2.01.2019 r

OKE 2018 CHALLENGE – ESWC 2018

Dokumentacja z realizacji zadania 2
Zastosowanie technologii informatycznych

Grupa projektowa:
Michał Gozdek 126811
Dominik Kaczmarek 126836
Hubert Kaszuba 126839
Konrad Michalak 126841

1. Wprowadzenie

Niniejsza dokumentacja zawiera informacje o realizacji zadania drugiego w konkursie OKE 2018 Challenge – ESWC 2018 przez członków grupy projektowej. Temat wyzwania koncentruje się wokół idei Sieci semantycznych, a konkretniej automatycznej ekstrakcji wiedzy z tekstu. Samo wyzwanie dzieli się na cztery zadania o rosnącym poziomie trudności, których ostatecznym celem jest dostarczenie systemu, który w efektywny i skuteczny sposób będzie w stanie rozwiązywać problem detekcji tzw. bytów nazwanych (ang. Named Entities), a także relacji między nimi.

Celem zadania drugiego jest identyfikacja bytów nazwanych w zdaniach oraz ujednoznacznienie zidentyfikowanych podmiotów do bazy wiedzy DBpedia. Byty mają być identyfikowane za pomocą indexu początkowego oraz końcowego. Wynik ma być wygenerowany w postaci RDF, który będzie formalizował powiązanie zidentyfikowanych podmiotów z bazą wiedzy DBpedia. W odróżnieniu od zadania pierwszego, gdzie do zidentyfikowania były jedynie trzy klasy ontologii, w tym zadaniu klas tych jest dużo więcej i podzielone są one na super klasy i sub klasy.

2. Dane

W tej części opisany został format danych wejściowych i wyjściowych zdefiniowany przez organizatorów konkursu OKE 2018.

Dane podzielone są na tzw. kontekst i frazy. Przez kontekst rozumiany jest tekst podany przez użytkownika, który ma zostać przeanalizowany przez parser, moduł CRF i z którego wyciągnięte mają zostać frazy wraz z odnośnikami do DBpedii. Kontekst (Rys 2.1) zbudowany jest z pól:

a - numer dokumentacji typu z którego zbudowany jest kontekst; typ; informacja, że pole to opisuje właśnie kontekst;
beginIndex - indeks od którego zaczyna się kontekst;
endIndex - indeks zakończenia kontekstu;
isString - zawiera pełną treść kontekstu w postaci łańcuchu znaków typu String;

Budowa frazy (Rys 2.2) również zawiera pole **a** i **beginIndex, endIndex,** które umożliwiają zlokalizowanie jej w kontekście. Dodatkowo zawiera informację:

anchorOf - zawiera pełną frazę w postaci ciągu znaków;
referenceContext - odnośnik do kontekstu;
taIdentRef - link do strony w DBpedii;

Przykładowym kontekstem może być: *“Florence May Harding studied at a school in Sydney, and with Douglas Robert Dundas , but in effect had no formal training in either botany or art.*” Frazami tego kontekstu są: *Florence May Harding, Sydney, Douglas Robert Dundas.*

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix itsrdf: <http://www.w3.org/2005/11/its/rdf#> .
@prefix nif:   <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .

<http://www.ontologydesignpatterns.org/data/oke-challenge-2017/task-2/sentence-10#char=0,332>
        a               nif:RFC5147String , nif:String , nif:Context ;
        nif:beginIndex  "0"^^xsd:nonNegativeInteger ;
        nif:endIndex    "332"^^xsd:nonNegativeInteger ;
        nif:isString    "So-called Islamic State (IS) has released a video online which claims to
        show two captured Turkish soldiers being burned alive. IS said they were killed in revenge
        for Turkish killing of Muslims. Turkey launched a campaign against IS in northern Syria in
        August and is currently fighting IS around the group's stronghold of al-Bab. "^^xsd:string .
```

Rys 2.1. Przykładowy plik wejściowy

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix itsrdf: <http://www.w3.org/2005/11/its/rdf#> .
@prefix nif:   <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .

<http://www.ontologydesignpatterns.org/data/oke-challenge-2017/task-2/sentence-53#char=0,541>
        a               nif:RFC5147String , nif:String , nif:Context ;
        nif:beginIndex  "0"^^xsd:nonNegativeInteger ;
        nif:endIndex    "541"^^xsd:nonNegativeInteger ;
        nif:isString    "JFK is a 1991 American historical legal-conspiracy thriller film directed by Oliver Stone. It
        examines the events leading to the assassination of President John F. Kennedy and alleged cover-up through the eyes
        of former New Orleans district attorney Jim Garrison (Kevin Costner). The film was adapted by Stone and Zachary Sklar
        from the books On the Trail of the Assassins by Jim Garrison and Crossfire: The Plot That Killed Kennedy by Jim
        Marrs. Stone described this account as a \"counter-myth\" to the Warren Commission's \"fictional myth.\"
        "^^xsd:string .

<http://www.ontologydesignpatterns.org/data/oke-challenge-2017/task-2/sentence-53#char=0,3>
        a                       nif:RFC5147String , nif:String , nif:Phrase ;
        nif:anchorOf            "JFK"^^xsd:string ;
        nif:beginIndex          "0"^^xsd:nonNegativeInteger ;
        nif:endIndex            "3"^^xsd:nonNegativeInteger ;
        nif:referenceContext    <http://www.ontologydesignpatterns.org/data/oke-challenge-2017/task-2/sentence-53#char=0,541> ;
        itsrdf:taIdentRef       <http://aksw.org/notInWiki/JFK> .

<http://www.ontologydesignpatterns.org/data/oke-challenge-2017/task-2/sentence-53#char=77,89>
        a                       nif:RFC5147String , nif:String , nif:Phrase ;
        nif:anchorOf            "Oliver Stone"^^xsd:string ;
        nif:beginIndex          "77"^^xsd:nonNegativeInteger ;
        nif:endIndex            "89"^^xsd:nonNegativeInteger ;
        nif:referenceContext    <http://www.ontologydesignpatterns.org/data/oke-challenge-2017/task-2/sentence-53#char=0,541> ;
        itsrdf:taIdentRef       <http://dbpedia.org/resource/Oliver_Stone> .
```

Rys 2.2. Przykładowy plik wyjściowy

3. Realizacja zadania

W tym rozdziale zostało przedstawione rozwiązanie zadania zaproponowane i zaimplementowane przez grupę projektową, a także podział zadań wewnątrz grupy.

3.1. Opis rozwiązania

Rozwiązanie zadanie zostało oparte o dwie części. Po pierwsze, statyczną analizę leksykalną zdania pod kątem słów i części mowy z wykorzystaniem dostępnych narzędzi. Po drugie, interpretacje potencjalnych ciągów słów, które mogą być bytem nazwanym, za

pomocą algorytmu uczenia maszynowego. Ostatecznie wynik konfrontuje się z rzeczywistą bazą bytów, w tym wypadku DBpedią.

Parser

Parserem, lub inaczej analizatorem składniowym, nazywa się program, który przyjmuje na wejściu określone, nieustrukturyzowane dane i dzieli je na mniejsze struktury tak, by były zrozumiałe dla kolejnych mechanizmów przetwarzania. Przykładem takiego parsera, może być XML Parser, który dla podanego pliku XML, przetwarza jego zawartość i generuje ją na wyjście w postaci zwykłego tekstu.

Idąc tą definicją, w projekcie zostały wykorzystane trzy mechanizmy parsujące tekst, dla którego mamy określić znajdujące się w nim byty nazwane (ang. Named Entities). Pierwszym z nich jest usuwanie znaków specjalnych w tekście, z wyjątkiem kropek, które pozostały w celu rozdzielania tekstu wejściowego na pojedyncze zdania, przy pomocy wyrażenia regularnego. Kolejnym mechanizmem, jest użycie Taggera, dostarczanego przez bibliotekę Stanford.NLP, który przetwarza wstępnie przygotowany tekst, dzieląc go na pojedyncze zdania, nadając każdemu wyrazowi część mowy, oraz podając jego indeksy, które określają jego pozycję w tekście. Ostatni mechanizm wyszukuje w tak przygotowanych zbiorach wyrazów nazw własnych, które mogą być potencjalnymi bytami nazwanymi, a następnie buduje z nich JSON'y wykorzystywane w dalszej części aplikacji.

Rys. 3.1 Schemat parsowania tekstu

Uczenie maszynowe

W ogólności, uczenie maszynowe to nauka o algorytmach i modelach usprawniających swoją wiedzę i wyniki wraz ze zdobywanym doświadczeniem, gdzie owe doświadczenie może być interpretowane jako odpowiednio przygotowane dane. Dzięki zwiększającej się mocy obliczeniowej maszyn, zdolnych do przetwarzania ogromnych ilości danych (klasy Big Data), temat ten zdobywa popularność na wielu polach związanych z m.in. nauką, medycyną, marketingiem, czy sprzedażą. W przypadku tego projektu wykorzystany został algorytm uczenia maszynowego do rozwiązania problemu klasyfikacji, czyli przewidywania/ przyporządkowania pewnej klasy, ze skończonego zbioru klas, do bytu poprzez analizę zdefiniowanych dla niego cech.

Aktualnie, takie podejście jest szeroko wykorzystywane, przy projektowaniu systemów wykrywania bytów nazwanych (ang. Named-Entity Recognize, w skrócie NER), w szczególności swoje zastosowanie mają różne architektury rekurencyjnych sieci neuronowych np. LSTM. Cechą charakterystyczną sieci rekurencyjnych jest wysoka efektywność podczas analizy ciągów, w których występowanie dwóch bytów obok siebie warunkują pewne wcześniej zdefiniowane zależności. Jest to szczególnie charakterystyczne dla języka naturalnego, w których słowa układane są według zdefiniowanej w danym

języku gramatyki i kontekstu. Głównym minusem architektur opartych o sieci jest potrzeba posiadania dużych zbiorów, otagowanych danych do ich poprawnego nauczenia.

Innym podejściem, jest wykorzystanie algorytmu Conditional Random Field (w skrócie CRF), który również bazuje na sekwencjach. Model zakłada, że cechy są zależne od siebie i uwzględnia przyszłe obserwacje podczas nauki. Wzór na wyliczenie CRF można zobaczyć poniżej:

$$p(\mathbf{y}|\mathbf{x}) = \underbrace{\frac{1}{Z(\mathbf{x})}}_{\text{Normalization}} \prod_{t=1}^{T} \exp \left\{ \sum_{k=1}^{K} \underbrace{\theta_k}_{\text{Weight}} \underbrace{f_k(y_t, y_{t-1}, \mathbf{x}_t)}_{\text{Feature}} \right\}$$

Wzór CRF dzieli się na dwie główne części, normalizację, a także definicję cech, w których x_t odpowiada pojedynczej wartości x ze zbioru cech X, a y_t klasą przyporządkowaną danej próbce i y_{t-1} klasą poprzedniej próbki.

Po przeglądzie literatury porównującej te dwa podejścia, ostatecznie w projekcie zastosowany został algorytm CRF. Czynnikami, które wpłynęły na taką decyzję były:

- nie duże różnicę pomiędzy wynikami uzyskanymi przez obydwa algorytmy (z przewagą LSTM),
- możliwość lepszej generalizacji przez algorytm CRF, niż LSTM dla małego zbioru danych przekazanego przez organizatorów,
- prostota implementacji algorytmu CRF w języku python.

W rzeczywistych systemach często korzysta się z zalet obydwu algorytmów i łączy je przekazując przetworzone wyjście z LSTM na wejście CRF.

Cechy skonstruowane na potrzeby uczenia zostały przedstawione poniżej:

```
'word.lower'    : słowo w ciągu
'word.isupper'  : czy słowo składało się w całości z dużych liter
'word.istitle'  : czy słowo zaczyna się z dużej litery
'word.isdigit'  : czy słowo jest liczbą
'word.tail'     : trzy ostatnie litery słowa np. go[ing]
'postag'        : część mowy
'position'      : pozycja w zdaniu - BOS (beginning of sentence) lub
                  EOS (end of sentence)
```

Poza cechami głównego słowa, w każdej próbce znajdują się analogiczne pola dla poprzedniego i następnego słowa w danym ciągu. Cechy poprzedniego słowa w ciągu oznaczamy przedrostkiem ‘-1’, natomiast cechy następnego ‘+1’. Ostatecznie pojedyncza próbka wygląda następująco:

```
'word.lower'       : słowo w ciągu
'word.isupper'     : czy słowo składało się w całości z dużych liter
'word.istitle'     : czy słowo zaczyna się z dużej litery
'word.isdigit'     : czy słowo jest liczbą
'word.tail'        : trzy ostatnie litery słowa np. go[ing]
'postag'           : część mowy
'position'         : pozycja w zdaniu - BOS lub EOS

'-1:word.lower'    : poprzednie słowo w ciągu
'-1:word.isupper'  : czy słowo składało się w całości z dużych liter
'-1:word.istitle'  : czy słowo zaczyna się z dużej litery
'-1:word.isdigit'  : czy słowo jest liczbą
'-1:word.tail'     : trzy ostatnie litery słowa np. go[ing]
'-1:postag'        : część mowy
'-1:position'      : pozycja w zdaniu - BOS lub EOS

'+1:word.lower'    : następne słowo w ciągu
'+1:word.isupper'  : czy słowo składało się w całości z dużych liter
'+1:word.istitle'  : czy słowo zaczyna się z dużej litery
'+1:word.isdigit'  : czy słowo jest liczbą
'+1:word.tail'     : trzy ostatnie litery słowa np. go[ing]
'+1:postag'        : część mowy
'+1:position'      : pozycja w zdaniu - BOS lub EOS
```

Każda obserwacja jest również oznakowana odpowiednią klasą, do której należy. W projekcie wyróżnione 17 klas, które pokrywają 12 super klas opisujących byty w bazie DBpedia, a także zawierają 3 dodatkowe podklasy, które często występowały wśród oznakowanych próbek, a którym brakowało powiązań z super klasami. Ostatnie 2 klasy są klasami pomocniczymi. “UNE” (Undefined Named Entity) oznacza próbkę, która jest bytem nazwanym jednak nie można przypisać jej żadnej z wcześniej wybranych klas i “O”, czyli oznaczenie, że dana próbka nie jest bytem nazwanym. Ostatecznie lista klas wygląda następująco:

- 'Activity'
- 'Agent'
- 'EthnicGroup'
- 'Organisation'
- 'Award'
- 'Language',
- 'Person'
- 'Disease'
- 'Event'
- 'MeanOfTransportation'
- 'PersonFunction'
- 'Place'
- 'ArchitecturalStructure'
- 'Species'
- 'Work’
- 'UNE'
- 'O'

Powodem dodania klasy “UNE” jest uwzględnienie przez algorytm, uwzględnia ogólnych powiązań, które mogą wskazywać, że dany ciąg jest potencjalnie bytem nazwanym, pomimo braku możliwości zakwalifikowania go do jednej z klas.

Przypisując klasę do danego słowa w ciągu jest mu również nadawany znacznik w zależności od jego pozycji. Słowo rozpoczynające ciąg posiada przypisaną klasę z

przedrostkiem “I-”, natomiast następujące po nim posiada przedrostek “O-”. Czyli wiedząc, że dany ciąg słów, przykładowo “Leonardo da Vinci”, wskazuje na klasę Person, to kolejne słowa w ciągu dostaną etykiety:

```
(Leonardo, I-Person), (da, O-Person), (Vinci, O-Person)
```

Taka technika ma naturalnie wskazywać algorytmowi sekwencje, po wykryciu przez algorytm słowa, do którego pasuje klasa z przedrostkiem “I-”, algorytm powinien zawęzić zbiór klas, podczas predykcji następnego słowa. Wybór ograniczony zostaje do dopasowania następnej etykiety do tej samej klasy z przedrostkiem “O-”, lub etykiety “O” oznaczającej koniec tego sekwencji dla tego bytu.

DBpedia oraz SPARQL

DBpedia to dosyć nowy projekt, zapoczątkowany w 2007 roku przez ludzi z Wolnego Uniwersytetu Berlina oraz Uniwersytetu w Lipsku. Ma on na celu usystematyzowanie i powiązanie danych zawartych w Wikipedii, a następnie udostępnienie tych danych w internecie. DBpedia wyodrębnia uporządkowane dane z infoboxów w Wikipedii i publikuje je w postaci RDF i kilku innych formatach. Ma to pozwolić na szybsze znajdowanie przez ludzi określonych bytów, a także ich powiązań.

Podstawowy model danych RDF operuje na pojęciach zasobów (ang. *resources*), właściwości (ang. *properties)* przyjmujące wartości (ang. *object*) określonych typów i stwierdzeń-trójek (ang. *statements*).

- Zasoby - dane opisywane (np. strony, dokumenty, kolekcje dokumentów) identyfikowane za pomocą URI.
- Właściwości - atrybuty, charakterystyki lub relacje związane z zasobami; właściwości przyjmują wartości określonych typów.
- Stwierdzenia - trójki: zasób (podmiot opisu), właściwość (atrybut podmiotu) i wartość właściwości (przedmiot); przedmiotami mogą być literały znakowe lub zasoby.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf =
"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:s = "http://description.org/schema/">
 <rdf:Description about = zasób >
    <s:właściwość> wartość </s:właściwość>
 </rdf:Description>
</rdf:RDF>
```

Dzięki takiemu formatowi zapisu danych, do odpytywania DBpedii można wykorzystać język SPARQL, który jest językiem zapytań Sieci Semantycznych. Pozwala on na

wykonanie zapytań do źródeł danych strukturalnych i semistrukturalnych, a także na eksplorację danych przez zadawanie pytań dotyczących nieznanych związków. Właśnie za pomocą tego języka wykonywane są zapytania do DBpedii w naszej aplikacji.

Jsony zwrócone przez CRFModule są wstępnie przetwarzane. Odrzucane są te elementy, które nie zostały zaklasyfikowane do żadnej klasy. Reszta natomiast przekazywana jest do modułu SPARQL, w którym to wykonywane są zapytania do DBpedii. Jako zmienne atrybuty przyjmują one nazwę elementu i klasę w jakiej ma ich szukać. Dokładniej zapytanie brzmi "Czy w klasie X znajduje się element o nazwie Y?". Z uzyskanych odpowiedzi tworzone są frazy, które składają się z kontekstu, linku do elementu w DBpedii, indexu początkowego oraz końcowego. Tak stworzone frazy są dodawane do listy, która następnie jest przekazywana do modułu odpowiedzialnego za zapis do pliku w formacie .ttl.

3.2. Podział zadań

Członkowie grupy projektowej przyjęli następujący podział zadań:

Michał Gozdek

- Implementacja komunikacji między C#-pową aplikacją parsera a CRFModule.
- Przetworzenie JSONów dostarczonych przez CRFModule.
- Implementacja modułu odpytującego DBpedie.

Dominik Kaczmarek

- Wygenerowanie zbioru uczącego na podstawie danych w formacie RDF.
- Implementacja modułu uczenia przy użyciu algorytmu CRF.
- Implementacja modułu serwowania predykcji z wykorzystaniem interfejsu komunikacyjnego opartego o bibliotekę Flask.

Hubert Kaszuba

- Implementacja Taggera przy użyciu biblioteki Standford.NLP, który przypisuje wyrazom części mowy.
- Utworzenie struktury JSON'a używanego do wysyłania nazw własnych (ang. NNP, Proper Noun) wykrytych w tekście, wraz z ich otoczeniem do modułu serwowania predykcji.
- Implementacja parsera, odpowiedzialnego za wykrywanie nazw własnych w tekście, budowania dla każdej JSON'a i zwracania ich w liście.

Konrad Michalak

- Implementacja interfejsu graficznego aplikacji.
- Odczytywanie plików wejściowych o różnych formatach.
- Implementacja biblioteki zapisującej wynik w formacie .ttl.

4. Implementacja

W tym rozdziale opisana została implementacja systemu, która zawiera informacje o wykorzystanych technologiach, a także najważniejszych klasach i metodach.

4.1. Zastosowane technologie

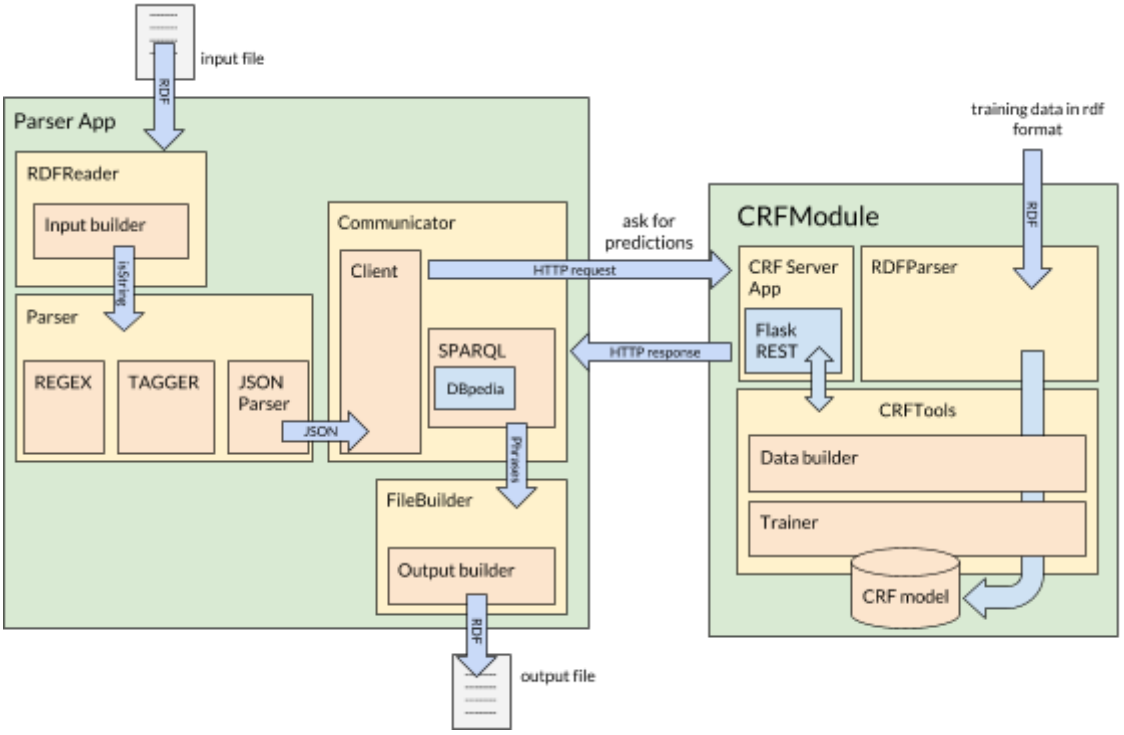
Do implementacji przedstawionego rozwiązania zostały użyte następujące języki i biblioteki:

- Python 3.6 - użyte biblioteki:
  - RDFLib 4.2.2 - biblioteka do obsługi RDF, współpracuje z wieloma formatami serializowania grafów, jak N3, RDF/XML, czy Turtle.
    Strona projektu: https://rdflib.readthedocs.io/en/stable/index.html
  - NLTK 3.3 - Natural Language Toolkit to biblioteka służąca do przetwarzania języka naturalnego, posiada wbudowany dostęp do ponad 50 korpusów językowych i zasobów leksykalnych, posiada narzędzia do takich operacji jak tokenizacja, lematyzacja, czy rozpoznawanie części mowy w zdaniu.
    Strona projektu: https://www.nltk.org/
  - SPARQLWrapper 1.8.2 - biblioteka występuje, jako część projektu RDFlib, pozwala na konstruowanie i komunikację przy pomocy języka w SPARQL w środowisku języka python.
    Strona projektu: https://rdflib.github.io/sparqlwrapper/
  - python-crfsuite 0.9.6 - biblioteka jest wrapperem do implementacji algorytmu Conditional Random Fields w języku C++, przystosowaną do języka python,.
    Strona projektu: https://github.com/scrapinghub/python-crfsuite
  - scikit-learn 0.19.1 - zestaw wielu przydatnych narzędzi do eksplorowania danych, implementacji znanych algorytmów uczenia maszynowego, a także metryk oceny wydajności modelu.
    Strona projektu: https://scikit-learn.org/stable/
  - pandas 0.23.0 - biblioteka pozwalająca na przetwarzanie i analizę danych w efektywny i prosty sposób.
    Strona projektu: https://pandas.pydata.org/
  - Flask 1.0.2 - biblioteka do definiowania podstawowego szkieletu aplikacji typu REST.
    Strona projektu: http://flask.pocoo.org/
- C# - wykorzystane biblioteki:
  - Newtonsoft.Json - biblioteka umożliwiająca konwersję między obiektami typu .NET i JSON.
    Strona projektu: https://www.newtonsoft.com/json

- Stanford NLP Parser - biblioteka, która umożliwia opracowanie struktury gramatycznej zdań, na przykład definiuje, które grupy słów są frazami, przedmiotami, podmiotami.
  Strona projektu: http://sergey-tihon.github.io/Stanford.NLP.NET/samples/Parser.html
- System.Net.Http - interfejs do komunikacji HTTP
  Strona projektu:
  https://docs.microsoft.com/pl-pl/dotnet/api/system.net.http?view=netframework-4.7.2
- dotNetRDF - biblioteka do parsowania, zarządzania, wysyłania zapytań i tworzenia plików RDF.
  Strona projektu: https://www.dotnetrdf.org/

4.2. Architektura

Architektura rozwiązania została podzielona na dwie wyraźne części. Moduł obsługi zapytań działający po stronie klienta i moduł uczenia maszynowego, indywidualny serwis udostępniający interfejs komunikacyjny. Takie rozwiązanie pozwoliło na uporządkowanie zadań wykonywanych, przez aplikacje, a także indywidualne podzielenie zadań między członków grupy. Niezależność modułu uczenia maszynowego pozwala również, na proste podmienienie go w przypadku chęci implementacji bardziej złożonego rozwiązania. Uproszczona architektura, wraz z przepływem sterowania, została przedstawiona poniżej:



Rysunek 4.1. Schemat architektury rozwiązania **(prawy przycisk -> edytuj)**

Przepływu sterowania w aplikacji (główny “pipeline”) prezentuje się następująco:

- wczytanie i przetworzenie wejścia (z pliku, lub okna aplikacji),
- wykrycie ciągów-kandydatów przez analizator składni,
- zapakowanie kandydatów do formatu JSON,
- wysyłanie żądań HTTP z kandydatami do oceny przez CRFModule,
- przetworzenie odpowiedzi w formacie JSON,
- odpytanie DBpedii z użyciem zebranych informacji (m.in. o klasie),
- zwrócenie odpowiedzi użytkownikowi.

Niezależnie od głównego strumienia sterowania w module CRF wykonuje się również, przynajmniej raz przed pierwszym uruchomieniem pełnego programu:

- wczytanie zbioru treningowego w formacie RDF,
- rozbicie zdań w zbiorze na próbki (pojedyncze słowa),
- wygenerowanie cech,
- etykietowanie próbek poprzez odpytywanie DBpedii,
- nauka algorytmu CRF,
- eksport do pliku,
- wczytanie pliku przez serwer CRF.

4.3. Parser App

Aplikacja Parser App napisana została w języku C# z wykorzystaniem interfejsu programowania graficznych aplikacji Windows Forms. Implementacja została podzielona na trzy części:

- Obsługę plików;
- Parsowanie danych wejściowych;
- Komunikację z modułem CRF oraz DBpedią;

Za obsługę plików odpowiedzialne są klasy MainForm.cs, Sentence.cs oraz FileBuilder.
Klasa **MainForm.cs** zawiera zdarzenia (ang. events) aktywowane w momencie danej interakcji użytkownika, odpowiedzialne za komunikację z pozostałymi klasami projektu.
Sentence.cs jest klasą macierzystą klas Context oraz Phrase. Zawiera pola takie jak uri, indeks początkowy i końcowy.

Jako context rozumiany jest cały tekst wejściowy dla parsera. Klasa ta zawiera metody generujące plik wejściowy w formacie przedstawionym na stronie konkursu, w przypadku gdy użytkownik nie wczyta pliku a wprowadzi tekst ręcznie.
Phrase jest obiektem części mowy znalezionej przez parser i zawiera pola informujące o indeksach danej frazy a także odnośniku do strony w DBpedii. Wszystkie pola powyższych klas budowane są w konstruktorach.
FileBuilder.cs jest klasą zawierającą tylko jedną metodę BuildOutput() łączącą wszystkie zebrane informacje o kontekście i frazach i generującą plik wyjściowy w formacie przedstawionym na stronie konkursu. Wywołanie metody wymaga podania dwóch

argumentów: obiektu Context który generuje się automatycznie wewnątrz Eventów zaimplementowanych w klasie MainForm.cs, oraz listy obiektów typu Phrase, stworzonej na podstawie pliku JSON pozyskanego z modułu CRF.

```
public static String BuildOutput(Context context, List<Phrase> phrases)
      {
          StringBuilder builder = new StringBuilder();
          builder.Append(context.oryginalInput);

          if (phrases != null)
          {
              foreach (var phrase in phrases)
              {
                  builder.Append(phrase);
              }
          }
          return builder.ToString();
      }
```

Za część odpowiadającą za parsowanie danych wejściowych odpowiada klasa **ParserController.cs** oraz **Entity.cs**. Ta pierwsza klasa składa się jedynie z konstruktora, w którym inicjalizowany jest obiekt Taggera, oraz z funkcji **List<string> Parse(string text)**, która odpowiedzialna jest kolejno za usunięcie z tekstów znaków specjalnych, za wyjątkiem kropek, przy użyciu wyrażenia regularnego, następnie za przypisanie każdemu wyrazowi części mowy, by ostatecznie dla każdej wykrytej nazwy własnej (NNP), zbudować JSON'a, dodać go do listy i na koniec zwrócić ją w wyniku wykonania. Struktura JSON'a została otrzymana przez utworzenie obiektu klasy **Entity.cs** i przekonwertowaniu go do tejże postaci.

```
struct wordAfter
{
    public string word;
    public string tag;
    public string placeInSentence;
}

struct wordBefore
{
    public string word;
    public string tag;
    public string placeInSentence;
}


class Entity
{
    public string word;
    public string tag;
    public int begin;
    public int end;
```

```
        public string placeInSentence;
        public wordBefore wordBefore;
        public wordAfter wordAfter;
    }
```

Komunikację z modułem CRF oraz DBpedią odpowiadając dwie klasy : Communication.cs oraz SPARQL.cs. Pierwsza z nich jako argument w konstruktorze otrzymuje tablicę JSONów uzyskaną od modułu parsującego dane wejściowe. Klasa składa się z dwóch metod Communicate oraz Internet. Odpowiedzialne są one za wysłanie JSONów do modułu CRF za pomocą HTTP oraz analiza uzyskanej odpowiedzi. Jeżeli któryś z elementów został przypisany do jakiejś klasy to przekazywany jest on do klasy SPARQL.

```
async public void Communicate(Context context)
      {
            List<Phrase> phrases = new List<Phrase>();
            for (int i = 0; i < jsons.Length; i++)
            {
            JSON_entity js = await Internet(jsons[i]);
            if (js.label != "O")
            {

                  SPARQL sparql = new SPARQL(js);
                  phrases.Add(sparql.ask(context));
            }
            }
            SaveResults(context, phrases);
      }
```

Klasa SPARQL odpowiedzialna jest za odpytanie DBpedii, za pomocą języka SPARQL. Dla każdego elementu, który został jej przekazany wysyła ona zapytanie “Czy w klasie X znajduje się element o nazwie Y?”. Po otrzymaniu odpowiedzi buduje ona obiekty klasy Phrase i zwraca je.

```
      public Phrase ask(Context context)
            {
                  Phrase phrase = null;
                  SparqlRemoteEndpoint endpoint = new SparqlRemoteEndpoint(new
Uri("http://dbpedia.org/sparql"), "http://dbpedia.org");
                  SparqlResultSet                  results                  =
endpoint.QueryWithResultSet(@"SELECT ?x WHERE { ?x a dbo:" + js.label + @". ?x
rdfs:label ?name . FILTER regex(str(?name), ""^" + js.name + @"$"")}LIMIT 1");
                  foreach (SparqlResult result in results)
                  {
```

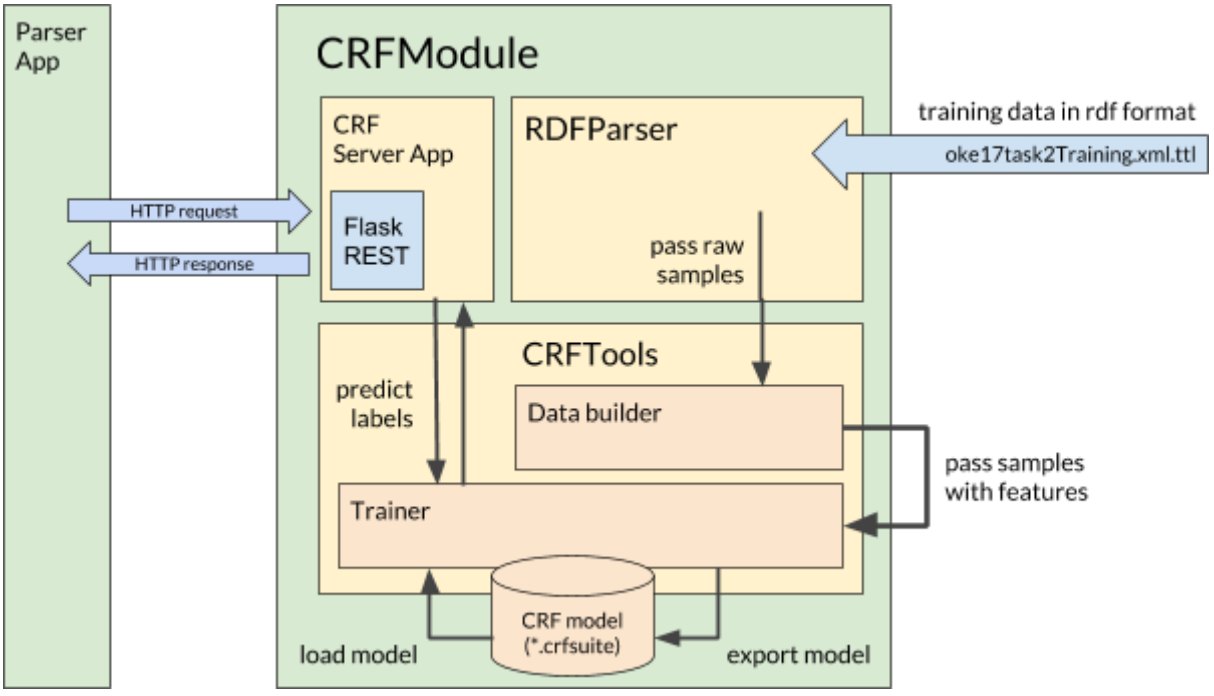
```
                    string res = result.ToString();
                    res = res.Replace("?x = ", string.Empty);
                    phrase  =  new  Phrase(context,  js.name,  js.begin_index,
js.end_index, res);
                    }
                    if (results.LongCount() == 0) {
                    js.name = js.name.Replace(" ", "_");
                    phrase  =  new  Phrase(context,  js.name,  js.begin_index,
js.end_index, "http://aksw.org/notInWiki/" + js.name);
                }
                    return phrase;}
```

4.4. CRFModule

Moduł CRF jest w całości napisany w języku python i wykorzystuje biblioteki przedstawione w podrozdziale 4.1. Technologie. Jest on indywidualną jednostką, która może również działać w oderwaniu od Parser App. Moduł składa się z trzech głównych podmodułów, gdzie każdy z reprezentuję odrębną funkcję. Są to kolejno pakiety:

- RDFParser - przetwarzanie danych wejściowych i tworzenie zbioru treningowego,
- CRFTools - tworzenie cech, nauka algorytmu i eksport modelu,
- CRF Server App - wczytywanie modelu i udostępnianie interfejsu komunikacyjnego.

Wypisane pakiety i ich umiejscowienie w module, wraz z przepływem sterowania zostały przedstawione na rysunku poniżej.



Rysunek 4.2. Schemat działania modułu CRF.

Opis działania

Zaczynając od prawej strony, proces nauki modelu rozpoczyna się od wczytania przez RDFParser pliku w formacie RDF. Na jego podstawie, przy pomocy biblioteki rdflib, konstruowane są obiekty typu Sentence, które zawierają informacje o pełnym zdaniu, a także związanymi z nim bytami nazwanymi. Każdy byt zapisywany jest wraz z URI kierującym do odpowiedniej strony w DBpedii. Na ich podstawie RDFParser może odwołać się bazy i przy pomocy języka SPARQL pobrać informacje o klasie danego bytu. Metoda konstruująca zapytanie została przedstawiona na poniższym listingu.

```
def return_type_sparql_query(name):
    return ("prefix itsrdf: <http://www.w3.org/2005/11/its/rdf#> "
            "select distinct ?type where { "
            "<{0}> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type . "
            "filter (strstarts(str(?type), \"http://dbpedia.org/ontology/\"))}")
            .replace("{0}", name)
```

Tak przygotowane próbki są wysyłane do CRFTools, gdzie blok Data builder wzbogaca je o dodatkowe cechy, przedstawione w rozdziale 3. Posiadając pełny zbiór, podzielony on zostaje na podzbiory treningowy i testowy. Treningowy zostaje podany na wejście algorytmu, natomiast testowy służy do jego oceny. Proces uczenia i ewaluacji odbywa się w bloku Trainer. Model zostaje wyeksportowany do pliku w formacie *.crfsuite i równocześnie od razu jest wczytywany przez serwer.

CRF Serwer udostępnia interfejs komunikacyjny typu REST do komunikacji z klientem. Każde zapytanie klienta zawiera surową próbkę w formacie JSON, która jest rozszerzana o nowe cechy przez blok Data Builder i kierowana do predykcji do wcześniej wyuczonego modelu.

Komunikacja

Do komunikacja z serwerem CRF wykorzystywany jest interfejs komunikacyjny typu REST, jego parametry zostały przedstawione w tabelce poniżej. W aktualnej wersji, serwer był testowany jedynie lokalnie, stąd też przedstawiony adres URL.

Łączenie z serwerem			
URL:	localhost	PORT:	17011
Żądanie przeładowania modelu uczenia maszynowego			
URL:	localhost:17011/crf/reload	TYP:	POST
Żądanie pobrania predykcji dla wysyłanych próbek			
URL:	localhost:17011/crf/predict	TYP:	GET, POST

Formaty danych i metadanych

W zależności od etapu przetwarzania danych wejściowych moduł CRF przyjmuje różne formaty danych. W przypadku pakietu RDFParser format danych wejściowych musi być kompatybilny z popularnymi formatami RDF takimi jak N3, RDF/XML, czy Turtle.

Przekształcone, surowe próbki z wyjścia RDFParsera są zapakowane w obiekty typu Sentence, których definicja została przedstawiona poniżej.

```
class Sentence:

    def __init__(self, sentence):
        self.sentence = sentence       # zdanie główne
        self.associated_entities = []  # powiązane byty
```

Każdy byt znajdujący się na liście `associated_entities` zapisany jest w postaci krotki (nazwa, url, początkowy index pozycji w zdaniu, końcowy index pozycji w zdaniu). Ostatecznie na wyjściu Parsera znajduję się lista słowników w formacie:

```
{
    'value': string,      # słowo
    'uri': string,        # uri do dbpedii
    'begin': int,         # index początkowy w zdaniu
    'end': int,           # index końcowy w zdaniu
    'label': string       # klasa
}
```

Próbki wzbogacone o nowe cechy, czyli takie, które znajdują się ostatecznie w zbiorze treningowym i testowym, przechowywane są w słowniku. Format słownika narzucony jest m.in. przez bibliotekę python-crfsuite. Pola i wartości w nich zawarte pokrywają się z formatem przedstawionym w rozdziale 3.1. Opis rozwiązania - uczenie maszynowe.

Ostatecznym formatem danych wartym przedstawienia i jest format zawartości żądań HTTP kierowanych do serwera. Jest on w formacie pośrednim pomiędzy surową próbką, a próbką wzbogaconą, gotową do uczenia. Jej dokładna forma została przedstawiona przy okazji komunikacji poprzez Parser App.

Ocena modelu

Poniżej zestawione zostały wyniki osiągnięte przez algorytm dla zbioru testowego wynoszącego 20% zbioru pierwotnego.

	precision	recall	f1-score	liczba w zbiorze
I-Activity	0,00	0,00	0,00	1
I-Agent	0,82	0,47	0,60	19
O-Agent	0,71	0,36	0,48	14
I-EthnicGroup	0,00	0,00	0,00	1
I-Event	0,00	0,00	0,00	1

O-Event	0,00	0,00	0,00	3
I-Organisation	0,00	0,00	0,00	1
O-Organisation	0,00	0,00	0,00	3
I-Person	0,57	0,44	0,50	62
O-Person	0,63	0,63	0,63	43
I-Place	0,63	0,52	0,57	33
O-Place	0,78	0,75	0,77	24
I-UNE	0,32	0,18	0,23	34
O-UNE	0,54	0,23	0,32	31
I-Work	0,00	0,00	0,00	3
O-Work	0,00	0,00	0,00	6
średnia	**0,625**	**0,4475**	**0,5125**	279

Wynik wskazują, że algorytm radzi sobie przeciętnie w wykrywaniu klas pojawiających się często w zbiorze treningowym i bardzo słabo dla klas bardziej niszowych jak I-EthnicGroup, czy I-Event. Powodu takiego efektu nauki można doszukiwać się w małej liczbie przykładów treningowych, patrząc chociaż na kolumnę “liczba w zbiorze” możemy zauważyć, że wiele klas jest bardzo rzadkich przez co algorytmowi trudniej wychwycić zależności pomiędzy nimi a dostarczonymi wartościami cech. Pozytywnym zaskoczeniem jest umiejętność rozpoznawania zależności pomiędzy ciągami słów, a ich przełożeniem na klasy. Znalezienie klasy I-Person sprawia, że prawdopodobieństwo przewidzenia O-Person przy kolejnym słowie znacznie wzrasta. Dobrze widać to na zestawieniu najbardziej popularnych przejść w sekwencjach klas, są one wczytane bezpośrednio z modelu CRF.

```
# najczęstsze sekwencje
I-Organisation -> O-Organisation 5.639523
I-Work -> O-Work  5.434968
O-Agent -> O-Agent 5.244446
I-Agent -> O-Agent 5.132974
I-UNE  -> O-UNE   4.959354
O-Organisation -> O-Organisation 4.817624
O-UNE  -> O-UNE   4.689628
I-Place -> O-Place 4.540918
I-Person -> O-Person 4.494990
O-Work -> O-Work  4.375145
O-Place -> O-Place 4.141426
O-ArchitecturalStructure -> O-ArchitecturalStructure 4.101650
O-Person -> O-Person 3.764574
I-ArchitecturalStructure -> O-ArchitecturalStructure 3.699854
O      -> O       3.447957

# najrzadsze sekwencje
Top unlikely transitions:
```

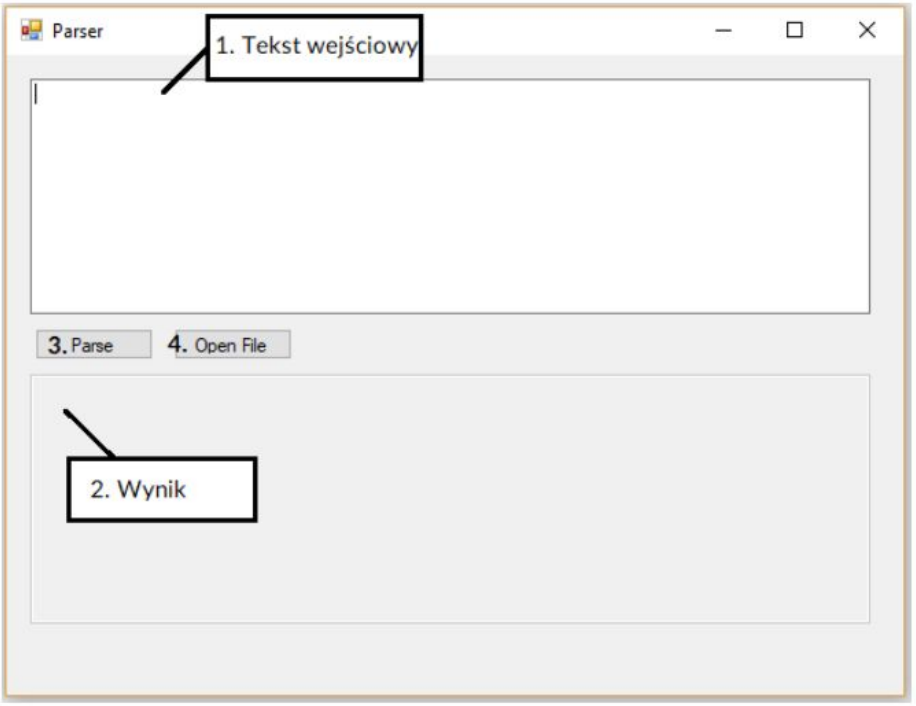
```
I-Disease -> O       -0.137898
O-Organisation -> O       -0.188770
I-Person -> O-UNE   -0.316351
I-EthnicGroup -> O       -0.433683
O      -> O-Disease -0.496678
O      -> O-Event -0.545572
O      -> O-EthnicGroup -0.612713
I-ArchitecturalStructure -> O       -0.752251
O      -> O-ArchitecturalStructure -0.846393
O      -> O-Organisation -0.876707
O      -> O-Work  -1.017217
O      -> O-Place -1.514418
O      -> O-Agent -2.033159
O      -> O-UNE   -2.358936
O      -> O-Person -2.445754
```

5. Uruchomienie

W tym rozdziale przedstawiony został sposób uruchomienia poszczególnych modułów w celu przeprowadzenia testu.

Parser App

Po uruchomieniu aplikacji Parser App, na wejściu należy wprowadzić tekst w języku angielskim z którego wyciągnięte mają zostać odpowiednie części mowy. Tekst może zostać wpisany ręcznie w polu numer 1 (Rys. 5.1), bądź wczytany z pliku po wybraniu przycisku “Open File”. Obsługiwane są cztery formaty wejściowe: .txt, .ttl, .rdf, .xml.
Jeżeli tekst został wprowadzony, można wcisnąć przycisk “Parse” w celu uzyskania wyniku, który wyświetlony zostanie w polu numer 2. Uzyskany wynik zostanie zapisany w formacie .ttl automatycznie po przeparsowaniu. Plik wyjściowy będzie zapisany w katalogu w którym zainstalowana jest aplikacja.



Rysunek 5.1. Okno aplikacji po uruchomieniu.

CRFModule

Moduł został przetestowany na systemie Ubuntu 18.04 z językiem Python w wersji 3.6.6 oraz na Windowsie 8.1 z językiem Python w wersji 3.7.1.

Aby uruchomić moduł CRF, należy zainstalować pakiety wymienione w pliku *requirements.txt*. Można to zrobić za pomocą menedżera pakietów pip dla języka Python.

Aby zainstalować je za pomocą narzędzia pip:

```
#!/usr/bin/env bash
pip3 install -r CRFModule/requirements.txt
```

Moduł składa się z trzech podmodułów: rdfparser, crftools i crfserving_app. Podmoduły należy uruchamiać w podanej kolejności:

1. *rdfparser* i *crftools* wykorzystują skrypt *setup.py* (można go uruchomić tylko raz, aby wygenerować zbiór uczący i wyeksportować wyszkolony model),
2. *crfserving_app* przy użyciu skryptu *crfserving_app.py* (serwer REST, który obsługuje zapytania o predykcje).

Aby uruchomić CRFmodule za pomocą bash:

```
#!/usr/bin/env bash
cd CRFModule
```

```
python3 setup.py
# wait until process ends
python3 crfserving_app.py
```

Aby uruchomić CRFmodule za pomocą linii komend (*cmd*) w systemie Windows:

```
cd CRFModule
windows-run.bat /full/path/to/folder/CRFModule setup.py
# wait until process ends
windows-run.bat /full/path/to/folder/CRFModule crfserving_app
```

6. Źródła

- “Complete tutorial on Text Classification using Conditional Random Fields Model (in Python)”,https://www.analyticsvidhya.com/blog/2018/08/nlp-guide-conditional-random-fields-text-classification/
- Charles Sutton, Andrew McCallum, “An Introduction to Conditional Random Fields”, http://homepages.inf.ed.ac.uk/csutton/publications/crftut-fnt.pdf
- Yves Peirsman, “Named Entity Recognition and the Road to Deep Learning”, http://nlp.town/blog/ner-and-the-road-to-deep-learning/
- Nikola Ljubesic, “Comparing CRF and LSTM performance on the task of morphosyntactic tagging of non-standard varieties of South Slavic languages”, http://aclweb.org/anthology/W18-3917