

Realizacja prostego silnika reguł walidacyjnych przy użyciu technik NLP

Mariusz Wójcik

16 czerwca 2019

Streszczenie

Dokument stanowi raport z realizacji projektu, którego celem było opracowanie metody skutecznego wykorzystania technik NLP do transkrypcji reguł walidacyjnych zapisanych językiem naturalnym na kod wykonywalny aplikacji.

Dokument składa się z trzech części:

1. Przygotowanie i trening modelu - skupia się ona na zagadnieniach związanych z przygotowaniem próbki uczącej użytej do treningu modelu.
2. Aplikacja - część ta dotyczy prezentacji wyników w oparciu o napisaną w tym celu aplikację.
3. Ocena wyników - część ta zawiera wykaz problemów, które ujawniły się podczas pracy nad aplikacją i które nie znalazły należytego rozwiązania.

Spis treści

I	Przygotowanie i trening modelu	3
1	Początki	4
2	Realizacja	4
3	Abstrakcyjny model reguły	5
4	Przygotowanie próbki uczącej	8
5	Trening modelu	9
II	Aplikacja	11
6	Architektura aplikacji	12
6.1	Sekcja danych wejściowych	12
6.2	Sekcja prezentacji wyników analizy NLP	13
6.3	Sekcja wygenerowanego kodu	14
7	Eksperymenty	15
7.1	Reguła z jednym warunkiem i jedną akcją	15
7.2	Reguła z podwójnym warunkiem	18
7.3	Reguła z akcją „w przeciwnym wypadku”	20
7.4	Reguła z wywołaniem innej reguły	22
8	Próbki zaburzone	25
8.1	Literówki i nieznane słowa	25
8.2	Zaburzenia struktury i niezgodność z abstrakcyjnym modelem reguły	27
8.3	Podsumowanie	28
III	Ocena wyników	29
9	Pytania bez odpowiedzi	30
9.1	Ocena jakości danych uczących	30
9.2	Rozszerzalność próbki danych uczących	30
9.3	Optymalny dobór parametrów uczenia	30
9.4	Odporność na błędy i zaburzenia	30
9.5	Optymalny dobór narzędzi i technologii	30

Część I

Przygotowanie i trening modelu

1 Początki

Jakiś czas temu miałem okazję obejrzeć film, który zrobił na mnie ogromne wrażenie. „*Arrival - Nowy Początek*” w reżyserii Denisa Villeneuve’a - to obraz niezwykle. Porusza on problematykę szerokopojętej, wielopłaszczyznowej komunikacji (a czasem skutków jej braku) . W niesamowicie sugestywny i obrazowy sposób pokazuje mechanizm kształtowania się podstaw wspólnego języka i nawiązywania kontaktu. Proces stopniowego budowania u wspólnionych modeli pojęciowych prowadzący do porozumiewania się tym samym językiem wydał mi się tak logiczny i uporządkowany, że sprawiał wrażenie niemal algorytmicznego... Pamiętam swoją myśl, że skoro możliwe jest tak precyzyjne określenie reguł stojących u podstaw nawiązania skutecznej komunikacji, to droga do zbudowania inteligentnych maszyn porozumiewających się z nami „ludzkim” językiem wydaje się już bardzo krótka.

Do niedawna wydawało mi się, że porozumiewanie się językiem naturalnym jest domeną przynależną wyłącznie człowiekowi. Miałem poczucie, że prace nad komputerowym przetwarzaniem języka naturalnego mają wymiar wyłącznie akademicki. Okazało się jednak, że dynamiczny rozwój algorytmów sztucznej inteligencji i przetwarzania maszynowego dotknął również tej dziedziny. Gdzieś na styku matematyki, informatyki i lingwistyki wykształciła się dziedzina, która funkcjonuje jako *NLP* (*ang. natural language processing*).

Techniki NLP koncentrują się na analizie, przekształcaniu i generowaniu języka naturalnego. Dzięki nim komputery nabywają umiejętności nie tylko analizy tekstu, ale również nauki i wyciągania wniosków. Dają one możliwość analizy nie tylko składni zdań, ale również doszukiwania się ich znaczeń i ukrytych pomiędzy słowami intencji. Czasami uświadamiam sobie że to wszystko razem brzmi jak czysta fantastyka. Bo jak niby sens, znaczenie i intencje można przeliczyć na liczby i twardo zakotwiczyć w dziedzinie algebry liniowej ?

Tajemnicy tej uchyla jedna z najciekawszych książek, jaką miałem przyjemność ostatnio czytać, mianowicie „*Natural Language Processing in Action*”. Jest to bardzo przystępnie napisany przewodnik, dzięki któremu łatwiej oswoić się z podstawowymi prawami rządzącymi światem NLP. Pozycja nie traktuje o rzeczach najłatwiejszych, a mimo to czyta się ją z dużą przyjemnością.

Z teorią często jest tak, że w którymś momencie chciałoby się ją zobaczyć w praktyce. Z tej potrzeby zrodził się pomysł na aplikację, którą możnaby zrealizować przy użyciu technik i algorytmów NLP. Przyszedł mi do głowy generator kodu aplikacji, który byłby w stanie przekształcić tekst napisany językiem zbliżonym do naturalnego bezpośrednio do kodu wykonywalnego. Oczywiście zakładałam że tego typu rozwiązanie miałoby zastosowanie do jakiegoś ściśle określonego aspektu działającej aplikacji, np. walidacji dokumentu, czy sprawdzania reguł poprawności modelu dziedziny. I tak właśnie powstał mój miniprojekt, którego celem jest zobaczenie o co tak naprawdę chodzi z tym NLP . :) . Zapraszam do zapoznania się z założeniami i otrzymanymi wynikami.

Mam świadomość, że jeśli chodzi o NLP, jestem na początku drogi. Nie mogę powiedzieć nawet tego, że udało mi się zrobić jeden krok, ale wiem jedno... podróż zapowiada się naprawdę imponująco...

2 Realizacja

Do realizacji moich założeń wybrałam napisaną w Javie bibliotekę *Apache OpenNLP*. Dostarcza ona narzędzi realizujących wiele aspektów przetwarzania języka naturalnego. W moim projekcie skupię się technice nazywanej *Named Entity Recognition (NER)*.

Polega ona na rozpoznawaniu w tekście określonych bytów nazwanych. Najczęściej są to imiona, nazwiska, nazwy własne itp. W moim przypadku chciałbym stworzyć własny model, który zostanie przyuczony do rozpoznawania poszczególnych elementów konstrukcji reguły walidacyjnej (takich jak słowa kluczowe rozpoczynające i kończące bloki, operatory, akcje i ich parametry).

Żeby to osiągnąć konieczne jest przygotowanie odpowiednio opisanej próbki uczącej, a następnie wykorzystanie jej do treningu modelu.

3 Abstrakcyjny model reguły

Przygotowanie próbki rozpocznę od opracowania schematu reguły.

Na początek wypiszę sobie kilka przykładowych reguł walidacyjnych.

1. *Jeśli wiek_pacjenta jest większy od 18 wtedy zgłoś błąd „Pacjent jest osobą dorosłą.”, w przeciwnym wypadku wyświetl komunikat „Pacjent został zakwalifikowany do leczenia pediatrycznego.”.*
2. *Jeśli data_kwalifikacji jest mniejsza od '01-01-2019' wtedy zgłoś wyjątek „Data sprzed roku 2019.”, w przeciwnym wypadku sprawdź regułę RS-001.*
3. *Gdy saldo_rachunku jest większe od 100 oraz saldo_rachunku jest mniejsze niż 1000 wtedy wyświetl komunikat „Saldo rachunku jest prawidłowe.”, w przeciwnym razie zgłoś błąd „Nieprawidłowe saldo rachunku”.*
4. *Jeśli data_teraz jest niewiększa niż data_ważności wyświetl komunikat „Wniosek jest aktualny.” w przeciwnym wypadku zgłaszaj błąd „Wniosek utracił ważność”.*

Przyjmuję uproszczenie, że każda rozpoznawana reguła składała się będzie z trzech wyróżnialnych bloków:

WARUNKI AKCJA_TAK (AKCJA_NIE)?

Poszczególne bloki oddzielone będą od siebie słowami kluczowymi oznaczającymi rozpoczęcie i zakończenie bloku.

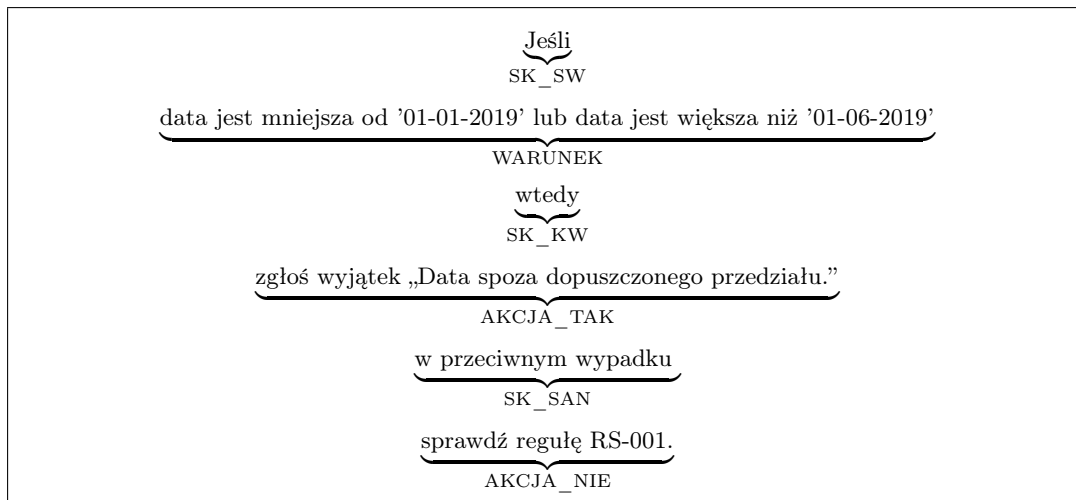
W celu ich wyróżnienia wprowadzam następujące oznaczenia:

1. SK_SW - Start sekcji warunku
2. SK_KW - Koniec sekcji warunku
3. SK_SAN - Start sekcji akcji wykonywanej przy niespełnionym warunku

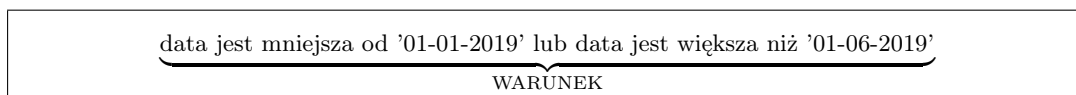
Schemat reguły przyjmuje następującą postać:

SK_SW WARUNKI SK_KW AKCJA_TAK (SK_SAN AKCJA_NIE)?

Rzut oka na przykład:



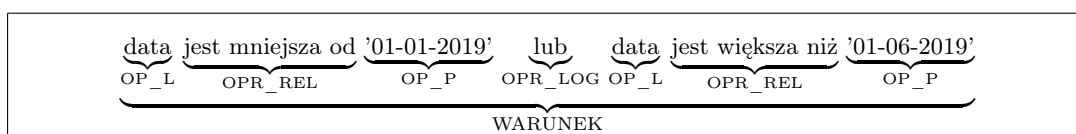
Ponieważ kluczowe jest właściwe rozpoznanie sekcji warunku, chciałbym wyłączyć go przed nawias i przez chwilę skupić się wyłącznie na nim.



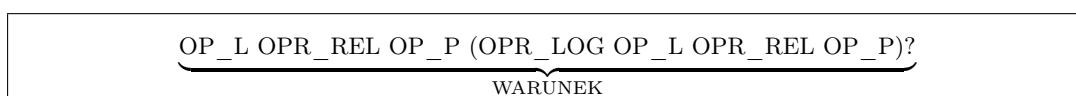
Na początek trzeba zauważyć, że powyższa sekcja składa się z dwóch niezależnych wyrażeń warunkowych połączonych operatorem logicznym *lub*. Każdy z pojedynczych warunków składa się z kolei z operatora relacyjnego (*jest mniejsza*, *jest większa*), oraz z dwóch operandów (*lewego i prawego*). Wprowadzam więc następujące oznaczenia:

1. OP_L - Operand lewy
2. OPR_REL - Operator relacyjny
3. OP_P - Operand prawy
4. OPR_LOG - Operator logiczny

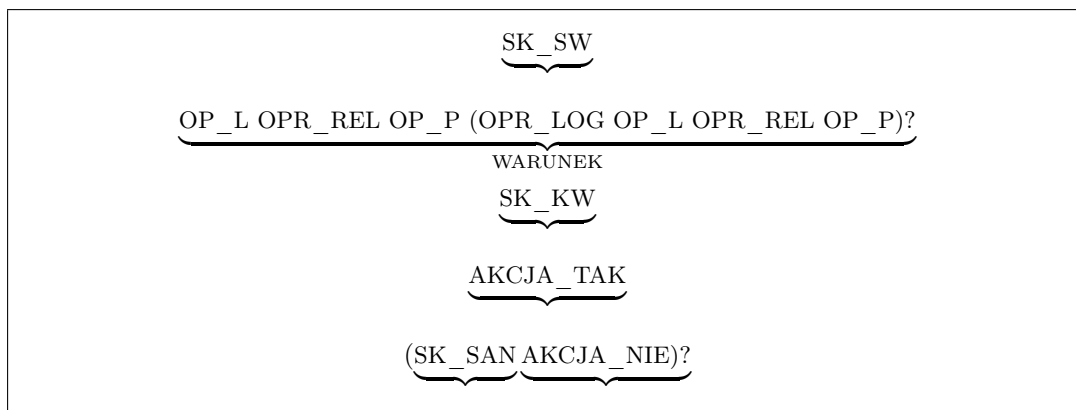
Po podstawieniu, przykładowy warunek można



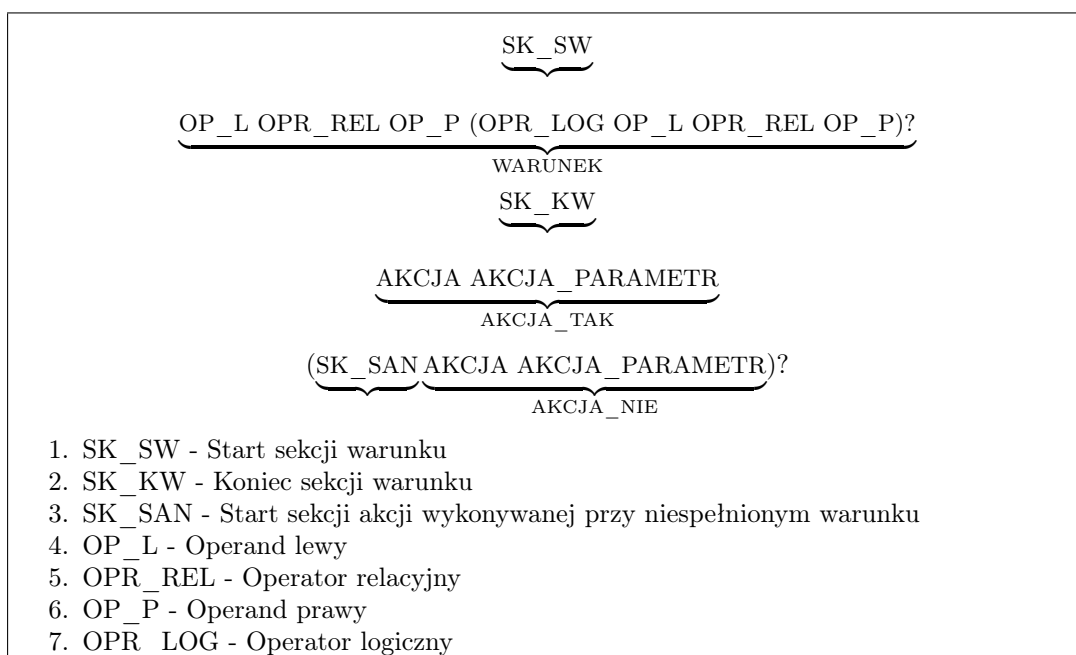
Zatem zapis symboliczny sekcji warunkowej będzie wyglądał następująco:



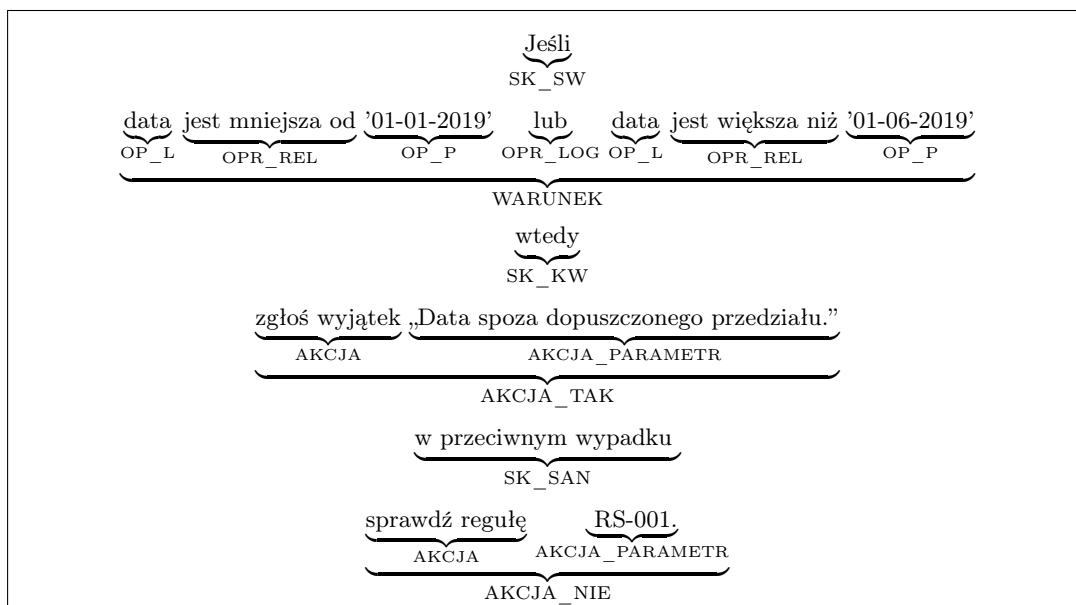
Po dokonaniu podstawienia w sekcji *WARUNKI* abstrakcyjny model reguły przyjmie następującą postać:



Jeśli chodzi o akcje, to sprawa wydaje się prostsza, bo każda z nich składa się z części mówiącej o tym co ma być zrobione (AKCJA), oraz z jakim parametrem ma być wykonane (AKCJA_PARAMETR). Ostatecznie więc, po wykonaniu podstawienia nasz model przyjmie następującą, ostateczną formę:



I jeszcze spojrzenie na przykład:



4 Przygotowanie próbki uczącej

Mechanizmy NER wchodzące w skład *OpenNLP* pozwalają na stworzenie własnego modelu i przyuczenie go do rozpoznawania specyficznych bytów domenowych. Próbka ucząca jest dosyć obszernym zbiorem przykładów (dokumentacja OpenNLP mówi o minimum 15 tyś. zdań), w których w specjalny sposób otagowane zostały kluczowe frazy.

```
<START:SK_SW> jeśli <END> <START:OP_L> xxx <END> <START:OPR_REL> jest
większy niż <END> <START:OP_P> xxx <END> <START:SK_KW> wtedy <END>
<START:AKCJA> zgłoś błąd <END> <START:AKCJA_PARAMETR> xxx <END> .
```

Byty, które docelowo mają być rozpoznawane przez model należy umieścić pomiędzy tagami *<START:NAZWA_BYTU>* i *<END>*. Dodatkowo każda reguła danych uczących zbudowana jest według schematu omawianego wcześniej abstrakcyjnego modelu reguły. Zgodne z nim są również nazwy encji. W przypadku tych części reguły, które są zmienne i specyficzne dla każdej instancji (takie jak komentarze, nazwy operandów, wszelkie parametry) użyłem frazy *xxx*, która oznacza że będzie tu coś, o czym na tym etapie nie możemy nic powiedzieć (znamy tylko pozycję tego tokena względem innych encji).

Wygenerowanie próbki uczącej okazało się zagadnieniem samym w sobie. Do tego celu napisałem aplikację pythonową, która w danych wejściowych otrzymuje przewidywane przykładowe wartości encji, a następnie otagowuje je, tworzy ich iloczyny kartezjańskie i ostatecznie konstruuje z nich zdanie zgodne z założonym schematem.

Poniżej znajdują się wartości poszczególnych encji użyte to wygenerowania danych uczących.

$$SK_SW = \begin{cases} \text{jeśli} \\ \text{gdy} \\ \text{jeżeli} \end{cases}$$

$$OPR_REL = \{ \text{nie} \} * \begin{cases} \text{jest równy} \\ \text{jest równa} \\ \text{jest równe} \\ \text{jest większy} \\ \text{jest mniejszy} \\ \text{jest różny} \\ \text{jest większa} \\ \text{jest mniejsza} \\ \text{jest różna} \\ \text{jest większe} \\ \text{jest mniejsze} \\ \text{jest różne} \end{cases} \begin{cases} \{ \text{niż} \} \\ \{ \text{od} \} \end{cases}$$

$$SK_KW = \begin{cases} \text{wtedy} \\ \text{to} \end{cases}$$

$$SK_SAN = \{ \text{w przeciwnym wypadku} \}$$

$$AKCJA = \left\{ \begin{array}{l} \text{zgłoś błąd} \\ \text{zgłoś błąd walidacji} \\ \text{raportuj błąd} \\ \text{wyświetl komunikat} \\ \text{sprawdź regułę} \\ \text{sprawdzaj regułę} \end{array} \right\}$$

Przykładowe, bardziej złożone reguły utworzone opisaną wcześniej techniką:

1. $\langle START:SK_SW \rangle$ jeśli $\langle END \rangle \langle START:OP_L \rangle xxx \langle END \rangle \langle START:OPR_REL \rangle$ jest większy niż $\langle END \rangle \langle START:OP_P \rangle xxx \langle END \rangle \langle START:SK_KW \rangle$ wtedy $\langle END \rangle \langle START:AKCJA \rangle$ zgłoś błąd $\langle END \rangle \langle START:AKCJA_PARAMETR \rangle xxx \langle END \rangle \langle START:SK_SAN \rangle$ w przeciwnym wypadku $\langle END \rangle \langle START:AKCJA \rangle$ sprawdzaj regułę $\langle END \rangle \langle START:AKCJA_PARAMETR \rangle xxx \langle END \rangle$.
2. $\langle START:SK_SW \rangle$ jeśli $\langle END \rangle \langle START:OP_L \rangle xxx \langle END \rangle \langle START:OPR_REL \rangle$ nie jest mniejsza od $\langle END \rangle \langle START:OP_P \rangle xxx \langle END \rangle \langle START:OPR_LOG \rangle$ oraz $\langle END \rangle \langle START:OP_L \rangle xxx \langle END \rangle \langle START:OPR_REL \rangle$ nie jest równy $\langle END \rangle \langle START:OP_P \rangle xxx \langle END \rangle \langle START:SK_KW \rangle$ wtedy $\langle END \rangle \langle START:AKCJA \rangle$ zgłoś błąd $\langle END \rangle \langle START:AKCJA_PARAMETR \rangle xxx \langle END \rangle \langle START:SK_SAN \rangle$ w przeciwnym wypadku $\langle END \rangle \langle START:AKCJA \rangle$ zgłoś błąd $\langle END \rangle \langle START:AKCJA_PARAMETR \rangle xxx \langle END \rangle$.
3. $\langle START:SK_SW \rangle$ jeśli $\langle END \rangle \langle START:OP_L \rangle xxx \langle END \rangle \langle START:OPR_REL \rangle$ jest większy niż $\langle END \rangle \langle START:OP_P \rangle xxx \langle END \rangle \langle START:OPR_LOG \rangle$ lub $\langle END \rangle \langle START:OP_L \rangle xxx \langle END \rangle \langle START:OPR_REL \rangle$ jest różny od $\langle END \rangle \langle START:OP_P \rangle xxx \langle END \rangle \langle START:SK_KW \rangle$ wtedy $\langle END \rangle \langle START:AKCJA \rangle$ zgłoś błąd $\langle END \rangle \langle START:AKCJA_PARAMETR \rangle xxx \langle END \rangle \langle START:SK_SAN \rangle$ w przeciwnym wypadku $\langle END \rangle \langle START:AKCJA \rangle$ zgłoś błąd walidacji $\langle END \rangle \langle START:AKCJA_PARAMETR \rangle xxx \langle END \rangle$.

Liczność próbki użytej do treningu modelu ustawiłem na poziomie ok 20 000 rekordów.

5 Trening modelu

Proces trenowania modelu realizowany jest przez niewielką aplikację (*Kotlin, OpenNLP NER API*), której zadaniem jest dostarczenie danych próbki, ustawienie parametrów uczenia, wystartowanie procesu trenowania, oraz zapisanie wygenerowanego binarnego pliku modelu. Najistotniejszy fragment aplikacji przedstawiony jest na listingu.

```
private fun trenujModel(aZbiorRegul:Path): TokenNameFinderModel {
    // reading training data
    var inputFactory: InputSteamFactory?
    try {
        inputFactory =
            MarkableFileInputStreamFactory(aZbiorRegul.toFile())
    } catch (e: FileNotFoundException) {
        throw IllegalArgumentException(e)
    }

    var sampleStream: ObjectStream<*>?

    sampleStream = NameSampleDataStream(
        PlainTextByLineStream(inputFactory, StandardCharsets.UTF_8))

    // setting the parameters for training
    val params = TrainingParameters()
    params.put(TrainingParameters.ITERATIONS_PARAM, 70)
    params.put(TrainingParameters.CUTOFF_PARAM, 1)

    // training the model using TokenNameFinderModel class
    var nameFinderModel: TokenNameFinderModel?
    try {
```

```

        nameFinderModel = NameFinderME.train("en"
        , null
        , sampleStream
        , params
        , TokenNameFinderFactory.create(null
        , null
        , mutableMapOf<String, Any>()
        , BioCodec()))

    return nameFinderModel

} catch (e: IOException) {
    throw IllegalArgumentException(e)
}
}

```

Raport podsumowujący pojedynczą sesję treningową wykonaną w oparciu o przygotowaną wcześniej próbkę wygląda następująco:

```

> Task :app-modul-silnik-regul:wytrenujModel
=====encje_reguly_probka_uczaca.reg
Indexing events with TwoPass using cutoff of 1

Computing event counts... done. 259140 events
Indexing... done.
Collecting events... Done indexing in 6,59 s.
Incorporating indexed data for training...
done.
Number of Event Tokens: 259140
Number of Outcomes: 13
Number of Predicates: 654
Computing model parameters...
Performing 70 iterations.
1: . (259100/259140) 0.9998456432816238
2: . (259140/259140) 1.0
3: . (259140/259140) 1.0
4: . (259140/259140) 1.0
5: . (259140/259140) 1.0
Stopping: change in training set accuracy less than 1.0E-5
Stats: (259140/259140) 1.0
...done.
Compressed 654 parameters to 322
79 outcome patterns
Trained model saved to file in location=>src/main/resources/modelnlp/encje_reguly_model.bin

```

Trening modelu jest krokiem kończącym etap przygotowawczy. Zapisany plik binarny jest gotowy do użycia go w aplikacji. W następnej części postaram się opisać jak można posłużyć się nim do rozwiązania konkretnego problemu.

Część II

Aplikacja

6 Architektura aplikacji

Aplikacja składa się z dwóch warstw-części GUI i części serwerowej. Część serwerowa odpowiedzialna jest za całościową analizę reguł, oraz generowanie kodu. Jest ona napisana w technologii Spring Boot, w języku Kotlin. Warstwa GUI stworzona została w Angularze. Komunikacja odbywa się przy pomocy serwisów REST. Z punktu widzenia użytkownika aplikacja składa się z trzech wyraźnie wyodrębnionych części

1. Sekcji danych wejściowych
2. Sekcji wyników analizy NLP
3. Sekcji wygenerowanego kodu

6.1 Sekcja danych wejściowych

W tej części aplikacji użytkownik może zarządzać danymi, jakie poddawane są analizie NLP. Interfejs aplikacji umożliwia dodawanie nowych reguł, modyfikację treści istniejących, oraz kasowanie reguł.



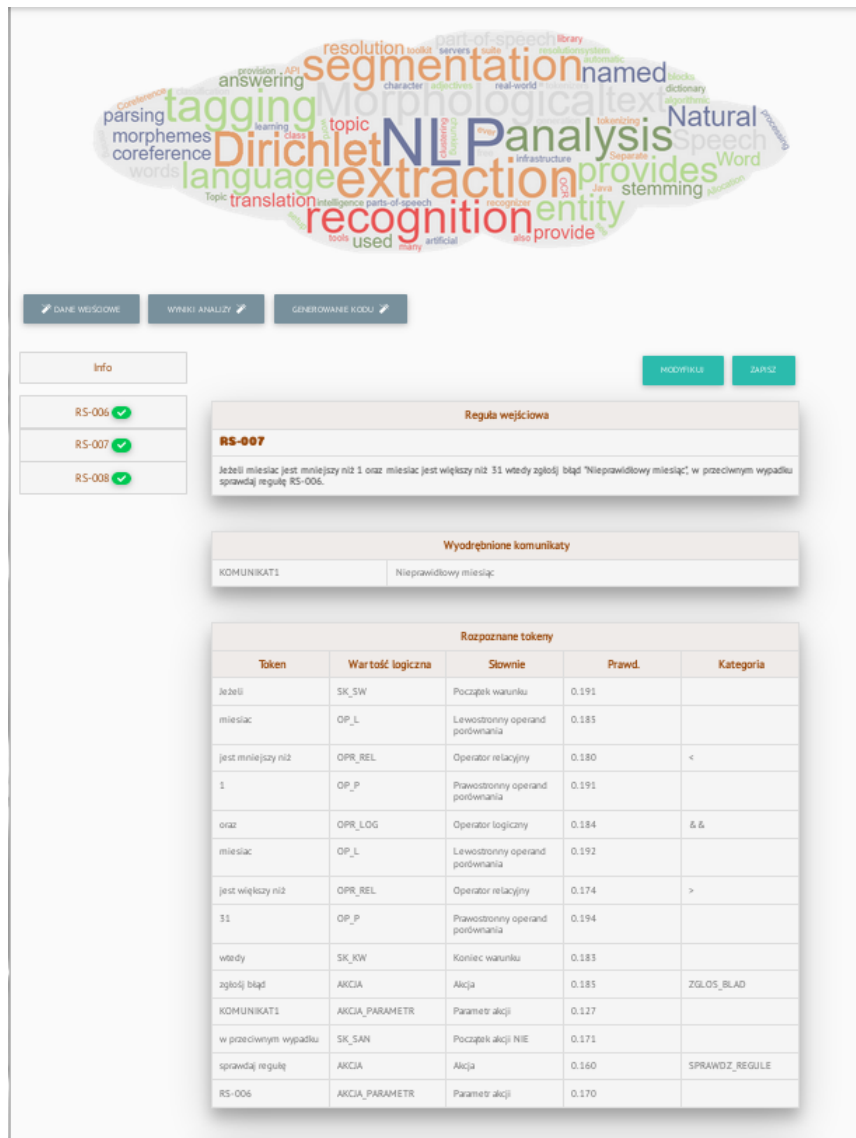
Rysunek 1: Główne okno aplikacji - sekcja definicji reguł wejściowych

6.2 Sekcja prezentacji wyników analizy NLP

Ten ekran jest nieco ciekawszy od poprzedniego. Prezentuje wyniki analizy reguł wykonanej w oparciu o algorytmy uczenia maszynowego dostępne w ramach biblioteki *OpenNLP*. Wyniki analizy prezentowane są w tabelach według następującego schematu:

1. **Reguła wejściowa** - informacja o treści reguły poddawanej analizie algorytmicznej.
2. **Wyodrębnione komunikaty** - jest to lista komunikatów użytkownika, które zostały odnalezione w regule. W celu uproszczenia przetwarzania sekwencji przez algorytmy NLP komunikaty są wyodrębniane i zastępowane słowem - markerem identyfikującym komunikat. Jako komunikat uznaje się ciąg znaków otoczonych znakiem podwójnego cudzysłowu.
3. **Rozpoznane tokeny** - jest to tabela pokazująca wyniki działania algorytmu NLP. Prezentuje ona regułę rozbity na pojedyncze tokeny. Przy każdym tokenie wskazana jest logiczna encja, do której przyporządkował go algorytm dokonujący analizy, oraz liczbowe prawdopodobieństwo z jaką nastąpiło dopasowanie. Ostatnia kolumna tej tabeli ma zastosowanie tylko do wybranych bytów logicznych i służy do normalizacji tokenów. Np. tokeny "żówny", "żówna", "jest równy" należą do jednej wspólnej kategorii oznaczającej równość "=", a tokeny "jest większy lub równy", "jest nie mniejszy niż" oznaczają kategorię "»="
4. **Rozpoznane parametry wejściowe** - w tabeli tej prezentowane są tokeny uznane jako parametry wejściowe do reguły. Zwykle występują one w wyrażeniach warunkowych. W przypadku gdy warunek reguły skonstruowany jest w taki sposób że następuje porównanie z wartością stałą (np. liczbą lub datą), wtedy taka wartość jest traktowana jako parametr wejściowy reguły z przypisaną wartością domyślną. Warunkiem poprawności analizy reguły jest właściwe przyporządkowanie typów danych do parametrów reguły. Jeśli nie uda się wnioskowanie automatyczne, to informacje te muszą być wprowadzone przez użytkownika w do drugiej kolumny tabeli parametrów.
5. **Rozpoznane wywołania innych reguł** - W tabeli tej prezentowane są odnalezione odwołania do innych reguł. Konieczne jest dokonanie mapowania parametrów wejściowych reguły wołającej i reguły wołanej.

Prezentacja wyników następuje po wybraniu reguły z listy w panelu bocznym. Zielony znaczek świadczy o tym że dopełnione zostały wszystkie doprecyzowania i mapowania parametrów, więc aplikacja posiada komplet informacji potrzebnych do wygenerowania kodu.



Rysunek 2: Główne okno aplikacji - sekcja analizy NLP

6.3 Sekcja wygenerowanego kodu

W trzeciej sekcji programu pokazany jest wynik przekształcenia reguły do kompilowalnego kodu *Kotlin*. Dla każdej reguły tworzona jest osobna metoda. Nazwa metody jest tożsama z kodem reguły. Metoda przyjmuje parametry, które zaprezentowane są w tabeli „Parametry wejściowe” na ekranie wyników analizy.



Rysunek 3: Główne okno aplikacji - sekcja wygenerowanego kodu

Uwaga! Sekcja ta jest dostępna tylko wtedy, gdy wszystkie reguły zaprezentowane na ekranie analizy danych 2 zostaną zwalidowane pozytywnie, czyli aplikacja uzyska wszystkie dane potrzebne do wygenerowania kodu.

7 Eksperymenty

Teraz chciałbym pokazać jak działa moja aplikacja. W tym celu spróbuję trochę poeksperymentować z danymi i pokazać to, co do tej pory udało się osiągnąć.

7.1 Reguła z jednym warunkiem i jedną akcją




Wyobraźmy sobie sytuację że robimy system rejestrujący badania medyczne wykonane na rzecz pacjenta. Chcielibyśmy się dowiedzieć, czy rejestrowane badanie jest badaniem refundowanym (nasza wyobrażona refundacja dotyczy tylko badań zarejestrowanych w roku 2019). O czasie rejestracji badania

mówi zmienna „data_rejestracji_badania”. W przypadku gdy badanie nie jest refundowane system ma wyświetlić komunikat informacyjny.

Na początek definicja reguły, niech brzmi ona następująco:

Jeżeli data_rejestracji_badania jest mniejsza niż '01-01-2019' wtedy wyświetl komunikat "Badanie sprzed okresu refundacji".

Wprowadzam ją do systemu, i wygląda to tak:

Reguły wejściowe		
RS-012	Jeżeli data_rejestracji_badania jest mniejsza niż '01-01-2019' wtedy wyświetl komunikat "Badanie sprzed okresu refundacji".	  

Teraz spójrzmy czego udało się o niej dowiedzieć:

Reguła wejściowa

RS-012

Jeżeli data_rejestracji_badania jest mniejsza niż '01-01-2019' wtedy wyświetl komunikat "Badanie sprzed okresu refundacji".

Wyodrębnione komunikaty

KOMUNIKAT1

Badanie sprzed okresu refundacji

Rozpoznane tokeny

Token	Wartość logiczna	Słownie	Prawd.	Kategoria
Jeżeli	SK_SW	Początek warunku	0.191	
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.184	
jest mniejsza niż	OPR_REL	Operator relacyjny	0.182	<
'01-01-2019'	OP_P	Prawostronny operand porównania	0.193	
wtedy	SK_KW	Koniec warunku	0.184	
wyświetl komunikat	AKCJA	Akcja	0.164	WYSWIETL_KOMUNIKAT
KOMUNIKAT1	AKCJA_PARAMETR	Parametr akcji	0.170	

Rozpoznane parametry wejściowe

Parametr	Typ	Domyślnie
data_rejestracji_badania	<div><div>Data</div><div></div></div>	
const_param2	<div><div>Data</div><div></div></div>	'01-01-2019'

+

-

- **Reguła wejściowa** - system jeszcze raz przypomina treść wprowadzonej reguły i prezentuje nadany jej kod, w naszym przypadku RS-012.
- **Wyodrębnione komunikaty** - tu zaprezentowane zostały komunikaty, które udało się wyszukać w regule. Każdy odnaleziony komunikat zostaje wycięty z reguły przed poddaniem jej analizie i zastąpiony swoim reprezentującym go identyfikatorem - w naszym przypadku odnaleziony został jeden komunikat i zastąpiony przez słowo „KOMUNIKAT1”.
- **Rozpoznane tokeny** - najistotniejsza część wyników analizy. Widzimy, że z tą regułą algorytm poradził sobie bezbłędnie. Tokeny zostały prawidłowo skojarzone z ich znaczeniem logicznym. Na chwilę uwagi zasługuje tylko kolumna „Kategoria” . Dotyczy ona tych tokenów, które mogą być określone na wiele sposobów. Musimy przyporządkować im jedną, uniwersalną kategorię. Jako przykład niech posłuży operator mniejszości - może on zostać opisany jako („jest mniejszy niż”, „jest

mniejszy od” „,jest mniejsza od”,...). Z punktu widzenia aplikacji, wszystkie te wartości reprezentują jedną kategorię, ja nazwałem ją po prostu „<” .

- Rozpoznane parametry wejściowe - w tym miejscu pokazywane są wartości, które uznane zostały za parametry wejściowe reguły. Trafiają tu tokeny, które biorą udział w porównaniach, lub są przekazywane jako parametry wejściowe do innych reguł (tę sytuację pokażę trochę później). W tym przypadku widzimy, że aplikacji udało się prawidłowo dopasować typy danych i określić wartość domyślną jednego z parametrów. Gdyby to się nie udało, czynność tę musi wykonać człowiek. Musi się to stać przed wygenerowaniem kodu.

I na koniec spójrzmy na wygenerowany kod metody walidującej regułę RS-012.

```
fun rs_012(  
    data_rejestracji_badiania: LocalDate,  
    const_param2: LocalDate = LocalDate.parse("01-01-2019" ),  
    KOMUNIKAT1: String = "Badanie sprzed okresu refundacji"  
): wynikDzialaniaReguly {  
    if(data_rejestracji_badiania <const_param2) {  
        return Komunikat(KOMUNIKAT1 )  
    }  
}
```




Wszystko wygląda ok, więc spróbujmy trochę skomplikować regułę.

7.2 Reguła z podwójnym warunkiem

Założmy że refundacji podlegają tylko te badania, które zostały wykonane w pierwszym kwartale 2019 roku. Chcielibyśmy zmienić treść reguły w taki sposób, by po stwierdzeniu tego typu sytuacji informowała użytkownika że jego badanie może zostać zrefundowane.

Jeżeli data_rejestracji_badiania jest mniejsza niż '01-01-2019' lub data_rejestracji_badiania nie jest większa niż '01-04-2019' wtedy wyświetl komunikat "Badanie sprzed okresu refundacji".

Modyfikuję regułę:

Reguły wejściowe		
RS-012	Jeżeli data_rejestracji_badiania jest większa niż '01-01-2019' i data_rejestracji_badiania jest mniejsza od '01-04-2019' wtedy wyświetl komunikat "Badanie podlega refundacji".	  

I oglądamy wyniki:

Wyodrębnione komunikaty				
KOMUNIKAT1	Badanie podlega refundacji			

Rozpoznane tokeny				
Token	Wartość logiczna	Słownie	Prawd.	Kategoria
Jeżeli	SK_SW	Początek warunku	0.191	
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.184	
jest większa niż	OPR_REL	Operator relacyjny	0.181	>
'01-01-2019'	OP_P	Prawostronny operand porównania	0.192	
i	OPR_LOG	Operator logiczny	0.185	&&
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.190	
jest mniejsza od	OPR_REL	Operator relacyjny	0.186	<
'01-04-2019'	OP_P	Prawostronny operand porównania	0.187	
wtedy	SK_KW	Koniec warunku	0.188	
wyświetl komunikat	AKCJA	Akcja	0.164	WYŚWIETL_KOMUNIKAT
KOMUNIKAT1	AKCJA_PARAMETR	Parametr akcji	0.170	

Rozpoznane parametry wejściowe		
Parametr	Typ	Domyślnie
data_rejestracji_badania	<input type="text" value="Data"/>	
const_param2	<input type="text" value="Data"/>	'01-01-2019'
const_param3	<input type="text" value="Data"/>	'01-04-2019'

+
-

Istotne zmiany doszły w tabeli rozpoznanych tokenów. Doszedł operator logiczny, oraz druga część warunku. Udało się również prawidłowo przyporządkować operatory do określających je kategorii. Dodatkowo pojawił się nowy parametr wejściowy, który definiuje drugą z porównywanych wartości.

Wygenerowany kod również wygląda na poprawny:

```

fun rs_012(
    data_rejestracji_badania: LocalDate,
    const_param2: LocalDate = LocalDate.parse("01-01-2019" ),
    const_param3: LocalDate = LocalDate.parse("01-04-2019" ),
    KOMUNIKAT1: String = "Badanie podlega refundacji"
): WynikDzialaniaReguly {
    if(data_rejestracji_badania >const_param2 && data_rejestracji_badania <const_param3) {
        return Komunikat(KOMUNIKAT1 )
    }
}




```

7.3 Reguła z akcją „w przeciwnym wypadku”

Założmy, że chcielibyśmy by w przypadku spełnienia warunku reguła wyświetlała stosowny komunikat informacyjny, a w przypadku gdy warunek nie zostanie spełniony zwracała błąd walidacji . Niech będzie ona miała następującą postać:

Jeżeli data_rejestracji_badania jest większa niż '01-01-2019' i data_rejestracji_badania jest mniejsza od '01-04-2019' wtedy wyświetl komunikat "Badanie podlega refundacji", w przeciwnym wypadku zgłoś błąd "Badanie nie może zostać zarejestrowane".

Tak jak poprzednio, dokonuję zmian w treści reguły:

Reguły wejściowe		
RS-012	Jeżeli data_rejestracji_badania jest większa niż '01-01-2019' i data_rejestracji_badania jest mniejsza od '01-04-2019' wtedy wyświetl komunikat "Badanie podlega refundacji", w przeciwnym wypadku zgłoś błąd "Badanie nie może zostać zarejestrowane" .	  

W tabeli rozpoznanych tokenów znowu pojawiły się nowe rekordy. Tym razem doszła sekcja związana z akcją „NIE”. Wygląda na to, że algorytm poradził sobie również z tą regułą i prawidłowo określił znaczenie poszczególnych słów.

Reguła wejściowa

RS-012

Jeżeli data_rejestracji_badiania jest większa niż '01-01-2019' i data_rejestracji_badiania jest mniejsza od '01-04-2019' wtedy wyświetl komunikat "Badanie podlega refundacji", w przeciwnym wypadku zgłoś błąd "Badanie nie może zostać zarejestrowane" .

Wyodrębnione komunikaty

KOMUNIKAT1

Badanie podlega refundacji

KOMUNIKAT2

Badanie nie może zostać zarejestrowane

Rozpoznane tokeny

Token	Wartość logiczna	Słownie	Prawd.	Kategoria
Jeżeli	SK_SW	Początek warunku	0.191	
data_rejestracji_badiania	OP_L	Lewostronny operand porównania	0.184	
jest większa niż	OPR_REL	Operator relacyjny	0.181	>
'01-01-2019'	OP_P	Prawostronny operand porównania	0.192	
i	OPR_LOG	Operator logiczny	0.185	& &
data_rejestracji_badiania	OP_L	Lewostronny operand porównania	0.190	
jest mniejsza od	OPR_REL	Operator relacyjny	0.186	<
'01-04-2019'	OP_P	Prawostronny operand porównania	0.187	
wtedy	SK_KW	Koniec warunku	0.188	
wyświetl komunikat	AKCJA	Akcja	0.161	WYSWIETL_KOMUNIKAT
KOMUNIKAT1	AKCJA_PARAMETR	Parametr akcji	0.116	
w przeciwnym wypadku	SK_SAN	Początek akcji NIE	0.178	
zgłoś błąd	AKCJA	Akcja	0.194	ZGLOS_BŁAD
KOMUNIKAT2	AKCJA_PARAMETR	Parametr akcji	0.178	

Zmiany zaszły również w wygenerowanym kodzie. Nasza instrukcja warunkowa otrzymała część „ELSE”, a metoda zwraca nowy typ obiektu.

```







fun rs_012(
    data_rejestracji_badania: LocalDate,
    const_param2: LocalDate = LocalDate.parse("01-01-2019" ),
    const_param3: LocalDate = LocalDate.parse("01-04-2019" ),
    KOMUNIKAT1: String = "Badanie podlega refundacji",
    KOMUNIKAT2: String = "Badanie nie może zostać zarejestrowane"
): WynikDzialaniaReguly {
    if(data_rejestracji_badania >const_param2 && data_rejestracji_badania <const_param3) {
        return Komunikat(KOMUNIKAT1 )
    }
    else {
        return BładWalidacji(KOMUNIKAT2 )
    }
}

```

7.4 Reguła z wywołaniem innej reguły

W tym scenariuszu założymy, że sprawdzanie warunków dat refundacji badania ma odbyć się tylko wtedy, gdy spełniony zostanie warunek odpowiedniego wieku pacjenta. Żeby go zrealizować definiuję nową regułę o następującej treści:

dy wiek_pacjenta jest większy od 18 wtedy zgłoś wyjątek "Przekroczony wiek refundacji", w przeciwnym wypadku sprawdzaj regułę RS-012 .

Reguły wejściowe		
RS-012	Jeżeli data_rejestracji_badania jest większa niż '01-01-2019' i data_rejestracji_badania jest mniejsza od '01-04-2019' wtedy wyświetl komunikat 'Badanie podlega refundacji', w przeciwnym wypadku zgłoś błąd 'Badanie nie może zostać zarejestrowane' .	  
RS-013	Gdy wiek_pacjenta jest większy od 18 wtedy zgłoś wyjątek 'Przekroczony wiek refundacji', w przeciwnym wypadku sprawdzaj regułę RS-012 .	  

Po przejściu do sekcji analizy wyników od razu można zauważyć że pojawiły się nam błędy walidacji i niemożliwe jest poprawne wygenerowanie kodu.

DANE WEJŚCIOWE

WYNIKI ANALIZY

GENEROWANIE KODU

Info

MODYFIKUJ

ZAPISZ

RS-012 ✓

RS-013 ✗

- Pole `wiek_pacjenta` wymaga określenia typu
- Brak mapowania parametru `WY data_rejestracji_badania`

Reguła wejściowa

RS-013
 Gdy `wiek_pacjenta` jest większy od 18 wtedy zgłoś wyjątek "Przekroczony wiek refundacji", w przeciwnym wypadku sprawdź regułę RS-012 .

Wydreżnione komunikaty

KOMUNIKAT1	Przekroczony wiek refundacji
------------	------------------------------

W sekcji tokenów wszystko wygląda w porządku. Warunki, słowa kluczowe oraz akcje rozpoznane zostały poprawnie.

Rozpoznane tokeny				
Token	Wartość logiczna	Słownie	Prawd.	Kategoria
Gdy	SK_SW	Początek warunku	0.192	
wiek_pacjenta	OP_L	Lewostronny operand porównania	0.190	
jest większy od	OPR_REL	Operator relacyjny	0.184	>
18	OP_P	Prawostronny operand porównania	0.193	
wtedy	SK_KW	Koniec warunku	0.188	
zgłoś wyjątek	AKCJA	Akcja	0.177	ZGLOS_BLAD
KOMUNIKAT1	AKCJA_PARAMETR	Parametr akcji	0.127	
w przeciwnym wypadku	SK_SAN	Początek akcji NIE	0.169	
sprawdź regułę	AKCJA	Akcja	0.160	SPRAWDZ_REGULE
RS-012	AKCJA_PARAMETR	Parametr akcji	0.170	

Pierwszy problem napotykamy w sekcji „Parametrów wejściowych”. Aplikacja prawidłowo rozpoznała zmienną „`wiek_pacjenta`” jako parametr, ale z treści reguły nie da się wywnioskować jakiego on jest typu. Wymagana jest pomoc użytkownika. Z listy rozwijanej należy wybrać opcję „Liczba”. To załatwia temat pierwszego z błędów.

Rozpoznane parametry wejściowe		
Parametr	Typ	Domyślnie
wiek_pacjenta	<input type="text"/>	
const_param2	Liczba <input type="text"/>	18

Kolejny problem dotyczy samego wywołania reguły „RS-012”. Reguła ta na wejściu oczekuje podania daty. W tabeli wywołania użytkownik musi zmapować który parametr wejściowy reguły „RS-013” ma zostać przekazany na wejście reguły „RS-012”. I tu pojawia się nowy problem, ponieważ reguła „RS-013” tej daty nie przyjmuje.

Odwołanie do reguły RS-012	
RS-012.data_rejestracji_badania	<input type="text"/>

Konieczne jest cofnięcie się do tabeli parametrów i użycie opcji dodania nowego parametru - niech nazywa się on „data_badania” i będzie miał typ „Data”. Po tym kroku możliwe staje się wykonanie odpowiedniego zmapowania i po zapisaniu błędy walidacji znikają.

Rozpoznane parametry wejściowe		
Parametr	Typ	Domyślnie
wiek_pacjenta	Liczba <input type="text"/>	
const_param2	Liczba <input type="text"/>	18
data_badania	Data <input type="text"/>	

Odwołanie do reguły RS-012	
RS-012.data_rejestracji_badania	data badania <input type="text"/>

Poniżej kod, który teraz już bez przeszkód udało się wygenerować:

```

fun rs_012(
    data_rejestracji_badiania: LocalDate,
    const_param2: LocalDate = LocalDate.parse("01-01-2019" ),
    const_param3: LocalDate = LocalDate.parse("01-04-2019" ),
    KOMUNIKAT1: String = "Badanie podlega refundacji",
    KOMUNIKAT2: String = "Badanie nie moÅ¼e zostaÅ zarejestrowane"
): WynikDzialaniaReguly {
    if(data_rejestracji_badiania >const_param2 && data_rejestracji_badiania <const_param3) {
        return Komunikat(KOMUNIKAT1 )
    }
    else {
        return BladWalidacji(KOMUNIKAT2 )
    }
}

/**
 * Gdy wiek_pacjenta jest wiÅszy od 18 wtedy zgÅoÅ wyÅtek "Przekroczony wiek refundacji", w
 * przeciwnym wypadku sprawdzaj reguÅ RS-012 .
 */
fun rs_013(
    wiek_pacjenta: Int,
    const_param2: Int = 18 ,
    data_badiania: LocalDate,
    KOMUNIKAT1: String = "Przekroczony wiek refundacji"
): WynikDzialaniaReguly {
    if(wiek_pacjenta >const_param2) {
        return BladWalidacji(KOMUNIKAT1 )
    }
    else {
        return rs_012(data_rejestracji_badiania =data_badiania)
    }
}

```

8 Próbki zaburzone

Teraz chciaÅbym jeszcze trochÅ poeksperymentowaÅ z treÅciÅ reguÅ poddawanych analizie i przyjrzeÅ im siÅ pod kÅtem odpornoÅci na rÅÅnego typu zaburzenia takie jak np. literÅwki czy niepoprawna struktura reguÅ.

8.1 LiterÅwki i nieznane sÅwa

Jako oznaczenie akcji, ktÅre majÅ zostaÅ wykonane wprowadzam okreÅlenia, ktÅre nigdy nie pojawiÅy siÅ w danych uczÅcych.

*JeÅeli data_rejestracji_badiania jest wiÅsza niÅ '01-01-2019' i data_rejestracji_badiania jest mniejsza od '01-04-2019' wtedy **zaprezentuj komunikat** "Badanie podlega refundacji", w przeciwnym wypadku **wyrzuÅ bÅd** "Badanie nie moÅe zostaÅ zarejestrowane".*

Rozpoznane tokeny				
Token	Wartość logiczna	Słownie	Prawd.	Kategoria
Jeżeli	SK_SW	Początek warunku	0.191	
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.184	
jest większa niż	OPR_REL	Operator relacyjny	0.181	>
'01-01-2019'	OP_P	Prawostronny operand porównania	0.192	
i	OPR_LOG	Operator logiczny	0.185	&&
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.190	
jest mniejsza od	OPR_REL	Operator relacyjny	0.186	<
'01-04-2019'	OP_P	Prawostronny operand porównania	0.187	
wtedy	SK_KW	Koniec warunku	0.184	
zaprezentuj komunikat	AKCIA	Akcja	0.161	WYSWIETL_KOMUNIKAT
KOMUNIKAT1	AKCIA_PARAMETR	Parametr akcji	0.116	
w przeciwnym wypadku	SK_SAN	Początek akcji NIE	0.177	
wyrzuć błąd	AKCIA	Akcja	0.186	ZGLOS_BLAD
KOMUNIKAT2	AKCIA_PARAMETR	Parametr akcji	0.174	

Jak widać algorytm znowu sobie poradził, niestety nie jest to jednak regułą. Tolerancja na błędy pojawia się tylko w niewielkim zakresie. Może jest to związane z niskimi prawdopodobieństwami, z jakimi algorytm rozpoznaje moje encje.

Dla przykładu wprowadźmy jeszcze jedną modyfikację, tym razem wyrażenie „jest mniejsza od” zamienię na „jest mniejsz od”.

*Jeżeli data_rejestracji_badania jest większa niż '01-01-2019' i data_rejestracji_badania **jest mniejsz od** '01-04-2019' wtedy **zaprezentuj komunikat** "Badanie podlega refundacji", w przeciwnym wypadku **wyrzuć błąd** "Badanie nie może zostać zarejestrowane".*

I ta niewielka zmiana wystarczy by algorytm zadziałał źle. Bo chociaż z jednej strony uznał, że fraza ta jest operatorem relacyjnym, to dokonał złej jego kategoryzacji, bo zamiast przypisać go do kategorii „<”, trafił on do kategorii „>”, niestety z punktu widzenia jakości wygenerowanego kodu, pomyłka ma zasadnicze znaczenie.

Rozpoznane tokeny				
Token	Wartość logiczna	Słownie	Prawd.	Kategoria
Jeżeli	SK_SW	Początek warunku	0.191	
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.184	
jest większa niż	OPR_REL	Operator relacyjny	0.181	>
'01-01-2019'	OP_P	Prawostronny operand porównania	0.192	
i	OPR_LOG	Operator logiczny	0.185	&&
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.190	
jest mniejsz od	OPR_REL	Operator relacyjny	0.186	>
'01-04-2019'	OP_P	Prawostronny operand porównania	0.187	
wtedy	SK_KW	Koniec warunku	0.184	
zaprezentuj komunikat	AKCJA	Akcja	0.161	WYSWIETL_KOMUNIKAT
KOMUNIKAT1	AKCJA_PARAMETR	Parametr akcji	0.116	
w przeciwnym wypadku	SK_SAN	Początek akcji NIE	0.177	
wyrzuc błąd	AKCJA	Akcja	0.186	ZGLOS_BLAD
KOMUNIKAT2	AKCJA_PARAMETR	Parametr akcji	0.174	

8.2 Zaburzenia struktury i niezgodność z abstrakcyjnym modelem reguły

Spróbujmy teraz przyjrzeć się reakcji algorytmu na dużo bardziej złożone zaburzenie - niezgodność z przyjętym schematem reguły. Założmy, że z reguły znika słowo „WTEDY”.

Jeżeli data_rejestracji_badania jest większa niż '01-01-2019' i data_rejestracji_badania jest mniejsza od '01-04-2019' wyświetl komunikat "Badanie podlega refundacji", w przeciwnym wypadku zgłoś błąd "Badanie nie może zostać zarejestrowane".

Rozpoznane tokeny				
Token	Wartość logiczna	Słownie	Prawd.	Kategoria
Jeżeli	SK_SW	Początek warunku	0.191	
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.184	
jest większa niż	OPR_REL	Operator relacyjny	0.181	>
'01-01-2019'	OP_P	Prawostronny operand porównania	0.192	
i	OPR_LOG	Operator logiczny	0.185	& &
data_rejestracji_badania	OP_L	Lewostronny operand porównania	0.190	
jest mniejsza od	OPR_REL	Operator relacyjny	0.185	<
'01-04-2019'	OP_P	Prawostronny operand porównania	0.140	
wyświetl	OPR_LOG	Operator logiczny	0.138	WYSWIETL_KOMUNIKAT
komunikat	OP_L	Lewostronny operand porównania	0.188	
KOMUNIKAT1	OPR_REL	Operator relacyjny	0.108	!=
w przeciwnym wypadku	SK_SAN	Początek akcji NIE	0.171	
zgłoś błąd	AKCJA	Akcja	0.194	ZGLOS_BLAD
KOMUNIKAT2	AKCJA_PARAMETR	Parametr akcji	0.178	

Jak można łatwo zauważyć, w tym przypadku algorytm sobie nie poradził. W poszukiwaniu znanego mu wzorca dokonał złych przyporządkowań. Ciekawe natomiast jest to, że mimo pomyłki sekcji rozpoznawania akcji TAK, części reguły leżące poza nią, w sekcji akcji NIE zostały rozpoznane prawidłowo.

8.3 Podsumowanie

Niestety w zakresie uodpornienia algorytmu na pomyłki i niezgodności z założoną strukturą pozostało wiele do zrobienia. Stosunkowo niewielkie pomyłki prowadzą do złej interpretacji reguły i wygenerowania wadliwego kodu.

Część III

Ocena wyników

9 Pytania bez odpowiedzi

Ze względu na fakt, że mój miniprojekt ma charakter czysto akademicki, to uzyskane wyniki oceniam jako bardzo ciekawe i zadowalające. Mam jednak świadomość (a może tylko mi się wydaje że mam . . .), że zaprezentowana przeze mnie metoda rozwiązywania problemu transkrypcji języka naturalnego na kod wykonywalny nie jest doskonała, a wiele pojawiających się zagadnień wymaga gruntownego przestudiowania. Poniżej omówię kilka zagadnień problemowych, które na tę chwilę pozostają bez rozwiązania.

9.1 Ocena jakości danych uczących

Pierwsza rzecz, która sprawia mi trudność to ocena jakości dostarczonej przeze mnie próbki uczącej. Jak ocenić, czy dane nie są zbyt schematyczne, a wytrenowana sieć zachowuje swoje zdolności generalizacyjne. Może dobór przykładów, lub lepsza ich konstrukcja wpłynęłaby na poprawę osiągniętych wyników. Nie znam sposobu na zinterpretowanie dosyć lakonicznego raportu z procesu uczenia 5. Zastanowienie budzi również niska wartość prawdopodobieństwa rozpoznawalności poszczególnych encji. Dlaczego bardzo rzadko jest ona większa niż 0.2 ?

9.2 Rozszerzalność próbki danych uczących

Kolejna rzecz, która wymaga poprawy to sposób rozszerzania zakresu rozpoznawalnych przez algorytmy schematów. Co należałoby zrobić by sieć zaczęła rozpoznawać inny typ warunku np. frazy typu „czy pole xxx zostało zdefiniowane”, albo jak sprawić by sieć potrafiła rozpoznać n - grup warunkowych połączonych operatorem logicznym (w tej chwili radzi sobie z dwoma warunkami połączonymi jednym operatorem). Czy dostarczenie kolejnej liczby przykładów obejmujących taki rodzaj warunku na pewno rozwiąże problem ? Jeśli tak to jak ocenić optymalną dla próbki liczbę nowodostarczonych przykładów. Czy liczba przykładów w nowym schemacie nie powinna zachować jakiejś proporcji względem liczby próbek w starym schemacie ? Tu obawiałbym się sytuacji, że zbyt duża liczba próbek w nowym schemacie mogłaby spowodować że stanie się on dominujący, co z kolei doprowadziłoby do pogorszenia dotychczasowych osiągnięć. Pozatym takie dołączanie kolejnych przypadków szybko doprowadzi do zbyt dużego rozdrobnienia przykładów i lawinowy rozrost liczności próbki.

9.3 Optymalny dobór parametrów uczenia

Sam proces treningu posiada możliwości parametryzacji. Ja skorzystałem z tych najbardziej podstawowych. Może bardziej świadomy ich dobór doprowadziłby do otrzymania lepszych wyników.

9.4 Odporność na błędy i zaburzenia

Tutaj największym problemem jest utrzymanie zgodności z założonym schematem, oraz ocena czy proces rozpoznania przebiegł poprawnie. Nie znam na to innego sposobu niż ocena ekspercka, oczyma człowieka. Jak ustrzec się przed tym, że osoba wpisująca regułę nie wzbogaci jej treści we frazy, które są całkowicie nierozpoznawalne. Proces rozpoznawania encji i tak się zakończy powodzeniem, ale tokeny mogą zostać rozpoznane nieprawidłowo. Jak zdiagnozować taką sytuację ?

9.5 Optymalny dobór narzędzi i technologii

W tym punkcie należałoby się zastanowić, czy wybór biblioteki *OpenNLP*, *techniki NER*, oraz algorytmu jest optymalny do rozwiązania tego typu zadań. Co prawda w świecie Java możliwości wyboru bibliotek są bardziej ograniczone niż w środowisku *Python*, to jednak należałoby zrobić pod tym kątem rozpoznanie chociażby narzędzi *Stanford NLP*.