UNIWERSYTET IM. ADAMA MICKIEWICZA W POZNANIU
WYDZIAŁ MATEMATYKI I INFORMATYKI

Milena Szklarska
numer albumu: 412780

Wykorzystanie cech morfologicznych języka polskiego w statystycznym tłumaczeniu automatycznym

Applying Morphological Features of the Polish Language in Statistical Machine Translation

Praca magisterska na kierunku:
Informatyka
Specjalność:
Systemy inteligentne
Promotor:
prof. UAM dr hab. Krzysztof Jassem

Poznań 2016

Poznań, dnia ....

OŚWIADCZENIE

Ja, niżej podpisany/a Milena Szklarska student/ka Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt: "Wykorzystanie cech morfologicznych języka polskiego w statystycznym tłumaczeniu automatycznym" napisałem/napisałam samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem/am z pomocy innych osób, a w szczególności nie zlecałem/am opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem/am tej rozprawy lub jej części od innych osób. Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[ ]* - wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[ ]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

....
(czytelny podpis studenta)

Streszczenie

W niniejszej pracy magisterskiej opisano wnioski wynikające z eksperymentów przeprowadzonych z wykorzystaniem dwóch narzędzi: środowiska do tworzenia systemów tłumaczenia automatycznego Moses oraz systemu uczenia maszynowego przystosowanego do pracy z dużą ilością cech Vowpal Wabbit. Celem przeprowadzonych eksperymentów było sprawdzenie, czy wykorzystanie klasyfikacji oraz wprowadzenie cech morfologicznych języka pozwoli na ulepszenie jakości tłumaczenia automatycznego. W rozdziałach drugim, trzecim, czwartym i piątym wprowadzono istotne pojęcia teoretyczne związane z tematyką pracy magisterskiej. Rozdziały szósty i siódmy zawierają opis stosowanego rozwiązania oraz uzyskane wyniki. Rozdział ósmy jest rozdziałem ostatnim i podsumowującym.

Słowa kluczowe

statystyczne tłumaczenie automatyczne, uczenie maszynowe, przetwarzanie języka naturalnego, klasyfikacja, systemy inteligentne

Abstract

This master's thesis presents conclusions of experiments carried out with the aid of two tools: the environment for creating statistical machine translation systems named Moses and a machine learning multi-feature toolkit named Vowpal Wabbit. The purpose of experiments was to verify the thesis that using the classifier based on morphological features improves the quality of statistical machine translation. In the second, third, fourth and the fifth chapters the author has introduced essential concepts related to the topic of the master's thesis. The sixth and the seventh chapters contain description of the solution and the received results. The eighth chapter is the last and summarizing chapter.

Keywords

statistical machine translation, machine learning, natural language processing, classification, intelligent systems

Spis treści

1 Wstęp **7**

2 Kluczowe pojęcia związane z tematyką pracy **9**

2.1 Wybrane zagadnienia statystycznego tłumaczenia automatycznego . . . 9

2.1.1 Pojęcie korpusu równoległego . . . 9

2.1.2 Teoria prawdopodobieństwa w statystycznym tłumaczeniu automatycznym . . . 11

2.1.3 Model log-liniowy w statystycznym tłumaczeniu automatycznym . . . 15

2.1.4 Metody ewaluacji statystycznego tłumaczenia automatycznego . . . 22

2.2 Wybrane aspekty uczenia maszynowego . . . 32

2.2.1 Zagadnienie regresji liniowej . . . 32

2.2.2 Zagadnienie klasyfikacji . . . 35

2.2.3 Zagadnienia algorytmiczne oraz model One Against All 39

2.3 Aspekty językoznawcze i morfologiczne . . . 43

3 Architektura systemu tłumaczenia automatycznego Moses **48**

3.1 Funkcje bezstanowe ze szczególnym uwzględnieniem modelu tłumaczenia . . . 49

3.2 Funkcje stanowe ze szczególnym uwzględnieniem modelu języka 53

3.3 Inne cechy wykorzystywane w systemie . . . 56

4 Vowpal Wabbit jako system uczący przystosowany do pracy z dużą ilością cech **61**

4.1 Ogólna charakterystyka działania oraz wykorzystywanych algorytmów . . . 63

4.2 Cechy zależne od klasy . . . 67

5 Morfologia w statystycznym tłumaczeniu automatycznym **70**

5.1 Przykłady w kontekście innych języków słowiańskich . . . 70

5.2 Narzędzia i zasoby dla języka polskiego 75

6 Opis proponowanego rozwiązania 81

6.1 Trenowanie korpusu za pomocą systemu Moses 81

6.2 Przewidywanie klas morfologicznych za pomocą systemu Vowpal Wabbit 89

7 Wyniki przeprowadzonych eksperymentów 105

7.1 Metoda ewaluacji 105

7.2 Dokładność klasyfikacji 107

7.3 Jakość tłumaczenia 109

8 Podsumowanie 110

9 Bibliografia 111

10 Lista rysunków 113

11 Lista tabel 115

12 Lista programów 116

1 Wstęp

Początki statystycznego tłumaczenia automatycznego sięgają już lat 50. XX wieku. Niewątpliwie duży przełom w tej dziedzinie nastąpił w latach 70. XX wieku, kiedy stworzono system METEO, służący do tłumaczenia prognozy pogody w Kanadzie. Zastosowanie tłumaczenia automatycznego zostało jednak spopularyzowane dopiero w latach 90. XX wieku dzięki grupie badawczej Thomas J. Watson Research Center.

Szczególnie od kilku lat można obserwować rozwój statystycznego tłumaczenia automatycznego. W dobie biegłego posługiwania się językami obcymi, trudno wyobrazić sobie proces tłumaczenia bez narzędzi ułatwiających tę często mozolną i trudną pracę. Obecnie niewiele osób korzysta z grubych, papierowych słowników, co można uznać za w pełni uzasadnione. Złoty okres przeżywa Google Translate, będący najchętniej wykorzystywanym tłumaczem, wyposażonym w wiele języków tłumaczenia i dostępnym także jako aplikacja mobilna.

W czasach przechowywania dużej ilości dokumentów cyfrowych, statystyczne tłumaczenie automatyczne wydaje się być bardzo przydatnym rozwiązaniem. Tłumaczenie automatyczne posiada jednak zarówno zwolenników, jak i przeciwników.

Pierwsza z grup twierdzi, że statystyczne tłumaczenie automatyczne za kilka lat stanie się jeszcze potężniejsze niż obecnie, a nawet dorówna tłumaczeniu dokonywanemu przez człowieka. Druga grupa jest z kolei przekonana, że jakość tłumaczenia automatycznego nigdy nie będzie aż tak dobra.

Wyniki statystycznego tłumaczenia automatycznego nie zawsze są doskonałe i należy mieć tego pełną świadomość. Stanowią obecnie solidny fundament do prowadzenia dalszych prac dotyczących ulepszenia jakości tłumaczenia maszynowego, która zależy od wielu czynników.

Celem niniejszej pracy magisterskiej jest sprawdzenie, czy klasyfikacja oraz dodanie cech morfologicznych języka pozwala na ulepszenie jakości statystycznego tłumaczenia automatycznego. W eksperymentach wykorzystano dwa narzędzia - środowisko do tworzenia systemów tłumaczenia automatycznego Moses oraz program uczenia maszynowego, przystosowany do pracy z

dużą ilością cech Vowpal Wabbit.

Wytrenowany w programie Moses system tłumaczenia pozwolił na otrzymanie wyników, będących podstawą do wprowadzania późniejszych ulepszeń. Vowpal Wabbit, wykorzystany jako narzędzie klasyfikujące, umożliwił dodawanie cech morfologicznych, takich jak część mowy lub lemat słowa oraz weryfikację, czy wprowadzone cechy przyczyniły się do ulepszenia jakości statystycznego tłumaczenia automatycznego.

W rozdziałach [2], [3], [4] oraz [5] zostaną przedstawione zagadnienia teoretyczne związane z tematyką pracy magisterskiej. W rozdziałach [6] oraz [7] zostaną opisane przeprowadzone eksperymenty. Rozdział [8] jest rozdziałem podsumowującym.

2 Kluczowe pojęcia związane z tematyką pracy

W niniejszym rozdziale zostaną przedstawione podstawowe terminy związane z teoriami statystycznego tłumaczenia automatycznego oraz uczenia maszynowego.

2.1 Wybrane zagadnienia statystycznego tłumaczenia automatycznego

2.1.1 Pojęcie korpusu równoległego

Definicja 1 *(Korpus równoległy). Korpus równoległy (ang. parallel corpus) to zbiór zawierający oryginalne teksty wraz z ich odpowiednikami w jednym lub wielu językach, różnych od języka źródłowego.*

Definicja 2 *(Język źródłowy). Język źródłowy to język, z którego dokonuje się tłumaczenia.*

Definicja 3 *(Język docelowy). Język docelowy to język, na który dokonuje się tłumaczenia.*

Najprostszy przykład korpusu równoległego zakłada występowanie oryginalnego tekstu wraz z jego tłumaczeniem w innym języku. Dwujęzyczne korpusy równoległe zasadniczo składają się z dwóch kolumn - pierwsza zawiera teksty w języku źródłowym, a druga - ich tłumaczenia. Za uzyskanie takiej struktury odpowiedzialny jest proces urównoleglania (ang. *alignment*), czyli właściwego zestawiania fragmentów tekstów w celu późniejszego wykorzystania. Przeważnie w każdym wierszu korpusu znajduje się pojedyncze zdanie wraz ze swoim odpowiednikiem w innym języku. Nie jest to jednak ogólnie obowiązująca reguła. Rysunek (1) prezentuje nieskomplikowany dwujęzyczny korpus równoległy, składający się z pięciu zdań w języku polskim wraz z odpowiednikami w języku angielskim. Jedno zdanie w języku docelowym jest odpowiednikiem jednego zdania w języku źródłowym.

Rozmawiałem wczoraj z Anną.	I talked with Anna yesterday.
Co powiedziała?	What did she say?
Niezbyt wiele.	Not too much.
Jest bardzo zajęta?	Is she very busy?
Pisze nową książkę.	Yes, she writes a new book.

Rysunek 1: Przykład prostego dwujęzycznego korpusu równoległego

Zdania oryginalnych tekstów mogą być również tłumaczone na więcej niż jeden język. Mówi się wówczas o korpusach równoległych wielojęzycznych.

Oprócz podziału uwzględniającego ilość wykorzystywanych w korpusie języków, obowiązuje także inna kategoryzacja zależna od kierunku, w którym odbywa się tłumaczenie. Istnieją korpusy równoległe jednokierunkowe, dwukierunkowe oraz wielokierunkowe.

Korpusy równoległe jednokierunkowe zawierają teksty w języku źródłowym A i tłumaczenia w języku docelowym B. Korpusy równoległe dwukierunkowe zawierają teksty w języku źródłowym A i tłumaczenia w języku docelowym B oraz teksty w języku źródłowym B i tłumaczenia w języku docelowym A. Korpusy równoległe wielokierunkowe zawierają teksty w języku źródłowym A oraz tłumaczenia w dwóch lub więcej językach.

Korpusy równoległe mogą także występować w rozmaitych formach i dotyczyć różnorodnej tematyki. Przykład (1) pokazuje, że struktura korpusu równoległego niekoniecznie musi być jednolita. Przykładowo, jednemu zdaniu języka źródłowego mogą odpowiadać dwa zdania języka docelowego.

Przykład 1 Jednym z korpusów równoległych udostępnionych w Internecie jest *OpenSubtitles*. Jego zawartość pochodzi z napisów filmowych. Fragment próbki w języku *xml* przedstawiono na rysunku (2). Zastosowane oznaczenia to *(src)* dla zdania języka źródłowego oraz *(trg)* dla zdania języka docelowego. Numery w cudzysłowach wskazują na obowiązującą kolejność zdań języka źródłowego oraz zdań języka docelowego.

(src)="1"> First , state your name , age , and the present condition of your health , and whether in your condition you could travel to attend in person to Beaver , the court now sitting there .
(trg)="2"> Po pierwsze :
(trg)="3"> Proszę podać swoje imię , wiek i obecny stan zdrowia , i czy w pańskim stanie może pan podróżować , aby stawić się osobiście w Beaver do tamtejszego sądu .

(src)="2"> I am Brigham Young .
(trg)="4"> Nazywam się Brigham Young .

Rysunek 2: Fragment próbki dwujęzycznego korpusu równoległego OpenSubtitles

2.1.2 Teoria prawdopodobieństwa w statystycznym tłumaczeniu automatycznym

Fundamentem statystycznego tłumaczenia automatycznego są obliczenia statystyczne. Wyznaczanie prawdopodobieństw tłumaczeń to istotny etap, który pozwala na wybór tłumaczenia uznawanego za właściwe i poprawne w określonym kontekście.

Systemy statystycznego tłumaczenia automatycznego opierają swoje działanie na analizie korpusów równoległych. Z tych korpusów czerpią najistotniejsze informacje, chociażby te dotyczące częstości występowania poszczególnych wyrazów.

Terminy przytoczone w definicjach (2) i (3) nie muszą pokrywać się z autentycznym cyklem powstawania dokumentów. Stosuje się je pod kątem wykorzystania korpusu. Przykładowo, chociaż korpus Europarl powstał poprzez tłumaczenie angielskich tekstów na inne języki, to w procesie tłumaczenia automatycznego można wytrenować translator odwrotny, w którym językiem źródłowym jest język polski.

We wspomnianym procesie tłumaczenia automatycznego istotne jest pojęcie prawdopodobieństwa tłumaczenia (ang. *translation probability*). Prawdopodobieństwo tłumaczenia słowa języka źródłowego na określone słowo języka docelowego można interpretować jako iloraz ilości występujących w korpusie tłumaczeń słowa języka źródłowego na określone słowo języka docelowego przez ilość wystąpień słowa w korpusie.

Przykład 2 Jeśli w korpusie równoległym zbudowanym z tekstów w językach angielskim i polskim, słowo *cat* zostało przetłumaczone na *kot* 4400 razy

przy 5000 wystąpień, to wówczas prawdopodobieństwo tłumaczenia angielskiego słowa *cat* na polskie słowo *kot* wynosi

$$p = \frac{4400}{5000} = 0.88 \tag{1}$$

Na podstawie wyrażonego prawdopodobieństwa możliwe jest dokonanie oszacowania rozkładu prawdopodobieństwa tłumaczenia leksykalnego (ang. *lexical translation probability distribution*). Formalizując, funkcję, która pozwoli na otrzymanie wspomnianego powyżej prawdopodobieństwa wyboru angielskiego tłumaczenia e, zapisano w formule (2).

$$p_s : e \to p_s(e) \tag{2}$$

Dla każdego tłumaczenia docelowego e słowa języka źródłowego s zwracana jest liczba określająca, jak prawdopodobne jest dane tłumaczenie. Najwyższa wartość prawdopodobieństwa zwracana przez funkcję dla słowa e oznacza, że słowo e jest potencjalnie najlepszym kandydatem na tłumaczenie, jeśli najczęściej pojawia się w korpusie. Niska wartość prawdopodobieństwa oznacza, że tłumaczenie rzadko pojawia się w korpusie, a zatem słowo e nie jest dobrym kandydatem na tłumaczenie. Należy podkreślić, że zwracane przez funkcję wartości powinny spełniać dwie istotne własności, określone w formułach (3) oraz (4).

$$\sum_e p_s(e) = 1 \tag{3}$$

$$\forall e : 0 \leq p_s(e) \leq 1 \tag{4}$$

W korpusie równoległym niekoniecznie musi występować wyłącznie jedno tłumaczenie słowa. Przykład (2) ograniczono do rozpatrzenia jednego tłumaczenia angielskiego słowa *cat*, którym było polskie słowo *kot*. Można przypuszczać, że oprócz polskiego tłumaczenia *kot*, wystąpiły w korpusie także inne tłumaczenia, takie jak *kotek* czy *kocur*.

Proces urównoleglania na poziomie wyrazów (ang. *word alignment*) jest odpowiedzialny za przyporządkowanie słów języka źródłowego do słów języ-

ka docelowego. W procesie istotną rolę pełni funkcja urównoleglająca (ang. *alignment function*), która działa w odwrotnym kierunku. Funkcja mapuje słowo w języku docelowym na słowo w języku źródłowym. Proces urównoleglania wraz z rozkładami prawdopodobieństw dla tłumaczeń leksykalnych pozwala na znalezienie ekwiwalentów słów w języku docelowym. W korpusach równoległych, dla pojedynczych słów języka źródłowego, możliwe jest znalezienie wielu takich ekwiwalentów.

Przykład 3 Jeśli słowo *cat* zostało przetłumaczone na słowo *kot* 4400 razy, to na potrzeby przykładu można założyć, że przetłumaczono je również na słowo *kotek* 450 razy oraz na słowo *kocur* 150 razy przy 5000 wystąpień w korpusie. Otrzymane prawdopodobieństwa zostały zawarte w tabeli (1).

$p_s(e)$	e
0.88	kot
0.09	kotek
0.03	kocur

Tabela 1: Rozkład prawdopodobieństw tłumaczeń leksykalnych

Otrzymane wyniki pozwalają stwierdzić, że w analizowanym korpusie najbardziej prawdopodobnym tłumaczeniem angielskiego słowa *cat* jest polskie słowo *kot*. Prawdopodobieństwa spełniają własności określone w formułach (3) i (4).

Prezentowana technika nosi nazwę estymacji maksymalnego prawdopodobieństwa (ang. *maximum likelihood estimation*) i stanowi bazę dla bardziej złożonych i wyrafinowanych metod wykorzystywanych w statystycznym tłumaczeniu automatycznym.

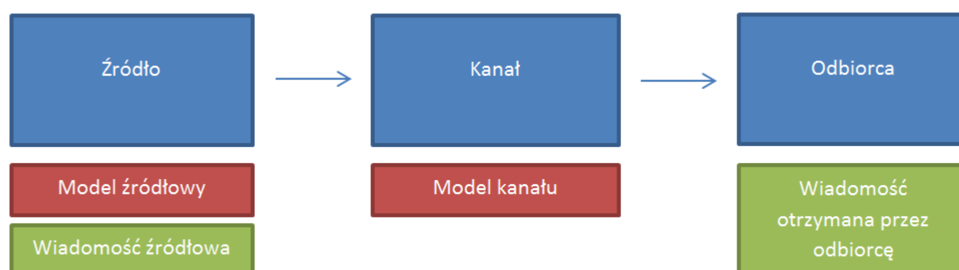
Główne założenia statystycznego tłumaczenia automatycznego zostały zawarte w równaniu określonym w formule (5).

$$\hat{e} = \underset{e}{\operatorname{argmax}} P(F = f|E = e)P(E = e) \tag{5}$$

Definicja 4 *(Podstawowe równanie statystycznego tłumaczenia automatycznego). Podstawowe równanie statystycznego tłumaczenia automatycznego to*

matematyczna formuła, która łączy w sobie model języka (ang. language model) oraz model tłumaczenia (ang. translation model), zapisywane kolejno jako $P(E = e)$ *oraz* $P(F = f|E = e)$.

Połączenie modelu języka z modelem tłumaczenia w taki sposób, w jaki przedstawia to definicja (4), nosi nazwę *modelu zaszumionego kanału* (ang. *noisy-channel model*) i zostało przedstawione na rysunku (3).



Rysunek 3: Model zaszumionego kanału - rysunek poglądowy (Koehn, 2010, [1])

Statystyczne tłumaczenie automatyczne jest traktowane jako szczególny przypadek modelu zaszumionego kanału. Właśnie ten model znalazł swoje zastosowanie również w rozpoznawaniu mowy. Wiadomość wysyłana do odbiorcy przez kanał ulega pewnym zniekształceniom, spowodowanym przez występowanie szumu w tym kanale. Proces rekonstrukcji wiadomości odbywa się za pomocą modelu źródłowego oraz modelu kanału, które można odnieść do występujących w statystycznym tłumaczeniu automatycznym pojęć modelu języka oraz modelu tłumaczenia. Kluczowa jest tutaj wiedza o rozkładzie prawdopodobieństwa wiadomości źródłowych oraz rozkładzie prawdopodobieństwa poszczególnych typów zakłóceń.

Podejście do statystycznego tłumaczenia automatycznego przy wykorzystaniu metod probabilistycznych jest możliwe głównie dzięki modelowi zaszumionego kanału. Rola modelu języka oraz modelu tłumaczenia, będących komponentami podstawowego równania statystycznego tłumaczenia automatycznego, zostanie przybliżona w podrozdziale [2.1.3].

2.1.3 Model log-liniowy w statystycznym tłumaczeniu automatycznym

Zastosowanie modelu języka, określanego jako $P(E = e)$ w podstawowym równaniu statystycznego tłumaczenia automatycznego, pozwala na otrzymanie zdań docelowych, wyróżniających się płynnością. Brane pod uwagę czynniki to nie tylko poprawność znaczeniowa, ale także kolejność słów. Model języka pomaga w dokonaniu wyboru odpowiedniego tłumaczenia spośród dostępnych alternatyw. Dobre modele przypisują wyższe prawdopodobieństwa bardziej naturalnie brzmiącym zdaniom. W statystycznym tłumaczeniu automatycznym wykorzystuje się modele n-gramowe.

Definicja 5 *(Model n-gramowy języka). Model n-gramowy języka to specyficzny rodzaj modelu języka, który wykorzystuje $n - 1$ poprzednich słów do obliczenia prawdopodobieństwa wystąpienia kolejnego słowa.*

Zgodnie z definicją (5), model 3-gramowy korzysta z dwóch poprzednich słów do obliczenia prawdopodobieństwa wystąpienia kolejnego słowa. Estymacja maksymalnego prawdopodobieństwa w przypadku modelu 3-gramowego sprowadza się do wzoru

$$p(w_3|w_1, w_2) = \frac{count(w_1, w_2, w_3)}{\sum_w count(w_1, w_2, w)} \tag{6}$$

Wzór (6) pozwala określić jak często w korpusie równoległym sekwencja słów w_1 i w_2 występuje za słowem w_3. W przykładzie (4) zaprezentowano praktyczne wykorzystanie wzoru (6).

Przykład 4 W przykładowym korpusie zanotowano 725 wystąpień sekwencji słów *the high*. Dane dotyczące słów występujących w korpusie najczęściej po tej sekwencji zawarto w tabeli (2).

the high (łącznie: 725)

Słowo	Zliczenia	Wynik estymaty
court	380	0.524
castle	175	0.241
heel	54	0.074
cost	22	0.03

Tabela 2: Estymaty maksymalnego prawdopodobieństwa tłumaczenia dla określonych trzywyrazowych sekwencji słów

Z modelami języka współpracują modele tłumaczenia. Działanie modeli tłumaczenia może opierać się na tłumaczeniu słów (ang. *word-based*) lub tłumaczeniu fraz (ang. *phrase-based*).

Definicja 6 *(Fraza). Fraza w statystycznym tłumaczeniu maszynowym to każda ciągła i spójna sekwencja słów, niekoniecznie będąca jednostką językową.*

Definicja 7 *(Model oparty na tłumaczeniu słów). Model oparty na tłumaczeniu słów to model tłumaczenia, w którym jednostkami atomowymi są słowa.*

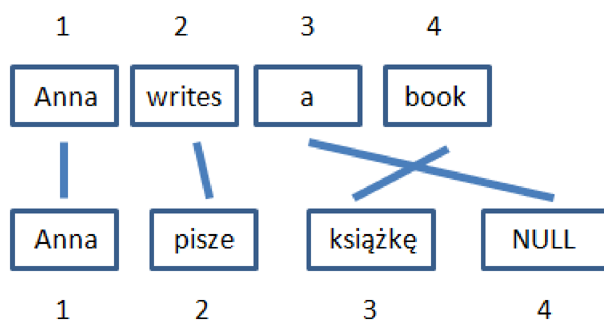
W korpusie równoległym słowa zdań w języku źródłowym są urównoleglane do słów w języku docelowym. Podczas procesu tłumaczenia kolejność słów może ulec zmianie. Słowa mogą być także usuwane lub dodawane. Model typu *word-based* korzysta z prawdopodobieństw leksykalnych tłumaczeń, które służą do znalezienia najlepszego tłumaczenia słowa.

Definicja 8 *(Model oparty na tłumaczeniu fraz). Model oparty na tłumaczeniu fraz to model tłumaczenia, w którym jednostkami atomowymi są frazy.*

Sekwencje fraz powstające w wyniku podziału zdania wejściowego są mapowane jeden do jednego do fraz wyjściowych, a ich kolejność może ulec zmianie.

Przykłady (5) oraz (6) dotyczą wykorzystywania w procesie tłumaczenia modeli *word-based* oraz *phrase-based.*

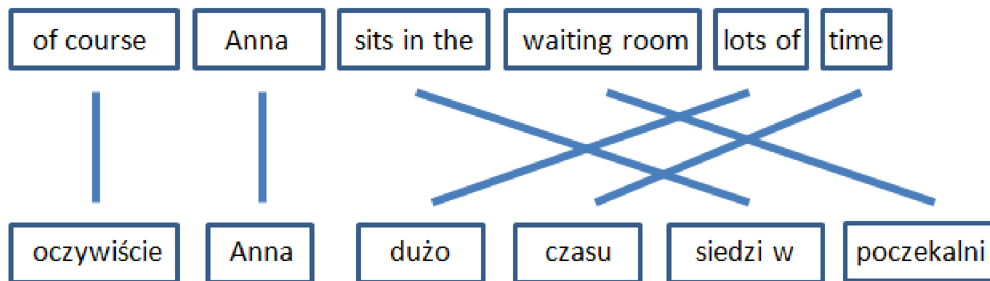
Przykład 5 Tekst *Anna writes a book* został przetłumaczony na język polski. Wybrano słowa języka docelowego, które uzyskały najwyższe wyniki leksykalnych prawdopodobieństw tłumaczeń. Przy wykorzystaniu modelu *word-based* słowa zdania języka źródłowego są traktowane jak jednostki atomowe i urównoleglane do słów zdania języka docelowego. Jedno słowo zdania języka źródłowego zostaje urównoleglone do jednego słowa zdania języka docelowego.



Rysunek 4: Przykład urównoleglenia dla tekstu *Anna writes a book*

Wykorzystany w tym przypadku token NULL pomaga radzić sobie z brakiem ekwiwalentów w języku docelowym dla słowa języka źródłowego *a*. Funkcja urównoleglania powinna być kompletna i w pełni zdefiniowana. Dokonanie tego w takiej sytuacji umożliwia właśnie token NULL.

Przykład 6 Tekst *of course Anna sits in the waiting room lots of time* w języku źródłowym został podzielony na sekwencje fraz. Każda z wyodrębnionych fraz jest traktowana jako jednostka atomowa i tłumaczona na język docelowy. Kolejność fraz może ulegać zmianie.



Rysunek 5: Przykład urównoleglenia dla tekstu *of course Anna sits in the waiting room lots of time*

Wśród modeli opartych na tłumaczeniu fraz wyróżnia się modele hierarchiczne (ang. *hierarchical translation models*) oraz modele składniowe (ang. *syntax-based translation models*).

W tradycyjnych modelach typu *phrase-based* frazę traktuje się jako ciągłą sekwencję słów. Modele hierarchiczne pomagają w sytuacjach, gdy słowa frazy zostają rozdzielone w zdaniu innymi wyrazami, co zdarza się dość powszechnie. Poprzez stosowanie reguł tłumaczenia (ang. *translation rules*) możliwe jest uwzględnienie odpowiednich przestawień poszczególnych słów.

Przykład 7 Dla zdania w języku angielskim *My best friends will be deported today* wyodrębniono wyrażenie *will be deported today*. Tłumaczeniem polskim wyrażenia *will be deported today* jest *zostaną dziś deportowani*. Regułę tłumaczenia dla przykładu przedstawiono w formule (7).

$$Y -> will\ be\ deported\ today \mid zostaną\ dziś\ deportowani \tag{7}$$

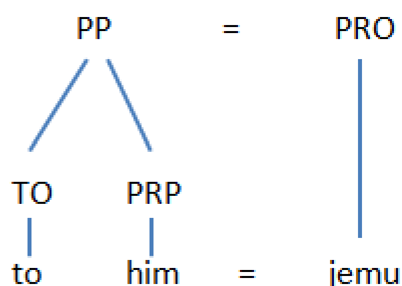
Reguła może zostać doprowadzona do ogólniejszej postaci dzięki wprowadzeniu symbolu nieterminalnego X. Symbol nieterminalny jest definiowalną zmienną. Postać reguły przedstawiono w formule (8).

$$Y- > \textit{will be deported } X \mid \textit{zostaną } X \textit{ deportowani} \tag{8}$$

Reguły tego typu, łączące w sobie symbole terminalne oraz nieterminalne, pozwalają poradzić sobie ze zmianami kolejności słów i łączeniem słów będących komponentami fraz, nawet jeśli w zdaniach nie są one ciągłymi sekwencjami.

Modele składniowe operują na regułach syntaktycznych, które oprócz słów zawierają także dodatkowe informacje dotyczące składni. Przykład (8) prezentuje budowanie reguły syntaktycznej.

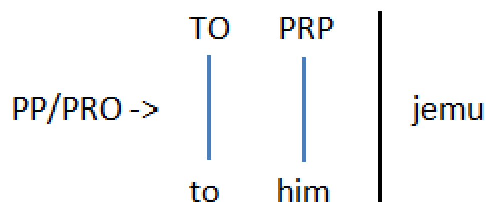
Przykład 8 Dla zdania języka angielskiego *Give it to him* wyodrębniono wyrażenie *to him*. Wyrażenie *to him* zostało urównoleglone do polskiego wyrażenia *jemu*. Dodatkowo zawarto informację o częściach mowy poszczególnych wyrazów i uzyskano mapowanie przedstawione na rysunku (6).



Rysunek 6: Mapowanie części mowy oraz słów

PRP to oznaczenie określające przyimek, a PRO - zaimek. Lewa strona rysunku (6) jest tak naprawdę drzewem. Fraza *to him* składa się z dwóch węzłów, będących potomkami frazy przyimkowej oznaczonej jako PP.

Na podstawie podanych informacji o częściach mowy i zaprezentowanego wnioskowania można określić regułę syntaktyczną przedstawioną na rysunku (7).



Rysunek 7: Reguła syntaktyczna

Modele tłumaczenia, w których wykorzystuje się drzewa syntaktyczne oraz reguły syntaktyczne, pozwalają na wykorzystanie składni, czyli wzajemnych powiązań między wyrazami oraz frazami w zdaniu w szczególnym stopniu. Znajomość tych powiązań wspiera proces tłumaczenia i pozwala na uzyskanie lepszej jakości. Jeśli porządek występujących słów lub fraz jest poprawny, zdania brzmią bardziej naturalnie.

Charakterystycznymi wyróżnikami modeli typu *phrase-based* są ich komponenty definiowane jako funkcje cech (ang. *feature functions*).

Definicja 9 *(Funkcje cech). Funkcje cech to funkcje określane na podstawie wykorzystanych własności tłumaczenia.*

Przykładami funkcji cech mogą być tabela tłumaczeń fraz oraz model zmiany kolejności.

Definicja 10 *(Tabela tłumaczeń fraz). Tabela tłumaczeń fraz to tabela przedstawiająca poszczególne propozycje tłumaczeń fraz wraz z prawdopodobieństwami tych tłumaczeń.*

Definicja 11 *(Model zmiany kolejności). Model zmiany kolejności to struktura, która zapamiętuje końcową pozycję poprzednio przetłumaczonej frazy w*

tekście wejściowym oraz startową pozycję bieżącej frazy w tekście wejściowym. Znajomość tych dwóch liczb umożliwia obliczenie kosztu zmiany kolejności (ang. reordering cost).

Funkcje cech wchodzą w skład modelu log-liniowego (ang. *log-linear model*) - typu modelu tłumaczenia także opartego na tłumaczeniu fraz.

Definicja 12 *(Model log-liniowy). Model log-liniowy to rodzaj modelu tłumaczenia opartego na frazach, składającego się z tabeli tłumaczeń fraz (ang. phrase translation table)* $\phi(\bar{f}|\bar{e})$*, modelu zmiany kolejności (ang. reordering model)* d *oraz modelu języka* $p_{LM}(e)$*.*

Wzór dla modelu log-liniowego zapisano w formule (9).

$$p(x) = exp \sum_{i=1}^{n} \lambda_i h_i(x) \tag{9}$$

Wzór (9) wykorzystuje odpowiednie wagi, które oznacza się jako λ_ϕ, λ_d oraz λ_{LM}, w zależności od komponentu. Każda z wag dotyczy innego komponentu modelu log-liniowego. λ_ϕ to waga dla tabeli tłumaczeń fraz, λ_d to waga dla modelu zmiany kolejności, a λ_{LM} to waga dla modelu języka.

Funkcje cech to: $h_1 = log\,\phi$, $h_2 = log\,d$, $h_3 = log\,p_{LM}$, a zatem $n = 3$. $h_1 = log\,\phi$ to funkcja dla tabeli tłumaczeń fraz, $h_2 = log\,d$ to funkcja dla modelu zmiany kolejności, a $h_3 = log\,p_{LM}$ to funkcja dla modelu języka. Oznaczenie x wskazuje na dowolną zmienną zdefiniowaną jako $x = (e, f, start, end)$.

Korzystając z tych informacji, wzór formuły (9) można rozwinąć do postaci formuły (10), co zostało przedstawione w przykładzie (9).

Przykład 9 Mając do dyspozycji trzy funkcje cech $h_1 = log\,\phi$, $h_2 = log\,d$, $h_3 = log\,p_{LM}$ oraz wagi λ_ϕ, λ_d, λ_{LM} można rozwinąć wzór formuły (9) do postaci

$$p(e,a|f) = exp\Big[\lambda_\phi \sum_{i=1}^{I} log\phi(\bar{f}_i|\bar{e}_i)$$
$$+\lambda_d \sum_{i=1}^{I} logd(a_i - b_{i-1} - 1) \quad (10)$$
$$+\lambda_{LM} \sum_{i=1}^{|e|} logp_{LM}(e_i|e_1...e_{i-1})\Big]$$

Dane są traktowane jako wektory cech, a model log-liniowy jako zestaw odpowiadających funkcji cech. Funkcje cech są rozpatrywane jako wzajemnie niezależne czynniki.

Za stosowaniem modelu log-liniowego przemawia przede wszystkim ulepszenie jakości tłumaczenia poprzez wykorzystywanie wag dla komponentów tego modelu. Dodatkową zaletą jest możliwość rozszerzania struktury o dodatkowe komponenty modelu, takie jak np. kierunki tłumaczenia, w postaci funkcji cech.

2.1.4 Metody ewaluacji statystycznego tłumaczenia automatycznego

Metody ewaluacji w statystycznym tłumaczeniu automatycznym służą do oceny jakości tłumaczenia reprezentowanego w języku docelowym.

Warto podkreślić, że jedną z podstawowych, nadal powszechnie stosowanych metod jest ewaluacja manualna. Osoby, które rozumieją język docelowy, oceniają system tłumaczenia automatycznego dokonując porównań do tłumaczeń referencyjnych (ang. *reference translation*).

Kryteria brane pod uwagę przy ewaluacji tłumaczenia to płynność (ang. *fluency*) oraz adekwatność (ang. *adequacy*). Płynność dotyczy poprawności gramatycznej i wyboru właściwych słów. Adekwatność to poprawność znaczeniowa względem zdania źródłowego.

Definicja 13 *(Tłumaczenie systemowe). Tłumaczenie systemowe to tłumaczenie będące wynikiem działania systemu statystycznego tłumaczenia auto-*

matycznego.

Ze względu na ogromne ilości przetwarzanych danych, ewaluacja manualna tłumaczenia systemowego jest zbyt czasochłonna. Stąd preferuje się stosowanie automatycznych metod ewaluacji. Każda automatyczna metoda ewaluacji tłumaczenia opiera się na stosowaniu porównań i badaniu podobieństwa tłumaczenia systemowego do tłumaczenia referencyjnego.

Podstawowymi miarami automatycznej ewaluacji tłumaczenia są precyzja (ang. *precision*) oraz pokrycie (ang. *recall*). W obu miarach tłumaczenie systemowe jest porównywane z tłumaczeniem referencyjnym.

Definicja 14 *(Precyzja). Precyzja to automatyczna miara ewaluacji tłumaczenia bazująca na ilorazie liczby słów pokrywających się w tłumaczeniu systemowym oraz tłumaczeniu referencyjnym (correct) przez liczbę słów tłumaczenia systemowego (outputLEN).*

Wzór wykorzystywany przy obliczaniu precyzji zapisano w formule (11).

$$precision = \frac{correct}{outputLEN} \tag{11}$$

Definicja 15 *(Pokrycie). Pokrycie to automatyczna miara ewaluacji tłumaczenia bazująca na ilorazie liczby słów pokrywających się w tłumaczeniu systemowym oraz tłumaczeniu referencyjnym (correct) przez liczbę słów tłumaczenia referencyjnego (referenceLEN).*

Wzór wykorzystywany przy obliczaniu pokrycia zapisano w formule (12).

$$recall = \frac{correct}{referenceLEN} \tag{12}$$

Obliczenia dla precyzji oraz pokrycia przedstawiono w przykładzie (10).

Przykład 10 Jeśli tłumaczeniem systemowym dla zdania *you are waiting for me at home now* jest *wy czekasz na mnie o domu*, a tłumaczeniem referencyjnym *teraz wy czekacie na mnie w domu*, to obliczenia dla precyzji i pokrycia będą wyglądały tak, jak zapisano w formułach (13) oraz (14).

$$precision = \frac{correct}{outputLEN} = \frac{4}{6} = 67\% \tag{13}$$

$$recall = \frac{correct}{referenceLEN} = \frac{4}{7} = 57\% \tag{14}$$

Specyficzną kombinacją precyzji oraz pokrycia jest miara *f-measure*. Wzór wykorzystywany przy *f-measure* przedstawiono w formule (15).

$$f - measure = \frac{precision * recall}{(precision + recall)/2} \tag{15}$$

Miarą wykorzystywaną w automatycznej ewaluacji tłumaczenia jest również *Word Error Rate (WER)*.

Definicja 16 *(Word Error Rate). Word Error Rate (WER) to automatyczna miara ewaluacji tłumaczenia, której działanie opiera się na mierzeniu odległości Levensteina zdefiniowanej jako minimalna liczba kroków niezbędnych do wykonania w celu doprowadzenia zdania do postaci z tłumaczenia referencyjnego.*

Wzór, który jest szczególnie istotny przy określaniu wartości *Word Error Rate* zapisano w formule (16). Zawarte w liczniku czynniki sumy wskazują na operacje, których wykonanie pozwoliłoby tłumaczeniu systemowemu przybliżyć się do postaci tłumaczenia referencyjnego. Wyszczególnione działania to zamiany (*subst*), wstawienia (*ins*) i usunięcia (*del*). Warto zauważyć, że wszystkie dotyczą operowania na słowach, a nie znakach.

$$WER = \frac{subst + ins + del}{referenceLEN} \tag{16}$$

Obliczenia *Word Error Rate* przedstawiono w przykładzie (11).

Przykład 11 Załóżmy, że dla zdania *They have a cat*, tłumaczenie systemowe to *Oni mieć tego kota*, a tłumaczenie referencyjne to *Oni mają kota*. *Word Error Rate* dla przykładu obliczono w formule (17).

$$WER = \frac{subst + ins + del}{referenceLEN} = \frac{1 + 0 + 1}{3} = \frac{2}{3} = 0.67 \tag{17}$$

Z perspektywy niniejszej pracy znacznie istotniejszą miarą jest *BLEU*, które wykorzystano do ewaluacji wyników przeprowadzonego eksperymentu.

Definicja 17 *(A Bilingual Evaluation Understudy). A Bilingual Evaluation Understudy (BLEU) to automatyczna miara ewaluacji tłumaczenia, biorąca pod uwagę dopasowania n-gramów pomiędzy tłumaczeniem systemowym, a tłumaczeniem referencyjnym oraz stosująca kary za zwięzłość tekstu, które redukują otrzymany wynik w przypadku zbyt krótkiego tłumaczenia.*

BLEU zdefiniowano matematycznie w formułach (18) i (19).

$$BLEU - n = brevity\ exp \sum_{i=1}^{n} \lambda_i \, log \, precision_i \tag{18}$$

$$brevity = min\left(1, \frac{outputLEN}{referenceLEN}\right) \tag{19}$$

Zazwyczaj liczba n osiąga wartość 4. Stąd mówi się o metryce *BLEU-4*. Stosowane *brevity* to kara za zwięzłość tekstu. Proces dopasowania n-gramów umożliwia obliczenie precyzji dla n-gramów, co we wzorze (18) wyrażono jako $precision_i$. Precyzja n-gramów to iloraz n-gramów tłumaczenia systemowego pokrywających się z n-gramami tłumaczenia referencyjnego przez całkowitą ilość n-gramów z tłumaczenia systemowego.

Przykład (12) przedstawia obliczenia dla precyzji n-gramów.

Przykład 12 Tłumaczeniem systemowym dla zdania *broccoli contain a lot of nutrients* jest *brokuły zawierają dużo odżywek*, a tłumaczeniem referencyjnym *brokuły zawierają dużo składników odżywczych*. Wyniki precyzji dla poszczególnych n-gramów przedstawiono w formule (20).

$$
\begin{aligned}
1 - gram - precision &: \frac{3}{4} \\
2 - gram - precision &: \frac{2}{3} \\
3 - gram - precision &: \frac{1}{2} \\
4 - gram - precision &: \frac{0}{1}
\end{aligned}
\tag{20}
$$

Stosowanie wag, oznaczonych w formule jako λ_i i ustawienie ich wartości na 1, znacząco upraszcza wzór wykorzystywany w mierze *BLEU-4*. Osiąga on wówczas postać zapisaną w formule (21).

$$
BLEU - 4 = min\left(1, \frac{outputLEN}{referenceLEN}\right) \prod_{i=1}^{4} precision_i \tag{21}
$$

Obliczenia dla *BLEU-4* przedstawiono w przykładzie (13).

Przykład 13 Tłumaczeniem systemowym dla zdania *broccoli contain a lot of nutrients* jest *brokuły zawierają dużo odżywek*, a tłumaczeniem referencyjnym *brokuły zawierają dużo składników odżywczych*. Wyniki dla *BLEU-4* przedstawiono w formule (22).

$$BLEU-4 = min\left(1, \frac{outputLEN}{referenceLEN}\right)\prod_{i=1}^{4} precision_i = min\left(1, \frac{4}{5}\right) * \frac{3}{4} * \frac{2}{3} * \frac{1}{2} * 0 = 0 \tag{22}$$

Tłumaczenie zostało ukarane za zwięzłość. Świadczy o tym wynik uzyskany w *brevity*. Iloraz długości otrzymanego tłumaczenia systemowego przez długość tłumaczenia referencyjnego wynosi $\frac{4}{5}$. Spośród dwóch wartości, zawsze wybierana jest mniejsza. Uzyskanie *brevity* równego jeden mogłoby mieć miejsce w dwóch sytuacjach - jeśli długość tłumaczenia systemowego byłaby większa od długości tłumaczenia referencyjnego lub jeśli długości tłumaczenia systemowego oraz tłumaczenia referencyjnego byłyby sobie równe. Tłumaczenie systemowe zawsze zostanie ukarane za zwięzłość, kiedy jego długość będzie mniejsza od długości tłumaczenia referencyjnego.

Należy też zauważyć, że w przypadku, gdy jakikolwiek czynnik *precision* osiągnie wartość 0, wówczas wynik *BLEU* także wyniesie 0. Oznacza to, że żadne z n-gramów o określonej długości nie zostały dopasowane w otrzymanym tłumaczeniu. W takiej sytuacji lepiej sprawdza się zastosowanie $n = 3$. Dzięki wykorzystaniu $n = 3$, możliwe jest pozbycie się elementu zerującego, odpowiedzialnego za 4-gramy. Wynik dla *BLEU-3* w przykładzie (13) wyniósłby 0.2.

W metryce *BLEU* wykorzystuje się wielokrotne tłumaczenia referencyjne dobrej jakości. Całkowita jakość tłumaczenia jest szacowana w obrębie całego korpusu. Miara *BLEU* zdecydowanie lepiej sprawdza się w przypadku obszernych korpusów niż pojedynczych zdań.

BLEU to miara, która kładzie szczególny nacisk na korelację pomiędzy tłumaczeniem systemowym, a tłumaczeniem referencyjnym. Tłumaczenie systemowe jest tym lepsze, im bardziej zbliżone do tłumaczenia referencyjnego.

W sytuacji gdy n-gram posiada dopasowanie w jakimkolwiek tłumaczeniu referencyjnym, uznawany jest za poprawny. Jeśli n-gram występuje

wiele razy w tłumaczeniu systemowym, powinien pojawić się w pojedynczym tłumaczeniu referencyjnym taką samą liczbę razy dla wszystkich wystąpień oznaczonych jako poprawne.

Funkcjonalność wykorzystania wielokrotnych tłumaczeń referencyjnych może powodować problemy z długościami referencyjnymi. Długości referencyjne są definiowane dla każdego zdania tłumaczenia systemowego jako długości najbliższe jakimkolwiek tłumaczeniom referencyjnym. W sytuacji gdy różnią się one nieznacznie, zawsze pod uwagę brana jest mniejsza długość. Przykład (14) prezentuje wybór długości referencyjnej.

Przykład 14 Jeśli zdanie uzyskanego tłumaczenia systemowego ma długość 8, a długości zdań referencyjnych wynoszą kolejno 6, 7, 9 i 13, to wówczas długość referencyjna dla tego zdania wynosi 7. Chociaż wielkości 7 i 9 leżą podobnie blisko, to zgodnie z definicją, właściwy wybór to mniejsza z wartości.

Wśród metryk ewaluacji statystycznego tłumaczenia maszynowego istotną rolę pełni także metryka *METEOR*, która kładzie szczególny nacisk na miarę pokrycia.

Definicja 18 *(Rdzeniowanie). Rdzeniowanie jest procesem wyodrębniania rdzenia z wyrazu, czyli jego głównej części, decydującej o znaczeniu.*

Definicja 19 *(Metric for Evaluation of Translation with Explicit ORdering). Metric for Evaluation of Translation with Explicit ORdering (METEOR) jest automatyczną miarą ewaluacji tłumaczenia maszynowego, która wykorzystuje precyzje n-gramów, ich pokrycia, rdzenie oraz synonimy.*

Pokrycie ma większe znaczenie, ponieważ zapewnia uzyskanie kompletnego, pod względem znaczeniowym, tłumaczenia. *METEOR* wykorzystuje proces rdzeniowania w celu porównywania znaczeń wyrazów oraz używa synonimów.

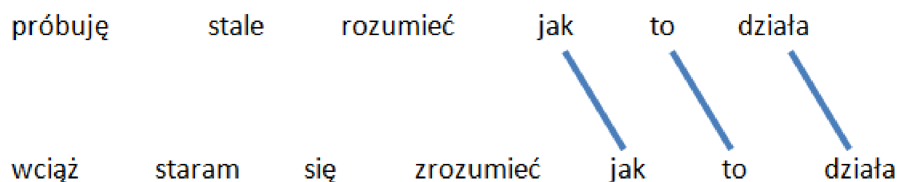
BLEU nie dokona obliczeń w przypadku próby dopasowania dwóch słów wyglądających inaczej, ale mających to samo znaczenie. Proces rdzeniowania prowadzi do redukowania słów poprzez wyodrębnianie ich rdzeni.

Krok ten jest szczególnie pomocny przy znajdowaniu właściwego dopasowania.

METEOR jest miarą, którą zaprojektowaną w celu poprawienia niedoskonałości znalezionych w *BLEU*. Stąd zaproponowanie rozwiązania wykorzystującego w większym stopniu pokrycie. Ewaluacja odbywa się na poziomie segmentów, będących zdaniami. Porównanie każdego z nich z tłumaczeniem referencyjnym, pozwala na uzyskanie wielu ocen. *METEOR* oferuje moduły *exact*, *stem*, *synonym* oraz *paraphrase*. Wszystkie służą do uzyskiwania najlepszych dopasowań pomiędzy słowami tłumaczenia systemowego, a tłumaczenia referencyjnego.

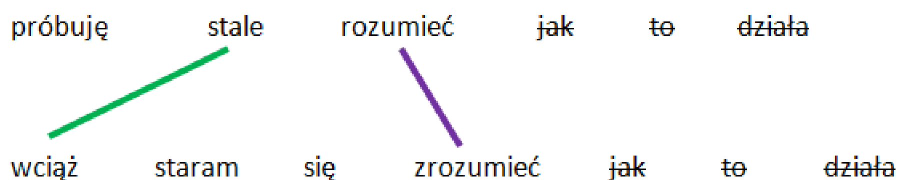
Przykład (15) obrazuje działanie modułów *exact*, *stem*, *synonym* oraz *paraphrase* dla miary *METEOR*.

Przykład 15 Tłumaczeniem systemowym dla zdania *I'm constantly trying to understand how it works* jest *próbuję stale rozumieć jak to działa*, a tłumaczeniem referencyjnym *wciąż staram się zrozumieć jak to działa*. Na rysunkach (8), (9) oraz (10) pokazano działanie modułów miary *METEOR*.



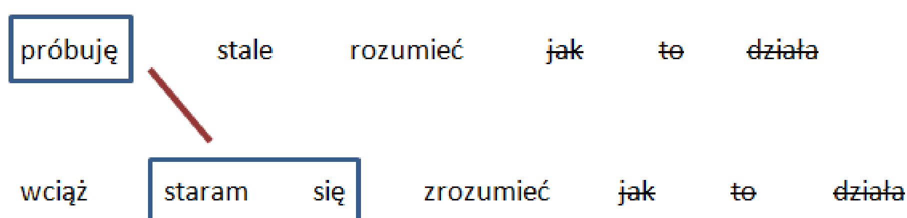
Rysunek 8: Uzyskiwanie dopasowania za pomocą modułu *exact* (na podstawie pracy [15])

Moduł *exact* znajduje dokładne słowne ekwiwalenty w języku docelowym. Moduły *stem* oraz *synonym* nie biorą pod uwagę słów dopasowanych przez działanie modułu *exact*.



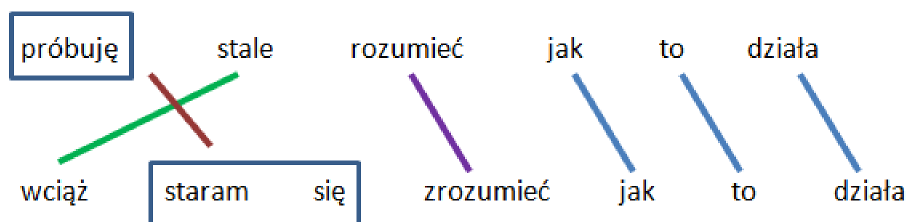
Rysunek 9: Uzyskiwanie dopasowania za pomocą modułów *stem* (fioletowa linia) i *synonym* (zielona linia) (na podstawie pracy [15])

Ze względu na dopasowywanie fraz zamiast pojedynczych słów, w module *paraphrase* brane są pod uwagę wszystkie słowa, również te dopasowane w module *exact*.



Rysunek 10: Uzyskiwanie dopasowania za pomocą modułu *paraphrase* (na podstawie pracy [15])

Następnie gromadzone są wszystkie dopasowania.



Rysunek 11: Zbiór wszystkich dopasowań (na podstawie pracy [15])

Wzór stosowany w mierze *METEOR* został przedstawiony w formule (23).

$$Score = \left(1 - \left(\gamma * \frac{ch}{m}\right)^{\beta}\right) * \left(\frac{P * R}{\alpha * P + (1 - \alpha) * R}\right) \tag{23}$$

P określa precyzję, a R - pokrycie. Parametr α pozwala na ustalenie wpływu precyzji i pokrycia na tłumaczenie. W mierze *METEOR* wykorzystuje się także karę za błędny szyk wyrazów, która służy do oceny poprawności gramatycznej tłumaczenia. Na karę za błędny szyk wyrazów składają się ilość zbitek wyrazowych (sekwencji słów) ch oraz liczba dopasowań m. Parametry γ oraz β to dodatkowe komponenty pomocnicze.

2.2 Wybrane aspekty uczenia maszynowego

2.2.1 Zagadnienie regresji liniowej

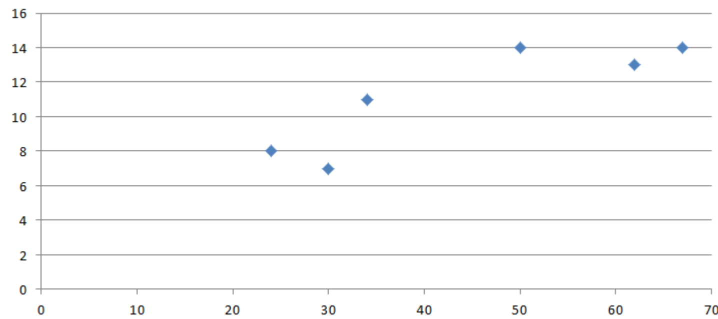
Zagadnienie regresji liniowej (ang. *linear regression*) jest jedną z metod uczenia maszynowego (ang. *machine learning*). W przykładzie (16) zaprezentowano działanie regresji liniowej.

Przykład 16 W tabeli (3) zaprezentowano przykładowy zestaw danych dotyczący długości najdłuższego słowa w zdaniu w zależności od ilości znaków w nim występujących.

Zdanie	Ilość znaków w zdaniu	Ilość znaków najdłuższego słowa w zdaniu
Nowa ustawa o cyberprzemocy została zawetowana przez opozycję.	62	13
Młodzi informatycy wygrali międzynarodowy konkurs.	50	14
Ministrowie wybrali się na obrady.	34	11
Współczesne kobiety chętnie biorą udział w licznych manifestacjach.	67	14
Matka kupiła mi starą książkę.	30	7
Bardzo ciekawa historia.	24	8
...	...	...

Tabela 3: Przykładowy zestaw danych dotyczący ilości znaków najdłuższego słowa w zdaniu w zależności od całkowitej ilości znaków w zdaniu

Dane z tabeli przeniesiono na wykres (Rysunek 12).



Rysunek 12: Wykres przedstawiający dane dotyczące ilości znaków najdłuższego słowa w zdaniu w zależności od całkowitej ilości znaków w zdaniu

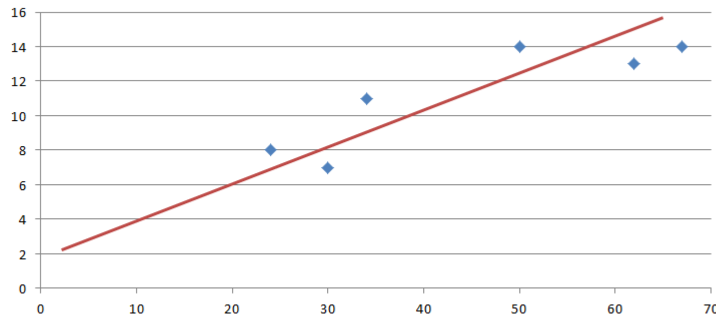
Definicja 20 *(Cecha). Cecha, w kontekście uczenia maszynowego, to indywidualna, mierzalna właściwość obserwowanego zjawiska.*

W przykładzie (16) wyróżnioną cechą jest ilość znaków w zdaniu. Każde zdanie definiuje się jako $x^{(i)}$, gdzie i wskazuje na numer rozpatrywanego zdania. W problemach uczenia maszynowego bazuje się na tysiącach, a nawet milionach cech.

Formuła (24) prezentuje zapis funkcji liniowej dla przykładu (16).

$$h_\theta(x) = \theta_0 + \theta_1 x_1 \tag{24}$$

Parametry θ nazywane są wagami. Odpowiedni dobór tych parametrów pozwala na znalezienie najlepszego dopasowania prostej do punktów przedstawionych na wykresie. Przykładowe dopasowanie prostej przedstawiono na rysunku (13).



Rysunek 13: Wykres przedstawiający linię regresji dla przykładowych danych.

Wzór funkcji liniowej pokazanej na wykresie (Rysunek 13) to $h_\theta(x) = 0.22 + 1.6x_1$.

Wzór wykorzystywany przy regresji liniowej można rozwinąć do postaci uwzględniającej wszystkie występujące cechy. Zapisano go w formule (25).

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x \tag{25}$$

W celu wygodniejszego operowania na dużych ilościach liczb, parametry θ_i przedstawia się w postaci transponowanego wektora θ.

Kluczowym zadaniem jest znalezienie takich parametrów θ, które pozwolą hiperpłaszczyźnie znaleźć się jak najbliżej przykładów oznaczonych na wykresie. W tym celu definiuje się funkcję kosztu (ang. *cost function*).

Definicja 21 *(Funkcja kosztu). Funkcja kosztu to określona funkcja, która dla każdej wartości parametru θ potrafi zmierzyć jak blisko $y^{(i)}$ znajdują się od $h(x^{(i)})$.*

Przykładowy wzór definiujący funkcję kosztu przedstawiono w formule (26).

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{26}$$

Funkcja kosztu określa, czy parametry θ zostały dobrane w prawidłowo. Należy dążyć do uzyskania jak najlepszych wyników, dlatego minimalizuje się funkcję kosztu $J(\theta_0, ..., \theta_n)$. Do tego celu służy metoda gradientu prostego.

Definicja 22 *(Metoda gradientu prostego). Metoda gradientu prostego jest algorytmem wykorzystywanym w uczeniu maszynowym, którego działanie polega na iteracyjnych zmianach zainicjalizowanych parametrów $\theta_0, ..., \theta_n$ aż do momentu osiągnięcia minimum funkcji.*

Pojedynczy krok aktualizujący dla metody gradientu prostego, przedstawiono w formule (27).

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \tag{27}$$

2.2.2 Zagadnienie klasyfikacji

Definicja 23 *(Klasa). Klasa jest pewną grupą obiektów, wyróżniających się wspólnymi cechami.*

Zagadnienie klasyfikacji polega na przewidywaniu dyskretnych wartości y. Najprostszym typem klasyfikacji jest klasyfikacja binarna (ang. *binary classification*). Wartości przyjmowane wówczas przez y to 0 lub 1. Liczby te reprezentują klasę negatywną oraz klasę pozytywną.

Znacznie bardziej zaawansowana forma klasyfikacji, czyli klasyfikacja wieloklasowa (ang. *multiclass classification*), pozwala na przypisanie do obiektu jednej z wielu dostępnych klas. Przykład (17) prezentuje działanie klasyfikacji.

Przykład 17 W tabeli (4) zaprezentowano przykładowy zestaw danych. Są nimi zdania wraz z wyodrębnionymi słowami, które sprowadzono do małych liter. Każde zdanie należy do innej kategorii tematycznej.

Zdanie	Słowa w zdaniu	Kategoria
Mieszkanie jest przytulne.	mieszkanie, jest, przytulne	Inne
Obserwuje się duży wzrost popytu.	obserwuje, się, duży, wzrost, popytu	Gospodarka
Politycy czekają.	politycy, czekają	Polityka

Tabela 4: Przykładowy zestaw danych sklasyfikowanych względem kategorii

W procesie klasyfikacji wykorzystuje się wektory cech (ang. *feature vectors*).

Definicja 24 *(Wektor cech). Wektor cech jest n-wymiarowym wektorem, który w sposób numeryczny określa cechy służące reprezentacji danego obiektu.*

Do reprezentacji wektorowej mogą zostać wykorzystane wszystkie słowa wchodzące w skład korpusu lub słowa kluczowe dla korpusu. Każdy wyraz, którego wystąpienie traktowane jest jako cecha, definiuje się w sposób numeryczny. Zdania interpretuje się w tym przypadku jako wektory cech równej długości, w których 1 oznacza wystąpienie wyrazu w zdaniu, a 0 brak takiego wystąpienia. Przykładowy wektor cech dla zdania *Mieszkanie jest przytulne* został przedstawiony w formule (28). Za pomocą occ_{word} oznaczono wystąpienie słowa *word* w zdaniu. Formalnie wartości cech można zapisać jako x_i. Indeks i określa numer cechy.

$$x = \begin{bmatrix} occ_{mieszkanie} \\ occ_{jest} \\ occ_{przytulne} \\ occ_{obserwuje} \\ occ_{się} \\ occ_{duży} \\ occ_{wzrost} \\ occ_{popytu} \\ occ_{politycy} \\ occ_{czekają} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{28}$$

Ilość cech dla przykładu wynosi 10, ponieważ ilość niepowtarzających się słów wyodrębnionych ze zdań zapisanych w tabeli (4) wynosi właśnie 10. Każda cecha służy określeniu wystąpienia pojedynczego wyrazu. Cechy w przykładzie odnoszą się do wyrazów składających się na zdania, zgodnie z kolejnością w tabeli. Wyrazy *mieszkanie*, *jest*, *przytulne* występują w rozpatrywanym zdaniu i właśnie dlatego w wektorze znalazły się jedynki. Wyrazy takie jak *politycy*, czy *obserwuje* nie wchodzą w skład zdania i dlatego w wektorze pojawiły się zera.

Kategorie tematyczne można rozpatrywać jako pewne klasy. Celem klasyfikacji jest przypisanie nowego obiektu, czyli zdania, do odpowiedniej klasy. Zdanie *Mieszkanie jest przepiękne* składa się ze słów *mieszkanie*, *jest*, *przepiękne*. Wektor cech dla tego zdania przedstawiono w formule (29).

$$x = \begin{bmatrix} occ_{mieszkanie} \\ occ_{jest} \\ occ_{przytulne} \\ occ_{obserwuje} \\ occ_{się} \\ occ_{duży} \\ occ_{wzrost} \\ occ_{popytu} \\ occ_{politycy} \\ occ_{czekają} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{29}$$

Porównanie wektora cech nowego zdania z wektorami cech zdań znajdujących się w korpusie, doprowadzi do pewnych wniosków. Wektory cech zdań *Mieszkanie jest przytulne* oraz *Mieszkanie jest przepiękne* różnią się na jednej pozycji. Wektory cech zdań *Obserwuje się duży wzrost popytu* oraz *Mieszkanie jest przepiękne* różnią się na siedmiu pozycjach. Wektory cech zdań *Politycy czekają* oraz *Mieszkanie jest przepiękne* różnią się na czterech pozycjach. Wektor cech nowego zdania *Mieszkanie jest przepiękne* jest najbardziej zbliżony do wektora cech zdania *Mieszkanie jest przytulne*, które przyporządkowano do klasy *Inne*. Ze względu na największe podobieństwo wektorów tych zdań, nowe zdanie zostanie przyporządkowane również do klasy *Inne*.

Przykład (17) przedstawia ogólną, uproszczoną ideę klasyfikacji polegającą na porównywaniu wektorów i określaniu ich wzajemnego podobieństwa. Do operowania na dużej ilości cech, wykorzystuje się regresję logistyczną (ang. *logistic regression*).

Wzór dla regresji logistycznej przedstawiono w formule (30).

$$g(z) = \frac{1}{1 + e^{-z}} \tag{30}$$

Wzór zapisany w formule (30) można rozwinąć do postaci w formule (31).

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad (31)$$

Funkcja zapisana w formule (31) nazywana jest funkcją logistyczną lub funkcją sigmoidalną.

Podobnie jak dla regresji liniowej, także dla regresji logistycznej definiuje się funkcję kosztu. Przedstawiono ją w formule (32).

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))] \quad (32)$$

Metoda gradientu prostego działa dla regresji logistycznej w taki sam sposób jak dla regresji liniowej. Inaczej wygląda wzór pojedynczej aktualizacji. Zapisano go w formule (33).

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (33)$$

2.2.3 Zagadnienia algorytmiczne oraz model One Against All

Podstawowe algorytmy uczenia maszynowego są odmianami algorytmu gradientu prostego (ang. *Gradient Descent*), przybliżonego w podrozdziale [2.2.1]. Należą do nich *Batch Gradient Descent*, *Mini-Batch Gradient Descent* oraz *Stochastic Gradient Descent*.

Batch Gradient Descent to algorytm, w którym aktualizacja parametrów θ następuje po przejściu przez określoną liczbę przykładów nazywaną batchem i będącą wsadem danych. W porównaniu do Gradient Descent, w *Batch Gradient Descent* aktualizacja następuje po przejściu przez pewną partię danych, zamiast wszystkie przykłady od razu.

Szczególną odmianą algorytmu *Batch Gradient Descent* jest algorytm *Mini-Batch Gradient Descent*. Rozmiar partii danych określa mini-batch. Rozmiar mini-batcha jest znacznie mniejszy od rozmiaru batcha stosowa-

nego w *Batch Gradient Descent*. Preferowany rozmiar mini-batcha wynosi 10. Po przejściu przez określoną liczbę przykładów następuje pojedyncza aktualizacja parametrów θ. Okno kodu (1) prezentuje pojedynczą aktualizację parametrów θ w algorytmie *Mini-Batch Gradient Descent*.

```
1 Loop {
2     for i=1 to m, {
3       θ_j := θ_j − α * 1/10 * Σ_{k=i}^{i+9} (h_θ(x^(k)) − y^(k)) x_j^(k)  for every j
4   }
5 }
```

Kod 1: Pojedyncza aktualizacja parametrów θ dla algorytmu Mini-Batch Gradient Descent (na podstawie [6])

Jeśli liczba przykładów treningowych m wynosi 10000, to wówczas należy wykonać 1000 kroków algorytmu *Mini-Batch Gradient Descent* o wielkości batcha równej 10, by przejść przez wszystkie te przykłady.

Algorytm *Mini-Batch Gradient Descent* jest szczególną odmianą algorytmu *Batch Gradient Descent*, w którym partia danych, zamiast dużej liczby przykładów, zawiera ich znacznie mniej - tyle, ile sugeruje rozmiar mini-batcha.

Alternatywą dla algorytmów *Batch Gradient Descent* oraz *Mini-Batch Gradient Descent* jest *Stochastic Gradient Descent*. Aktualizacja parametrów θ następuje po przejściu przez pojedynczy przykład treningowy.

Celem działania algorytmu *Stochastic Gradient Descent* jest jak najszybsze znalezienie parametrów θ. Parametry te, w przeciwieństwie do innych odmian algorytmu Gradient Descent, są znajdowane bardzo szybko, a ich wartości są zbliżone do poszukiwanego minimum. Okno kodu (2) prezentuje pojedynczą aktualizację parametrów θ w algorytmie *Stochastic Gradient Descent*.

```
Loop {
  for i=1 to m, {
    θ_j := θ_j + α(y^(i) − h_θ(x^(i)))x_j^(i) for every j
  }
}
```

Kod 2: Pojedyncza aktualizacja parametrów θ dla algorytmu Stochastic Gradient Descent (na podstawie [6])

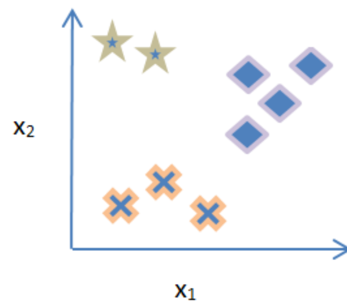
Niewątpliwą zaletą algorytmu *Stochastic Gradient Descent* jest jego wysoka wydajność.

Należy zwrócić uwagę, że każdy z opisanych w niniejszym rozdziale algorytmów działa na podstawie określonego batcha. Dla *Batch Gradient Descent* rozmiar batcha jest większy niż w przypadku *Mini-Batch Gradient Descent*. Dla *Stochastic Gradient Descent* rozmiar batcha wynosi 1, ponieważ aktualizacja następuje po każdym przykładzie z zestawu treningowego.

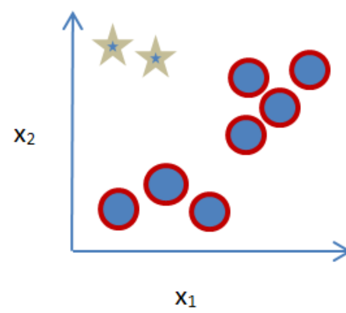
Mini-Batch Gradient Descent jest odmianą algorytmu *Batch Gradient Descent*, więc trudno bezpośrednio porównać oba algorytmy. *Mini-Batch Gradient Descent* może okazać się lepszy od *Stochastic Gradient Descent*. Kluczowa w tym przypadku jest wektoryzacja. Zastosowanie wektorów może pozwolić na jednoczesne przetwarzanie większej liczby przykładów.

Z punktu widzenia klasyfikacji wieloklasowej, bardzo istotna jest możliwość zastosowania modelu jeden przeciwko wszystkim (ang. *One Against All*).

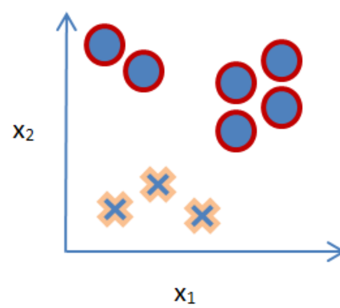
Klasyfikacja wieloklasowa wyróżnia więcej niż dwie klasy. Do każdej z klas mogą należeć obiekty reprezentowane przez przykłady treningowe. Model jeden przeciwko wszystkim pozwala przełożyć problem klasyfikacji wieloklasowej na kilka równorzędnych problemów klasyfikacji binarnej. Na rysunku (14) przedstawiono trzy klasy wyróżnione w zestawie treningowym, oznaczone symbolicznie za pomocą gwiazdek, rombów i krzyżyków. Zastosowanie modelu *jeden przeciwko wszystkim* będzie polegało na skorzystaniu z trzech klasyfikatorów binarnych. Aktualnie rozpatrywane obiekty będą należały do klasy pozytywnej, a pozostałe obiekty - do klasy negatywnej. Działanie klasyfikatorów binarnych pokazano na rysunkach (15), (16), (17).



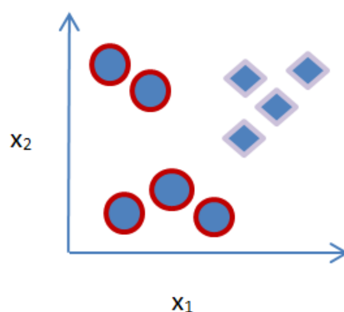
Rysunek 14: Klasy wyróżnione spośród zestawu treningowego



Rysunek 15: Proces klasyfikacji binarnej przeprowadzany w ramach modelu One Against All (1)



Rysunek 16: Proces klasyfikacji binarnej przeprowadzany w ramach modelu One Against All (2)



Rysunek 17: Proces klasyfikacji binarnej przeprowadzany w ramach modelu One Against All (3)

Dużą rolę odgrywa tutaj regresja logistyczna, ponieważ dla każdej klasy trenowany jest klasyfikator regresji logistycznej. Celem działania klasyfikatora jest przewidzenie prawdopodobieństwa, że $y = i$, gdzie i określa numerycznie klasę.

2.3 Aspekty językoznawcze i morfologiczne

W statystycznym tłumaczeniu automatycznym stosowane są pojęcia łączące w sobie zarówno statystykę, jak i lingwistykę komputerową. Większość pojęć nie byłaby jasna, gdyby nie terminy leżące u podstaw nauki o języku.

Definicja 25 *(Morfologia). Morfologia jest dziedziną lingwistyki, która zajmuje się wewnętrzną strukturą wyrazu, czyli budową form odmiany wyrazu oraz wyrazem jako jednostką słownikową.*

Na tym poziomie należy rozróżnić pojęcia słowa i wyrazu. Słowo to ciąg znaków, odseparowany za pomocą odstępów lub znaków interpunkcyjnych. Wyraz to natomiast jednostka słownikowa języka.

Morfologia zajmuje się morfemami.

Definicja 26 *(Morfem). Morfem jest podstawową i najmniejszą jednostką, będącą elementem konstrukcyjnym wyrazu, który ma znaczenie i/lub funkcję gramatyczną.*

Morfemy są komponentami wyrazów. Przez funkcję gramatyczną rozumie się część mowy. *Część mowy* to pewna klasa wyrazów, wyodrębniona m.in. na podstawie kryteriów morfologicznych.

Definicja 27 *(Leksem). Leksem to jednostka językowa reprezentowana przez formy wyrazowe.*

Definicja 28 *(Analiza morfologiczna). Analiza morfologiczna to zautomatyzowany proces komputerowy nieuwzględniający kontekstu wystąpienia słowa, służący określeniu dla słowa wszystkich form leksemów, dla których stanowi ono wykładnik.*

Analiza morfologiczna pozwala na uzyskanie informacji o formach słów występujących w zdaniu lub wyrażeniu. Najczęściej wykorzystuje się w tym celu programy komputerowe. Jednym z nich jest *Morfeusz Polimorf*, który swoje działanie opiera na użyciu słownika *Polimorf*. Wyniki analizy morfologicznej dla przykładowej frazy, uzyskane za pomocą programu *Morfeusz Polimorf*, przedstawiono w przykładzie (18).

Przykład 18 Dla frazy *niebieska chmura* uruchomiono analizę morfologiczną w programie *Morfeusz Polimorf*. Zrzut ekranu z wynikami analizy przedstawiono na rysunku (18).

Analiza morfologiczna:

O➔	➔O	Forma	Lemat	Tag	Nazwa	Kwalifikatory
0	1	**niebieska**	niebieski	adj:sg:nom.voc:f:pos		
1	2	**chmura**	chmura	subst:sg:nom:f	*nazwa pospolita*	

Rysunek 18: Wyniki analizy morfologicznej w formie zrzutu ekranu

Najistotniejszą część otrzymanych w przykładzie (18) wyników przechowuje kolumna *Tag*, która za pomocą symboli odseparowanych od siebie dwukropkami lub kropkami określa informacje o występującej formie wyrazu. Przykładowo dla wyrazu *chmura* program zwraca wynik w postaci *subst:sg:nom:f*, co oznacza, że wyraz jest rzeczownikiem (subst) występującym w liczbie pojedynczej (sg) w mianowniku (nom) i jest rodzaju żeńskiego (f).

W przykładzie (18) wykorzystanie analizy morfologicznej umożliwiło otrzymanie dość jednoznacznych wyników. Nie zawsze można to osiągnąć. Zadaniem analizy morfologicznej jest zwrócenie wszystkich znalezionych form dla analizowanego wyrazu.

Definicja 29 *(Rodzina wyrazów). Rodzina wyrazów jest grupą wyrazów, które wyróżnia wspólne pochodzenie. Oznacza to, że wyrazy z jednej rodziny wywodzą się od jednego wyrazu podstawowego.*

Rodzina wyrazów to grupa, która zawiera ten sam rdzeń.

Definicja 30 *(Rdzeń). Rdzeń jest głównym morfemem wyrazu decydującym o jego znaczeniu.*

Procesem służącym pozyskaniu rdzenia z wyrazu jest *rdzeniowanie*. Rdzenie są modyfikowane przez afiksy.

Definicja 31 *(Afiks). Afiks jest dodatkiem do rdzenia, który modyfikuje jego znaczenie. Do afiksów należą prefiksy (wstawiane przed rdzeniem), sufiksy (wstawiane za rdzeniem) oraz infiksy (wstawiane wewnątrz rdzenia).*

Definicja 32 *(Forma fleksyjna). Forma fleksyjna jest formą wynikającą z odmiany wyrazu, np. przez przypadki.*

W dużym uproszczeniu analiza morfologiczna może być również traktowana jako pozyskiwanie lematu wyrazu.

Definicja 33 *(Lemat). Lemat jest formą podstawową wyrazu.*

Pozyskiwanie formy podstawowej, czyli lematu wyrazu nosi nazwę *lematyzacji*.

Przykład (19) służy podsumowaniu wszystkich pojęć opisanych dotychczas w niniejszym podrozdziale.

Przykład 19 Jeśli wyrazem podstawowym jest *czytać*, to przykładowa rodzina wyrazów obejmuje wyrazy *przeczytać*, *poczytać*, *czytanka*, *wczytać*, *czytelnictwo*. Rdzeń to *czyt*. Rdzeń jest elementem stałym we wszystkich wyrazach wchodzących w skład rodziny wyrazów. Przykład wyrazu modyfikowanego przez afiks to *przeczytać*, gdzie prefiksem jest *prze-*, a sufiksem *-ać*. Do form fleksyjnych wyrazu *czytać* należy *czytasz*. Lematem wyrazów *czytać* oraz *czytasz* jest wyraz *czytać*.

Z punktu widzenia lingwistyki komputerowej, istotnym procesem jest również tagowanie części mowy (ang. *POS-tagging*), które w przeciwieństwie do analizy morfologicznej uwzględnia kontekst wystąpienia słowa. Kontekst dotyczy rozpatrzenia wzajemnych powiązań słów występujących w zdaniu lub wyrażeniu. Przykład (20) przezentuje przebieg procesu tagowania części mowy.

Przykład 20 Dla zdania *Zabieram je dzisiaj na długą wyprawę* możliwe jest określenie następujących części mowy:

- zabieram - czasownik
- je - zaimek
- dzisiaj - przysłówek
- na - przyimek
- długą - przymiotnik
- wyprawę - rzeczownik

Uwzględniony kontekst pozwala na określenie słowa *je* jako zaimka, a nie czasownika w trzeciej osobie liczby pojedynczej od formy podstawowej

jeść. Analiza morfologiczna zwróciłaby wszystkie możliwe formy, z których jedna byłaby tą właściwą i poprawną. W prawidłowym tagowaniu częścią mowy słowa *je* może być brana pod uwagę osoba czasownika (pierwsza osoba liczby pojedynczej - *ja zabieram*) oraz przypadek występującego po nim wyrazu (*kogo lub co zabieram*).

Komputerowe narzędzia służące do wyznaczania lematów słów nazywa się *lematyzatorami*. *Lematyzator* pozwala na uogólnianie słów i sprowadzanie ich do podstawowej formy lematu. *Tager* to z kolei narzędzie odpowiadające za ujednoznacznienie, podające informację o właściwej formie słowa występującego w określonym kontekście.

3 Architektura systemu tłumaczenia automatycznego Moses

Moses [10] jest systemem statystycznego tłumaczenia automatycznego stworzonym przez Hieu Hoang'a i Philippa Koehn'a. Danymi wejściowymi dla systemu *Moses* są dwujęzyczne korpusy równoległe urównoleglone na poziomie zdań. Oznacza to, że jedno zdanie języka źródłowego jest urównoleglone do jednego zdania języka docelowego. Zasadniczo *Moses* składa się z dwóch komponentów - kanału treningowego i dekodera.

Kanał treningowy (ang. *training pipeline*) to zbiór narzędzi, służących do wytrenowania modelu języka oraz modelu tłumaczenia na podstawie danych wejściowych, czyli korpusów. *Trenowanie* to proces, w którym system na podstawie określonej liczby przykładów (zestawu treningowego) doskonali się, a następnie nabywa nową wiedzę. Jeszcze przed procesem trenowania, korpusy równoległe są oczyszczane - zdania, które są według systemu zbyt długie lub niedopasowane zostają skrócone lub usunięte.

Model języka zapewnia płynność tłumaczenia. *Moses* wykorzystuje zewnętrzne narzędzia do tworzenia modelu języka. Do najpopularniejszych należą *IRST* oraz *KenLM*. Domyślnie *Moses* korzysta z *KenLM*. Opcję wyboru narzędzia do tworzenia modelu języka można modyfikować w pliku konfiguracyjnym systemu *Moses* - *moses.ini*.

Wykorzystywane w systemie *Moses* modele tłumaczenia to modele oparte na tłumaczeniu fraz: modele hierarchiczne oraz składniowe. Wymienione w poprzednim zdaniu modele szczególnie wspierają proces tłumaczenia w radzeniu sobie z nieciągłymi sekwencjami słów. System *Moses* wzbogacono także o tłumaczenie oparte na tzw. *faktorach*. Każde słowo jest reprezentowane przez wektor faktorów. *Faktory* mogą być np. tagami części mowy. Wszelkie dodatkowe informacje, których nośnikami są *faktory*, mogą znacząco wpłynąć na ulepszenie jakości tłumaczenia.

Proces dekodowania wykonywany jest przez *dekoder* (ang. *decoder*). *Dekodowanie* pozwala na znalezienie najlepszego tłumaczenia w języku docelowym dla danego zdania w języku źródłowym. *Dekoder* systemu *Moses* umożliwia użytkownikom wprowadzenie modyfikacji do procesu dekodowa-

nia. Jednym ze sposobów jest dokonanie zmian w modelu tłumaczenia, co można osiągnąć poprzez dodanie cech. Cechy są nośnikami dodatkowych informacji, w które zostaje wyposażony proces tłumaczenia.

W systemie *Moses* dużą rolę odgrywa model *log-liniowy*. Każdy z komponentów wchodzących w skład modelu jest reprezentowany przez jedną lub więcej cech. Cechy te posiadają odpowiednie wagi i są przemnażane przez siebie. Procesem, który służy w systemie *Moses* do dostrajania tych wag, jest *tuning*. Dostrojone wagi są wykorzystywane w procesie dekodowania.

Cechy w statystycznym tłumaczeniu automatycznym mogą być powiązane z prawdopodobieństwami modelu języka lub tagami części mowy. W systemie *Moses* implementacją cech są funkcje cech, które zwracają odpowiedni wynik. Funkcje cech dzielą się na *funkcje bezstanowe* oraz *stanowe*. System *Moses* udostępnia własne szkielety, które ułatwiają implementację funkcji bezstanowych oraz stanowych.

3.1 Funkcje bezstanowe ze szczególnym uwzględnieniem modelu tłumaczenia

Definicja 34 *(Cecha bezstanowa). Cecha bezstanowa jest cechą, która zależy wyłącznie od bieżącego tłumaczenia frazy.*

Przykładem cechy bezstanowej może być kara za słowa (ang. *word penalty*).

Definicja 35 *(Kara za słowa). Kara za słowa jest komponentem modelu log-liniowego oraz przykładem cechy bezstanowej, która zapewnia, że tłumaczenie nie będzie zbyt długie lub zbyt krótkie.*

Kara za słowa bierze pod uwagę wyłącznie stan aktualny - bieżącą frazę. Nie sugeruje się tłumaczeniami wcześniej występujących fraz. Kara za słowa jest bezpośrednio związana z log-liniowym modelem tłumaczenia. W systemie *Moses* jest ona zaimplementowana jako `WordPenalty`.

Implementacją cechy bezstanowej w systemie *Moses* jest funkcja bezstanowa (ang. *stateless feature function*).

Definicja 36 *(Funkcja bezstanowa). Funkcja bezstanowa jest funkcją zawierającą implementację dla cechy bezstanowej w środowisku systemu Moses.*

Logikę dotyczącą cech bezstanowych można zatem łatwo przenieść na logikę funkcji bezstanowych. Funkcje bezstanowe reprezentują cechy bezstanowe, więc korzystają wyłącznie z informacji o bieżącym tłumaczeniu.

W systemie *Moses* cechy są reprezentowane jako klasy, a funkcje cech jako metody. Implementacji dokonuje się w języku C++. Do definiowania cechy bezstanowej wykorzystuje się dwa pliki. Jeden z nich (o rozszerzeniu *.h) jest plikiem nagłówkowym, zawierającym deklaracje zmiennych, klasy (czyli cechy) i metod. Drugi plik (o rozszerzeniu *.cpp) zawiera właściwe i w pełni rozwinięte implementacje metod zadeklarowanych w pliku nagłówkowym.

Klasa `StatelessFeatureFunction` stanowi fundament dla budowania cech bezstanowych. Nowo wprowadzana cecha bezstanowa będzie reprezentowana przez klasę dziedziczącą z `StatelessFeatureFunction`. W oknie kodu (3) zaprezentowano szkielet klasy dla cechy bezstanowej.

```
1 (...)
2 class SkeletonStatelessFF : public StatelessFeatureFunction
3   {
4     public:
5     SkeletonStatelessFF(const std::string &line);
6     (...)
7   }
8 (...)
```

Kod 3: Szkielet klasy dla cechy bezstanowej (na podstawie [11])

Deklarację należy umieścić w pliku o rozszerzeniu *.h. Nazwa klasy `SkeletonStatelessFF` powinna zostać zastąpiona nazwą adekwatną do nazwy wprowadzanej cechy. Zaprezentowany szkielet uwzględnia konstruktor. W pliku o rozszerzeniu *.cpp, gdzie znajduje się implementacja, konstruktor powinien wywoływać funkcję `StatelessFeatureFunction(...)` oraz dedykowaną metodę `ReadParameters()`. Metoda `ReadParameters()` jest dziedziczona z klasy `FeatureFunction` i nie powinna zostać nadpisana w procesie własnej implementacji. W oknie kodu (4) zaprezentowano szkielet implementacji dla cechy beztsanowej.

```
(...)
SkeletonStatelessFF::SkeletonStatelessFF(const std::string &line)
:StatelessFeatureFunction(2, line)
  {
    ReadParameters();
  }
(...)
```

Kod 4: Szkielet implementacji dla cechy bezstanowej w pliku *.cpp (na podstawie [11])

Deklaracja klasy cechy bezstanowej powinna uwzględniać metodę

`IsUseable`.

Należy pozostawić wartość `true`, kiedy faktory nie są wykorzystywane. W pliku nagłówkowym powinny się znaleźć także deklaracje dla metod

`EvaluateInIsolation`,
`EvaluateWithSourceContext`,
`EvaluateWhenApplied`.

Metody należą do klasy `FeatureFunction`. Ich definicje przedstawiono w oknie kodu (5).

```
void EvaluateInIsolation(const Phrase &source
, const TargetPhrase &targetPhrase
, ScoreComponentCollection &scoreBreakdown
, ScoreComponentCollection &estimatedScores) const;

void EvaluateWithSourceContext(const InputType &input
, const InputPath &inputPath
, const TargetPhrase &targetPhrase
, const StackVec *stackVec
, ScoreComponentCollection &scoreBreakdown
, ScoreComponentCollection *estimatedScores = NULL) const;

void EvaluateWhenApplied(const Hypothesis& hypo,
ScoreComponentCollection* accumulator) const;

void EvaluateWhenApplied(const ChartHypothesis &hypo,
ScoreComponentCollection* accumulator) const;
```

Kod 5: Deklaracje metod (na podstawie [11])

Tworząc implementacje w pliku o rozszerzeniu *.cpp powinno się nadpisywać metody z kodu (5). Wprowadzane modyfikacje dotyczą głównie zmian punktowania cechy. W procesie *tuningu* określana jest ważność cechy i na tej podstawie nadaje się jej odpowiednią wagę. Ważone wyniki otrzymane w obrębie cech są wykorzystywane dalej w procesie dekodowania. Funkcje cech powinny zatem zwracać wagi, a zmiana punktowania dotyczy zmiany wagi danej cechy.

Każda z metod działa nieco inaczej i jest wywoływana w innym momencie. W większej części przypadków nadpisaniu powinna ulec pierwsza z metod, czyli `EvaluateInIsolation`. Wyniki ewaluowane za jej pomocą są wykorzystywane w szacowaniu przyszłego kosztu w modelu opartym na tłumaczeniu fraz. Wywołuje się ją jeszcze przed procesem wyszukiwania, w momencie tworzenia reguły tłumaczenia. Metodę `EvaluateWithSourceContext` nadpisuje się w przypadku, gdy funkcje bezstanowe wymagają znajomości całego zdania, by móc dokonać oceny. Jej wywołanie odbywa się przed rozpoczęciem wyszukiwania. Obydwie metody `EvaluateWhenApplied` wykorzystuje się, gdy funkcja bezstanowa wymaga segmentacji źródła lub jakichkolwiek informacji dotyczących kontekstu wystąpienia. Wywołuje się je podczas wyszukiwania.

Przykładowa implementacja dla metody `EvaluateInIsolation` została zaprezentowana w oknie kodu (6).

```
void SkeletonStatelessFF::EvaluateInIsolation(const Phrase &source
, const TargetPhrase &targetPhrase
, ScoreComponentCollection &scoreBreakdown
, ScoreComponentCollection &estimatedScores) const
  {
    // dense scores
    vector<float> newScores(m_numScoreComponents);
    newScores[0] = 1.5;
    newScores[1] = 0.3;
    scoreBreakdown.PlusEquals(this, newScores);

    // sparse scores
    scoreBreakdown.PlusEquals(this, "sparse-name", 2.4);
  }
```

Kod 6: Przykładowa implementacja dla metody EvaluateInIsolation (na podstawie [11])

Przykładowa implementacja przedstawiona w oknie kodu (6) pokazuje uaktualnianie wartości liczbowych (sposobu punktowania) dla cechy bezstanowej.

3.2 Funkcje stanowe ze szczególnym uwzględnieniem modelu języka

Definicja 37 *(Cecha stanowa). Cecha stanowa jest cechą, która zależy od wcześniejszych decyzji podjętych względem tłumaczenia.*

Specyficznym przykładem cechy stanowej jest model języka, który opiera swoje działanie na n-gramach, co wiąże się z potrzebą znajomości n wcześniejszych słów.

Cechy stanowe są implementowane w systemie *Moses* w postaci funkcji stanowych (ang. *stateful feature functions*).

Definicja 38 *(Funkcja stanowa). Funkcja stanowa jest funkcją zawierającą implementację dla cechy stanowej w środowisku systemu Moses.*

W odniesieniu do cech stanowych, funkcje stanowe korzystają z informacji o wcześniejszych decyzjach podjętych względem tłumaczenia. Proces budowania cechy stanowej wygląda tak samo, jak w przypadku cechy bezstanowej. W pliku o rozszerzeniu *.h powinna zostać zapisana deklaracja klasy stanowej. Klasa `StatefulFeatureFunction` stanowi fundament dla budowania cech stanowych. Nowo wprowadzona cecha będzie z niej dziedziczyć. W oknie kodu (7) zaprezentowano szkielet klasy dla cechy stanowej.

```
(...)
class SkeletonStatefulFF : public StatefulFeatureFunction
  {
    public:
    SkeletonStatefulFF(const std::string &line);
    (...)
  }
(...)
```

Kod 7: Szkielet klasy dla cechy stanowej (na podstawie [11])

Implementacja w pliku o rozszerzeniu *.cpp powinna uwzględniać wywołanie metody `StatefulFeatureFunction(...)` oraz `ReadParameters()`. Podobnie, jak w przypadku funkcji bezstanowych, metoda ta nie może zostać nadpisana w procesie implementacji. W oknie kodu (8) zaprezentowano szkielet implementacji dla cechy stanowej.

```
(...)
SkeletonStatefulFF::SkeletonStatefulFF(const std::string &line)
:StatefulFeatureFunction(3, line)
  {
    ReadParameters();
  }
(...)
```

Kod 8: Szkielet implementacji dla cechy stanowej w pliku *.cpp (na podstawie [11])

W pliku nagłówkowym powinny znaleźć się deklaracje metody `isUseable` oraz dwóch metod o nazwie `EvaluateWhenApplied`. Definicje metod przedstawiono w oknie kodu (9).

```
FFState* EvaluateWhenApplied(
const Hypothesis& cur_hypo,
const FFState* prev_state,
ScoreComponentCollection* accumulator) const;

FFState* EvaluateWhenApplied(
const ChartHypothesis& /* cur_hypo */,
int /* featureID - used to index the state in the previous hypotheses */,
ScoreComponentCollection* accumulator) const;
```

Kod 9: Deklaracje metod (na podstawie [11])

Metody o tej samej nazwie pojawiły się także w podrozdziale [3.1] o funkcjach bezstanowych. Główną różnicą jest dodatkowe zwrócenie stanu poprzedniego dla funkcji stanowych. Kierowanie się wcześniejszymi decyzjami podjętymi względem tłumaczenia wymusza obecność wskaźnika do stanu poprzedniego. Pierwsza z przedstawionych metod `EvaluateWhenApplied` dotyczy modelu wykorzystywanego w standardowym tłumaczeniu opierającym się na frazach, a druga - modelu opierającym się na tłumaczeniu hierarchicznym i składniowym.

Implementacja modelu języka w systemie Moses nadpisuje przede wszystkim metodę `EvaluateInIsolation`. Metoda `EvaluateInIsolation` została omówiona szerzej w podrozdziale [3.1] o funkcjach bezstanowych. Tutaj służy do oceny reguł tłumaczenia i jest zaimplementowana w klasie `LanguageModel`, co bezpośrednio wskazuje na wykorzystywaną cechę, czyli model języka. W oknie kodu (10) przedstawiono przykładową implementację dla metody `EvaluateInIsolation`.

```
FFState* SkeletonStatefulFF::EvaluateWhenApplied(
const Hypothesis& cur_hypo,
const FFState* prev_state,
ScoreComponentCollection* accumulator) const
  {
    // dense scores
    vector<float> newScores(m_numScoreComponents);
    newScores[0] = 1.5;
    newScores[1] = 0.3;
    newScores[2] = 0.4;
    accumulator->PlusEquals(this, newScores);
    // sparse scores
    accumulator->PlusEquals(this, "sparse-name", 2.4);
```

```
14
15      return new SkeletonState(0);
16    }
```

Kod 10: Przykładowa implementacja dla metody EvaluateWhenApplied (na podstawie [11])

Podobnie jak w przypadku cech bezstanowych, także tutaj punktowanie dla cechy zostaje uaktualnione za pomocą wartości liczbowych. Dodatkowo zwracany jest także stan.

3.3 Inne cechy wykorzystywane w systemie

Zadaniem cech wykorzystywanych w systemie *Moses* jest wpłynięcie na wydajność pracy oraz ulepszenie jakości tłumaczenia. W podrozdziale wyszczególniono kilka istotnych cech, które zostały wprowadzone do systemu *Moses*. Skorzystanie z nich w systemie wymaga wprowadzenia odpowiedniej konfiguracji do pliku *moses.ini*. W sekcji *[feature]* znajdują się deklaracje wszystkich funkcji cech.

System tłumaczenia automatycznego *Moses* korzysta z tabel fraz. Tabele zawierają pary fraz, dzięki którym możliwe jest obliczenie prawdopodobieństwa tłumaczenia. Tabele fraz w systemie *Moses* mogą przybierać różną postać. Wyróżnia się następujące typy tabel fraz:

- `PhraseDictionaryMemory`
- `PhraseDictionaryCompact`
- `PhraseDictionaryOnDisk`

`PhraseDictionaryMemory` wczytuje tabelę fraz bezpośrednio do pamięci. Chociaż ten typ tabeli działa dość szybko, to zużywa dużo pamięci RAM. `PhraseDictionaryCompact` wykorzystuje specjalne kodowanie, które nie powoduje znaczących strat danych i pozwala na zmniejszenie rozmiaru tabel fraz. `PhraseDictionaryOnDisk` to implementacja binarnych tabel fraz, które przekształcono do bazy danych. Wyłącznie część tabel fraz niezbędna do przetłumaczenia zdania języka wejściowego na zdanie języka docelowego jest ładowana do pamięci.

W modelach opartych na tłumaczeniu fraz proces dekodowania pozwala na generowanie zdania od lewej do prawej strony poprzez dodawanie nowo przetłumaczonych fraz na końcu, po tłumaczeniach pośrednich. Na tej podstawie powstają wykresy, które składają się z reguł składniowych. Ich komponentami są drzewa składniowe. `PhraseDictionaryOnDisk` umożliwia dekodowanie wykresów.

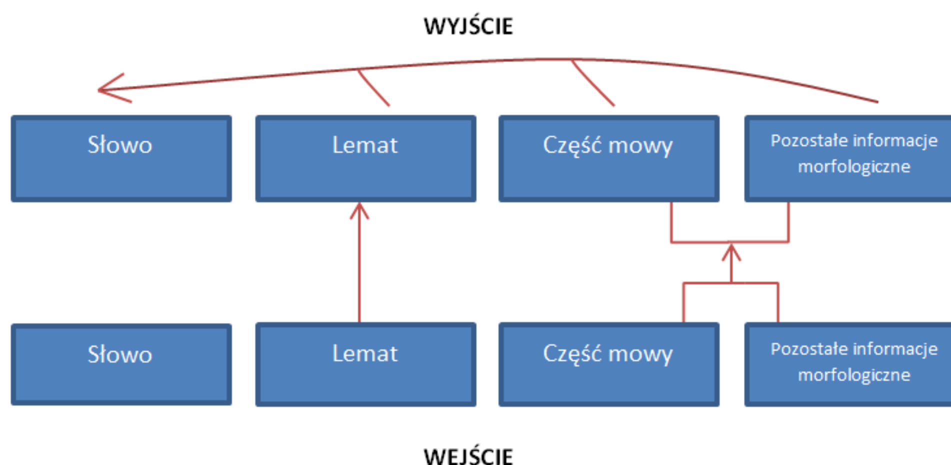
Bezpośrednio na proces tłumaczenia mogą wpływać funkcje leksykalne odnoszące się do tłumaczenia słów.

Cecha `WordTranslationFeature` wskazuje, czy określone słowo w języku źródłowym zostało przetłumaczone na określone słowo w języku docelowym. Cecha `TargetWordInsertionFeature` informuje, czy dane słowo w języku docelowym nie zostaje urównoleglone do żadnego słowa w języku źródłowym. Cecha `SourceWordDeletionFeature` sprawdza, czy określone słowo w języku źródłowym nie zostaje urównoleglone do żadnego słowa w języku docelowym (nie posiada punktu urównoleglenia).

Bardzo ważna z punktu widzenia pracy magisterskiej jest możliwość stosowania informacji morfologicznych, takich jak tagi części mowy, czy lematy słów. Te dodatkowe informacje są implementowane w systemie*Moses* w postaci faktorów. *Faktor* jest napisem reprezentowanym przez klasę `Factor`. Faktory są przypisywane na poziomie słów i zazwyczaj nie występują w postaci pojedynczych wartości, lecz wektorów wartości.

W korpusie słowa są traktowane jako tokeny, czyli zupełnie odrębne wyrazy. Oznacza to, że przykładowo słowa *cat* oraz *cats* będą dla modelu tłumaczenia zupełnie niezależne. W językach bogatych morfologicznie, takich jak język polski, może to stanowić dość duży problem.

W systemie *Moses* zaimplementowano modele tłumaczenia, opierające swoje działanie na faktorach. Nazywa się je faktorowymi modelami tłumaczenia. Zastosowanie takich modeli pozwala na odrębne tłumaczenie lematów, części mowy oraz innych informacji morfologicznych i odtwarzanie na ich podstawie właściwych form słów. Rysunek (19) przedstawia w uproszczeniu działanie faktorowego modelu tłumaczenia.



Rysunek 19: Wykres prezentujący działanie faktorowego modelu tłumaczenia

Zarówno lemat, jak i część mowy oraz pozostałe informacje morfologiczne są tłumaczone niezależnie. Na podstawie tych tłumaczeń udaje się uzyskać formę słowa w języku docelowym.

Mapowanie w faktorowych modelach tłumaczenia zostało podzielone na trzy etapy następujące po sobie - tłumaczenie lematów wejściowych na lematy wyjściowe, tłumaczenie faktorów reprezentujących tagi części mowy oraz inne informacje morfologiczne i generowanie właściwych form wyrazowych na podstawie wcześniej przetłumaczonych lematów i faktorów. Proces mapowania pozwala na wygenerowanie słów wyjściowych będących odpowiednikami słów wejściowych przy uwzględnieniu reprezentacji faktorowej.

Faktorowe modele tłumaczenia to dodatkowo udoskonalone modele oparte na tłumaczeniu fraz. Tutaj również zdania wejściowe są dzielone na frazy, czyli ciągłe sekwencje słów, ale proces tłumaczenia składa się z wyszczególnionych kroków. Przykład (21) prezentuje działanie faktorowego modelu tłumaczenia.

Przykład 21 Angielska fraza składająca się z jednego słowa *tables* posiada reprezentację, w której właściwą formą słowa jest *tables*, lematem jest *table*, a częścią mowy jest rzeczownik (oznaczenie: NN). Dodatkowe informacje morfologiczne dla słowa to liczba mnoga, rodzaj nijaki oraz przypadek mianownik. Notacja dla tego przykładu stosowana w systemie *Moses* mogłaby

wyglądać nastepująco: surface-form tables | lemma table | part-of-speech NN | count plural | case nominative | gender neutral. Znak | jest traktowany jako separator, oddzielający kolejno podawane informacje.

Pierwszy etap procesu tłumaczenia to tłumaczenie lematów.

table -> stół, stolik, tabela, tablica

Drugi etap procesu tłumaczenia to tłumaczenie informacji morfologicznych.

NN | plural-nominative-neutral -> NN | plural, NN | singular

Zapis odnosi się do formy *tables*. Słowo *tables* jest rzeczownikiem (NN) występującym w liczbie mnogiej (plural), w mianowniku (nominative) i będącym rodzaju nijakiego (neutral). Stosowanie mapowania dla frazy wejściowej jest nazywane rozszerzaniem (ang. *expansion*). Dla każdego kroku mapowania istnieje wiele możliwości tłumaczenia. Każda fraza wejściowa może być rozszerzana do listy dostępnych tłumaczeń. W pewnym sensie rozszerzanie odzwierciedla niejednoznaczność tłumaczenia. Stosowanie rozszerzania pozwala na uzyskanie w przykładzie (21) dwóch informacji morfologicznych, mówiących o występowaniu rzeczownika (NN) o podanym przypadku i rodzaju w liczbie mnogiej (plural) oraz pojedynczej (singular).

Ostatnim etapem jest określanie właściwych form słów na podstawie informacji wyznaczonych w wyniku przejścia przez dwa poprzednie kroki mapowania.

table | NN | singular -> stół
table | NN | plural -> stoły
table | NN | singular -> stolik
table | NN | plural -> stoliki
table | NN | singular -> tabela
table | NN | plural -> tabele
table | NN | singular -> tablica
table | NN | plural -> tablice

Bezpośrednie tłumaczenie skomplikowanych wejściowych reprezentacji faktorowych na reprezentacje wyjściowe może przyczynić się do wystąpienia problemu rzadkości danych. Wprowadzenie mapowania ogranicza jego skalę. Pozwala na pozyskanie szerokiej listy tłumaczeń, które mogą zostać wykorzystane w procesie tłumaczenia.

Standardowe modele tłumaczenia oparte na tłumaczeniu fraz nie wykorzystują informacji lingwistycznych, takich jak informacje morfologiczne, czy składniowe. Wprowadzanie takich informacji do procesu tłumaczenia pozwala na wykorzystanie bardziej ogólnych form słów, czyli lematów i uzyskiwanie na tej podstawie odmian, co prowadzi do uzupełnienia luk występujących w zbiorach danych, czyli ograniczenia problemu rzadkości danych. Informacje lingwistyczne wspomagają proces tłumaczenia, pozwalając na modelowanie składniowych i morfologicznych aspektów tłumaczenia, co pozytywnie wpływa na jego jakość.

4 Vowpal Wabbit jako system uczący przystosowany do pracy z dużą ilością cech

Vowpal Wabbit to projekt obecnie rozwijany przez Microsoft Research będący zaawansowanym systemem uczącym. *Vowpal Wabbit* jest uruchamiany z poziomu konsoli za pomocą komend wpisywanych do wiersza poleceń.

W programie *Vowpal Wabbit*, na podstawie przygotowanego zestawu treningowego, możliwe jest wytrenowanie modelu, który będzie służył do testowania na zestawie testowym. Przed rozpoczęciem trenowania dane powinny zostać przedstawione w odpowiednim formacie wejściowym, którego schemat jest następujący:

[Etykieta] [Ważność] [Podstawa] [Tag]|Przestrzeń_ nazw Cechy ...

Vowpal Wabbit umożliwia przeprowadzenie procesu klasyfikacji, na co bezpośrednio wskazuje preferowany format wejściowy. *Etykieta* reprezentuje przewidywaną wartość liczbową. W procesie klasyfikacji będzie ona klasą, do której przypisuje się obiekty na podstawie określonych dla nich cech. *Ważność* jest liczbą nieujemną wskazującą na znaczenie przykładu względem pozostałych. *Podstawa* to dodatkowy parametr regresji, domyślnie przyjmujący wartość 0. *Tag* służy do identyfikowania przykładu, ale nie musi być wartością unikalną. Domyślnie jest pustym napisem. *Przestrzeń nazw* to z kolei identyfikator źródła danych. *Cechy* są reprezentowane jako *nazwa_ cechy:wartość_ cechy*, gdzie *nazwa_ cechy* to napis, a *wartość_ cechy* to wartość liczbowa cechy. *Etykieta* oraz *Cechy* są podstawowymi komponentami formatu wejściowego. Pozostałe elementy schematu mogą zostać pominięte.

Przykład 22 Tabela (5) przedstawia zestaw danych składający się z trzech przykładów. Dane pochodzą z popularnego zbioru danych dotyczącego irysów. Postawionym celem jest przewidywanie gatunku irysa dla nowego przykładu.

Gatunek	Dł. działki kielicha	Szer. działki kielicha	Dł. płatka	Szer. płatka
Iris setosa	5.1	3.5	1.4	0.2
Iris versicolor	5.9	3.2	4.8	1.8
Iris virginica	6.9	3.2	5.7	2.3

Tabela 5: Trzy przykłady treningowe z zestawu danych dotyczącego irysów

Dane z tabeli (5) przedstawione w formacie wejściowym dla systemu *Vowpal Wabbit* mają postać zapisaną poniżej.

1 | f0:5.1 f1:3.5 f2:1.4 f3:0.2
2 | f0:5.9 f1:3.2 f2:4.8 f3:1.8
3 | f0:6.9 f1:3.2 f2:5.7 f3:2.3

Etykiety 1, 2, 3 oznaczają klasy, do których przyporządkowano przykłady treningowe. Każdy z irysów należy do innego gatunku. Każdy przykład wyróżniają także cztery cechy, którymi są długość i szerokość działki kielicha oraz długość i szerokość płatka. Każda z cech ma przypisaną wartość liczbową.

Uruchomienie narzędzia *Vowpal Wabbit* jest możliwe dzięki poleceniu `vw`. Polecenie może zostać wzbogacone o dodatkowe opcje, w zależności od preferowanego zastosowania. Na podstawie danych wejściowych, program przebiega przykład po przykładzie. Wynikiem działania procesu trenowania jest utworzony model. Model służy do testowania na zestawie danych testowych, co skutkuje otrzymaniem predykcji dla nieznanych programowi danych na podstawie wiedzy wyuczonej w procesie trenowania. Predykcje są przyporządkowaniami przykładów testowych do odpowiednich klas.

Vowpal Wabbit jest doceniany za bogate wyposażenie w różnorodne opcje, nie tylko związane z problemami uczenia maszynowego, oraz szybkość działania. Na szybkość działania oraz wydajność programu wpływają przede wszystkim zastosowane algorytmy.

4.1 Ogólna charakterystyka działania oraz wykorzystywanych algorytmów

Vowpal Wabbit jest narzędziem konsolowym, które można uruchomić poprzez wpisanie odpowiedniego polecenia do linii komend. Podstawowym komponentem polecenia jest napis `vw` modyfikowany przez dostępne opcje.

Wykorzystanie narzędzia *Vowpal Wabbit* można ograniczyć do trzech podstawowych kroków - przedstawienia danych w formacie wejściowym, uruchomienia procesu trenowania na zbiorze treningowym oraz uruchomienia procesu testowania na zbiorze testowym.

Zestaw danych należy przetworzyć w taki sposób, by format wejściowy przykładów wchodzących w jego skład stał się akceptowalny dla programu *Vowpal Wabbit*. Często zestaw danych jest dużych rozmiarów. Należy zadbać o automatyzację procesu przetwarzania. Można tego dokonać poprzez napisanie programu w wybranym języku programowania. W przetwarzaniu tekstu szczególnie dobrze sprawdzają się języki skryptowe, takie jak *Python*, czy *Perl*.

Dysponując przykładami w określonym formacie wejściowym, zestaw danych można podzielić na zestaw treningowy oraz zestaw testowy. Proces trenowania powinien odbywać się na odpowiednio dużym zestawie treningowym, by efekty działania były zadowalające. Można wykorzystać proporcje 80:20 lub 90:10, które oznaczają, że odpowiednio 80% lub 90% zestawu danych będzie budować zbiór treningowy oraz 20% lub 10% będzie wchodziło w skład zestawu testowego.

Uruchomienie procesu trenowania w narzędziu *Vowpal Wabbit* sprowadza się do użycia polecenia

```
vw train_set
```

Argument `train_set` określa zestaw treningowy i jest jednocześnie nazwą pliku. Wynikiem procesu trenowania jest utworzenie modelu. Pomocnym krokiem będzie zapisanie modelu do pliku o przykładowej nazwie `train.model`, co pozwoli na późniejsze wykorzystanie modelu w procesie testowania. Polecenie służące do zapisu modelu to

```
vw train_set -f train.model
```

Z pomocą wytrenowanego modelu można przeprowadzić proces testowania na zestawie testowym. Proces testowania pozwala na uzyskanie predykcji, czyli przewidywanych wartości klas. Polecenie służące do uzyskania predykcji to

```
vw -i train.model -t test_set -p predicts
```

Argument `train.model` oznacza wytrenowany model, a `test_set` oznacza zestaw testowy. Opcja `-t` służy do włączenia wbudowanego trybu testowego. Opcja `-p` pozwala na uzyskanie predykcji i zapisanie ich do pliku `predicts`.

Po uzyskaniu predykcji w procesie testowania, można dokonać oceny otrzymanych wyników. W tym celu wykorzystuje się miary ewaluacji, np. dokładność. Jeśli wyniki nie są satysfakcjonujące, można postarać się o ulepszenie modelu, a następnie przeprowadzić ponowne testowanie i ewaluację.

Vowpal Wabbit oferuje możliwość skorzystania z wielu wariantów algorytmicznych. Algorytmem domyślnym dla narzędzia *Vowpal Wabbit* jest *Stochastic Gradient Descent*. Opcję `--sgd`, odpowiadającą za wykorzystanie algorytmu, można dodatkowo wprowadzić do polecenia konsolowego.

Zamiast *Stochastic Gradient Descent*, można użyć *Mini-Batch Gradient Descent*. Rozmiar batcha określa się jako argument `arg` opcji `minibatch arg`. Argument powinien być liczbą całkowitą.

Stochastic Gradient Descent oraz *Mini-Batch Gradient Descent* są dwoma z wielu ciekawych wariantów algorytmicznych. Oprócz opcji algorytmicznych, istotną rolę pełnią opcje klasyfikacji. Jedną z nich jest opcja `--oaa`, umożliwiająca obsługę modelu *jeden przeciwko wszystkim*.

Opcja `--oaa arg` dodana do polecenia konsolowego pozwala na użycie modelu klasyfikacji *jeden przeciwko wszystkim*. Argument `arg` określa ilość klasyfikatorów binarnych potrzebnych do przeprowadzenia procesu. Polecenie służące do trenowania z wykorzystaniem modelu *jeden przeciwko wszystkim* to

```
vw --oaa arg train_set -f train.model
```

Argument `arg` określa ilość klasyfikatorów binarnych i jest liczbą całkowitą. Argument `train_set` oznacza zestaw treningowy, a argument `train.model` wraz z opcją `-f` pozwala na zapisanie modelu do pliku.

Przykład 23 Zestaw danych dotyczący gatunków irysów Iris-versicolor, Iris-setosa oraz Iris-virginica składa się ze 150 przykładów. Przykłady te są zapisane w pliku w następującym formacie:

7.0,3.2,4.7,1.4,Iris-versicolor
4.7,3.2,1.6,0.2,Iris-setosa
6.4,2.9,4.3,1.3,Iris-versicolor
(...)

Pierwsze cztery wartości liczbowe odseparowane przecinkami są wartościami cech. Ostatnią wartością jest klasa reprezentowana przez napis.

Dane można przetworzyć za pomocą skryptu, by uzyskać format odpowiedni dla programu *Vowpal Wabbit*. Klasy powinny mieć reprezentację numeryczną.

2 | f0:7.0 f1:3.2 f2:4.7 f3:1.4
1 | f0:4.7 f1:3.2 f2:1.6 f3:0.2
2 | f0:6.4 f1:2.9 f2:4.3 f3:1.3
(...)

Iris-versicolor został oznaczony jako klasa 1, *Iris-setosa* jako klasa 2, a *Iris-virginica* jako klasa 3.

Zestaw danych został podzielony na zestaw treningowy, składający się ze 120 przykładów oraz zestaw testowy, składający się z 30 przykładów. Trening uruchomiono za pomocą polecenia

```
~/vowpal_wabbit/vowpalwabbit/vw iris.train.vw --oaa 3 \
                                -f iris.model.train
```

Dodanie opcji `--oaa` z argumentem 3 pozwala na wykorzystanie modelu *jeden przeciwko wszystkim* dla trzech klas. Argument `iris.train.vw` określa

plik z zestawem danych, służącym do trenowania, a argument `iris.model.train` wraz z opcją `-f` służy do zapisania wytrenowanego modelu do pliku. Poniższe polecenie umożliwiło uzyskanie predykcji.

```
~/vowpal_wabbit/vowpalwabbit/vw -i iris.model.train -t \
                              iris.test.vw -p predicts
```

W pliku `predicts` zapisane zostały predykcje dla poszczególnych przykładów. W tabeli (6) przedstawiono predykcje programu *Vowpal Wabbit* dla pięciu przykładów z zestawu testowego.

Predykcja	Przykład z zestawu testowego
1.000000	1 | f0:4.6 f1:3.2 f2:1.4 f3:0.2
3.000000	2 | f0:6.3 f1:3.3 f2:4.7 f3:1.6
1.000000	1 | f0:5.0 f1:3.5 f2:1.6 f3:0.6
1.000000	1 | f0:4.7 f1:3.2 f2:1.3 f3:0.2
3.000000	3 | f0:6.3 f1:2.5 f2:5.0 f3:1.9

Tabela 6: Porównanie predykcji programu Vowpal Wabbit z przykładami z zestawu testowego

Vowpal Wabbit prawidłowo przyporządkował do klas przykłady znajdujące się w pierwszym, trzecim, czwartym oraz piątym wierszu tabeli (6). Dokładność osiągnięta na całym zestawie testowym, składającym się z trzydziestu przykładów, wyniosła 60%.

Oprócz standardowej wersji modelu jeden przeciwko wszystkim, *Vowpal Wabbit* oferuje także opcję `--csoaa`. Opcja służy do wykorzystania modelu jeden przeciwko wszystkim wrażliwego na koszt (ang. *Cost Sensitive One Against All, CSOAA*). Przykład (24) prezentuje działanie modelu *CSOAA*.

Przykład 24 Poniżej zapisano trzy przykłady sprowadzone do formatu wejściowego akceptowalnego przez program Vowpal Wabbit.

1:1 2:0 3:1 4:1 | a b c

1:1 2:1 3:1 4:0 | d c b

1:0 2:1 3:1 4:1 | a e b

Oznaczenia literowe reprezentują cechy. Cechy mogą być zapisane w tradycyjny sposób. Możliwe jest stosowanie opcjonalnych wag, cech numerycznych oraz przestrzeni nazw. Na wieloklasowość wskazują cztery występujące klasy - 1, 2, 3 oraz 4. Każdą etykietę powiązano z kosztem, będącym wartością odwrotną do wagi. Przykładowo, dla pierwszej, trzeciej i czwartej etykiety w pierwszym przykładzie koszt wynosi 1, a dla drugiej etykiety 0. Przy wykorzystaniu modelu *CSOAA*, klasyfikacja sprowadzana jest do regresji logistycznej. W tym przypadku model konstruuje cztery problemy regresji logistycznej na podstawie danych wejściowych. Poniżej przedstawiono rozpatrywane problemy regresji dla przykładu pierwszego.

1 | 1_a 1_b 1_c
0 | 2_a 2_b 2_c
1 | 3_a 3_b 3_c
1 | 4_a 4_b 4_c

Podsumowując, w modelu *CSOAA* każdy przykład treningowy może zostać opatrzony wieloma etykietami reprezentującymi klasy. Każda etykieta jest powiązana z kosztem wyrażonym za pomocą wartości liczbowej. Klasyfikację sprowadza się do problemów regresji logistycznej, dzięki której finalnie, na etapie testowania, możliwy jest wybór klasy z najniższym kosztem.

4.2 Cechy zależne od klasy

W omówieniu modelu *CSOAA* w podrozdziale [4.1] wyraźnie podkreślono, że problem klasyfikacji wieloklasowej można sprowadzić do kilku równorzędnych problemów regresji logistycznej. Zupełnie innym podejściem jest wytrenowanie dla każdej z wyróżnionych klas odrębnych klasyfikatorów. Każdy z nich ma wówczas swoją własną przestrzeń cech.

Wykorzystanie cech zależnych od klasy (ang. *label dependent features*) pozwala na dzielenie pewnych parametrów pomiędzy klasami.

Przykład 25 Załóżmy, że zbiór danych dotyczy książek i czasopism, papierowych oraz elektronicznych. Cztery klasy możliwe do wyróżnienia i wyrażone w sposób słowny to książki papierowe, książki elektroniczne, czasopisma

papierowe oraz czasopisma elektroniczne. Klasy można oznaczyć w sposób numeryczny, po uwzględnieniu powyższej kolejności, jako 1, 2, 3, 4. Jakakolwiek podstawowa cecha przykładu, oznaczona tutaj jako f lub g, może zostać sprowadzona do trzech cech określających dany obiekt. Poniżej przedstawiono dwa przykłady w formacie wejściowym.

1:1 | Paper_f Book_f PaperBook_f
2:0 | Electronic_f Book_f ElectronicBook_f
3:1 | Paper_f Magazine_f PaperMagazine_f
4:1 | Electronic_f Magazine_f ElectronicMagazine_f

1:1 | Paper_g Book_g PaperBook_g
2:1 | Electronic_g Book_g ElectronicBook_g
3:1 | Paper_g Magazine_g PaperMagazine_g
4:0 | Electronic_g Magazine_g ElectronicMagazine_g

Wykorzystany format wejściowy to format wieloliniowy, w którym każda klasa danego przykładu jest uwzględniana w oddzielnej linii. Linie puste służą odseparowaniu przykładów od siebie. W każdej linii znajdują się etykieta wraz z przypisanym kosztem oraz wyróżnione cechy.

Polecenie służące do uruchomienia procesu trenowania z wykorzystaniem cech zależnych od klasy to

```
vw --csoaa_ldf multiline < file
```

Argument `file` określa plik z danymi wejściowymi. Opcja `csoaa_ldf` pozwala na wykorzystanie cech zależnych od klasy.

Przykłady są przekształcane do problemów regresji. Do uzyskania wyników wykorzystuje się przypisane koszty.

Cechy z jednej przestrzeni mogą być współdzielone przez inne przykłady z zestawu, co można zapisać w następujący sposób:

1:1 | g Paper_f Book_f PaperBook_f
2:0 | g Electronic_f Book_f ElectronicBook_f

3:1 | g Paper_f Magazine_f PaperMagazine_f
4:1 | g Electronic_f Magazine_f ElectronicMagazine_f

Uproszczony zapis wygląda tak, jak przedstawiono poniżej.

shared | g
1:1 | Paper_f Book_f PaperBook_f
2:0 | Electronic_f Book_f ElectronicBook_f
3:1 | Paper_f Magazine_f PaperMagazine_f
4:1 | Electronic_f Magazine_f ElectronicMagazine_f

Zapis pozwala na przekazanie programowi *Vowpal Wabbit* informacji o tym, że cecha g jest współdzielona przez wszystkie linie reprezentujące klasy dla przykładu.

Zamiast konstruowania problemów regresji, model $CSOAA$ może działać wraz z argumentem mc pozwalającym na wykorzystanie wielu klasyfikatorów zamiast modeli regresji. Jedyną wystepującą różnicą w zapisie jest pozbycie się kosztów. Pozostawia się wyłącznie etykiety reprezentujące klasy.

5 Morfologia w statystycznym tłumaczeniu automatycznym

Dysponowanie informacjami morfologicznymi może okazać się przydatne i wpłynąć na jakość tłumaczenia.

5.1 Przykłady w kontekście innych języków słowiańskich

Do grupy języków bogatych morfologicznie należy przede wszystkim grupa języków słowiańskich. Języki te wyróżnia bogactwo form fleksyjnych. Oprócz języka polskiego, do grupy tej należy włączyć także m.in. języki rosyjski, czeski oraz słowacki.

Głównie z myślą o językach bogatych morfologicznie, opracowano metodę służącą do przewidywania odmienionych form wyrazowych. Według twórców artykułu [4] metoda ma istotne znaczenie, ponieważ ogranicza braki w tłumaczeniu z języka źródłowego na język docelowy, co bezpośrednio przekłada się na ograniczenie problemu rzadkości danych i przyczynia się do ulepszenia jakości tłumaczenia. Informacje morfologiczne wraz z wytrenowanym modelem służą do przewidywania odmienionych form wyrazowych. Autorzy artykułu [4] zajmują się tłumaczeniem z języka angielskiego na język rosyjski.

W systemie *Moses* faktory składają się z pewnych wartości reprezentujących informacje morfologiczne, wchodzących w skład wektorów. Proponowana metoda pozwala na składanie w pełne zdania generowanych form wyrazowych. Analiza morfologiczna, wnosząca istotne informacje, jest reprezentowana jako wektor wartości a. Wymiary wektora, są uzależnione od rozmiaru stosowanego leksykonu L. Leksykon L_s to zbiór danych dla języka źródłowego, a leksykon L_t to zbiór danych dla języka docelowego. Wszystkie pojedyncze wykładniki morfologiczne wchodzą w skład zbioru A_w. Zbiór A_w zawiera wykładniki morfologiczne słowa w, będącego daną formą odmienioną. Dodatkowo zdefiniowane zbiory to S_w jako zbiór potencjalnych rdzeni słowa w oraz I_w jako zbiór form wyrazowych mających ten sam rdzeń, co wyraz w.

Zdanie w języku docelowym składa się z sekwencji form wyrazowych

$w_1, ..., w_n$ i może zostać sprowadzone do sekwencji zbiorów rdzeni $S_1, ..., S_n$, gdzie każdy zbiór rdzeni odpowiada pojedynczej formie wyrazowej. Dla każdego zbioru rdzeni istnieje możliwość przewidzenia odmiany y_t na podstawie zbioru form wyrazowych I_t, mających ten sam rdzeń (form odmienionych).

Zastosowany przez twórców model probabilistyczny korzysta z maksymalnej entropii Markowa i bierze pod uwagę dwie poprzednie predykcje na podstawie słów poprzedzających. Prawdopodobieństwo dla przewidywanej odmiany można opisać wzorem zawartym w formule (34).

$$p(\bar{y}|\bar{x}) = \prod_{t=1}^{n} p(y_t|y_{t-1}, y_{t-2}, x_t), y_t \in I_t \tag{34}$$

We wzorze uwzględniono dodatkowo kontekst wystąpienia zarówno w języku źródłowym, jak i w języku docelowym. W formule x_t określa kontekst na pozycji t, a y_t - odmianę wybraną ze zbioru wszystkich form odmienionych I_t.

Twórcy metody wykorzystują cechy będące nośnikami informacji morfologicznych. Cechy dotyczą kontekstu wystąpienia oraz wybranej formy y_t.

Przykład 26 Dla formy odmienionej y_t oraz danego kontekstu para cech wykorzystanych przez twórców opisywanej metody, może wyglądać tak, jak przedstawiono w formułach (35) oraz (36).

$$\phi_k = \begin{cases} 1\ \textit{jeśli formą odmienioną}\ y_t\ \textit{jest}\ y'\ \textit{oraz}\ s' \in S_{t+1} \\ 0\ \textit{w przeciwnym przypadku} \end{cases} \tag{35}$$

$$\phi_{k+1} = \begin{cases} 1\ \textit{jeśli}\ Przypadek(y_t) = \text{“}Nom\text{“}\ \textit{oraz}\ Przypadek(y_{t-1}) = \text{“}Nom\text{“}) \\ 0\ \textit{w przeciwnym przypadku} \end{cases} \tag{36}$$

Do przewidywania formy odmienionej y_t wykorzystuje się sąsiadujący zbiór stemów S_{t+1}. Sąsiadujący zbiór służy jako cecha kontekstu. Pokazano to w formule (35). Druga formuła (36) uwzględnia przypadek wcześniejszego

słowa. Zastosowany model pozwala na uzyskanie cech kontekstu z uwzględnieniem dwóch poprzednich predykcji, czyli przewidywań poprzednich słów.

Autorzy artykułu [4] korzystają z cech skategoryzowanych jako jednojęzyczne i dwujęzyczne. Cechy jednojęzyczne dotyczą kontekstu wyłącznie w języku docelowym, a cechy dwujęzyczne także informacji o zdaniach w języku źródłowym. Obie kategorie dzielą się na kolejne podkategorie cech leksykalnych, morfologicznych i składniowych. Do cech leksykalnych należą chociażby rdzenie słów w języku docelowym oraz zbiór urównoleglonych słów, do cech morfologicznych - tagi części mowy, osoba, rodzaj, czas. Cechy składniowe uwzględniają wzajemne relacje między wyrazami.

Do eksperymentów wykorzystano milionowy korpus angielsko-rosyjski. Finalnie leksykon składał się z 14000 rdzeni i średnio około 4 form wyrazowych na jeden rdzeń. Przeprowadzono dwa podstawowe eksperymenty. Pierwszy z nich zakładał wybór losowej formy odmienionej spośród zbioru I_t. W drugim wykorzystano trigramowy model języka. Oba eksperymenty służyły do późniejszego porównania wyników. Dalsze eksperymenty polegały na sprawdzeniu jakości tłumaczenia przy wykorzystaniu miary dokładności, po wzbogacaniu procesu tłumaczenia o dodatkowe cechy. Testy odbywały się przy odrębnym aplikowaniu cech jednojęzycznych oraz dwujęzycznych.

Wykorzystywaną miarą ewaluacji była dokładność. Tylko przy wykorzystaniu cech leksykalnych jednojęzycznych udało się podnieść wynik dokładności z 31.7% uzyskany w przypadku eksperymentu podstawowego z wyborem losowej formy odmienionej oraz z 77.6% w przypadku eksperymentu podstawowego z użyciem trigramowego modelu języka do 85.1%. Z kolei przy wykorzystaniu wszystkich cech dwujęzycznych udało się osiągnąć wynik dokładności na poziomie 91.5%. Twórcy artykułu udowodnili poprzez przeprowadzane eksperymenty, że ich metoda przynosi oczekiwane efekty i pozytywnie wpływa na jakość tłumaczenia z języka angielskiego na język rosyjski, czyli z języka ubogiego pod względem morfologicznym na język bogaty morfologicznie.

W artykule naukowym [5], którego autorami są Sharon Goldwater oraz David McClosky przedstawiono z kolei inne podejście przyczyniające się do

ulepszenia jakości tłumaczenia z języka czeskiego na język angielski.

Twórcy prezentują metodę, która polega na zmianie parametrów modelu języka. Pierwszą z opcji modyfikacji jest zastosowanie lematów zamiast właściwych form słów. Ze względu na to, że proces lematyzacji może usuwać przydatne informacje z właściwych form słów, zastosowano dwie metody lematyzacji. Pierwsza polega na lematyzacji wyłącznie niektórych części mowy, co pozwala na ograniczenie ubytku wspomnianych informacji. Drugą metodą lematyzacji jest lematyzowanie wyłącznie rzadziej występujących w korpusie form słów.

Twórcy artykułu wykorzystują termin pseudosłów, które służą reprezentacji dodatkowych informacji morfologicznych wyrażonych w postaci tagów. Przykładowo, PER_1 oznacza pierwszą osobę.

Przykład 27 Przytoczony przez twórców artykułu [5] przykład obrazujący działanie metody zakłada określenie lematów dla form wyrazowych, wprowadzenie pseudosłów określających informacje morfologiczne w postaci tagów oraz modyfikowanie lematów za pomocą informacji, których nośnikami są tagi.

Etap 1 - wejściowe formy słowne: Pro někoho by její provedení mělo smysl.
Etap 2 - określenie lematów: pro někdo být jeho provedení mít smysl.
Etap 3 - wprowadzenie pseudosłów: pro někdo být PER_3 jeho provedení mít PER_X smysl.
Etap 4 - modyfikowanie lematów pseudosłowami: pro nékdo být+PER_3 jeho provedení mít+PER_X smysl.

Lemat słowa *být* zostaje zmodyfikowany przez trzecią osobę, a lemat słowa *mít* - przez jakąkolwiek osobę.

Urównoleglenia na poziomie słów są przydatne dla twórców i umożliwiają stosowanie modelu tłumaczenia opartego na morfemach. We wzorze $f_j = f_{j0}, ..., f_{jK}$, f_{j0} jest lematem słowa f_j, a $f_{j1}, ..., f_{jK}$ są morfemami. Morfemy wygenerowano na podstawie tagów powiązanych ze słowem f_j. Każde słowo posiada K morfemów. Model zakłada, że jeśli w korpusie słowo w

języku docelowym zostaje urównoleglone do słów w języku źródłowym posiadających odpowiedni tag, to bardziej prawdopdobnym jest, że zostanie ono urównoleglone także do innego słowa posiadającego ten sam tag, niż inny tag. Przykładowo, jeśli słowo w języku angielskim zostaje urównoleglone do większości form czasu teraźniejszego, to z dużym prawdopodobieństwem może zostać urównoleglone do innego słowa będącego także formą czasu teraźniejszego, a nie do formy czasu przeszłego lub przyszłego.

W przeprowadzanych eksperymentach wykorzystano korpus składający się z 21000 oznaczonych morfologicznie zdań. Podstawowy eksperyment polegający na wytrenowaniu standardowego modelu tłumaczenia pozwolił na uzyskanie wyniku *BLEU* na poziomie 31.1% dla zestawu treningowego oraz 27% dla zestawu testowego. Model tłumaczenia był zatem trenowany na zestawie treningowym, a wyniki tego trenowania sprawdzono na zestawie testowym. Proponowane metody spowodowały osiągnięcie wyniku *BLEU* na poziomie 39% dla zestawu treningowego oraz 33.3% dla zestawu testowego.

5.2 Narzędzia i zasoby dla języka polskiego

Jednym z narzędzi, wspomnianym już w podrozdziale [2.3], jest *Morfeusz*, służący do analizy morfologicznej. Program ten wykonuje analizę morfologiczną dla języka polskiego.

Program *Morfeusz* występuje w trzech wersjach. Każda z wersji opiera się na innych źródłach. *Morfeusz SGJP* korzysta z danych pochodzących ze *Słownika gramatycznego języka polskiego*. *Morfeusz Polimorf* korzysta ze słownika *Polimorf*, łączącego w sobie dane ze słownika SGJP oraz dane pochodzące ze społeczności sjp.pl. Trzecią i jednocześnie najstarszą wersją jest *Morfeusz SIaT*, którego dane pochodzą ze *Schematycznego indeksu a tergo polskich form wyrazowych*.

Najbardziej rozbudowaną i bogatą wersją programu *Morfeusz* jest wersja *Morfeusz Polimorf*. *Morfeusz* to jednak nie tylko samodzielnie uruchamiany program komputerowy. *Morfeusz* to biblioteka, którą programista może wykorzystać we własnym programie.

Najnowsza wersja programu *Morfeusz Polimorf* nosi numer 1.9.2. Podawane zdanie jest pewnym ciągiem wejściowym składającym się z wyrazów. Program podaje dla każdego z tych wyrazów, czyli elementów ciągu wejściowego, znalezione formy wraz z lematami i tagowaniem. Nieskomplikowany przykład analizy morfologicznej realizowanej za pomocą programu *Morfeusz* został przytoczony w podrozdziale [2.3]. Warto jednak przyjrzeć się bliżej bardziej rozbudowanym fragmentom tekstów.

Przykład 28 Dla zdania *Anna ma dużego psa* przeprowadzono analizę morfologiczną za pomocą programu *Morfeusz Polimorf*. Wyniki przedstawiono na rysunku (20).

Analiza morfologiczna:

O→	→O	Forma	Lemat	Tag	Nazwa	Kwalifikatory
0	1	**Anna**	Anna	subst:sg:nom:f	*imię*	
1	2	**ma**	mieć	fin:sg:ter:imperf	*pospolita*	
			mój	adj:sg:nom.voc:f:pos	*pospolita*	
2	3	**dużego**	duży	adj:sg:acc:m1.m2:pos	*pospolita*	
			duży	adj:sg:gen:m1.m2.m3.n1.n2:pos	*pospolita*	
3	4	**psa**	pies	subst:sg:acc:m1	*pospolita*	
			pies	subst:sg:acc:m2	*pospolita*	
			pies	subst:sg:gen:m1	*pospolita*	
			pies	subst:sg:gen:m2	*pospolita*	
4	5	**.**	.	interp		

Rysunek 20: Wyniki analizy morfologicznej przeprowadzonej programem Morfeusz Polimorf (Morfeusz Polimorf, [16])

Kolumna *Nazwa* wyróżniła imię spośród nazw pospolitych. Imię i znak interpunkcyjny w postaci kropki wydają się dość jednoznaczne, stąd zwrócenie wyłącznie jednego wyniku. Wyraz *ma* został określony w tagowaniu jako forma nieprzeszła (fin) od lematu *mieć* oraz przymiotnik (adj) od lematu (mój). Przymiotnik (adj) *dużego* może występować w rodzaju męskim (m1.m2 lub m1.m2.m3) i nijakim (n1.n2) oraz w dwóch przypadkach - bierniku (acc) oraz dopełniaczu (gen). Rzeczownik (subst) *psa* występuje tylko w liczbie pojedynczej (sg), ale w dwóch przypadkach - bierniku (acc) oraz dopełniaczu (gen) oraz rodzaju męskim (m1 lub m2). Wskazanie dodatkowych cyfr dookreśla kategorię dla rodzaju.

Oprócz analizy morfologicznej, program *Morfeusz Polimorf* posiada również wbudowany generator, który na podstawie podanego lematu zwraca wszystkie formy od niego pochodzące.

Przykład 29 Dla lematu *uwielbiać* uruchomiono generator programu *Morfeusz Polimorf*. Wyniki przedstawiono na rysunku (21).

Wygenerowane formy:

▼ Forma	Lemat	Tag	Nazwa	Kwalifikatory
uwielbiają	uwielbiać	fin:pl:ter:imperf	pospolita	
uwielbiając	uwielbiać	pcon:imperf	pospolita	
uwielbiająca	uwielbiać	pact:sg:nom.voc:f:imperf:aff	pospolita	
uwielbiającą	uwielbiać	pact:sg:acc.inst:f:imperf:aff	pospolita	
uwielbiające	uwielbiać	pact:pl:nom.acc.voc:m2.m3.f.n1.n2.p2.p3:imperf:aff	pospolita	
uwielbiające	uwielbiać	pact:sg:nom.acc.voc:n1.n2:imperf:aff	pospolita	
uwielbiającego	uwielbiać	pact:sg:acc:m1.m2:imperf:aff	pospolita	
uwielbiającego	uwielbiać	pact:sg:gen:m1.m2.m3.n1.n2:imperf:aff	pospolita	
uwielbiającej	uwielbiać	pact:sg:gen.dat.loc:f:imperf:aff	pospolita	
uwielbiającemu	uwielbiać	pact:sg:dat:m1.m2.m3.n1.n2:imperf:aff	pospolita	
uwielbiający	uwielbiać	pact:pl:nom.voc:m1.p1:imperf:aff	pospolita	
uwielbiający	uwielbiać	pact:sg:acc:m3:imperf:aff	pospolita	
uwielbiający	uwielbiać	pact:sg:nom.voc:m1.m2.m3:imperf:aff	pospolita	
uwielbiających	uwielbiać	pact:pl:acc:m1.p1:imperf:aff	pospolita	
uwielbiających	uwielbiać	pact:pl:gen.loc:m1.m2.m3.f.n1.n2.p1.p2.p3:imperf:aff	pospolita	
uwielbiającym	uwielbiać	pact:pl:dat:m1.m2.m3.f.n1.n2.p1.p2.p3:imperf:aff	pospolita	
uwielbiającym	uwielbiać	pact:sg:inst.loc:m1.m2.m3.n1.n2:imperf:aff	pospolita	
uwielbiającymi	uwielbiać	pact:pl:inst:m1.m2.m3.f.n1.n2.p1.p2.p3:imperf:aff	pospolita	
uwielbiajcie	uwielbiać	impt:pl:sec:imperf	pospolita	
uwielbiajmy	uwielbiać	impt:pl:pri:imperf	pospolita	
uwielbiali	uwielbiać	praet:pl:m1.p1:imperf	pospolita	
uwielbiał	uwielbiać	praet:sg:m1.m2.m3:imperf	pospolita	

Rysunek 21: Wyniki dla generowania form w programie Morfeusz Polimorf (Morfeusz Polimorf, [16])

Generator uwzględnia wszystkie formy pochodzące od lematu *uwielbiać*. W wynikach można odnaleźć zarówno formy przeszłe oraz przyszłe czasowników, jak i imiesłowy (ppas).

Oprócz programu *Morfeusz*, będącego zaawansowanym narzędziem do analizy morfologicznej, istnieją także inne narzędzia służące do tagowania dla języka polskiego.

Tree Tagger oznacza tekst tagami części mowy oraz lematami. Program jest dostępny zarówno z poziomu konsoli systemu Linux, jak i z poziomu systemu Windows.

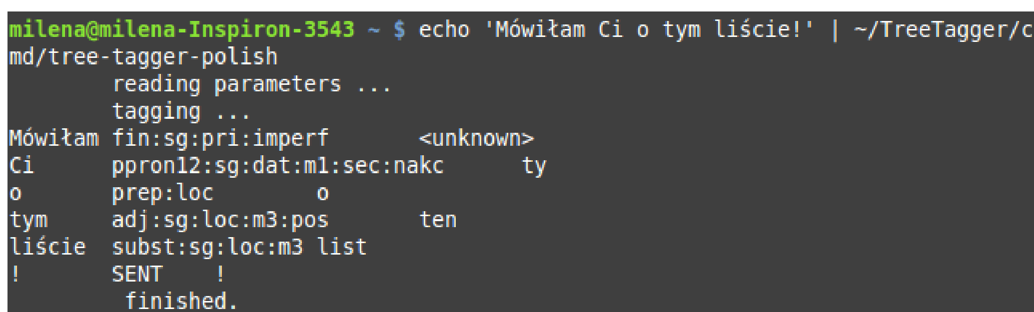
Ważnym krokiem konfiguracyjnym, który pozwala na uruchomienie pro-

gramu, jest ustawienie ścieżek do katalogów. W katalogach znajdują się pliki programu. Ścieżki ustawia się w pliku dedykowanym dla danego języka - w tym przypadku *tree-tagger-polish*. Bezpośrednie uruchomienie programu jest możliwe dzięki poleceniu zapisanemu poniżej.

```
echo sentence | sh cmd/tree-tagger-polish
```

Argument *sentence* określa zdanie wejściowe dla *Tree Taggera*.

Przykład 30 Dla zdania wejściowego *Mówiłam Ci o tym liście!* uruchomiono program *Tree Tagger*. Wyniki działania programu przedstawiono na rysunku (22).



Rysunek 22: Wyniki działania programu Tree Tagger (Tree Tagger, [17])

Program *Tree Tagger* znakuje tekst wyłącznie tagami dotyczącymi części mowy i formy wyrazu oraz podaje lematy dla poszczególnych słów. Dokonuje ujednoznacznienia, co szczególnie dobrze widać w przypadku niejednoznacznego słowa *liście*, którego lemat *list* oraz forma zostały poprawnie zinterpretowane. *Tree Tagger* nie dokonuje analizy morfologicznej.

Należy nadmienić, że istnieją również inne narzędzia tagujące dla języka polskiego. Zarówno *WMBT* (Wrocław Memory-Based Tagger), jak i *WCRFT* (Wrocław CRF Tagger) oraz *TaKIPI* (Tager Korpusu IPI PAN) są wspierane w działaniu przez Morfeusza. Dla *WMBT* oraz *WCRFT* istnieje dodatkowy moduł o nazwie *MACA* (Morphological Analysis Converter and Aggregator), który dokonuje analizy morfologicznej.

Z punktu widzenia pracy magisterskiej najistotniejszy będzie tager *Clarin Pelcra* [8]. Program jest dostępny w sieci i umożliwia tagowanie dla języka

polskiego. Wyniki tagowania mogą zostać zapisane do pliku w formacie *.xml lub *.json.

Przykład 31 Dla zdania w języku polskim *Pisała w liście.* uruchomiono tagowanie za pomocą tagera Clarin Pelcra. Otrzymano wyniki przedstawione na rysunku (23).


```
[
  [
    {
      "orth": "Pisała",
      "lexes": [
        {
          "CTag": "praet:sg:f:imperf",
          "alias": "verb",
          "base": "pisać",
          "disamb": true
        }
      ]
    },
    {
      "orth": "w",
      "lexes": [
        {
          "CTag": "prep:loc:nwok",
          "alias": "prep",
          "base": "w",
          "disamb": true
        }
      ]
    },
    {
      "orth": "liście",
      "lexes": [
        {
          "CTag": "subst:sg:loc:m3",
          "alias": "noun",
          "base": "list",
          "disamb": true
        }
      ]
    },
    {
      "orth": ".",
      "lexes": [
        {
          "CTag": "interp",
          "alias": "punct",
          "base": ".",
          "disamb": true
        }
      ]
    }
  ]
]
```


Rysunek 23: Wyniki tagowania przeprowadzonego za pomocą programu Clarin Pelcra w formacie *.json (Clarin Pelcra, [8])

Wyniki są przedstawione w formacie *.json. W `orth` znajduje się forma wyrazu pobrana ze zdania. W `CTag` znajdują się informacje morfologiczne odseparowane dwukropkami, dotyczące formy występującego wyrazu, np. część mowy, rodzaj, liczba. W części `base` znajduje się forma podstawowa słowa, czyli jego lemat. Przykładowo dla słowa *pisała* wyszczególniono, że jest to pseudoimiesłów (praet) w liczbie pojedynczej (sg), rodzaju żeńskiego i w trybie niedokonanym (imperf). Lematem słowa *pisała* jest słowo *pisać*.

Tager *Clarin Pelcra* jest narzędziem, w którym stosowana notacja tagów opiera się na tej wykorzystywanej w Narodowym Korpusie Języka Polskiego. Ze względu na aktualność oraz prostotę użycia, tager będzie narzędziem pomocnicznym przy wykonywaniu eksperymentów, dokonującym niezbędnego ujednoznacznienia.

6 Opis proponowanego rozwiązania

Celem eksperymentów przeprowadzanych w ramach projektu magisterskiego jest wykazanie, że zastosowanie klasyfikacji oraz dodanie cech morfologicznych może pozytywnie wpłynąć na jakość tłumaczenia, szczególnie w przypadku języków bogatych pod względem morfologicznym. W eksperymentach wykorzystano system tłumaczenia automatycznego *Moses* oraz narzędzie *Vowpal Wabbit*. Badano tłumaczenie z języka angielskiego na język polski.

6.1 Trenowanie korpusu za pomocą systemu Moses

Pierwszym i ważnym krokiem przeprowadzonego eksperymentu było wytrenowanie korpusu za pomocą systemu *Moses*. Do tego celu wykorzystano korpus równoległy Europarl, który zawiera fragmenty dokumentów wyodrębnionych ze strony Parlamentu Europejskiego. Korpus *Europarl* składa się z dwóch plików. Pierwszy z nich `europarl-v7.pl-en.en` zawiera zdania w języku angielskim, a drugi - `europarl-v7.pl-en.pl` - ich polskie odpowiedniki. Rozmiar wykorzystanego korpusu sięga 632565 urównoleglonych zdań.

Cały zestaw danych podzielono na zestaw treningowy oraz zestaw testowy. W skład zestawu treningowego wchodzi 622565 zdań. Do zestawu testowego wykorzystano 10000 zdań z końca całego zestawu danych.

Do utworzenia zestawu treningowego oraz testowego wykorzystano polecenia konsolowe `head` i `tail`.

```
head -n 622565 europarl-v7.pl-en.en > train.pl-en.en
head -n 622565 europarl-v7.pl-en.pl > train.pl-en.pl

tail -n 10000 europarl-v7.pl-en.en > test.pl-en.en
tail -n 10000 europarl-v7.pl-en.pl > test.pl-en.pl
```

Odpowiednią ilość zdań wyodrębniono z plików korpusu i zapisano do plików `train.pl-en.en` i `train.pl-en.pl`, które stały się plikami zestawu treningowego oraz do plików `test.pl-en.en` i `test.pl-en.pl`, które stały się plikami zestawu testowego.

Następnie wykonano szereg operacji służących oczyszczeniu korpusu. Oczyszczanie odbywa się dzięki dostępnym w systemie *Moses* skryptom. Tokenizacja, którą wykonuje skrypt `tokenizer.perl`, umożliwia podzielenie tekstu na mniejsze cząstki, czyli tokeny, najczęściej odseparowane od siebie np. spacjami. Najpierw dokonano tokenizacji za pomocą wbudowanego skryptu `tokenizer.perl`.

```
~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l pl < \
~/ExperimentsMGR/Europarl/train.pl-en.pl > \
~/ExperimentsMGR/Europarl/train.pl-en.tok.pl

~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l en < \
~/ExperimentsMGR/Europarl/train.pl-en.en > \
~/ExperimentsMGR/Europarl/train.pl-en.tok.en
```

Wyniki działania skryptu tokenizującego tekst zostały zapisane w plikach `train.pl-en.tok.pl` oraz `train.pl-en.tok.en`.

Kolejnym etapem oczyszczania korpusów było sprowadzanie pierwszych liter słów występujących na początkach zdań do postaci najczęściej występującej w korpusie. Skrypt `train-truecaser.perl` umożliwił przeprowadzenie procesu trenowania. Skrypt `truecase.perl` pozwolił na sprowadzenie liter do ich właściwej postaci.

```
~/mosesdecoder/scripts/recaser/train-truecaser.perl --model \
~/ExperimentsMGR/truecase-model.pl --corpus \
~/ExperimentsMGR/Europarl/train.pl-en.tok.pl

~/mosesdecoder/scripts/recaser/train-truecaser.perl --model \
~/ExperimentsMGR/truecase-model.en --corpus \
~/ExperimentsMGR/Europarl/train.pl-en.tok.en

~/mosesdecoder/scripts/recaser/truecase.perl --model \
~/ExperimentsMGR/truecase-model.pl \
< ~/ExperimentsMGR/Europarl/train.pl-en.tok.pl > \
```

```
~/ExperimentsMGR/Europarl/train.pl-en.true.pl
```

```
~/mosesdecoder/scripts/recaser/truecase.perl --model \
~/ExperimentsMGR/truecase-model.en \
< ~/ExperimentsMGR/Europarl/train.pl-en.tok.en > \
~/ExperimentsMGR/Europarl/train.pl-en.true.en
```

Ostateczne wyniki działania skryptu sprowadzającego litery do ich odpowiednich wielkości zostały zapisane w plikach `train.pl-en.true.pl` oraz `train.pl-en.true.en`.

Ostatnim krokiem procesu oczyszczania było skracanie zbyt długich zdań. Czynność tą wykonano w celu uniknięcia problemów w kanale treningowym. Do skracania zdań wykorzystano skrypt `clean-corpus-n.perl`.

```
~/mosesdecoder/scripts/training/clean-corpus-n.perl \
~/ExperimentsMGR/Europarl/train.pl-en.true en pl \
~/ExperimentsMGR/Europarl/train.pl-en.clean 1 80
```

Po oczyszczeniu korpusu, przystąpiono do trenowania trigramowego modelu języka.

```
~/mosesdecoder/bin/lmplz -o 3 < \
~/ExperimentsMGR/Europarl/train.pl-en.true.pl > \
train.pl-en.arpa.pl
```

W wyniku procesu trenowania modelu języka powstały pliki o rozszerzeniu *.arpa. Model języka został zapisany w pliku `train.pl-en.arpa.pl`. W celu szybszego ładowania, pliki zostały zbinaryzowane, czyli skompresowane.

```
~/mosesdecoder/bin/build_binary train.pl-en.arpa.pl \
train.pl-en.blm.pl
```

Po uzyskaniu pliku z modelem języka, możliwe było uruchomienie długiego procesu trenowania systemu tłumaczenia.

```
~/mosesdecoder/scripts/training/train-model.perl -cores 5 -parallel \
-sort-batch-size 253 -sort-compress gzip -root-dir train -corpus \
~/ExperimentsMGR/Europarl/train.pl-en.clean -f en -e pl -alignment \
grow-diag-final-and -reordering msd-bidirectional-fe -lm \
0:3:$HOME/ExperimentsMGR/Europarl/lm/train.pl-en.blm.pl:8 \
-external-bin-dir ~/mosesdecoder/tools
```

Do trenowania systemu tłumaczenia wykorzystano wbudowany skrypt `train-model.perl`. Opcja `-cores arg` pozwala na wspieranie procesu trenowania poprzez uruchomienie pracy na wielu rdzeniach - w tym przypadku pięciu. W procesie trenowania systemu tłumaczenia korzysta się z wytrenowanego wcześniej modelu języka `train.pl-en.blm.pl`. Stosowane polecenie ma postać podstawową, rekomendowaną w dokumentacji systemu *Moses*.

W wyniku procesu trenowania systemu tłumaczenia powstał istotny plik konfiguracyjny *moses.ini*. Plik umożliwia uruchamianie wytrenowanego systemu tłumaczenia oraz udoskonalanie go poprzez wprowadzanie modyfikacji.

Po przejściu przez etap trenowania, system został uruchomiony w celu sprawdzenia.

```
~/mosesdecoder/bin/moses -f \
~/ExperimentsMGR/Europarl/working/train/model/moses.ini
```

W celu zredukowania czasu uruchamiania systemu tłumaczenia, wprowadzono modyfikacje. Skorzystano z tablic kompaktowych, kompresujących rozbudowane tabele fraz. W ten sposób zmniejszono rozmiar tabel fraz, co z kolei przełożyło się na znacznie szybsze uruchamianie systemu tłumaczenia.

Całość została zmodyfikowana z poziomu folderu

`~/ExperimentsMGR/Europarl/working/train/model/`.

Zastosowano polecenie służące do zmniejszenia rozmiaru tabel fraz.

```
~/mosesdecoder/bin/processPhraseTableMin -in phrase-table.gz \
-out phrase-table -nscores 4 -threads 4
```

Zmian dokonano także w pliku konfiguracyjnym *moses.ini*. W sekcji `[feature]` ustawiono cechę `PhraseDictionaryCompact` wraz z odpowiednią ścieżką dla skompresowanych tablic fraz.

Zestaw testowy, składający się z 10000 zdań, przygotowano do przeprowadzenia procesu testowania. W tym celu konieczne było przejście przez etapy tokenizacji i sprowadzania pierwszych liter pierwszych słów w zdaniach do odpowiedniej wielkości.

```
~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l pl < \
~/ExperimentsMGR/Europarl/test.pl-en.pl > \
~/ExperimentsMGR/Europarl/test.pl-en.tok.pl

~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l en < \
~/ExperimentsMGR/Europarl/test.pl-en.en > \
~/ExperimentsMGR/Europarl/test.pl-en.tok.en

~/mosesdecoder/scripts/recaser/train-truecaser.perl --model \
~/ExperimentsMGR/Europarl/truecase-test-model.pl --corpus \
~/ExperimentsMGR/Europarl/test.pl-en.tok.pl

~/mosesdecoder/scripts/recaser/train-truecaser.perl --model \
~/ExperimentsMGR/Europarl/truecase-test-model.en --corpus \
~/ExperimentsMGR/Europarl/test.pl-en.tok.en

~/mosesdecoder/scripts/recaser/truecase.perl --model \
~/ExperimentsMGR/truecase-model.pl < \
~/ExperimentsMGR/Europarl/test.pl-en.tok.pl > \
~/ExperimentsMGR/Europarl/test.pl-en.true.pl

~/mosesdecoder/scripts/recaser/truecase.perl --model \
~/ExperimentsMGR/truecase-model.en < \
~/ExperimentsMGR/Europarl/test.pl-en.tok.en > \
~/ExperimentsMGR/Europarl/test.pl-en.true.en
```

Pliki `test.pl-en.true.pl` oraz `test.pl-en.true.en` są oczyszczonymi plikami korpusu. Posłużyły do przetestowania działania wytrenowanego systemu tłumaczenia automatycznego.

```
~/mosesdecoder/bin/moses -f \
~/ExperimentsMGR/Europarl/working/train/model/moses.ini < \
~/ExperimentsMGR/Europarl/test.pl-en.true.en > \
~/ExperimentsMGR/Europarl/test.pl-en.translated.pl
```

Uzyskane tłumaczenia zapisano w pliku `test.pl-en.translated.pl`. Dokonano ewaluacji otrzymanych tłumaczeń za pomocą miary *BLEU*. Było to możliwe, dzięki wbudowanemu skryptowi `multi-bleu.perl`.

```
~/mosesdecoder/scripts/generic/multi-bleu.perl -lc \
~/ExperimentsMGR/Europarl/test.pl-en.true.pl < \
~/ExperimentsMGR/Europarl/test.pl-en.translated.pl
```

Otrzymany wynik *BLEU* dla zestawu testowego wyniósł 24.23.

W celu uzyskania lepszych tłumaczeń i poprawienia wyniku *BLEU*, przeprowadzono dodatkowy tuning. W tym celu wykorzystano Minimum Error Rate Training (MERT). MERT zakłada, że poprzez porównanie tłumaczenia systemowego z jego odpowiednikiem referencyjnym, można uzyskać liczbę błędów występującą w tłumaczeniu systemowym. Dla pewnego zbioru zdań, liczbę błędów można oszacować poprzez zsumowanie błędów występujących we wszystkich zdaniach, wchodzących w skład zbioru. Celem tuningu było zatem osiągnięcie jak najmniejszego oszacowania błędów na zbiorze zdań, w którym dla każdego zdania podany jest jest jego odpowiednik referencyjny oraz zbiór kandydatów na jego tłumaczenie.

Tuning został przeprowadzony na mniejszym zestawie danych. Wyodrębniono 3000 zdań z początku zestawu testowego oraz 3000 zdań z końca zestawu testowego. Pobrane partie zdań posłużyły kolejno jako zestaw deweloperski oraz zestaw testowy. Zestaw deweloperski służył do sprawdzenia działania systemu z innymi opcjami - w tym przypadku z wykorzystaniem *MERT*. Do utworzenia obu zestawów wykorzystano kolejno polecenia `head` oraz `tail`.

```
head -n 3000 test.pl-en.pl > \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.pl

head -n 3000 test.pl-en.en > \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.en

tail -n 3000 test.pl-en.en > \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.en

tail -n 3000 test.pl-en.pl > \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.pl
```

Również w przypadku plików korpusów dla *MERT*, należało przeprowadzić oczyszczanie.

```
~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l pl < \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.pl > \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.tok.pl

~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l en < \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.en > \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.tok.en

~/mosesdecoder/scripts/recaser/train-truecaser.perl --model \
~/ExperimentsMGR/Europarl/mert/truecase-model.pl --corpus \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.tok.pl

~/mosesdecoder/scripts/recaser/train-truecaser.perl --model \
~/ExperimentsMGR/Europarl/mert/truecase-model.en --corpus \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.tok.en

~/mosesdecoder/scripts/recaser/truecase.perl --model \
~/ExperimentsMGR/Europarl/mert/truecase-model.pl < \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.tok.pl > \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.true.pl
```

```
~/mosesdecoder/scripts/recaser/truecase.perl --model \
~/ExperimentsMGR/Europarl/mert/truecase-model.en < \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.tok.en > \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.true.en
```

Po oczyszczeniu korpusu, przeprowadzono proces tuningu.

```
~/mosesdecoder/scripts/training/mert-moses.pl \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.true.en \
~/ExperimentsMGR/Europarl/mert/dev-mert.pl-en.true.pl \
~/mosesdecoder/bin/moses train/model/moses.ini --mertdir \
~/mosesdecoder/bin
```

Po przeprowadzeniu procesu tuningu, zestaw testowy przygotowano do procesu testowania. Wykorzystano wielokrotnie stosowane skrypty do oczyszczania korpusu.

```
~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l pl < \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.pl > \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.tok.pl

~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l en < \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.en > \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.tok.en
```

```
~/mosesdecoder/scripts/recaser/train-truecaser.perl --model \
~/ExperimentsMGR/Europarl/mert/truecase-model.pl --corpus \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.tok.pl

~/mosesdecoder/scripts/recaser/train-truecaser.perl --model \
~/ExperimentsMGR/Europarl/mert/truecase-model.en --corpus \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.tok.en
```

```
~/mosesdecoder/scripts/recaser/truecase.perl --model \
~/ExperimentsMGR/Europarl/mert/truecase-model.pl < \
```

```
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.tok.pl > \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.true.pl

~/mosesdecoder/scripts/recaser/truecase.perl --model \
~/ExperimentsMGR/Europarl/mert/truecase-model.en < \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.tok.en > \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.true.en
```

Po uzyskaniu oczyszczonych plików zestawu testowego, możliwe było rozpoczęcie procesu testowania.

```
~/mosesdecoder/bin/moses -f ~/ExperimentsMGR/Europarl/mert-work/moses.ini < \
~/ExperimentsMGR/Europarl/test-mert.pl-en.true.en > \
~/ExperimentsMGR/Europarl/test-mert.pl-en.translated.pl
```

Tłumaczenia zostały zapisane w pliku `test-mert.pl-en.translated.pl`. Za pomocą skryptu `multi-bleu.perl` przeprowadzono ewaluację.

```
~/mosesdecoder/scripts/generic/multi-bleu.perl -lc \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.true.pl < \
~/ExperimentsMGR/Europarl/mert/test-mert.pl-en.translated.pl
```

Uzyskany wynik *BLEU* wynosi 26.11. Będzie on stanowił punkt odniesienia dla dalszych działań przeprowadzanych w ramach projektu magisterskiego.

6.2 Przewidywanie klas morfologicznych za pomocą systemu Vowpal Wabbit

Wykorzystanie w eksperymencie systemu *Vowpal Wabbit* umożliwiło dokonanie próby klasyfikacji oraz wprowadzenie dodatkowych cech morfologicznych. Pozwoliło to na sprawdzenie, czy skorzystanie z klasyfikacji może wesprzeć proces tłumaczenia.

Otrzymany na potrzeby pracy magisterskiej plik treningowy zawierał zdania w postaci przedstawionej na rysunku (24).

<s>
2a 2a subst:sg:nom:n
. . interp
Informację informacja subst:sg:acc:f
o o prep:loc
obowiązku obowiązek subst:sg:loc:m3
złożenia złożyć ger:sg:gen:n:perf:aff
przez przez prep:acc:nwok
zagranicznego zagraniczny adj:sg:acc:m1:pos
dostawcę dostawca subst:sg:acc:m1
oferty oferta subst:sg:gen:f
offsetowej offsetowy adj:sg:gen:f:pos
zamieszcza zamieszczać fin:sg:ter:imperf
się się qub
w w prep:loc:nwok
specyfikacji specyfikacja subst:sg:loc:f
istotnych istotny adj:pl:gen:m3:pos
warunków warunek subst:pl:gen:m3
zamówienia zamówienie subst:sg:gen:n
(...)
</s>

Rysunek 24: Fragment nieprzetworzonego pliku treningowego

Znaczniki `<s>` oraz `</s>` określają początek i koniec zdania. Dodatkowo zdania zostały odseparowane od siebie pojedynczym znakiem nowej linii. Każda linia zawiera właściwą formę słowa, lemat oraz informacje morfologiczne dotyczące np. części mowy, liczby, czy przypadku. Informacje morfologiczne są odseparowane za pomocą znaku dwukropka.

Każdą linię pliku treningowego należało sprowadzić do reprezentacji preferowanej przez program *Vowpal Wabbit*. W celu zautomatyzowania procesu napisano skrypt w języku *Python*, który umożliwił przekształcenie zdań do postaci formatu wejściowego. Skrypt przedstawiono w oknie kodu (11).

```
import sys
import os
import re

dict_for_lemmas = {}
dict_for_original = {}
python_dict_for_sentences = {}

def get_arguments():
  arguments = sys.argv
  number_arguments = len(arguments)
  fit_dictionary()
  for arg_num in range(1,number_arguments):
    get_sentences_return_morph_info(str(arguments[arg_num]))

def get_sentences_return_morph_info(file_name):
  file_to_process = open(file_name, 'r')
  number = 0
  for line in file_to_process:
    array_of_words = line.split( )
    if len(array_of_words) == 0:
      continue
    else:
      original_word = array_of_words[0]
      if len(array_of_words) == 1:
        lemma_of_word = original_word
        pos_of_word = []
        forms_of_word = []
      else:
        lemma_of_word = array_of_words[1]
        pos_of_word = array_of_words[2].split(":")[0]
        forms_of_word = array_of_words[2].replace(".", ":").split(":")[1:]
        find_word_in_dictionary(original_word, lemma_of_word, pos_of_word,
    forms_of_word)

def fit_dictionary():
  dictionary = open('polimorf-20150220.tab', 'r')
  for line in dictionary:
    line = re.sub(r"n[0-9]\.*n*[0-9]*", "n", line)
    array_dictionary_words = line.split( )
    original = array_dictionary_words[0]
    lemma = array_dictionary_words[1]
    pos = array_dictionary_words[2].split(":")[0]
    forms = array_dictionary_words[2].replace(".", ":").split(":")[1:]
    if lemma not in dict_for_lemmas:
      dict_for_lemmas[lemma] = {pos : { original : [forms] }}
    else:
      if pos in dict_for_lemmas[lemma].keys():
        if original in dict_for_lemmas[lemma][pos].keys():
```

```
49            dict_for_lemmas[lemma][pos][original].append(forms)
50          else:
51            dict_for_lemmas[lemma][pos][original] = [forms]
52        else:
53          dict_for_lemmas[lemma][pos] = { original : [forms] }
54
55
56  def find_word_in_dictionary(original_word, lemma_of_word, pos_of_word,
      forms_of_word):
57    array_of_words_in_dict = []
58    VW_string_format = ""
59    if ":" in lemma_of_word:
60      lemma_of_word = "&#58;"
61    if ":" in original_word:
62      original_word = "&#58;"
63    if original_word == lemma_of_word and pos_of_word == [] and forms_of_word
      == []:
64      print "shared |s " + original_word
65      print "1111:0 |t " + lemma_of_word
66    if original_word == "</s>":
67      print "\n###### EOS ######"
68    else:
69      array_of_words_in_dict = []
70      print "shared |s " + lemma_of_word + " " + pos_of_word
71      if lemma_of_word in dict_for_lemmas:
72        if pos_of_word in dict_for_lemmas[lemma_of_word].keys():
73          for one_word in dict_for_lemmas[lemma_of_word][pos_of_word].keys():
74            array_of_words_in_dict.append(one_word)
75
76          for one_elem in array_of_words_in_dict:
77            new_elem  = str(one_elem)
78
79            forms_values = dict_for_lemmas[lemma_of_word][pos_of_word][
      new_elem]
80
81          for one_form_set in forms_values:
82            VW_string_format = VW_string_format = "|t " + one_elem + " "
83            for one in one_form_set:
84              VW_string_format += "f^" + one + " "
85            if set(forms_of_word).issubset(set(one_form_set)):
86              print "1111:0 " + VW_string_format
87            else:
88              print "1111:1 " + VW_string_format
89
90        else:
91          print "1111:0 |t " + lemma_of_word
92      else:
93        print "1111:0 |t " + lemma_of_word
94
95    print "\r"
```

```
96
97 get_arguments()
```

Kod 11: Skrypt służący do przedstawienia danych w postaci preferowanej przez program Vowpal Wabbit

Plik zawierający zdania w formacie przedstawionym na rysunku (24) przetworzono za pomocą skryptu zaprezentowanego w oknie kodu (11), uruchamianego poleceniem konsolowym.

```
python script_input_VW.py train.txt > train_set_VW.txt
```

Plik `train.txt` jest plikiem wejściowym, przetworzonym za pomocą skryptu. Otrzymane wyniki zapisano w pliku `train_set_VW.txt`, a jego fragment przedstawiono na rysunku (25).

```
shared |s <s>
1111:0 |t <s>

shared |s 2a subst
1111:0 |t 2a

shared |s . interp
1111:0 |t .

shared |s informacja subst
1111:0 |t informację f^sg f^acc f^f
1111:1 |t informacjami f^pl f^inst f^f
1111:1 |t informacjom f^pl f^dat f^f
1111:1 |t informacjo f^sg f^voc f^f
1111:1 |t informacji f^sg f^loc f^f
1111:1 |t informacji f^sg f^gen f^f
1111:1 |t informacji f^sg f^dat f^f

1111:1 |t informacji f^pl f^gen f^f
1111:1 |t informacjach f^pl f^loc f^f
1111:1 |t informacje f^pl f^voc f^f
1111:1 |t informacje f^pl f^nom f^f
1111:1 |t informacje f^pl f^acc f^f
1111:1 |t informacyj f^pl f^gen f^f
1111:1 |t informacją f^sg f^inst f^f
1111:1 |t informacja f^sg f^nom f^f

shared |s o prep
1111:0 |t o f^loc
1111:1 |t o f^acc

shared |s obowiązek subst
1111:1 |t obowiązku f^sg f^voc f^m3
1111:0 |t obowiązku f^sg f^loc f^m3
1111:1 |t obowiązku f^sg f^gen f^m3
1111:1 |t obowiązkiem f^sg f^inst f^m3

(...)
```

Rysunek 25: Fragment pliku treningowego w formacie wejściowym dla narzędzia Vowpal Wabbit

Skrypt pobiera argument w postaci nazwy pliku z linii poleceń. Następnie wyodrębnia podane informacje dotyczące formy, lematu oraz morfologii i szuka ich w słowniku *Polimorf*, który został przekształcony w programie w słownik języka *Python* w celu optymalizacji i przyspieszenia działania.

Każde zdanie składa się z wielu list form odseparowanych pustymi liniami. Na początku każdej listy wyróżniono przestrzeń `shared`, która zawiera lemat słowa oraz część mowy. Informacje te są współdzielone przez wszystkie formy dla danego słowa. Każda określona forma słowa ma ten sam lemat oraz część mowy - różni się wyłącznie infomacjami morfologicznymi, które reprezentowane są jako cechy według wzorca `f^inf`, gdzie `inf` to podana informacja morfologiczna, będąca np. przypadkiem lub liczbą.

Każda z form posiada przypisany koszt. Niższy koszt równy 0 przypisano właściwej formie słowa względem wystąpienia w zdaniu. Wyższy koszt równy 1 posiadają wszystkie formy będące niewłaściwymi w danym kontekście, ale poprawnymi dla danego słowa.

Proponowana reprezentacja danych służy przede wszystkim przeprowadzeniu procesu trenowania z wykorzystaniem modelu *CSOAA* i cech zależnych od klasy.

Na pliku przedstawiającym dane w postaci akceptowalnej przez program Vowpal Wabbit, uruchomiono trenowanie modelu.

```
~/vowpal_wabbit/vowpalwabbit/vw --csoaa_ldf m \
-f model_train.ml < train_set_VW.txt
```

Wytrenowany model został zapisany w pliku `model_train.ml`. Wykorzystano opcję `--csoaa_ldf`.

Wytrenowanie modelu pozwoliło na przeprowadzenie procesu testowania. Wykorzystany w eksperymencie plik testowy to odpowiednio przetworzone tłumaczenia zdań uzyskane w wyniku działania systemu tłumaczenia automatycznego *Moses*.

Plik z tłumaczeniami `test-mert.pl-en.translated.pl` zawierał dane w postaci przedstawionej na rysunku (26).

te zalecenia w pełni zgadzam się , i dlatego chciałbym ponownie wyrazić uznanie dla jakości tego sprawozdania , które udało się uzyskiwały szerokie , ponadpartyjne poparcie .
na piśmie . - (LT) z zadowoleniem przyjąłem ten dokument , ponieważ wieloletnich ram finansowych na lata 2007-2013 stają się celem współpracy terytorialnej w jednym z trzech filarów polityki spójności UE , zastępując wspólnotowej inicjatywy INTERREG .
od tamtej pory stało się spójności terytorialnej , na mocy art. 174 Traktatu o funkcjonowaniu Unii Europejskiej , jednym z trzech elementów polityki spójności , obok spójności gospodarczej i społecznej .
(...)

Rysunek 26: Fragment pliku test-mert.pl-en.translated.pl

Jedna linia zawierała jedno zdanie tłumaczenia. Plik z tłumaczeniami należało na początku sprowadzić do postaci

```
forma lemat informacje_morfologiczne
```

Tager *Clarin Pelcra* posłużył do dokonania ujednoznacznienia morfologicznego. Otrzymane wyniki tagowania przeniesiono manualnie do pliku w formacie *.json. Pobranie odpowiednich informacji z pliku *.json było możliwe dzięki napisanemu skryptowi w języku Python, przedstawionemu w oknie kodu (12).

```
def get_words_and_lemmas():

  with file('tagged_corpus_to_modify.json') as data_file:
    data = json.load(data_file)

  for sentence in data:
    print "<s>"
    print "1111:0 |t <s>"
    for word in sentence:
      form = word["orth"]
      lemma = word["lexes"][0]["base"]
      morphologic = word["lexes"][0]["CTag"]
      clean_morphologic = re.sub(r"n[0-9]\.*n*[0-9]*", "n", morphologic)
      print form + " " + lemma + " " + clean_morphologic
    print "</s>"
    print "\n".rstrip()

  get_words_and_lemmas()
```

Kod 12: Skrypt pobierający dane z pliku *.json i przedstawiający je w postaci *słowo lemat informacje_ morfologiczne*

Skrypt uruchomiono poleceniem konsolowym.

```
python morph_script.py > vowpal_wabbit_corpus.txt
```

Uzyskane wyniki zapisano w pliku `vowpal_wabbit_corpus.txt`.

Mając do dyspozycji dane w postaci identycznej jak na rysunku (24), możliwe było uruchomienie skryptu z okna kodu (11).

```
python script_input_VW.py vowpal_wabbit_corpus.txt > \
vowpal_wabbit_corpus_input.txt
```

Plik z danymi wejściowymi dla systemu *Vowpal Wabbit* został zapisany jako `vowpal_wabbit_corpus_input.txt`.

W celu uniknięcia błędów klasyfikacji, napisano skrypt, który umożliwia pozbycie się nadmiarowych linii i zbędnych duplikatów. Skrypt przedstawiono w oknie kodu (13).

```
def remove_duplicates():

  filename = str(sys.argv[1])
  file_to_read = open(filename, 'rw')
  list_of_lines = []
  temp_arr = []
  counter = 1

  file_lines = file_to_read.readlines()

  for single_line in file_lines:
    temp_arr.append(single_line)
    if single_line == "\r\n":
      unique_elements = sorted(np.unique(temp_arr), reverse=True)
      for single_elem in unique_elements:
        if single_elem == "\n":
          unique_elements.remove(single_elem)
        if "1111:0" in single_elem:
          counter += 1
          if counter > 1:
            unique_elements.remove(single_elem)
      counter = 0
      for single_elem in unique_elements:
        list_of_lines.append(single_elem.rstrip())
      temp_arr = []
      unique_elements = []

  for elem in list_of_lines:
    if elem == "1111:0 |t </s>":
      print str(elem) + "\n"
    else:
      print str(elem)

remove_duplicates()
```

Kod 13: Skrypt pozwalający na usunięcie duplikatów i nadmiarowych linii

Skrypt uruchomiono za pomocą polecenia konsolowego, a dane zapisano do pliku `vowpal_wabbit_corpus_input_no_dupl.txt`

```
python remove_duplicates.py vowpal_wabbit_corpus_input.txt > \
vowpal_wabbit_corpus_input_no_dupl.txt
```

Na pliku uruchomiono proces pozyskiwania predykcji w programie *Vowpal Wabbit*.

```
~/vowpal_wabbit/vowpalwabbit/vw -t -i model_train.ml -r \
preds.txt < vowpal_wabbit_corpus_input_no_dupl.txt
```

Uzyskane przy wykorzystaniu wytrenowanego modelu predykcje zostały zapisane do pliku `preds.txt`. W celu uniknięcia błędów na pliku z predykcjami uruchomiono polecenie konsolowe usuwające puste linie.

```
sed '/^$/d' preds.txt > preds_clean.txt
```

Spośród predykcji programu *Vowpal Wabbit* dla każdej formy słowa spośród list form, należało wybrać najniższą wartość predykcji. W celu dokonania tego, połączono plik z formami pobranymi ze słownika *Polimorf* z plikiem zawierającym predykcje. Napisano skrypt, który umożliwił wykonanie tej czynności, przedstawiony w oknie kodu (14).

```
1  import sys
2
3  file_with_preds = file(str(sys.argv[1]))
4  file_with_lines = file(str(sys.argv[2]))
5
6  arr_preds = []
7
8  lines_pred = file_with_preds.readlines()
9  lines_text = file_with_lines.readlines()
10
11 counter = 0
12
13 for line in lines_text:
14   if "1111:" in line:
15     print line.strip() + " /*****/ " + lines_pred[counter].strip()
16     counter += 1
17   else:
18     print line.strip()
```

Kod 14: Skrypt łączący plik z predykcjami oraz plik z formami pobranymi ze słownika Polimorf

Skrypt uruchamia się poleceniem konsolowym. Przyjmuje dwa argumenty w postaci nazw plików do połączenia, czyli pliku z predykcjami oraz pliku z danymi wejściowymi dla narzędzia *Vowpal Wabbit*.

```
python merging.py preds_clean.txt vowpal_wabbit_corpus_input_no_dupl.txt > \
merged_corpus.txt
```

Dane zostały zapisane w pliku `merged_corpus.txt`. Postać pliku wynikającego z połączenia została przedstawiona na rysunku (27).

```
shared |s w prep
1111:1 |t we f^loc f^wok /*****/ 1111:0.969061
(...)
1111:0 |t w f^loc f^nwok /*****/ 1111:0.49468

shared |s pełnia subst
1111:1 |t pełń f^pl f^gen f^f /*****/ 1111:1.042
(...)
1111:0 |t pełni f^sg f^loc f^f /*****/ 1111:0.725721

shared |s zgadzać fin
1111:1 |t zgadzasz f^sg f^sec f^imperf /*****/ 1111:1.02688
(...)
1111:0 |t zgadzam f^sg f^pri f^imperf /*****/ 1111:0.751831

shared |s się qub
1111:0 |t się /*****/ 1111:-0.299767

shared |s , interp
1111:0 |t , /*****/ 1111:-0.109747

shared |s i conj
1111:0 |t i /*****/ 1111:-0.442224

shared |s dlatego adv
1111:0 |t dlatego /*****/ 1111:-0.00200382
```

Rysunek 27: Postać pliku wynikająca z połączenia pliku z formami pobranymi ze słownika Polimorf oraz pliku z predykcjami

W celu późniejszego eksperymentowania z dodatkowymi cechami, napisano skrypt, który porównuje predykcję dla formy poprawnej (oznaczonej jako `1111:0`) z pozostałymi predykcjami i weryfikuje, czy *Vowpal Wabbit* określił najniższą predykcję dla właściwej formy. Głównym zadaniem skryptu jest obliczenie miary dokładności dla prezentowanych danych, czyli sprawdzenie w jak wielu przypadkach predykcje programu *Vowpal Wabbit* były zgodne z pierwotnymi oznaczeniami. Skrypt zaprezentowano w oknie kodu (15).

```
import sys

file_to_read = file(str(sys.argv[1]))
lines = file_to_read.readlines()
temp_arr = []
arr_for_one = []
counter = 0

for line in lines:
  if "/*****/" in line:
    new_elem = line.split("/*****/")
    assign = new_elem[0].split(" ")[0].split(":")[1]
    predict = float(new_elem[1].split(":")[1])
    pair = [predict, assign]
    arr_for_one.append(pair)
  elif line == "\n" or line == "\r\n" or "# EOS #" in line:
    temp_arr.append(sorted(arr_for_one))
    arr_for_one = []

number_of_elems = len(temp_arr)

for elem in temp_arr:
  if len(elem) != 0:
  first_part = elem[0]
  assign = first_part[1]
    if assign == '0':
      counter += 1

accuracy = (float(counter)/float(number_of_elems))*100

print "Accuracy: " + str(accuracy) + "%"
```

Kod 15: Skrypt umożliwiający obliczenie dokładności dla zestawu danych

Uruchomienie skryptu umożliwia polecenie konsolowe.

```
python accuracy.py merged_corpus.txt
```

Ostatnim etapem jest pozyskiwanie tłumaczeń i ponowne uruchomienie *BLEU* za pomocą skryptu dostępnego z poziomu systemu *Moses*.

Na pliku `merged_corpus.txt` uruchomiono skrypt `translate.py` pozwalający na odtworzenie tłumaczeń. Skrypt przedstawiono w oknie kodu (16).

```
import sys

file_to_read = file(sys.argv[1])
lines = file_to_read.readlines()

temp_arr = []
arr_for_one = []
counter = 0

for line in lines:
  if "/*****/" in line:
    new_elem = line.split("/*****/")
    assign = new_elem[0].split(" ")[0].split(":")[1]
    predict = float(new_elem[1].split(":")[1])
    word = new_elem[0].split(" ")[2]
    pair = [predict, assign, word]
    arr_for_one.append(pair)
  elif line == "\n" or line == "\r\n" or "# EOS #" in line:
    temp_arr.append(sorted(arr_for_one))
    arr_for_one = []

number_of_elems = len(temp_arr)

arr = []

for elem in temp_arr:
  if len(elem) == 0:
    continue
  else:
  word = elem[0][2]
    if word == "<s>" or word == "</s>" or word == "\n":
      continue
    elif word == ".":
      print "."
    else:
      print word,
```

Kod 16: Skrypt pozwalający na odtworzenie tłumaczeń

Skrypt jest uruchamiany poleceniem konsolowym. Tłumaczenia są za-

pisywane do pliku `translations_vw.txt`.

```
python translate.py merged_corpus.txt > translations_vw.txt
```

Przykładowa postać pliku z tłumaczeniami otrzymanymi za pomocą programu *Vowpal Wabbit* została przedstawiona na rysunku (28).

ten zalecenia w pełni zgadza się , i dlatego chciałbym ponownie wyrazić uznanie dla jakość ten sprawozdanie , któren udało się uzyskiwał szerokim , ponadpartyjnym poparcia .
na pisma - (LT) z zadowolenie przyjąłem ten dokumentów , ponieważ wieloletnim ramach finansowych na roków 2007 - 2013 staje się celu współpracy terytorialnego w jednym z trzy filarze polityki spójności UE , zastępując wspólnotowego inicjatywy INTERREG .
od tamtego pory stało się spójności terytorialnego , na mocy art 174 traktatu o funkcjonowania unii europejskiej , jednym z trzy elementem polityki spójności , obok spójności gospodarczej i społecznego .

Rysunek 28: Przykładowe tłumaczenia uzyskane dzięki klasyfikacji

Na uzyskanym pliku z tłumaczeniami ponownie uruchomiono skrypt `multi-bleu.perl`, aby sprawdzić, czy klasyfikacja pozwoliła na poprawę wyników.

```
~/mosesdecoder/scripts/generic/multi-bleu.perl -lc \
~/Studia/SEM/TESTY/Korpus\ 3000/test-mert.pl-en.true.pl < \
~/Studia/SEM/TESTY/Korpus\ 3000/translations_vw.txt
```

Oprócz eksperymentu podstawowego wykonano również kilka dodatkowych eksperymentów, polegających na wprowadzaniu dodatkowych cech.

By móc szybko dodawać nowe cechy, napisano skrypt w języku *Python*, przedstawiony w oknie kodu (17).

```
1 import sys
2 
3 file_to_read = file(sys.argv[1])
4 lines = file_to_read.readlines()
5 
6 temp_arr = []
7 arr_for_one = []
```

```
num_sentence = 0

counter = 0

for line in lines:
  if "shared" in line:
    new_elem = line.split(" ")
    print new_elem
    #if len(new_elem) > 3:
    if len(new_elem) > 4:
      #pos = new_elem[3]
      lemma = new_elem[2]
    else:
      #pos = new_elem[2]
      lemma = new_elem[2]
    #arr_for_one.append(pos)
    arr_for_one.append(lemma)
  elif "# EOS #" in line:
    temp_arr.append(arr_for_one)
    arr_for_one = []

for line in lines:
  if "shared" in line:
    if counter == 0:
      #print line.rstrip() + " posnext^" + temp_arr[num_sentence][counter
    +1].rstrip()
      print line.rstrip() + " lemmanext^" + temp_arr[num_sentence][counter
    +1].rstrip()
    elif counter > 0 and counter < len(temp_arr[num_sentence])-1:
      #print line.rstrip() + " posprev^" + temp_arr[num_sentence][counter
    -1].rstrip() + " posnext^" + temp_arr[num_sentence][counter+1].rstrip()
      print line.rstrip() + " lemmaprev^" + temp_arr[num_sentence][counter
    -1].rstrip() + " lemmanext^" + temp_arr[num_sentence][counter+1].rstrip
    ()
    elif counter == len(temp_arr[num_sentence])-1:
      #print line.rstrip() + " posprev^" + temp_arr[num_sentence][counter
    -1].rstrip()
      print line.rstrip() + " lemmaprev^" + temp_arr[num_sentence][counter
    -1].rstrip()
    counter += 1
  elif "# EOS #" in line:
    print line.rstrip()
    counter = 0
    num_sentence += 1
  else:
    print line.rstrip()
```

Kod 17: Skrypt umożliwiający dodawanie nowych cech

Kod (17) umożliwił dodawanie nowych cech w postaci form słów, lema-

tów i części mowy. W trakcie wykonywania eksperymentów ulegał drobnym modyfikacjom.

Skrypt można uruchomić poleceniem konsolowym.

```
python add_new_features.py vowpal_wabbit_corpus_input_no_dupl.txt > \
corpus_with_new_features.txt
```

Po dodaniu nowej cechy, możliwe było testowanie, odtworzenie tłumaczeń oraz uruchomienie *BLEU*. Proces powtarzano wielokrotnie, by sprawdzić, czy dodawanie nowych cech pozwala na uzyskanie lepszego wyniku dokładności oraz *BLEU*.

7 Wyniki przeprowadzonych eksperymentów

7.1 Metoda ewaluacji

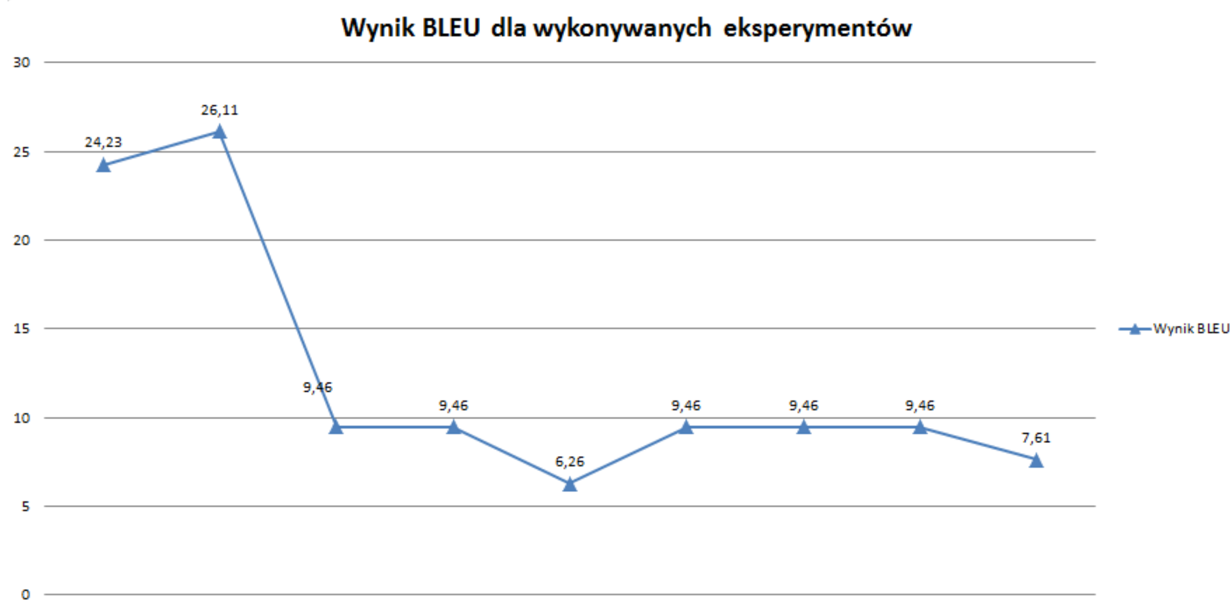
Metodą ewaluacji wybraną do oceny jakości tłumaczenia jest miara *BLEU*. Wbudowany skrypt `multi-bleu.perl` umożliwił natychmiastowe obliczenie wyniku dla kolejno uzyskiwanych tłumaczeń, wynikających z dodawania nowych cech do pliku wejściowego dla programu *Vowpal Wabbit*.

Uzyskane wyniki miary *BLEU* dla poszczególnych eksperymentów zostały przedstawione w tabeli (7).

Opis eksperymentu	Wynik BLEU
Eksperyment podstawowy Standardowy trening korpusu	24.23
Eksperyment podstawowy Standardowy trening korpusu + tuning MERT	26.11
Klasyfikacja Brak dodanych nowych cech	9.46
Klasyfikacja Dodanie poprzedniej i kolejnej części mowy do przestrzeni shared	9.46
Klasyfikacja Dodanie poprzedniej i kolejnej części mowy do przestrzeni shared z wykorzystaniem cech kwadratowych (opcja qst)	6.26
Klasyfikacja Dodanie poprzedniej i kolejnej części mowy oraz poprzedniego i kolejnego lematu do przestrzeni shared	9.46
Klasyfikacja Dodanie dwóch poprzednich oraz dwóch kolejnych lematów do przestrzeni shared	9.46
Klasyfikacja Dodanie dwóch poprzednich oraz dwóch kolejnych lematów oraz dwóch poprzednich i dwóch kolejnych części mowy do przestrzeni shared	9.46
Klasyfikacja Dodanie poprawnej formy słowa poprzedniego i następnego do bieżącego, poprawnego słowa	7.61

Tabela 7: Wyniki BLEU dla przeprowadzonych eksperymentów

Dodatkowo, zależność pomiędzy wynikami *BLEU* można zobaczyć na wykresie (Rysunek 29).



Rysunek 29: Wynik BLEU dla przeprowadzonych eksperymentów

7.2 Dokładność klasyfikacji

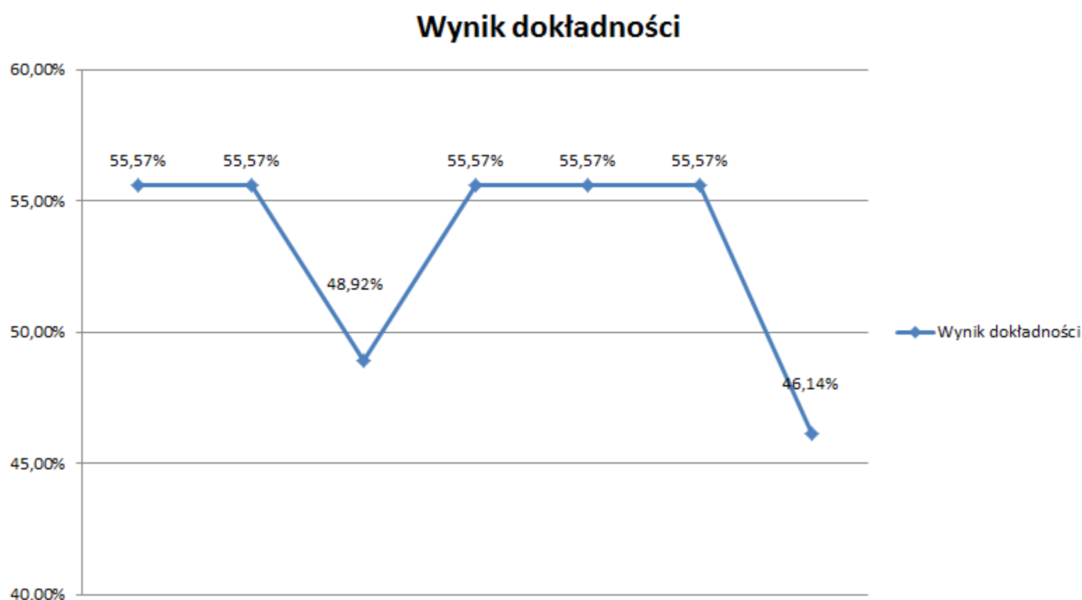
Miara dokładności służyła do jak najwłaściwszego doboru cech, które mogłyby pozytywnie wpłynąć na jakość tłumaczenia. Skrypt obliczający dokładność pozwalał na wstępne sprawdzenie, czy podany zestaw cech może przynieść oczekiwane rezultaty i ewentualnie znaleźć swoje odzwierciedlenie w wyniku *BLEU*.

Uzyskane wyniki miary *dokładności* dla poszczególnych eksperymentów zostały przedstawione w tabeli (8).

Opis eksperymentu	Wynik dokładności
Klasyfikacja Brak dodanych nowych cech	55.5711802004%
Klasyfikacja Dodanie poprzedniej i kolejnej części mowy do przestrzeni shared	55.5711802004%
Klasyfikacja Dodanie poprzedniej i kolejnej części mowy do przestrzeni shared z wykorzystaniem cech kwadratowych (opcja qst)	48.9210472459%
Klasyfikacja Dodanie poprzedniej i kolejnej części mowy oraz poprzedniego i kolejnego lematu do przestrzeni shared	55.5711802004%
Klasyfikacja Dodanie dwóch poprzednich oraz dwóch kolejnych lematów do przestrzeni shared	55.5711802004%
Klasyfikacja Dodanie dwóch poprzednich oraz dwóch kolejnych lematów oraz dwóch poprzednich i dwóch kolejnych części mowy do przestrzeni shared	55.5711802004%
Klasyfikacja Dodanie poprawnej formy słowa poprzedniego i następnego do bieżącego, poprawnego słowa	46.1367355287%

Tabela 8: Wyniki dokładności dla przeprowadzonych eksperymentów

Zależności pomiędzy uzyskanymi wynikami dokładności, można zobaczyć na wykresie (Rysunek 30).



Rysunek 30: Wynik dokładności dla przeprowadzonych eksperymentów

7.3 Jakość tłumaczenia

Uzyskana w wyniku procesu klasyfikacji jakość tłumaczenia nie jest wystarczająco zadowalająca. Po dokonaniu klasyfikacji, wynik *BLEU* znacznie spadł, co przełożyło się również na jakość tłumaczenia.

Na wykresach (Rysunek 29, Rysunek 30) widać istniejącą współzależność pomiędzy wynikami dokładności, a wynikami *BLEU*. Jeśli wynik dokładności spada, to przekłada się to bezpośrednio także na wynik *BLEU*.

Dodawanie nowych cech morfologicznych spowodowało zmniejszenie lub brak zmiany wyniku dokładności oraz zmniejszenie lub brak zmiany wyniku *BLEU*.

Ostatecznie, wykorzystanie klasyfikacji oraz wprowadzanie cech morfologicznych nie przyczyniło się do ulepszenia jakości tłumaczenia automatycznego.

8 Podsumowanie

Celem niniejszej pracy magisterskiej było wykazanie, że wykorzystanie klasyfikacji oraz dodawanie nowych cech morfologicznych w postaci lematów słów, czy części mowy, przyczynia się do ulepszenia jakości tłumaczenia automatycznego.

W wyniku przeprowadzonych eksperymentów otrzymano wyniki pozwalające stwierdzić, że wykorzystanie klasyfikacji nie pomaga w ulepszeniu jakości statystycznego tłumaczenia automatycznego.

Przyczyn takiego wyniku można z pewnością dopatrywać się w wielu miejscach. Być może wykorzystywanie procesu klasyfikacji nie jest dobrym rozwiązaniem dla ulepszania jakości tłumaczenia. Być może zastosowany model nie sprawdził się w przypadku ulepszania jakości tłumaczenia i dobrym pomysłem byłoby skorzystanie z innego modelu. Być może zestaw stosowanych cech był zbyt mały, by wpłynąć pozytywnie na jakość tłumaczenia i należałoby wprowadzić jeszcze więcej cech.

Niezależnie od ostatecznego wyniku eksperymentów, mogą one stanowić solidną podstawę do dalszych badań na temat ulepszania jakości tłumaczenia automatycznego poprzez wykorzystanie klasyfikacji oraz wprowadzanie cech morfologicznych języka do tłumaczenia.

9 Bibliografia

[1] Koehn, P. (2010). *Statistical Machine Translation.* Cambridge.

[2] Junczys-Dowmunt, M. (2008). *Wprowadzenie do metod statystycznych w tłumaczeniu automatycznym.* http://www.staff.amu.edu.pl/~inveling/pdf/Marcin_Junczys-Dowmunt_inve16.pdf [dostęp: 10.01.2016, 21:30].

[3] Ng, A. *CS229 Lecture notes.* http://cs229.stanford.edu/notes/cs229-notes1.pdf [dostęp: 06.03.2016, 19:50].

[4] Minkov, E., Toutanova, K., Suzuki, H. *Generating Complex Morphology for Machine Translation.* http://www.cs.cmu.edu/~einat/acl-07.pdf [dostęp: 03.05.2016, 00:05].

[5] Goldwater, S., McClosky, D. *Improving Statistical MT through Morphological Analysis.* http://dl.acm.org/citation.cfm?id=1220660 [dostęp: 29.05.2016, 11:30]

[6] Ng, A. *Mini-Batch Gradient Descent.* https://class.coursera.org/ml-003/lecture/106 [dostęp: 29.05.2016, 19:50]

[7] Langford, J. *Vowpal Wabbit.* https://github.com/JohnLangford/vowpal_wabbit/wiki/ [dostęp: 29.05.2016, 20:20]

[8] *Tager dla języka polskiego Clarin Pelcra.* http://clarin.pelcra.pl/tools/tagger [dostęp: 11.06.2016, 12:20]

[9] Przepiórkowski, A. *Zestaw znaczników morfosyntaktycznych.* http://nkjp.pl/poliqarp/help/plse2.html [dostęp: 11.06.2016, 13:00]

[10] *Statistical Machine Translation System Moses Official Website.* http://www.statmt.org/moses/ [dostęp: 21.06.2016, 22:26]

[11] *Moses Github Official Repository.* https://github.com/moses-smt/mosesdecoder [dostęp: 21.06.2016, 22:28]

[12] *Multiclass Classification in Vowpal Wabbit.* http://www.umiacs.umd.edu/~hal/tmp/multiclassVW.html [dostęp: 21.06.2016, 22:38]

[13] *Adding a Feature to Moses.* http://kentonmurray.com/blogs/addingafeaturetomoses.html [dostęp: 21.06.2016, 22:40]

[14] *Multi-class classification: One-vs-all.* https://class.coursera.org/ml-005/lecture/38 [dostęp: 21.06.2016, 22:42]

[15] Langa, N., Wojak, A. *Ewaluacja systemów tłumaczenia automatycznego.* https://psi.wmi.amu.edu.pl/uploads/Langa-mgr.pdf [dostęp: 21.06.2016, 22:44]

[16] *Analizator morfologiczny Morfeusz.* http://sgjp.pl/morfeusz/ [dostęp: 21.06.2016, 23:05]

[17] *Tree Tagger - a part-of-speech tagger for many languages.* http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/ [dostęp: 21.06.2016, 23:08]

10 Lista rysunków

1 Przykład prostego dwujęzycznego korpusu równoległego 10
2 Fragment próbki dwujęzycznego korpusu równoległego OpenSubtitles 11
3 Model zaszumionego kanału - rysunek poglądowy (Koehn, 2010, [1]) 14
4 Przykład urównoleglenia dla tekstu *Anna writes a book* 17
5 Przykład urównoleglenia dla tekstu *of course Anna sits in the waiting room lots of time* 18
6 Mapowanie części mowy oraz słów 19
7 Reguła syntaktyczna 20
8 Uzyskiwanie dopasowania za pomocą modułu *exact* (na podstawie pracy [15]) 29
9 Uzyskiwanie dopasowania za pomocą modułów *stem* (fioletowa linia) i *synonym* (zielona linia) (na podstawie pracy [15]) . 30
10 Uzyskiwanie dopasowania za pomocą modułu *paraphrase* (na podstawie pracy [15]) 30
11 Zbiór wszystkich dopasowań (na podstawie pracy [15]) 30
12 Wykres przedstawiający dane dotyczące ilości znaków najdłuższego słowa w zdaniu w zależności od całkowitej ilości znaków w zdaniu 33
13 Wykres przedstawiający linię regresji dla przykładowych danych. 34
14 Klasy wyróżnione spośród zestawu treningowego 42
15 Proces klasyfikacji binarnej przeprowadzany w ramach modelu One Against All (1) 42
16 Proces klasyfikacji binarnej przeprowadzany w ramach modelu One Against All (2) 42
17 Proces klasyfikacji binarnej przeprowadzany w ramach modelu One Against All (3) 43
18 Wyniki analizy morfologicznej w formie zrzutu ekranu 44
19 Wykres prezentujący działanie faktorowego modelu tłumaczenia 58

20 Wyniki analizy morfologicznej przeprowadzonej programem Morfeusz Polimorf (Morfeusz Polimorf, [16]) 76
21 Wyniki dla generowania form w programie Morfeusz Polimorf (Morfeusz Polimorf, [16]) 77
22 Wyniki działania programu Tree Tagger (Tree Tagger, [17]) . . 78
23 Wyniki tagowania przeprowadzonego za pomocą programu Clarin Pelcra w formacie *.json (Clarin Pelcra, [8]) 79
24 Fragment nieprzetworzonego pliku treningowego 90
25 Fragment pliku treningowego w formacie wejściowym dla narzędzia Vowpal Wabbit 94
26 Fragment pliku test-mert.pl-en.translated.pl 96
27 Postać pliku wynikająca z połączenia pliku z formami pobranymi ze słownika Polimorf oraz pliku z predykcjami 100
28 Przykładowe tłumaczenia uzyskane dzięki klasyfikacji 103
29 Wynik BLEU dla przeprowadzonych eksperymentów 107
30 Wynik dokładności dla przeprowadzonych eksperymentów . . . 109

11 Lista tabel

1 Rozkład prawdopodobieństw tłumaczeń leksykalnych 13

2 Estymaty maksymalnego prawdopodobieństwa tłumaczenia dla określonych trzywyrazowych sekwencji słów 16

3 Przykładowy zestaw danych dotyczący ilości znaków najdłuższego słowa w zdaniu w zależności od całkowitej ilości znaków w zdaniu 32

4 Przykładowy zestaw danych sklasyfikowanych względem kategorii 36

5 Trzy przykłady treningowe z zestawu danych dotyczącego irysów 62

6 Porównanie predykcji programu Vowpal Wabbit z przykładami z zestawu testowego 66

7 Wyniki BLEU dla przeprowadzonych eksperymentów 106

8 Wyniki dokładności dla przeprowadzonych eksperymentów . . 108

12 Lista programów

1 Pojedyncza aktualizacja parametrów θ dla algorytmu Mini-Batch Gradient Descent (na podstawie [6]) 40
2 Pojedyncza aktualizacja parametrów θ dla algorytmu Stochastic Gradient Descent (na podstawie [6]) 41
3 Szkielet klasy dla cechy bezstanowej (na podstawie [11]) . . . 50
4 Szkielet implementacji dla cechy bezstanowej w pliku *.cpp (na podstawie [11]) 51
5 Deklaracje metod (na podstawie [11]) 52
6 Przykładowa implementacja dla metody EvaluateInIsolation (na podstawie [11]) 53
7 Szkielet klasy dla cechy stanowej (na podstawie [11]) 54
8 Szkielet implementacji dla cechy stanowej w pliku *.cpp (na podstawie [11]) 54
9 Deklaracje metod (na podstawie [11]) 55
10 Przykładowa implementacja dla metody EvaluateWhenApplied 55
11 Skrypt służący do przedstawienia danych w postaci preferowanej przez program Vowpal Wabbit 91
12 Skrypt pobierający dane z pliku *.json i przedstawiający je w postaci *słowo lemat informacje_morfologiczne* 97
13 Skrypt pozwalający na usunięcie duplikatów i nadmiarowych linii 98
14 Skrypt łączący plik z predykcjami oraz plik z formami pobranymi ze słownika Polimorf 99
15 Skrypt umożliwiający obliczenie dokładności dla zestawu danych 101
16 Skrypt pozwalający na odtworzenie tłumaczeń 102
17 Skrypt umożliwiający dodawanie nowych cech 103