sprecyzowanych słów kluczowych.

2.1.3 Proces konstruowania modelu

Pakiet ***Document Categorizer*** **[11]** z biblioteki *OpenNLP* reprezentuje wynikowy model za pomocą pliku binarnego (*.bin*).

Wyjściowy plik modelu tworzony jest za pomocą wejściowego pliku tekstowego, który musi być sformatowany następująco:

<**wynikowa kategoria**><spacja><**treść**><koniec linii>

Im większa liczba treści w wejściowym pliku tekstowym, tym większe prawdopodobieństwo poprawy jakości klasyfikacji przez tworzony model.

2.1.4 Przygotowanie danych do uczenia modelu

Program łączy się z serwerem Twittera i za pomocą biblioteki *Twitter4J* dokonuje zapytania HTTP z parametrami określonymi według specyfikacji oficjalnego API [3]. Parametry zostały tak dobrane, żeby uwzględniały wszelkie ograniczenia opisane w 2.1.2.

Wyszukiwanie treści odbywa się za pomocą sprecyzowanego zapytania (ang. ***query***) korzystającego ze słów kluczowych.

Program korzysta z utworzonego wejściowego **pliku z kategoriami**, w którym zapisano wszystkie wynikowe kategorie, który ma się posługiwać konstruowany model wraz z odpowiadającymi im słowami kluczowymi.

Podczas działania programu wykryto, że do treści wiadomości dokładana jest również nazwa użytkownika rozpoczynająca się od znaku @ jak i adres url danego tweeta. Są to treści nieistotne pod kątem uczenia modelu. W związku z tym, należało je usunąć. Dokonano tego za pomocą następującej procedury:

```
// remove urls "https://..."
String trimmed = s.getText().replaceAll("https?://\\S+\\s?", "");
// remove users "@..."
trimmed = trimmed.replaceAll("@\\S+\\s?", "");
```

Pobrane i wstępnie przetworzone treści tweetów są zapisywane do plików wyjściowych według formatu określonego do konstrukcji modelu, czyli:

<**wynikowa kategoria**><spacja><**treść**><koniec linii>

Te pliki zostały wykorzystane do konstruowania modelu.

2.1.5 Tworzenie (uczenie) modelu

Podczas uczenia modelu poczyniono następujące założenia:

1. Nie każdy tweet dotyczy wypadku bądź katastrofy. Model musi być w stanie rozróżnić treść wiadomości i ocenić czy faktycznie dotyczy ona wypadku, czy nie. W związku z tym, zaistniała potrzeba odseparowania zbędnych treści. W tym celu **dodano trzecią kategorię nazwaną „Other"**. To podejście umożliwia rozróżnienie treści tweetów pod kątem poszukiwań treści faktycznie dotyczących wypadków.
2. Model, który został nauczony większą liczbą wiadomości, to model lepszy.
3. Większa liczba wiadomości typu „Other" pozwoli na lepszą klasyfikację treści, gdyż liczba wiadomości ogólnych zdecydowanie przewyższa liczbę wiadomości dotyczących konkretnych wypadków bądź katastrof.
4. Model musi być nauczony możliwie najlepszymi wiadomościami. Przez to rozumiemy, że wiadomość dotycząca kategorii „pożar" (ang. *fire*) będzie faktycznie dotyczyła pożaru (na przykład budynku: *„#NewYork #Albany #Buffalo Fire at Franklin County paper mill "*), a nie treści z pożarem niezwiązanych (na przykład nazwy utworu: *„I'm listening to Drake — She's on fire"*).

W konsekwencji stworzono model o następujących parametrach:

```
TOTAL AMOUNT OF TWEETS GATHERED (tweets.txt) = 3175
Other ~= 1500
Fire ~= 900
Car_crash ~= 900

Keywords:
Fire:
    fire evacuation
    fire forest
    fire building
    fire police
    fire firemen
```

```
Car_crash:
    crash route
    car crash
    car collision

Other:
    the
    a
    I
    in
    yes
    no
```

Powyżej określono liczbę wiadomości wyszukiwanych po konkretnych słowach kluczowych dla trzech kategorii (***„Other", „Fire", „Car_ crash"***).

Dla kategorii „*Fire*" jak i „*Car_ crash*" starano się dobrać jak najbardziej precyzyjne, szczegółowe słowa kluczowe.

Natomiast dla kategorii „*Other*" poszukiwano treści zawierających najbardziej ogólne słowa kluczowe — najczęściej występujące słowa w języku angielskim, czyli między innymi: *the, a, no.*

Nauczony model został zapisany do pliku binarnego (*.bin*). Tym modelem posłużono się następnie w tworzonym systemie agentowym.

2.2 Projekt i implementacja systemu agentowego

W tym punkcie opisano architekturę systemu agentowego. Uwzględniono rodzaje agentów, określono ich zachowanie oraz opisano ich komunikację.

2.2.1 Dobór narzędzi programistycznych

Utworzono osobny projekt dla systemu agentowego w środowisku *IntelliJ IDEA 2016*, z użyciem systemu wspomagania zarządzania projektami *Apache Maven*. Cały system opiera się na platformie agentowej *JADE*.

2.2.2 Architektura systemu agentowego — wstęp

Projektowany system posiada własne środowisko uruchomieniowe, w którym mogą funkcjonować poszczególni agenci. Cały system określany jest mianem platformy. Platforma posiada tak zwany *główny kontener*, który służy do rejestrowania innych kontenerów wraz z agentami. Agenci są identyfikowani unikalnymi nazwami, co pozwala na ich identyfikację.

Główny kontener zawiera domyślnie trzech agentów spełniających specjalne funkcje. Są oni niezbędny w celu prawidłowego działania platformy.

AMS — (ang. *Agent Management System*) — jest odpowiedzialny za zapewnienie unikalności nazw dla każdego agenta żyjącego w systemie.

DF — (ang. *Directory Facilitator*) — agent udostępniający usługę *Yellow Pages* (ogłaszanie się agentów wraz z możliwością ich odnalezienia w celu wykonania konkretnego zadania przez innych agentów). W projektowanym systemie nie zastosowano tej usługi ze względu na konkretną specyfikację agentów określoną poniżej.

RMA — agent odpowiedzialny za zarządzanie platformą z poziomu graficznego interfejsu użytkownika.

Proces uruchamiania systemu wymaga wstępnej kompilacji całego projektu. Wykonywane jest to następującą komendą w katalogu źródłowym projektu.

```
mvn compile
```

Uruchomienie systemu polega na początkowym uruchomieniu środowiska uruchomieniowego wraz z platformą. Następnie dołączani są agenci za pomocą specjalnych komend. Poniżej przedstawiono komendy wywoływane w powłoce systemowej w projektowanym systemie.

```
mvn -Pjade-main exec:java
mvn -Pjade-crawler exec:java
mvn -Pjade-process exec:java
```

Pierwsza komenda odpowiedzialna jest za **uruchomienie całej platformy** wraz z głównym kontenerem. Powołuje również trzech specjalnych agentów do życia: AMS, DF oraz

RMA. Uruchamiany jest graficzny interfejs użytkownika pozwalający na monitorowanie pracy agentów.

Natomiast druga i trzecia komenda **powołuje do życia agentów** określonych w specjalnych plikach konfiguracyjnych. Określa się w nich nazwy agentów oraz podaje wartości parametrów.

Przykładowy plik konfiguracyjny wygląda następująco:

```
agents=\
     crawler1-agent:org.sag.wedt.crawler.CrawlerAgent(
     categorizer1-agent,Crash,car,crash);\
     crawler2-agent:org.sag.wedt.crawler.CrawlerAgent(
     categorizer2-agent,Fire,fire,disaster);
port=10099
host=localhost
main=false
no-display=true
```

Powyższa konfiguracja spowoduje uruchomienie dwóch agentów nazwanych odpowiednio *crawler1-agent* oraz *crawler-2-agent*. Każdy z nich przyjmuje zestaw czterech parametrów wejściowych określonych w okrągłych nawiasach.

Rodzaje oraz role agentów w projektowanym systemie opisano w następnym podpunkcie.

2.2.3 Architektura systemu agentowego — role agentów

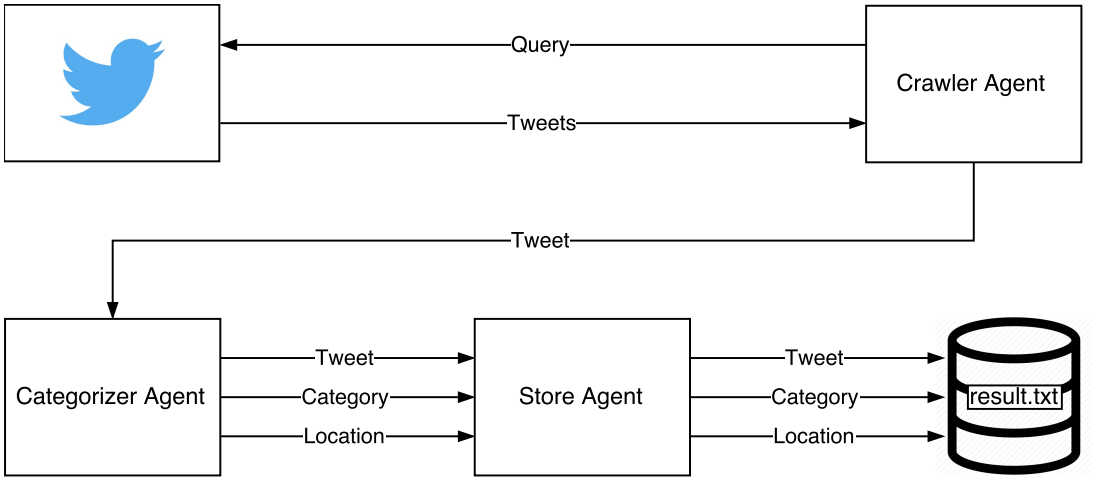
W projektowanym systemie istnieją trzy rodzaje wyspecjalizowanych agentów:

CrawlerAgent — Szukający tweetów w zasobach serwisu Twitter

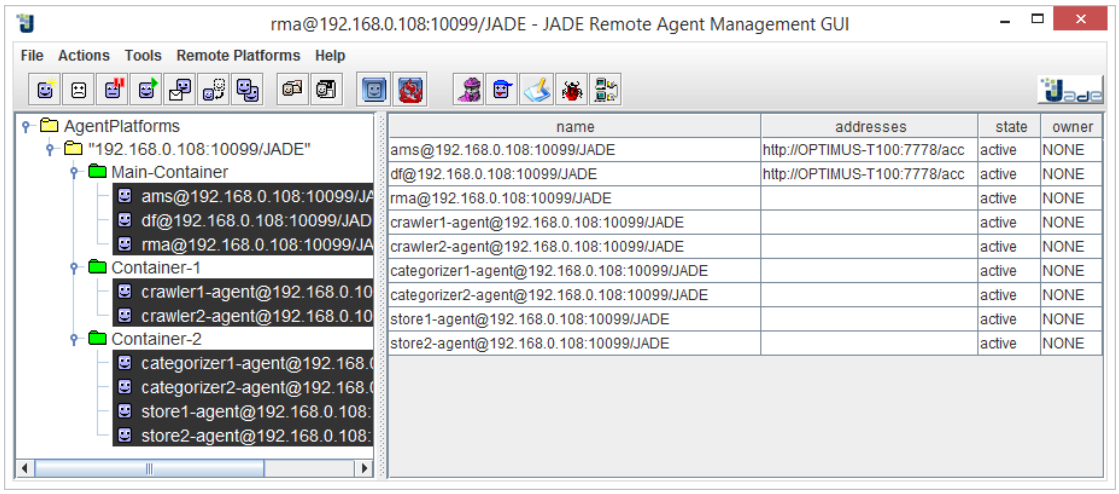
CategorizerAgent — Przydzielający tweetom kategorię i wyszukujący informację o lokalizacji, której tweety dotyczą.

StoreAgent — Zapisujący rezultaty na dysku.

Ich interakcje przedstawia schemat 1.
Na rysunku 2 przedstawiono interfejs graficzny środowiska *JADE*, wraz z aktywnymi agentami.



Rysunek 1: Architektura systemu agentowego



Rysunek 2: Interfejs środowiska JADE

2.2.4 Architektura systemu agentowego — zachowania agentów

Agenci zaprogramowane mają następujące zachowania:

CrawlerAgent:

CrawlTwitterBehaviour *(CyclicBehaviour)* — Przeszukuje serwis Twitter przy pomocy *Twitter4J* i przekazuje znalezione tweety do *SendTweetsBehaviour* (i dodaje to zachowanie do agenta).

SendTweetsBehaviour *(OneShotBehaviour)* — Wysyła pakiet z treścią tweetu do *CategorizerAgent*.

CategorizerAgent:

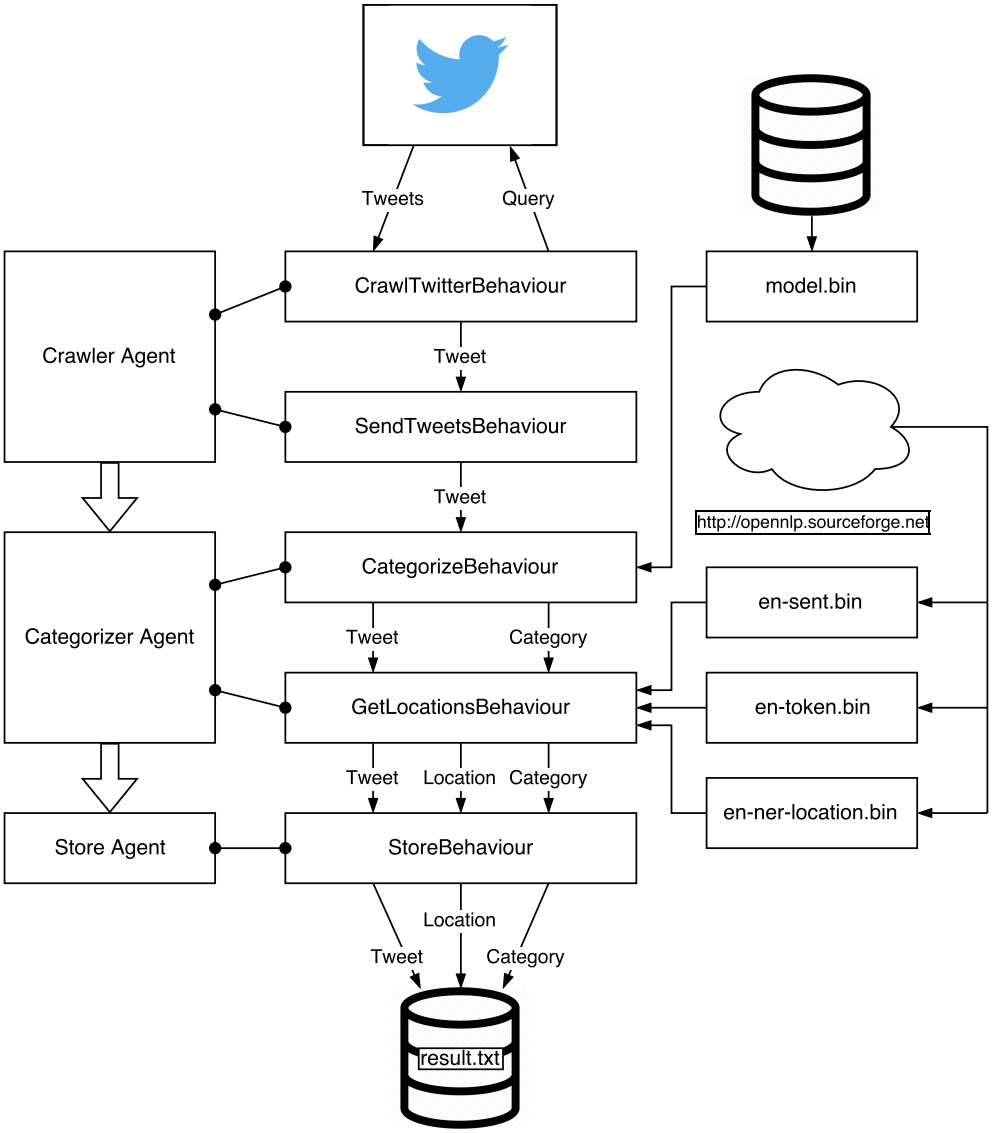
CategorizeBehaviour *(CyclicBehaviour)* — Czeka na pakiet z treścią tweetu od *CrawlerAgent*. Gdy go otrzyma, kategoryzuje wiadomość zgodnie z przyjętym modelem (*model.bin*). Następnie uruchamia *GetLocationsBehaviour*.

GetLocationsBehaviour *(OneShotBehaviour)* — Wyszukuje w treści tweetu lokalizację zdarzenia (metodą NER), a następnie wysyła do *StoreAgent* pakiet zawierający treść tweetu, kategorię mu przypisaną, oraz znalezione w nim lokalizacje.

StoreAgent:

StoreBehaviour *(CyclicBehaviour)* — Czeka na pakiet zawierający treść tweetu, kategorię mu przypisaną, oraz znalezione w nim lokalizacje od *CategorizerAgent*. Gdy go otrzyma, zapisuje go na dysku (w pliku *result.txt*)

Schematyczny zapis powyższych informacji ilustruje rysunek 3



Rysunek 3: Zachowania agentów i stosowanie modeli

2.2.5 Pobieranie tweetów

System zakłada znajdywanie informacji o wypadkach w możliwie krótkim czasie od ich wystąpienia. Z tego względu, pobieranie tweetów do przetworzenia realizowane jest przez użycie udostępnionego przez serwis Twitter Streaming API [10]. Pozwala on na pobieranie tweetów w czasie rzeczywistym. Użyty został wariant filtrowania, który zwraca wszystkie wiadomości dla których spełnione są parametry zapytania. Dla standardowego uwierzy-

telnionego dostępu nie ma ograniczenia maksymalnej ilości przesyłanych wiadomości, pobierane wiadomości powinny jednak stanowić mniej niż 1% wszystkich wiadomości pojawiających się w systemie Twitter.

2.3 Wykrywanie bytów w treści — NER

Uwzględniając trudności wynikające ze zbyt małej liczby wiadomości zawierających parametr geolocation oraz faktu, że najczęściej dotyczy on miejsca, skąd tweet został wysłany, a nie jakiego miejsca on dotyczy, posłużono się procedurą NLP zwaną NER (ang. *named entity recognition*).

2.3.1 Dobór narzędzi programistycznych

Zastosowano pakiet dostępny w ramach biblioteki *openNLP*, zwany ***Name Finder [11]***, pozwalający na wykrycie bytów w treści. Posłużono się również gotowym, wytrenowanym modelem udostępnianym przez stronę internetową *openNLP*. Został on wytrenowany z użyciem korpusów w języku angielskim oraz z nastawieniem na wyszukiwanie lokalizacji (między innymi nazw własnych: miast, dzielnic, obszarów).

2.3.2 Użycie modelu

Z modelu korzystają agenci typu *CategorizerAgent* wykonując zachowanie *GetLocations-Behaviour*. Z dokumentacji [11] wynika, że przed zastosowaniem NER model należy podzielić na zdania, a następnie na słowa. W tym celu należy użyć dwóch innych modeli, **wykrywacza zdań** (*en-sent.bin*) oraz **tokenizera** (*en-token.bin*). Ich użycie zostało uwzględnione na schemacie 3.

2.3.3 Wynik zastosowania modelu

Rezultatem zastosowania NER w systemie jest uzyskanie niezbędnej informacji o lokalizacji, do której nawiązuje treść danego tweetu. Po przejrzeniu wyników okazuje się, że **NER działa całkiem dobrze**, jednak nie doskonale — część lokalizacji jest pomijana. Ponadto, niektóre słowa uznane jako lokalizacja, nie zawsze faktycznie odnoszą się do niej.

Wynikowy plik (*result.txt*) zawiera wszystkie zebrane tweety, zapisane w formacie, kolejno:

1. Przydzielona przez system kategoria,
2. Kategoria, po której tweet został wyszukany treść,
3. [Wykryta lokalizacja 1, wykryta lokalizacja 2, ...],
4. Treść tweetu — ze spacjami, do końca linii.

Wszystkie pozycje oddzielone są od siebie spacjami, na koniec ustawiany jest znak końca linii.

3 Wnioski i wyniki

Zaprojektowany system agentowy **wykonuje założone zadanie projektowe** — pobiera treść wiadomości z serwisu Twitter, przetwarza ją za pomocą metod NLP oraz zapisuje wyniki do pliku. Dokonuje próby wykrycia wypadku (pożar, stłuczki samochodowe) oraz znalezienia odpowiadającej lokalizacji.

Podczas projektowania systemu napotkano różne **trudności** jak i ograniczenia wynikające z zastosowanej technologii lub z samej natury rzeczy. Poniżej opisano je szczegółowo.

3.1 Dobór języka treści

Z uwagi na ogromne rozpowszechnienie języka angielskiego w sieci, jego wybór był naturalny. Sprzyjało to zarówno dostępności materiałów [7] (korpusy, a w konsekwencji modele: tokenizer, sentencer, NER), jak ilości znalezionych tweetów — spośród języków, którymi posługują się autorzy (w rezultacie: są w stanie zweryfikować poprawność klasyfikacji), w języku angielskim jest ich zdecydowanie najwięcej [9].

Drugą przyczyną takiego wyboru jest prostota angielskiej gramatyki. W rezultacie — ponownie, dużo łatwiej o działający model dla języka angielskiego, niż np. polskiego.

3.2 Wyszukiwanie treści

W projekcie uznano, że wiadomości będą wyszukiwane po słowach kluczowych w nich występujących (ang. *keywords*). Rozważono wyszukiwanie wiadomości za pomocą **hasztagów**, jednak takie podejście powodowałoby pobieranie zbyt ogólnych wiadomości, które nie byłyby najczęściej związane z wypadkami.

3.3 Klasyfikacja treści wiadomości

Podjęto próbę stworzenia klasyfikatora dla treści wiadomości zawartych w Tweetach. Klasyfikator budowany był w oparciu o trzy główne klasy — „*Fire*", „*Car_ crash*", „*Other*". Wprowadzenie trzeciej klasy „*Other*" ma na celu odrzucenie wszelkiej treści nie związanej z wypadkami.

Próbowano pobrać jak największą liczbę wiadomości reprezentatywnych dla danych klas. Ograniczeniem tutaj była maksymalna liczba wiadomości do pobrania w ramach jednego żądania HTTP — 100 oraz nałożone na Twitter API ramy czasowe — możliwość pobierania wiadomości maksymalnie sprzed dwóch tygodni.

W związku z tym w przeciągu miesiąca uruchamiano program do pobierania wiadomości w celu znalezienia różniącej się treści.

Postarano się, aby dane trenujące model były unikalne — w tym celu odfiltrowano wiadomości typu „*retweeted*".

Szukano możliwie najbardziej szczegółowych wiadomości dotyczących pożarów oraz wypadków samochodowych w celu znalezienia reprezentatywnych danych trenujących. Użyto w tym celu wspomniane wcześniej słowa kluczowe (ang. *keywords*) dla dwóch kategorii. Przykładowe pobrane wiadomości:

- Fire — *Fire in Gray asssited living facility forces early morning evacuation*
- Fire — *1030 Ouellette Windsor Regional Hospital small fire on 3rd floor residents room. Evacuation of 3rd floor wing underway fire is out. *JL*
- Car_crash — *BREAKING: Crash closes Route 15 in Hamden Stay ahead of the delays:*
- Car_crash — *Crash delaying traffic near Higgins, Route53*

Natomiast dla kategorii „*Other*" starano się znaleźć jak największą liczbę wiadomości o możliwie najbardziej ogólnej treści. W związku z tym poszukiwano wiadomości po następujących słowach kluczowych: *the, a, I, in, yes, no* — najczęściej stosowane słowa w języku angielskim. Przykładowe pobrane wiadomości:

- Other — *Looking for #Chefs in #LosAngeles who are badass and edgy!! Last two days of casting!!!*
- Other — *at this point in life I'm convinced I should never have a child that could possibly go through what I've experienced*

Takie podejście pozwoliło na relatywnie dobre wytrenowanie klasyfikatora. Jednak problemem jest zbyt mała liczba wiadomości trenujących (około 3200) oraz fakt, że wiadomości nie związanych z dwoma specyficznymi kategoriami jest bardzo wiele. Należałoby znaleźć **dużą pulę ogólnych wiadomości** (szacunkowo około 20000) oraz spróbować znaleźć bardziej **szczegółowe słowa kluczowe**.

3.4 Wykrywanie lokalizacji

Okazało się, że parametr ***geolocation*** dla Tweetów nie jest użyteczny, gdyż jest on określony w około 3% wiadomości. Ponadto, dotyczy on najczęściej miejsca wysłania wiadomości. To sprawia, że istnieje ryzyko wystąpienia sytuacji prowadzących do błędów w przypisywaniu lokalizacji do zdarzenia: użytkownik serwisu Twitter, będący reporterem, pisze z lokalizacji A (np. studia redakcyjnego, na terenie którego nie zdarzył się jeszcze żaden pożar ani wypadek samochodowy) o wypadkach/katastrofach z lokalizacji B (np. znajdującej się po drugiej stronie kontynentu). W konsekwencji uznano ten parametr za mało istotny.

W projekcie zastosowano podejście ***NER***, które polega, kolejno, na:

1. Podziale tekstu na zdania,
2. Podziale zdań na tokeny (słowa),
3. Wyszukaniu słów, które określają lokalizację (właściwy NER, na podstawie modelu).

Poniżej przedstawiono rezultaty:

- Fire Fire [Tampa] *fire damage water removal Tampa, FL #firedamage Listed at*:
- Fire Fire [Victoria Park, Toronto] *Trains are holding on Line 2, Bloor Danforth, both ways at Victoria Park for a Toronto Fire investigation.*
- Car_crash Crash [Main Street] *Car accident right at the 55 south bound ramp on Main Street in Imperial. Be careful out there!*
- Car_crash Crash [Kholo Road] *RT [131940_Metro] Warrego Highway, North Ipswich — Crash Multi-vehicle Westbound At Kholo Road, Delays expected, Use alternat...*

Podejście NER umożliwia **skuteczne wykrycie lokalizacji w wiadomości**. Należy jednak zaznaczyć, że występują również przypadki niepoprawnego wykrycia lokalizacji. Przykładem może być następujący wynik:

- Car_crash Crash [Little] *RT Little man riding in the car today looking good like always & having some good dreams!! #SayCheese!!*

Powyższy przykład ukazuje również niepoprawne działanie zbudowanego klasyfikatora — ta wiadomość nie dotyczy wypadku samochodowego.

Następny przykład obrazuje możliwość wykrycia dwóch lokalizacji jednocześnie:

- Fire Fire [Oregon, Columbia River Gorge] *Derailed: Train carrying volatile oil from the Bakkens derails. Spills and fire in Oregon's Columbia River Gorge*

Warto również zwrócić uwagę na fakt zmiany klasyfikacji próbki — poniższy tweet znaleziony został jako związany z wypadkiem samochodowym, ale sklasyfikowany jest (poprawnie) jako dotyczący pożaru (który jest tu istotniejszy). Ponadto, gdyby klasyfikator miał możliwość przydzielania wielu kategorii, klasyfikacja do obu tych grup nie byłaby błędem Wypadek dotyczył kolei, jednak w istocie miał miejsce i był wypadkiem komunikacyjnym. Widać tu jednak dwie kolejne słabości użytego modelu NER. Po pierwsze, (tu należy porównać z poprzednim przykładem), należałoby zastosować jeszcze jeden etap

przetwarzania. Na chwilę obecną system uznaje, że pożary miały miejsce w *„Oregonie”* oraz w *„Ore.”* — warto nauczyć go, że to jest to samo miejsce. Po drugie, model nie wykrył *„PORTLAND”*. Możliwe, że przyczyną było stosowanie wielkich liter, choć w dobrze skonstruowanym modelu nie powinno to mieć znaczenia (*case-insensitivity*)

- Car_crash Fire [Ore .] *Oil train's derailment sparks massive fire: PORTLAND, Ore. — A train towing cars full of oil derailed Fri...*

3.5 Tweety „retweeted”

Przy nauce modelu należy dostarczyć mu jak najwięcej różnych próbek tekstu. Tweety oznaczone jako „retweeted” są więc na tym etapie zbędne. Nie ma jednak istotnego powodu, by odrzucać je na etapie wyszukiwania informacji poprzez system agentowy. To, że informacja powtarza się wielokrotnie, oznacza, że wiele osób o niej napisało — co sugeruje, że opisywane w niej zdarzenie jest ważne, popularne.

API Twittera [8] pozwala na odrzucanie retweetowanych tweetów.

3.6 Zastosowany system agentowy

Użycie systemu agentowego w tym projekcie przyniosło korzyści przede wszystkim dydaktyczne — autorzy nauczyli się, jak taki system funkcjonuje i jak go zaprojektować. Z tego punktu widzenia cel został w pełni osiągnięty — system działa i realizuje swoje zadania. W kategorii korzyści funkcjonalnych — jedyną osiągniętą jest równoległe przetwarzanie danych, co wpływa na jego szybkość. Można by to było jednak osiągnąć również poprzez zastosowanie klasycznej wielowątkowości.

Do grona realnych (nie zawartych w tym projekcie) korzyści, które daje system agentowy (a klasyczna wielowątkowość nie), można zaliczyć m.in. możliwość rozproszenia funkcjonalności systemu (a co za tym idzie — agentów) na różne maszyny.

3.7 Podsumowanie

System miał za zadnie wykrywać wypadki i katastrofy na podstawie wpisów z Twittera. Projekt realizuje założone cele, w czasie bliskim rzeczywistemu znajduje potencjalnie interesujące wpisy i przypisuje do nich lokalizację.

System do użytku pozanaukowego na pewno wymaga dopracowania. Ulepszona powinna zostać jakość klasyfikacji modelu oraz wyszukiwanie lokalizacji. Prawdopodobnie, w celu ułatwienia, jako końcowa faza przetwarzania powinno zostać dodane grupowanie wpisów o tych samych wydarzeniach do jednej grupy.

Realizacja projektu przybliżyła autorom od strony praktycznej zastosowanie systemów agentowych oraz metody przetwarzania języka naturalnego.

Bibliografia

[1] Adam M. Stark, Mark D. Plumbley, Twitter moves away from 140 characters, ditches confusing and restrictive rules [online], 24.05.2016, http://techcrunch.com/2016/05/24/twitter-moves-away-from-140-characters-ditches-confusing-and-restrictive-rules/ , dostęp: 04.06.2016r.

[2] *Twitter4J* — A Java library for the Twitter API [online], 2007, http://twitter4j.org/en/index.html , dostęp: 04.06.2016r.

[3] *Twitter* Developers [online], 2016, https://dev.twitter.com/overview/documentation , dostęp: 04.06.2016r.

[4] *Apache OpenNLP* — Welcome to Apache OpenNLP [online], 2010, https://opennlp.apache.org/ , dostęp: 04.06.2016r.

[5] *Jade* Site | Java Agent DEvelopment Framework [online], 2016, http://jade.tilab.com/ , dostęp: 04.06.2016r.

[6] *Maven* | Welcome to Apache Maven [online], 02.02.2016, https://maven.apache.org/ , dostęp: 04.06.2016r.

[7] *OpenNLP* Tools Models [online], 04.05.2011, http://opennlp.sourceforge.net/models-1.5/ , dostęp: 04.06.2016r.

[8] *REST APIs* | Twitter Developers [online], 2016, https://dev.twitter.com/rest/public , dostęp: 04.06.2016r.

[9] *Statista*: Most-used languages on Twitter as of September 2013 [online], 12.2013, http://www.statista.com/statistics/267129/most-used-languages-on-twitter/ , dostęp: 04.06.2016r.

[10] *The Streaming APIs* | Twitter Developers [online], 2016, https://dev.twitter.com/streaming/overview , dostęp: 04.06.2016r.

[11] *Apache OpenNLP* Developer Documentation [online], 23.02.2012, https://opennlp.apache.org/documentation/manual/opennlp.html , dostęp: 04.06.2016r.