Łukasz Dębowski

Tagowanie i dezambiguacja morfosyntaktyczna. Przegląd metod i oprogramowania

Nr 934

Warszawa, listopad 2001

Streszczenie

Przedstawiono aktualne metody i dostępne oprogramowanie do automatycznego tagowania i uczenia się tagerów. Omówiono skuteczność zastosowań przedstawionych metod i oprogramowania do dezambiguacji morfosyntaktycznej tekstów w językach naturalnych.

Słowa kluczowe: statystyczne przetwarzanie języka naturalnego, tagowanie częściami mowy, uczenie się maszynowe, ukryte modele Markowa, modelowanie maksimum entropii, uczenie się oparte na transformacjach, uczenie się oparte na pamięci

Tagging and morphosyntactic disambiguation. A review of methods and software

Abstract

We present state-of-the-art methods and available software for automatic tagging and tagger learning. Furthermore, we discuss applicability of presented methods and software for morphosyntactic disambiguation of texts in natural languages.

Keywords: statistical natural language processing, part-of-speech tagging, machine learning, hidden Markov models, maximum entropy modelling, transformation-based learning, memory-based learning

Spis treści

Spis diagramów

Spis tabel

Wstęp

W ogólności, teksty w języku naturalnym i ich przetwarzanie nie poddają się łatwemu modelowaniu matematycznemu. Przy zmniejszaniu do zera przeciętnej liczby błędów popełnianych przez program przetwarzający język naturalny zwykle następuje gwałtowny wzrost złożoności czasowej i pamięciowej algorytmu, który ten program musi implementować. Istnieje jednak klasa zadań, które daje się wykonać dość dobrze zakładając prawdziwość stosunkowo prostych modeli języka. Wiele z tych zadań można przedstawić jako kaskady segmentacji i tagowania. Zadaniami należacymi do tej klasy są np.: rozpoznawanie mowy, rozpoznawanie pisma (OCR), tagowanie częściami mowy, dezambiguacja morfosyntaktyczna [25, 29]. Dwa pierwsze zadania można przedstawić jako złożenie segmentacji i tagowania (tokenami są fragmenty sygnałów akustycznych lub danych graficznych, tagami są słowa lub litery). Dwa ostatnie zadania w większości języków sprowadzają się wyłącznie do tagowania (tokenami są słowa lub analizy morfologiczne słów, tagami są analizy morfosyntaktyczne).

Automatyczne tagowanie jest praktycznym zastosowaniem lingwistyki komputerowej, w którym od lat 90. XX wieku intensywnie wykorzystuje się i testuje różne techniki uczenia się maszynowego. Pierwszym celem niniejszego raportu jest jednolite i dokładne przedstawienie rozwiniętych dotychczas metod automatycznego tagowania oraz metod maszynowego uczenia się tagowania. Metody te omawiamy w rozdziale 1. Większość metod opiera się na standardowych metodach uczenia się klasyfikacji i może być stosowana do różnych problemów. Osobom bardziej zainteresowanym maszynowym uczeniem się i lingwistyką komputerową proponujemy podręczniki [29, 31]. Drugim celem raportu jest przedstawienie dostępnego oprogramowania implementującego omówione metody na potrzeby tagowania częściami mowy oraz dezambiguacji morfosyntaktycznej. Oprogramowanie, specyfikę zagadnienia i przeprowadzone dotychczas testy dyskutujemy w rozdziale 2. W szczególności interesuje nas przydatność dostępnego oprogramowania do dezambiguacji morfosyntaktycznej tekstów w językach o bogatej fleksji (np. słowiańskich). Niniejszy raport jest prawdopodobnie pierwszym opracowaniem w języku polskim dotyczącym automatycznego tagowania. Dla terminów, którymi się posługujemy, często podajemy w nawiasach odpowiedniki angielskie. Schematy algorytmów przedstawiamy w pseudokodzie o składni zbliżonej do składni języka Perl.

Niniejsze opracowanie było częściowo finansowane z grantu KBN nr 7 T11C 043 20.

1 Metody i algorytmy

1.1 Tagowanie jako problem uczenia się klasyfikacji

W przetwarzaniu języka naturalnego, tagowanie (tagging) M to odwzorowanie $W \mapsto M(W)$ tekstu segmentowanego $W = (W_1, W_2, ..., W_L)$, $W_i \in \mathcal{W}$, w ciąg tagów $M(W) = (M(W)_1, M(W)_2, ..., M(W)_L)$, $M(W)_i \in \mathcal{T}$. Token (token) W_i to i-ty element tekstu segmentowanego $W = (W_1, W_2, ..., W_L)$, $W_i \in \mathcal{W}$. Tag (tag) T_i to i-ty element ciągu tagów $T = (T_1, T_2, ..., T_L)$, $T_i \in \mathcal{T}$. Pojedynczy tag T_i traktowany jest jako przypisany pojedynczemu tokenowi W_i, lecz zależność między wartościami T_i i W_i może zależeć od kontekstu. Parę $(W, M(W))$ nazywamy tekstem anotowanym. Dla języków zapisywanych pismem alfabetycznym w wielu zastosowaniach tokeny W_i odpowiadają kolejnym słowom graficznym (spacja lub dowolny znak przestankowy rozpoczyna nowy token). Zbiór tagów $\mathcal{T}$, nazywany także tagsetem (tagset), zależy od szczególnego przeznaczenia tekstu anotowanego $(W, M(W))$. Program obliczający tagowanie nazywamy tagerem (tagger).

W maszynowym uczeniu rozpatruje się problem zwany klasyfikacją, który jest zbliżony do tagowania. Klasyfikacja (classification) Q to odwzorowanie $D \mapsto Q(D)$ odwzorowujące ciąg kontekstów $D = (D_1, D_2, ..., D_L)$, $D_i \in \mathcal{D}$, w ciąg tagów $Q(D) = (Q(D)_1, Q(D)_2, ..., Q(D)_L)$, $Q(D)_i \in \mathcal{T}$. Program obliczający klasyfikację nazywa się klasyfikatorem (classifier). Teoretyczna różnica między klasyfikacją a tagowaniem jest następująca. Dla klasyfikacji zakłada się, że przy danym D_i wartość $Q(D)_i$ może zależeć wyłącznie od częstości wartości $(D_j, Q(D)_j)$ w $(D, Q(D))$. Dla tagowania zakłada się, że przy danym W_i wartość $M(W)_i$ zależy nie tylko od częstości wartości $(W_j, M(W)_j)$ w $(W, M(W))$, ale także od prawidłowości w ich liniowym uporządkowaniu. Z różnego rodzaju rozważań teoretyczno-lingwistycznych wynika, że zależność wartości $M(W)_i$ od liniowego uporządkowania ciągu tokenów W może być bardzo dalekozasięgowa.

W praktyce, w przypadku wielu rodzajów tagowania w lingwistyce komputerowej spełniona jest następująca empiryczna zależność: Dla tagowania M tekstu segmentowanego $W = (W_1, ..., W_L)$ dokonywanego przez ekspertów (językoznawców) dla przytłaczającej większości [1] $i \in \{1, ..., L\}$ zachodzi

$$M(W)_i = F(W_{i-C}, ..., W_{i+C}, M(W)_{i-C}, ..., M(W)_{i-1}, M(W)_{i+1}, ..., M(W)_{i+C}), \quad (1)$$

gdzie $C < \infty$, F zaś jest pewną funkcją. Oznacza to, że tagowanie M można przybliżać przez klasyfikację Q, gdzie konteksty mają postać $D_i = (W_{i-C}, ..., W_{i+C}, M(W)_{i-C}, ..., M(W)_{i-1}, M(W)_{i+1}, ..., M(W)_{i+C})$,

[1] Powyżej 95%.

$Q(D)_i = M(W)_i$. [2] Odpowiednio, istnieją algorytmy, które wykorzystując statystyczną tendencję (1) obliczają pewne przybliżenie $M(W)$ dla danego W w czasie liniowym $O(L)$. Zwyczajowo tagerami nazywa się implementacje algorytmów o wyłącznie takiej złożoności czasowej. Algorytmy takich tagerów oparte są zwykle na programowaniu dynamicznym lub na skończonych kaskadach automatów przepisujących (transducers). Algorytmy te składają się one ze stosunkowo prostej struktury ogólnej sparametryzowanej znaczną liczbą nieznanych parametrów. [3] Tager implementujący algorytm K o parametrach $\theta \in \Theta$ zapisujemy jako $K(\theta)$. Algorytm K nazywamy także klasą parametryczną K. Tagowanie obliczane przez tager A oznaczamy jako M_A.

Praktycznie żaden program przetwarzający język naturalny nie działa bezbłędnie, tzn. wyniki przetwarzania danych przez program odbiegają od wyników podawanych odpowiednio przez ekspertów. W tej sytuacji pojawia się problem porównywania działania różnych programów wykonujących to samo zadanie. Oznaczmy zdanie „Program A wykonuje dane zadanie nie gorzej niż B" jako $A \succ B$. Jeżeli relacja $\succ$ jest porządkiem liniowym, tzn. $\forall A, B : (A \succ B) \vee (A \succ B)$ oraz $\forall A, B, C : (A \succ B) \wedge (B \succ C) \implies (A \succ C)$, to istnieje funkcja oceniająca U o wartościach rzeczywistych taka, że $U(A) \geqslant U(B) \iff A \succ B$. [4] Znajomość wartości funkcji oceniającej $U(A)$ i $U(B)$ pozwala porównać działanie dwóch różnych programów A i B. Dla klasy parametrycznej K można zdefiniować program $\hat{K}$, który wykonuje zadanie najlepiej,

$$\hat{K} = K(\hat{\theta}), \quad \hat{\theta} = \arg\max_{\theta \in \Theta} U(K(\theta)). \tag{2}$$

Algorytmy tagerów sparametryzowane są dużą liczbą parametrów, gdzie tylko niektóre wartości parametrów, nieznane a priori, zadają programy produkujące dostatecznie dobre wyniki. Pożądane jest zautomatyzować nie tylko tagowanie, ale także poszukiwanie optymalnych parametrów $\hat{\theta}$ dla danego algorytmu K. Automatyczne poszukiwanie optymalnych parametrów nazywa się uczeniem tagera. Aby uczenie tagera było wykonalne, przestrzeń Θ powinna dawać się efektywnie przeszukiwać w celu maksymalizacji $U(K(\theta))$, $\theta \in \Theta$, a także powinno istnieć łatwo obliczalne i dobre przybliżenie funkcji U. Przybliżenia funkcji U powinny mieć coś wspólnego z empirią, dlatego też stosowane przybliżenia U mają postać

$$U(A) \approx -d\left(M_A(W); M(W)\right) + \text{const}, \tag{3}$$

gdzie $(W, M(W))$ jest pewnym tekstem anotowanym przez ekspertów, d zaś jest pewną pseudoodległością, tzn. $d(x; x) = 0$ oraz $d(x; y) > 0$ dla $x \neq y$.

1.2 Podział algorytmów tagowania

Podstawowe algorytmy tagowania stosowane obecnie można podzielić w sposób następujący:

- oparte na modelach probabilistycznych (oparte na niestacjonarnych modelach Markowa):
  - oparte na ukrytych modelach Markowa,
  - oparte na modelowaniu maksimum entropii,
- nieoparte na modelach probabilistycznych (jawnie):
  - oparte na pamięci,
  - oparte na transformacjach,

Wszystkie wymienione algorytmy posiłkują się wnioskowaniem ilościowym na podstawie danych albo zarówno w czasie uczenia się i tagowania nowych tekstów (tagery oparte na modelach probabilistycznych i oparte na pamięci), albo wyłącznie w czasie uczenia się (tagery oparte na transformacjach).

Algorytmy oparte na modelach probabilistycznych wyznaczają $M_{\hat{K}}(W) = \hat{T}$ jako najbardziej prawdopodobny ciąg tagów przy założeniu pewnego modelu prawdopodobieństwa $p(T|W)$ ciągu tagów T przy danym tekście segmentowanym W, tzn.

$$\hat{T} = \arg\max_{T} p(T|W) \tag{4}$$

[2] Podobnie przez klasyfikację można przybliżać inne zadanie z przetwarzania języka naturalnego nazywane segmentacją. Segmentacja (segmentation) S to odwzorowanie $A \mapsto S(A)$ tekstu ciągłego $A \in \mathcal{A}^*$, $|\mathcal{A}| < \infty$, w tekst segmentowany $S(A) = (S(A)_1, S(A)_2, ..., S(A)_L)$, $S(A)_i \in \mathcal{W}$, gdzie $\mathcal{W} \subset \mathcal{A}^*$ oraz zachodzi równość konkatenacji $S(A)_1 S(A)_2 ... S(A)_L = A$. Segmentacja zależy od języka naturalnego i przeznaczenia tekstu segmentowanego. Aby sprowadzić segmentację do tagowania (a następnie do klasyfikacji), każdemu elementowi $A_i \in \mathcal{A}$ tekstu ciągłego $A = A_1 ... A_N$ należy przypisać tag $M(A)_i \in \{0, 1\}$, gdzie $M(A)_i = 1$ wtedy i tylko wtedy, gdy istnieje j takie, że $A_1 ... A_i = S(A)_1 ... S(A)_j$. Program obliczający segmentację nazywamy tokenizerem (tokenizer).

[3] Zauważmy, że w prostej zależności (1) występuje także znaczna liczba nieznanych parametrów, którymi są wartości funkcji f na całej jej dziedzinie.

[4] Warto zauważyć, że jeżeli $U(A) \geqslant U(B) \iff A \succ B$, to także $V(A) \geqslant V(B) \iff A \succ B$, gdzie $V(A) = g(U(A))$, a g jest dowolną funkcją monotonicznie rosnącą.

Wszystkie tagery oparte na modelach probabilistycznych wykorzystują założenie, że model prawdopodobieństwa $p(T|W)$ jest niestacjonarnym modelem Markowa skończonego rzędu. To znaczy, zakłada się, że dla pewnego $n < \infty$ zachodzi

$$p(T|W) = \prod_{i=1}^{L} p^{(i)}(T_i|W, T_{i-1}, ..., T_{i-n}), \tag{5}$$

gdzie $T = (T_1, ..., T_L)$, $W = (W_1, ..., W_L)$.

Algorytmy oparte na pamięci wyznaczają tagowanie $M_{\hat{K}}(W) = \hat{T}$ jako ciąg tagów, w którym każde $\hat{T}_i$ jest tagiem T_k^{tr} pewnej pary $(D_k^{\mathrm{tr}}, T_k^{\mathrm{tr}})$ ze zbioru wcześniej zapamiętanych przykładów, dla której wcześniej określona odległość $\Delta(D_k^{\mathrm{tr}}, \hat{D}_i)$ od bieżącego kontekstu $\hat{D}_i$ jest najmniejsza. To znaczy,

$$\hat{T}_i = T_k^{\mathrm{tr}}, \quad \text{gdzie} \quad k = \underset{j \in \{1,...,L_{\mathrm{tr}}\}}{\arg\min} \Delta(D_j^{\mathrm{tr}}, \hat{D}_i) \tag{6}$$

oraz $\hat{D}_i = (W_{i-C}, ..., W_{i+C}, \hat{T}_{i-1}, ..., \hat{T}_{i-n})$ dla pewnego $n < \infty$. Tagery oparte na pamięci nie są jawnie oparte na modelowaniu probablistycznym, jakkolwiek możliwa jest ich interpretacja w terminach modeli probabilistycznych.

Algorytmy oparte na transformacjach rozpoczynają od wyznaczenia pewnego trywialnego tagowania początkowego $T^{(0)}$, a następnie obliczają

$$T_i^{(n+1)} = F_{n+1}(W_{i-C}, ..., W_{i+C}, T_{i-C}^{(n)}, ..., T_{i+C}^{(n)}), \tag{7}$$

gdzie $(F_1, ..., F_N)$ jest ciągiem bardzo prostych przekształceń symbolicznych, natomiast $M_{\hat{K}}(W) = T^{(N)}$.

Dla każdego z algorytmów tagowania istnieją swoiste algorytmy uczenia się tagera. Szczegółowe zasady działania i uczenia poszczególnych typów tagerów omawiamy w podrozdziałach 1.5-1.12.

1.3 Proste funkcje oceniające

W celu oceny działania tagerów stosuje się proste funkcje oceniające U oparte na liczbie błędów popełnianych przez tager. Zakłada się, że rankingi programów produkowane przez takie funkcje oceniające dość dobrze pokrywają się z uszeregowaniami subiektywnie dokonywanymi przez ludzi. Definicje funkcji oceniających opartych na liczbie błędów zależą od tego, czy anotacje produkowane przez tager A dla poszczególnych tokenów są pojedynczymi wartościami wskazywanymi przez ekspertów $(M_A(W)_i \in \mathcal{T}, M(W)_i \in \mathcal{T})$, czy też ich zbiorami $(M_A(W)_i \in 2^{\mathcal{T}}, M_A(W)_i \in \mathcal{T})$. W definicjach przyjmujemy, że anotowany tekst segmentowany W składa się z L tokenów $(W = W_1, ..., W_L)$.

1. W przypadku, gdy $M_A(W)_i \in \mathcal{T}, M(W)_i \in \mathcal{T}$, definiuje się

   - ułamek błędów (error rate) [5]

   $$\mathrm{Err}(A) = \frac{\sum_{i=1}^{L} \delta(M_A(W)_i \neq M(W)_i)}{L}, \tag{8}$$

   - akuratność (accuracy)

   $$\mathrm{Acc}(A) = \frac{\sum_{i=1}^{L} \delta(M_A(W)_i = M(W)_i)}{L} = 1 - \mathrm{Err}(A). \tag{9}$$

   Przyjmuje się $U(A) \approx \mathrm{Acc}(A)$ [29]. Zachodzi $\mathrm{Err}(A), \mathrm{Acc}(A) \in [0, 1]$.

2. W przypadku, gdy $M_A(W)_i \in 2^{\mathcal{T}}, M(W)_i \in \mathcal{T}$, definiuje się

   - zwrot (recall)

   $$R(A) = \frac{\sum_{i=1}^{L} \delta(M(W)_i \in M_A(W)_i)}{L}, \tag{10}$$

[5] Funkcję $\delta(\phi)$ definiujemy jako

$$\delta(\phi) = \begin{cases} 1, & \text{jeśli } \phi \text{ jest prawdziwe,} \\ 0, & \text{jeśli } \phi \text{ jest fałszywe.} \end{cases}$$

- precyzję (precision)

$$P(A) = \frac{\sum_{i=1}^{L} \delta(M(W)_i \in M_A(W)_i)}{\sum_{i=1}^{L} |M_A(W)_i|}, \tag{11}$$

- miarę F (F measure)

$$\frac{1}{F(A,\alpha)} = \alpha \frac{1}{P(A)} + (1-\alpha)\frac{1}{R(A)}, \qquad \alpha \in \{0,1\}. \tag{12}$$

Przyjmuje się $U(A) \approx F(A,\alpha)$. Zachodzi $R(A)$, $P(A)$, $F(A) \in [0,1]$. Wybór stałej α zależy od tego, o ile bardziej w zastosowaniu liczy się obecność właściwego $M(W)_i$ w $M_A(W)_i$ od małej liczności zbioru $M_A(W)_i$ [29].

3. Pewien przypadek nierozważany w literaturze poświęconej automatycznemu tagowaniu zachodzi, gdy wartości tagów są elementami pewnej hierarchii typów. Wówczas $M_A(W)_i \in 2^{\mathcal{T}}, M(W)_i \in 2^{\mathcal{T}}$, gdzie $\mathcal{T}$ jest zbiorem typów maksymalnych. Ponieważ zbiory możliwych wartości $M_A(W)_i$ i $M(W)_i$ a priori są równe, teoretycznie przypadek ten sprowadza się do przypadku 1. W praktyce tak nie jest, gdyż akuratność $\mathrm{Acc}(A)$ karze program jednakowo za dowolną nierówność $M_A(W)_i \neq M(W)_i$. Dlatego też proponujemy zdefiniować

- niedopatrzenie

$$G(A) = \frac{\sum_{i=1}^{L} |M_A(W)_i \setminus M(W)_i|}{\sum_{i=1}^{L} |M(W)_i|}, \tag{13}$$

- nadgorliwość

$$S(A) = \frac{\sum_{i=1}^{L} |M(W)_i \setminus M_A(W)_i|}{\sum_{i=1}^{L} |M(W)_i|}, \tag{14}$$

- ocenę wypadkową

$$V(A,\alpha) = -\left[\alpha S(A) + (1-\alpha)G(A)\right], \qquad \alpha \in \{0,1\}. \tag{15}$$

Przyjmujemy $U(A) \approx V(A,\alpha)$. Zachodzi $G(A) \geqslant 0$, $S(A) \in [0,1]$, $V(A) \leqslant 0$. Wybór stałej α zależy od tego, o ile bardziej na ogólnej ocenie programu ważą przypadki, gdy właściwy tag jest bardziej szczegółowy ($M_A(W)_i \supset M(W)_i$), w stosunku do przypadków, gdy właściwy tag jest mniej szczegółowy ($M_A(W)_i \subset M(W)_i$).

1.4 Trenowanie, testowanie i przeuczenie

W praktyce tager z dowolnej klasy parametrycznej K uczony jest na pewnym tekście anotowanym $(W^{\mathrm{tr}}, M(W^{\mathrm{tr}}))$, nazywanym danymi treningowymi (training data). To znaczy, optymalizacja (2) dokonywana jest automatycznie dla przybliżenia (3), gdzie $W = W^{\mathrm{tr}}$. Tekst treningowy W^{tr} jest niewielkim reprezentantem wszystkich tekstów segmentowanych W, do których tagowania tager jest przeznaczony. Jest tak, gdyż po pierwsze tagowanie $M(W^{\mathrm{tr}})$ przygotowane przez ekspertów jest drogie, a po drugie tager ma właśnie służyć do automatycznego tagowania tekstów jeszcze nie anotowanych.

Oznaczmy $U_\alpha = U$ dane przybliżeniem (3) dla $W = W^\alpha$. Oznaczmy $\hat{\theta}_\alpha = \hat{\theta}$, $\hat{K}_\alpha = \hat{K}$ dane definicją (2) dla $U = U_\alpha$. Z definicji wynika, że $U_\beta(\hat{K}_\alpha) \leqslant U_\beta(\hat{K}_\beta)$. Sformułowania, że klasa parametryczna K jest podatna na przeuczenie, używa się w dwóch nieco odmiennych znaczeniach:

- W pierwszym znaczeniu mówimy, że klasa parametryczna K jest podatna na przeuczenie, gdy dla typowych W_α, W_β program $\hat{K}_\alpha$ wyuczony na $(W_\alpha, M(W_\alpha))$ przetwarza dane W_β znacznie gorzej niż program $\hat{K}_\beta$ wyuczony na $(W_\alpha, M(W_\alpha))$, tzn. $U_\beta(\hat{K}_\alpha) \ll U_\beta(\hat{K}_\beta)$.

- W drugim znaczeniu przeuczeniem nazywa się pewną szczególną sytuację mogącą wystąpić, gdy proces uczenia się programu jest rozciągnięty na iteracje. W wyniku tych iteracji otrzymuje się pewne programy $\hat{K}_\alpha^i$, $i \in N$, gdzie $\hat{K}_\alpha = \hat{K}_\alpha^n$, a n jest największą liczbą taką, że $U_\alpha(\hat{K}_\alpha^i) > U_\alpha(\hat{K}_\alpha^{i-1})$ dla wszystkich $i \in \{1,...,n\}$. Niech m będzie największą liczbą taką, że $U_\beta(\hat{K}_\alpha^i) > U_\beta(\hat{K}_\alpha^{i-1})$ dla wszystkich $i \in \{1,...,m\}$. Mówimy, że klasa parametryczna K jest podatna na przeuczenie, gdy $m \ll n$ i $U_\beta(\hat{K}_\alpha^m) > U_\beta(\hat{K}_\alpha^j)$ dla każdego $j \in \{m+1,...,n\}$.

Modele parametryzacji obecnych tagerów są stosunkowo proste i nadal mają mało wspólnego z rzeczywistym sparametryzowaniem języka naturalnego. Z tego powodu często są one podatne na przeuczenie. Praktyczna ocena działania programu zależy jednakże od tego, jak dobrze wykonuje on tagowanie wszystkich danych, do których przetwarzania jest on przeznaczony, a nie wyłącznie tych, na których był on uczony. Rankingi klas parametrycznych K sporządza się zatem według wartości $U_{\text{test}}(\hat{K}_{\text{tr}})$, a nie według wartości $U_{\text{tr}}(\hat{K}_{\text{tr}})$, gdzie $(W^{\text{test}}, M(W^{\text{test}}))$ jest pewnym tekstem anotowanym nazywanym danymi testowymi (test data). Dane testowe $(W^{\text{test}}, M(W^{\text{test}}))$ muszą być rozłączne z danymi treningowymi $(W^{\text{tr}}, M(W^{\text{tr}}))$ i zwykle są znacznie krótsze. Zwykle przyjmuje się, że długość danych testowych L_{test} stanowi 5%-10% długości danych treningowych L_{tr} [29].

Podatność na przeuczenie dla niektórych klas parametrycznych $K^{(k)}$ często idzie w parze z relatywnie wysokimi wartościami $U_{\text{tr}}(\hat{K}^{(k)}_{\text{tr}})$. Dla skończonej liczby programów $\hat{K}^{(1)}_{\text{tr}}, ..., \hat{K}^{(n)}_{\text{tr}}$ z różnych klas oznaczmy przez $\hat{k}$ indeks programu najlepiej przetwarzającego dane treningowe, tzn.

$$\hat{k} = \arg\max_{k \in \{1,...,n\}} U_{\text{tr}}(\hat{K}^{(k)}_{\text{tr}}).$$

Niekiedy możliwe jest skonstruowanie w oparciu o programy $\hat{K}^{(1)}_{\text{tr}}, ..., \hat{K}^{(n)}_{\text{tr}}$ nowej klasy parametrycznej K, gdzie $K(\theta) = K(\hat{K}^{(1)}_{\text{tr}}, ..., \hat{K}^{(n)}_{\text{tr}}; \theta)$, takiej, że najlepszy program $\hat{K}_\alpha$ ma wartość $U_{\text{tr}}(\hat{K}_\alpha)$ zbliżoną do $U_{\text{tr}}(\hat{K}^{(\hat{k})}_{\text{tr}})$, a zarazem klasa K jest mniej podatna na przeuczenie w sensie pierwszym niż klasa $K^{(\hat{k})}$. Konstrukcję tę nazywa się wygładzaniem. Klasycznym przypadkiem wygładzania jest interpolacja liniowa stosowana w modelach probabilistycznych, o której więcej piszemy w podrozdziale 1.7. Wygładzone tagery $K(\theta)$ są sparametryzowane pewnymi nowymi parametrami θ. Aby wyznaczyć program najlepszy, trzeba wyznaczyć optymalną wartość $\hat{\theta}$ jak we wzorze (2). Dla najczęściej używanych parametryzacji, wartości tej nie można wyznaczyć używając do oceniania danych treningowych $(W^{\text{tr}}, M(W^{\text{tr}}))$, gdyż dla tych parametryzacji zachodzi

$$\hat{K}_{\text{tr}} = \hat{K}^{(\hat{k})}_{\text{tr}}.$$

Praktycznym rozwiązaniem jest wyznaczenie $\hat{K} = \hat{K}_{\text{ho}}$, gdzie $(W^{\text{ho}}, M(W^{\text{ho}}))$ jest pewnym tekstem anotowanym nazywanym danymi dodatkowymi (held-out data). Dane dodatkowe $(W^{\text{ho}}, M(W^{\text{ho}}))$ muszą być rozłączne z danymi treningowymi $(W^{\text{tr}}, M(W^{\text{tr}}))$ i testowymi $(W^{\text{test}}, M(W^{\text{test}}))$. Zwykle przyjmuje się, że długość danych dodatkowych L_{ho} stanowi 10% długości danych treningowych L_{tr} [29]. W przypadku konieczności używania danych dodatkowych ostateczną ocenę programu daje $U_{\text{test}}(\hat{K}_{\text{ho}})$.

W przypadku przeuczenia w sensie drugim, przeuczenie także można zmniejszyć w prosty sposób, jeżeli dysponuje się danymi dodatkowymi $(W^{\text{ho}}, M(W^{\text{ho}}))$. Zamiast kłaść $\hat{K} = \hat{K}^n_{\text{tr}}$, gdzie n jest największą liczbą taką, że $U_{\text{tr}}(\hat{K}^i_{\text{tr}}) > U_{\text{tr}}(\hat{K}^{i-1}_{\text{tr}})$ dla wszystkich $i \in \{1, ..., n\}$, kładzie się $\hat{K} = \hat{K}^m_{\text{tr}}$, gdzie m jest największą liczbą taką, że $U_{\text{ho}}(\hat{K}^i_{\text{tr}}) > U_{\text{ho}}(\hat{K}^{i-1}_{\text{tr}})$ dla wszystkich $i \in \{1, ..., m\}$.

1.5 Ukryte modele Markowa

Tagery oparte na ukrytych modelach Markowa (hidden Markov models) są historycznie najstarszym rodzajem tagerów. Tak jak inne tagery oparte na modelach probabilistycznych wyznaczają one $M_{\hat{K}}(W) = \hat{T}$ jako najbardziej prawdopodobny ciąg tagów przy założeniu pewnego modelu prawdopodobieństwa $p(T|W)$ ciągu tagów $T = (T_1, ..., T_L)$ przy danym tekście segmentowanym $W = (W_1, ..., W_L)$, zgodnie z równaniem (5). Korzystając z definicji prawdopodobieństwa warunkowego, można przepisać równoważnie

$$\hat{T} = \arg\max_T p(T|W) = \arg\max_T \frac{p(T,W)}{p(W)} = \arg\max_T \frac{p(W|T)p(T)}{p(W)} = \arg\max_T p(W|T)p(T). \tag{16}$$

Ukrytym sensem przepisania (16) jest to, że niezależne przybliżanie $p(W|T)$ i $p(T)$ często daje lepsze rezulaty niż bezpośrednie przybliżanie $p(T|W)$. Podobnie można przepisać równoważnie

$$p(T) = \prod_{i=1}^{L} p(T_i|T_{i-1}, ..., T_1), \tag{17}$$

$$p(W|T) = \prod_{i=1}^{L} p(W_i|T, W_{i-1}, ..., W_1). \tag{18}$$

We wzorach (17), (18) zakłada się, że $T_i \equiv W_i \equiv \epsilon$, dla $i \notin \{1, ..., L\}$. Z praktycznego punktu widzenia, wzory (17), (18) są bezużyteczne, jeśli nie dokona się w nich jakiegoś przybliżenia. Przybliżając prawdopodobieństwa

$p(T_i|...)$, $p(W_i|...)$ ukrytym modelem Markowa n-tego rzędu zakłada się, że

$$p(T_i|T_{i-1},...,T_1) \approx p_n(T_i|T_{i-1},...,T_{i-n}), \tag{19}$$

$$p(W_i|T,W_{i-1},...,W_1) \approx p_1(W_i|T_i). \tag{20}$$

Model n-tego rzędu nazywany jest także modelem $(n+1)$-gramowym.

Dla ukrytego modelu Markowa skończonego rzędu wyszukiwanie najbardziej prawdopodobnego ciągu tagów $\hat{T}$ realizuje algorytm Viterbiego opisany w podrozdziale 1.6. Uczenie tagera na danych treningowych $(W^{\rm tr}, M(W^{\rm tr}))$ oparte jest zwykle na estymacji najbardziej wiarogodnej (maximum likelihood estimation), to znaczy na podstawieniu

$$p_n(T_i|T_{i-1},...,T_{i-n}) = \frac{c^{\rm tr}_{n+1}(T_i,...,T_{i-n})}{c^{\rm tr}_n(T_{i-1},...,T_{i-n})}, \tag{21}$$

$$p_1(W_i|T_i) = \frac{c^{\rm tr}_e(W_i,T_i)}{c^{\rm tr}_1(T_i)}, \tag{22}$$

gdzie

$$c^{\rm tr}_n(t_1,...,t_n) = \sum_{j=1}^{L_{\rm tr}} \delta\left((t_1,...,t_n) = (M(W^{\rm tr})_j,...,M(W^{\rm tr})_{j-n+1})\right), \tag{23}$$

$$c^{\rm tr}_e(w,t) = \sum_{j=1}^{L_{\rm tr}} \delta\left((w,t) = (W^{\rm tr}_j, M(W^{\rm tr})_j)\right). \tag{24}$$

We wzorze (21) zakłada się, że $M(W^{\rm tr})_j \equiv \epsilon$, dla $j \notin \{1,...,L_{\rm tr}\}$.

Przy estymacji najbardziej wiarogodnej, im wyższy jest rząd modelu Markowa, tym bardziej jest on podatny na przeuczenie. Gwałtowny wzrost podatności wynika z tego, że zbiór n-tek kolejnych tokenów lub tagów występujących więcej niż jeden raz w zbiorze wszystkich danych dostępnych empirycznie dąży do zbioru pustego dla $n \to n_{max}$, gdzie n_{max} jest pewną skończoną liczbą. Przykładowo, w języku angielskim szóstki słów graficznych są unikatowe, jeśli nie są częścią sformułowań zwyczajowych bądź generowanych automatycznie. Empiryczna wartość n_{max} maleje gwałtownie ze wzrostem odpowiednio mocy zbioru tagów $|\mathcal{T}|$ lub mocy zbioru tokenów $|\mathcal{W}|$. We wszystkich zastosowaniach modeli Markowa (tagowanie częściami mowy, rozpoznawanie mowy i pisma) zakłada się, że $|\mathcal{T}| < \infty$. W żadnym z wymienionych zadań rzeczywisty zbiór tokenów otrzymanych w wyniku segmentacji nie jest skończony. W rozpoznawaniu mowy i pisma wszystkie tokeny są praktycznie unikatowe, a zatem stosuje się modele $p_1(W_i|T_i)$ bardziej złożone niż podstawienie (22) oparte na częstościach w danych treningowych [25, 1]. W tagowaniu częściami mowy, wszystkie tokeny nienależące do pewnego skończonego zbioru $\mathcal{W}'$ najczęstszych tokenów reprezentuje się w tekście segmentowanym jako jeden token OOV (out of vocabulary). Wówczas $\mathcal{W} = \mathcal{W}' \cup \{OOV\}$, $|\mathcal{W}| < \infty$. Jeżeli $|\mathcal{W}|$ jest dostatecznie małe, wtedy można stosować wzór (22).

Klasycznym remedium na przeuczenie stosowanym w modelach Markowa jest interpolacja liniowa, opisana w podrozdziale 1.7. Obecnie dla języka angielskiego w zastosowaniach komercyjnych, w rozpoznawaniu mowy i tagowaniu częściami mowy stosuje się prawie wyłącznie modele trigramowe ($n = 2$) z interpolacją liniową. W rozpoznawaniu pisma stosuje się modele wyższych rzędów.

Modelując tagowanie ukrytym procesem Markowa n-tego rzędu zakłada się, że najlepszą możliwą podstawę do przewidzenia wartości bieżącego tagu daje znajomość pewnego ustalonego kontekstu, który jest dany przez ciąg wartości n poprzednich tagów oraz bieżącego tokenu. Przy estymacji najbardziej wiarogodnej, konteksty różniące się wartością choć jednego z tych elementów zadają a priori całkowicie odmienne prawdopodobieństwa warunkowe. W rzeczywistości optymalne reguły wnioskowania o wartości bieżącego tagu mieć zupełnie inną postać. W podrozdziałach 1.8–1.11 przedstawiamy pewną jednolitą metodę budowania modelu prawdopodobieństwa w oparciu o racjonalny wybór dowolnych prawidłowości w zależności od ich mocy wyjaśniania empirycznego rozkładu danych.

1.6 Algorytm Viterbiego

Zarówno przybliżenie (21), (22) jak też przybliżenie (35), (22) zadają ukryte modele Markowa n-tego rzędu. Modele te są szczególnym przypadkiem niestacjonanego modelu Markowa n-tego rzędu, tzn. modelu postaci

$$p(T|W) = \prod_{i=1}^{L} p^{(i)}(T_i|W,T_{i-1},...,T_{i-n}). \tag{25}$$

Dla dowolnego niestacjonarnego modelu Markowa skończonego rzędu wyszukiwanie najbardziej prawdopodobnego ciągu tagów $\hat{T}$ redukuje się do programowania dynamicznego, znanego także jako algorytm Viterbiego [25, 29]. Algorytm ten wygląda następująco. Dla modelu n-tego rzędu definiuje się ciąg stanów $\Sigma = \Sigma_1, ..., \Sigma_L$, gdzie $\Sigma_i = (T_i, ..., T_{i-n+1}) \in \mathcal{S}_i$. $\mathcal{S}_i = \mathcal{T}^i \times \{\epsilon\}^{n-i}$ dla $0 \leqslant i \leqslant n$, $\mathcal{S}_i = \mathcal{T}^n$ dla $i \geqslant n$. Następnie, definiuje się macierz przejścia $\phi_i(T_{i-n}, \Sigma_i)$ tak, by spełniona była proporcjonalność

$$-\log p(T|W) \propto \sum_{i=1}^{L} \phi_i(T_{i-n}, \Sigma_i). \tag{26}$$

Dla ukrytego modelu Markowa (21), (22) wygodna macierz przejścia ma postać

$$\phi_i(T_{i-n}, \Sigma_i) = -\log\left[p_1(W_i|T_i)p_n(T_i|T_{i-1}, ..., T_{i-n})\right] \geqslant 0, \qquad i \in \{1, ..., L\}. \tag{27}$$

W ogólności dla dowolnego modelu typu (25) macierz przejścia ma postać

$$\phi_i(T_{i-n}, \Sigma_i) = -\log p^{(i)}(T_i|W, T_{i-1}, ..., T_{i-n}), \qquad i \in \{1, ..., L\}. \tag{28}$$

Aby obliczyć $\hat{T}$, dane wzorem (16), najpierw wypełnia się tablice warunkowych sum częściowych $\Phi_i(\Sigma_i)$ i warunkowych poprzedników $\Delta_i(\Sigma_i)$ dla każdego $\Sigma_i \in \mathcal{S}_i$. Odpowiednie wzory mają postać

$$\Phi_0(\Sigma_0) = 0, \tag{29}$$

$$\Phi_i(\Sigma_i) = \Phi_{i-1}(\Sigma_{i-1}) + \phi_i(T_{i-n}, \Sigma_i), \qquad i \in \{1, ..., n\}, \tag{30}$$

$$\Phi_i(\Sigma_i) = \min_{T_{i-n}} \left[\Phi_{i-1}(\Sigma_{i-1}) + \phi_i(T_{i-n}, \Sigma_i)\right], \qquad i \in \{n+1, ..., L\}, \tag{31}$$

$$\Delta_i(\Sigma_i) = \arg\min_{T_{i-n}} \left[\Phi_{i-1}(\Sigma_{i-1}) + \phi_i(T_{i-n}, \Sigma_i)\right], \qquad i \in \{n+1, ..., L\}. \tag{32}$$

Po wypełnieniu tablic odczytuje się optymalny ciąg tagów $\hat{T} = (\hat{T}_1, ..., \hat{T}_L)$ wykorzystując fakt, że

$$\hat{\Sigma}_L = \arg\min_{\Sigma_L} \Phi_L(\Sigma_L), \tag{33}$$

$$\hat{T}_{i-n} = \Delta_i(\hat{\Sigma}_i), \qquad i \in \{n+1, ..., L\}, \tag{34}$$

gdzie $\hat{\Sigma}_i = (\hat{T}_i, ..., \hat{T}_{i-n+1})$. Odczyt $\hat{T}_i$ następuje w kierunku przeciwnym do kierunku wypełniania $\Phi_i(\Sigma_i)$ i $\Delta_i(\Sigma_i)$. Złożoność pamięciowa algorytmu Viterbiego wynosi $O(|\mathcal{T}|^n(L + |\mathcal{T}|))$, a złożoność czasowa wynosi $O(|\mathcal{T}|^{n+1}(L+1))$.

1.7 Interpolacja liniowa

Interpolacja liniowa jest najpopularniejszą metodą wygładzania stosowaną w modelach Markowa. Polega ona na przybliżeniu

$$p(T_i|T_{i-1}, ..., T_1) \approx p_n^{\lambda}(T_i|T_{i-1}, ..., T_{i-n}) = \sum_{k=0}^{n} \lambda_k p_k(T_i|T_{i-1}, ..., T_{i-k}), \quad \lambda_k \geqslant 0, \quad \sum_{k=0}^{n} \lambda_k = 1. \tag{35}$$

gdzie prawdopodobieństwa $p_k(T_i|T_{i-1}, ..., T_{i-k})$ estymowane są wzorem (21). W szczególności, prawdopodobieństwo $p_0(T_i)$ zdefiniowane jest jako $p_0(T_i) = c_1(T_i)/L_{\text{tr}}$. Dla $P(W_i|T_i)$ nadal stosuje się przybliżenie (20) i wzór (22), jeżeli token W_i pojawia się danych treningowych. Jeżeli W_i nie pojawiło się w danych treningowych stosuje się różne heurystyki [4].

W przybliżeniu (35) parametry λ_k zwykle nie są wyznaczane są tak, aby bezpośrednio zmaksymalizować akuratność tagera $\text{Acc}(K(\lambda_0, ..., \lambda_n, \mu_0, \mu_1))$ na danych dodatkowych $(W^{\text{ho}}, M(W^{\text{ho}}))$, lecz w oparciu o algorytm oczekiwania-maksymalizacji (expectation-maximization) [25, 29]. Postać tego algorytmu przedstawia diagram 1. W algorytmie oczekiwania-maksymalizacji końcowe wartości λ_k mogą zależeć od wyboru początkowych wartości c_k.

W popularnym tagerze TnT [4], w celu wyznaczenia λ_k zamiast algorytmu oczekiwania-maksymalizacji stosuje się uproszczony algorytm interpolacji usunięć (deleted interpolation). Dla $n = 2$ algorytm ten sprowadza się do podstawienia

$$\lambda_2 = \frac{1}{L_{\text{tr}}} \sum_{(t_1,t_2,t_3)\in\mathcal{T}^3} c_3^{\text{tr}}(t_1, t_2, t_3)\delta\left(p_2(t_1|t_2, t_3) = \max\left(p_2(t_1|t_2, t_3), p_1(t_1|t_2), p_0(t_1)\right)\right), \tag{36}$$

$$\lambda_1 = \frac{1}{L_{\text{tr}}} \sum_{(t_1,t_2,t_3)\in\mathcal{T}^3} c_3^{\text{tr}}(t_1, t_2, t_3)\delta\left(p_1(t_1|t_2) = \max\left(p_2(t_1|t_2, t_3), p_1(t_1|t_2), p_0(t_1)\right)\right), \tag{37}$$

$$\lambda_0 = \frac{1}{L_{\text{tr}}} \sum_{(t_1,t_2,t_3)\in\mathcal{T}^3} c_3^{\text{tr}}(t_1, t_2, t_3)\delta\left(p_0(t_1) = \max\left(p_2(t_1|t_2, t_3), p_1(t_1|t_2), p_0(t_1)\right)\right). \tag{38}$$

Diagram 1: **Algorytm oczekiwania-maksymalizacji w interpolacji liniowej**

odpowiednio zainicjalizuj $c_k > 0$ dla każdego $k \in \{1, ..., n\}$;
dla każdego $i \in \{1, ..., k\}$ wykonaj {
  podstaw $\lambda_k \leftarrow \frac{c_k}{\sum_{m=1}^{n} c_m}$;
}
powtarzaj {
  dla każdego $k \in \{1, ..., n\}$ wykonaj {
    podstaw $c_k \leftarrow \sum_{j=1}^{L_{\mathrm{ho}}} \frac{\lambda_k p_k(M(W^{\mathrm{ho}})_j | M(W^{\mathrm{ho}})_{j-1}, ..., M(W^{\mathrm{ho}})_{j-k})}{\sum_{m=1}^{n} \lambda_m p_m(M(W^{\mathrm{ho}})_j | M(W^{\mathrm{ho}})_{j-1}, ..., M(W^{\mathrm{ho}})_{j-m})}$;
    podstaw $\lambda_k^{\mathrm{old}} \leftarrow \lambda_k$;
    podstaw $\lambda_k \leftarrow \frac{c_k}{\sum_{m=1}^{n} c_m}$;
  }
} dopóki $\max_{k \in \{1,...,n\}} |\lambda_k - \lambda_k^{\mathrm{old}}| > \epsilon$;

Zaletą algorytmu interpolacji usunięć jest to, że nie wymaga danych dodatkowych $(W^{\mathrm{ho}}, M(W^{\mathrm{ho}}))$ i jest on znacznie prostszy obliczeniowo.

Interpolacja liniowa nie jest optymalną metodą wygładzania. Więcej informacji na temat różnych metod wygładzania i ich skuteczności w zastosowaniu do modelowania języka angielskiego zawiera raport [8].

1.8 Modelowanie maksimum entropii w teorii

Załóżmy, że dysponujemy pewnym skończonym ciągiem $(f_i)_{i=1}^{k}$ liniowo niezależnych funkcji $f_i : X \to R$ określonych na pewnym zbiorze wartości danych X i przyjmujących wartości rzeczywiste. Dany jest także pewien nieskończony ciąg danych $(x_j)_{j \in N}$, $x_j \in X$. Załóżmy, że średnie wartości wszystkich funkcji f_i są asymptotycznie stałe dla ciągu $(x_j)_{j \in N}$, tzn.

$$\lim_{L \to \infty} \frac{1}{L} \sum_{j=1}^{L} f_i(x_j) = \bar{f}_i, \qquad i \in \{1, ..., k\}\,. \tag{39}$$

Oznaczmy przez C_k zbiór wszystkich rozkładów prawdopodobieństwa $p : X \to [0, 1]$ spełniających warunek

$$\sum_{x \in X} p(x) f_i(x) = \bar{f}_i, \qquad i \in \{0, 1, ..., k\}\,, \tag{40}$$

gdzie $f_0(x) = 1$ oraz $\bar{f}_0 = 1$. Modelowanie maksimum entropii (maximum entropy modeling) to odwzorowanie danego ciągu danych $(x_j)_{j \in N}$, $x_j \in X$ w taki rozkład prawdopodobieństwa $\hat{p}_k \in C_k$, który maksymalizuje entropię rozkładu $H(p)$, tzn.

$$\hat{p}_k = \arg\max_{p \in C_k} H(p), \tag{41}$$

gdzie entropia rozkładu H zdefiniowana jest jako

$$H(p) = -\sum_{x \in X} p(x) \log p(x) \geqslant 0. \tag{42}$$

Sens warunku (40) jest oczywisty. Poszukujemy takiego modelu prawdopodobieństwa, dla którego średnie wartości funkcji f_i pokrywają się z wartościami średnimi obserwowanymi dla ciągu $(x_j)_{j \in N}$. Sens warunku (41) jest bardziej subtelny. Dla $k = 0$ entropia jest maksymalizowana dla rozkładu jednorodnego $p(x) = \mathrm{const}$. Intuicyjnie, jeśli nie wiemy o danych nic, rozkład jednorodny stanowi najlepszy model, gdyż jest to model najprostszy. Podobnie rozkład o największej entropii jest najprostszym modelem przy zadanych innych warunkach. W podrozdziale 1.9 pokażemy dodatkowo, że rozkład prawdopodobieństwa o największej entropii jest rozkładem najbardziej zbliżonym do rozkładu empirycznego w pewnej ograniczonej klasie modeli parametrycznych.

Z rozważań analitycznych wynika, że wyznaczenie $\hat{p}_k$ redukuje się do znalezienia argumentów $(\hat{p}_k, \hat{\lambda})$ spełniających układ równań

$$p_\lambda = \arg\max_{p \in C_0} L(p, \lambda), \tag{43}$$

$$\hat{\lambda} = \arg\min_{\lambda} L(p_\lambda, \lambda), \tag{44}$$

$$\hat{p}_k = p_{\hat{\lambda}}, \tag{45}$$

gdzie $\lambda = (\lambda_1, ..., \lambda_k)$, $\hat{\lambda} = (\hat{\lambda}_1, ..., \hat{\lambda}_k)$, a langranżjan L zdefiniowany jest jako

$$L(p, \lambda) = H(p) + \sum_{i=1}^{k} \lambda_i \left[\sum_{x \in X} p(x) f_i(x) - \bar{f} \right]. \tag{46}$$

Zauważmy, że maksymalizacja w równaniu (43) odbywa się w całej przestrzeni rozkładów unormowanych C_0, a nie tylko w podprzestrzeni $C_k \subset C_0$. Równanie (43) można rozwiązać analitycznie, co daje

$$p_\lambda(x) = \frac{1}{Z_\lambda} \exp\left[\sum_{i=1}^{k} \lambda_i f_i(x) \right], \tag{47}$$

$$Z_\lambda = \sum_{x \in X} \exp\left[\sum_{i=1}^{k} \lambda_i f_i(x) \right], \tag{48}$$

$$L(p_\lambda, \lambda) = \log Z_\lambda - \sum_{i=1}^{k} \lambda_i \bar{f}_i. \tag{49}$$

Dla $p \in C_k$ zachodzi $L(p, \cdot) = H(p)$. Zgodnie z równaniami (43)–(45), wyznaczenie optymalnego rozkładu $\hat{p}_k$ spowadzone zostało do znalezienia λ minimalizującego $L(p_\lambda, \lambda)$, gdzie $L(p_\lambda, \lambda)$ dane jest wzorem (49). Dla zadanej zbioru wartości danych X, ciągu funkcji $(f_i)_{i=1}^k$ i ciągu wartości średnich $(\bar{f}_i)_{i=1}^k$ minimalizację tę wykonuje się numerycznie. Stosuje się do tego celu standardowe algorytmy znajdowania minimów funkcji wielu zmiennych (minimalizowanie względem kolejnych zmiennych, metoda sprzężonych gradientów itp.) bądź algorytmy specjalnie dostosowane do minimalizacji funkcji postaci (49) (iteracyjne skalowanie). Więcej informacji na ten temat podano w artykułach [3], [2]. [6] Algorytm iteracyjnego skalowania opisujemy w podrozdziale 1.10.

1.9 Funkcje oceniające oparte na entropii

Naturalną pseudoodległością rozkładu prawdopodobieństwa p od innego rozkładu p_e jest dywergencja Kullbacka-Leiblera $D(p_e||p)$ (entropia względna) zdefiniowana jako

$$D(p_e||p) = -\sum_{x \in X} p_e(x) \log \frac{p(x)}{p_e(x)}. \tag{50}$$

Jeżeli $\sum_{x \in X} p_e(x) = \sum_{x \in X} p(x) = 1$ to $D(p_e||p) \geqslant 0$. [7] $D(p_e||p) = 0$ zachodzi wyłącznie dla $p = p_e$. Miarą odstępstwa teoretycznego rozkładu p od empirycznego rozkładu p_e wygodniejszą niż entropia względna jest entropia krzyżowa $H(p_e||p)$ (cross entropy) dana wzorem

$$H(p_e||p) = -\sum_{x \in X} p_e(x) \log p(x) = H(p_e) + D(p_e||p) \geqslant 0. \tag{51}$$

[6] Często cytowany w literaturze artykuł [3] zawiera pomyłkę polegającą na zdefiniowaniu funkcji $\Psi(\lambda) = L(p_\lambda, \lambda)$, podczas gdy w dalszej dyskusji pojawia się $\Psi(\lambda)$ w znaczeniu $-L(p_\lambda, \lambda)$ kolejno w odpowiednikach wzorów (44), (49) i (53). W punkcie $(\hat{p}_k, \hat{\lambda})$ znikają pochodne funkcji $L(p, \lambda)$ po wszystkich argumentach (p, λ). Warto jednak zauważyć, że punkt ten nie jest maksimum ani minimum, lecz punktem siodłowym. W szczególności łatwo pokazać, że

$$\frac{\partial L(p_\lambda, \lambda)}{\partial \lambda_i} = L'_i = \langle f_i \rangle_\lambda - \bar{f}_i, \qquad \frac{\partial^2 L(p_\lambda, \lambda)}{\partial \lambda_i \partial \lambda_j} = L''_{ij} = \langle [f_i - \langle f_i \rangle_\lambda] [f_j - \langle f_j \rangle_\lambda] \rangle_\lambda.$$

gdzie $\langle g \rangle_\lambda = \sum_{x \in X} p_\lambda(x) g(x)$. Macierz L''_{ij} jest dodatnio określona, gdyż $\sum_{i=1}^k \sum_{j=1}^k a_i L''_{ij} a_j = \langle \left[\sum_{i=1}^k a_i [f_i - \langle f_i \rangle_\lambda] \right]^2 \rangle_\lambda \geqslant 0$. Zatem funkcja $\lambda \mapsto L(p_\lambda, \lambda)$ jest wypukła i wszystkie jej ekstrema są minimami globalnymi.

[7] Zachodzi $-\sum_{x \in X} p_e(x) \log \frac{p(x)}{p_e(x)} \geqslant -\sum_{x \in X} p_e(x) \left[\frac{p(x)}{p_e(x)} - 1 \right] = 0$, gdyż $\log y \leqslant y - 1$ z równością dla $y = 1$.

Przy ustalonym p_e, $H(p_e||p)$ jest minimalizowane dla $p = p_e$. O wygodzie $H(p_e||p)$ przesądza możliwość zdefiniowania entropii krzyżowej $H((x_j)_{j\in N}||p)$ również wtedy, gdy empiryczny rozkład p_e zadany jest w sposób uwikłany za pomocą ciągu danych $(x_j)_{j\in N}$. Wówczas

$$H((x_j)_{j\in N}||p) = -\lim_{L\to\infty}\frac{1}{L}\sum_{j=1}^{L}\log p(x_j). \tag{52}$$

W oparciu o empiryczną entropię krzyżową $H((x_j)_{j\in N}||p)$ definiuje się logarytm wiarogodności (log-likelihood) $U((x_j)_{j\in N}||p) = -H((x_j)_{j\in N}||p)$, który jest naturalną funkcją oceniającą dla modeli probabilistycznych. Dodatkowo, ze wzorów (39), (47), (49) wynika, że

$$H((x_j)_{j\in N}||p_\lambda) = L(p_\lambda, \lambda). \tag{53}$$

Oznaczmy przez C_k^* zbiór wszystkich rozkładów prawdopodobieństwa $p_\lambda : X \to [0,1]$ danych wzorem (47). Ponieważ $L(p_\lambda, \lambda) = H((x_j)_{j\in N}||p_\lambda)$, zatem zgodnie z równaniami (43)–(45), rozkład $p = \hat{p}_k$ maksymalizujący entropię rozkładu $H(p)$ dla $p \in C_k$ jest jednocześnie rozkładem minimalizującym entropię krzyżową $H((x_j)_{j\in N}||p)$ dla $p \in C_k^*$ (maksymalizującym logarytm wiarogodności $U((x_j)_{j\in N}||p)$ dla $p \in C_k^*$). Ponieważ $L(p,\cdot) = H(p)$ dla $p \in C_k$, zatem $C_k^* \cap C_k = \{\hat{p}_k\}$.

Należy zauważyć, że zbiór rozkładów C_k^* jest znacznie bardziej ograniczony niż zbiór rozkładów C_k. C_k jest zbiorem wszystkich rozkładów p, dla których spełnione są zależności (40). C_k^* jest zbiorem wszystkich tych rozkładów p, dla których można obliczyć $H((x_j)_{j\in N}||p)$ wiedząc o ciągu $(x_j)_{j\in N}$ tylko to, że spełnione są zależności (39). Zauważmy, że zachodzą nierówności $C_{k+1}^* \supset C_k^*$, $C_{k+1} \subset C_k$, a także

$$H((x_j)_{j\in N}||\hat{p}_{k+1}) = H(\hat{p}_{k+1}) \quad\leqslant\quad H((x_j)_{j\in N}||\hat{p}_k) = H(\hat{p}_k). \tag{54}$$

Im więcej jest funkcji $f_i \in F_k$, dla których znane są granice ich średnich wartości na ciągu $(x_j)_{j\in N}$, tym wierniej model $\hat{p}_k$ odzwierciedla empiryczny rozkład danych $(x_j)_{j\in N}$.

1.10 Algorytm iteracyjnego skalowania

Algorytm iteracyjnego skalowania to algorytm minimalizacji lagranżjanu $L(p_\lambda, \lambda)$, $\lambda = (\lambda_1, ..., \lambda_k)$, danego wzorem (49). Algorytm ten wykonuje względem kolejnych argumentów λ_i optymalne kroki zapewniające maksymalne zmniejszanie się pewnego górnego ograniczenia wartości $L(p_\lambda, \lambda)$.

Oznaczmy krok argumentu λ jako $\delta = (\delta_1, ..., \delta_k)$. Z powodu nierówności $\log y \leqslant y - 1$ zachodzi związek

$$L(p_{\lambda+\delta}, \lambda+\delta) - L(p_\lambda, \lambda) \leqslant A(\lambda,\delta) = \frac{Z_{\lambda+\delta}}{Z_\lambda} - 1 - \sum_{i=1}^{k}\delta_i\bar{f}_i = \sum_{x\in X} p_\lambda(x)\exp\left[\sum_{i=1}^{k}\delta_i f_i(x)\right] - 1 - \sum_{i=1}^{k}\delta_i\bar{f}_i. \tag{55}$$

Funkcja $y \mapsto \exp y$ jest wypukła, zatem $\exp\left[\sum_{i=1}^{k} n_i g_i\right] \leqslant \sum_{i=1}^{k} n_i \exp g_i$ dla $n_i \geqslant 0$, $\sum_{i=1}^{k} n_i = 1$. Jeżeli zachodzi

$$f_i(x) \geqslant 0, \qquad i \in \{1, ..., k\}, \tag{56}$$

wówczas oznaczając $f_+(x) = \sum_{i=1}^{k} f_i(x)$ i podstawiając $n_i = f_i(x)/_+ f(x)$ oraz $g_i = \delta_i f_+(x)$ otrzymuje się

$$A(\lambda,\delta) \leqslant B(\lambda,\delta) = \sum_{x\in X} p_\lambda(x)\sum_{i=1}^{k}\frac{f_i(x)}{f_+(x)}\exp\left[\delta_i f_+(x)\right] - 1 - \sum_{i=1}^{k}\delta_i\bar{f}_i. \tag{57}$$

Odpowiednio zachodzi także

$$\frac{\partial B(\lambda,\delta)}{\partial\delta_i} = B'(\lambda,\delta_i) \quad=\quad \sum_{x\in X} p_\lambda(x) f_i(x)\exp\left[\delta_i f_+(x)\right] - \bar{f}_i, \tag{58}$$

$$\frac{\partial^2 B(\lambda,\delta)}{\partial\delta_i^2} = B''(\lambda,\delta_i) \quad=\quad \sum_{x\in X} p_\lambda(x) f_i(x) f_+(x)\exp\left[\delta_i f_+(x)\right]. \tag{59}$$

Optymalny bezpieczny krok δ_i przy minimalizowaniu $L(p_\lambda, \lambda)$ odpowiada warunkowi $B'(\lambda, \delta_i) = 0$. Stąd wynika postać ulepszonego algorytmu iteracyjnego skalowania (improved iterative scaling algorithm) przedstawiona na diagramie 2 [2, 3]. Podany algorytm wykorzystuje metodę Newtona do rozwiązywania równania $B'(\lambda, \delta_i) = 0$ względem δ_i.

Diagram 2: **Ulepszony algorytm iteracyjnego skalowania**

Można stosować wyłącznie, gdy $f_i(x) \geqslant 0$ *dla każdego* $i \in \{1, ..., k\}$.
podstaw $\lambda_i \leftarrow 0$ dla wszystkich $i \in \{1, ..., k\}$;
powtarzaj {
  dla każdego $i \in \{1, ..., k\}$ wykonaj {
    odpowiednio zainicjalizuj δ_i;
    dopóki $\left|\frac{B'(\lambda, \delta_i)}{B''(\lambda, \delta_i)}\right| > \epsilon$ powtarzaj {
      podstaw $\delta_i \leftarrow \delta_i - \frac{B'(\lambda, \delta_i)}{B''(\lambda, \delta_i)}$;
    }
    podstaw $\lambda_i \leftarrow \lambda_i + \delta_i$;
  }
} dopóki $\max_{i \in \{1,...,k\}} |\delta_i| > \epsilon$;
podstaw $\hat{\lambda}_i \leftarrow \lambda_i$ dla wszystkich $i \in \{1, ..., k\}$;

1.11 Klasyfikacja oparta na modelowaniu maksimum entropii

Historycznie pierwszym zastosowaniem modelowania maksimum entropii była mechanika statystyczna [24]. W ostatnich latach modelowanie maksimum entropii znajduje szersze zastosowanie w problemie maszynowego uczenia się klasyfikacji [3].

Uczenie się klasyfikacji to uczenie się wnioskowania o wartości tagu $Q(D)_j \in \mathcal{T}$ odpowiadającej pewnemu kontekstowi $D_j \in \mathcal{D}$. Zakłada się, że zbiór wszystkich tagów $\mathcal{T}$ jest skończony. W celu przewidywania $Q(D)_j$ dla dowolnego D_j można posłużyć się modelowaniem probabilistycznym. Dysponując pewnym modelem prawdopodobieństwa $p(t|d)$, $t \in \mathcal{T}$, $d \in \mathcal{D}$, można automatycznie przybliżać $Q(D)_j$ jako najbardziej prawdopodbną wartość $\hat{T}_j$, gdzie

$$\hat{T}_j = \arg\max_{T_j \in \mathcal{T}} p(T_j|D_j). \tag{60}$$

Pewien model prawdopodobieństwa $p(t|d)$ można skonstruować w oparciu o modelowanie maksimum entropii dysponując danymi treningowymi $(D^{\mathrm{tr}}, Q(D^{\mathrm{tr}}))$, gdzie $D^{\mathrm{tr}} = (D_1^{\mathrm{tr}}, ..., D_{L_{\mathrm{tr}}}^{\mathrm{tr}})$, oraz skończonym ciągiem $(f_i)_{i=1}^k$ funkcji charakterystycznych (features) $f_i : \mathcal{T} \times \mathcal{D} \to \{0, 1\}$. Oznaczmy

$$\bar{f}_i^{\alpha} = \frac{1}{L_\alpha} \sum_{j=1}^{L_\alpha} f_i(Q(D^\alpha)_j, D_j^\alpha), \qquad i \in \{1, ..., k\}, \tag{61}$$

$$p^{\alpha}(d) = \frac{1}{L_\alpha} \sum_{j=1}^{L_\alpha} \delta(D_j^\alpha = d), \qquad \forall d \in \mathcal{D}, \tag{62}$$

gdzie $\alpha = (\mathrm{tr}, \mathrm{ho}, \mathrm{test})$. Rozkład $p(t|d)$ wyznaczymy jako taki rozkład prawdopodobieństwa, który jest unormowany, spełnia warunek

$$\sum_{t \in \mathcal{T}} \sum_{d \in \mathcal{D}} p(t|d) p^{\mathrm{tr}}(d) f_i(t, d) = \bar{f}_i^{\mathrm{tr}}, \qquad \forall i \in S, \quad S \subset \{1, ..., k\} \tag{63}$$

i ma maksymalną entropię. Warto zauważyć, że dla rozkładu $p(t|d)$ maksymalizującego entropię przy warunku (63) wzory (47), (48), (49), (58), (59) modyfikują się odpowiednio do postaci

$$p_\lambda(t|d) = \frac{1}{Z_\lambda(d)} \exp\left[\sum_{i \in S} \lambda_i f_i(t, d)\right], \tag{64}$$

$$Z_\lambda(d) = \sum_{t \in \mathcal{T}} \exp\left[\sum_{i \in S} \lambda_i f_i(t, d)\right], \tag{65}$$

$$L_\alpha(p_\lambda, \lambda) = \sum_{d \in \mathcal{D}} p^\alpha(d) \log Z_\lambda(d) - \sum_{i \in S} \lambda_i \bar{f}_i^{\alpha}, \tag{66}$$

$$f_+(t,d) = \sum_{i\in S} f_i(t,d), \tag{67}$$

$$B'_\alpha(\lambda,\delta_i) = \sum_{d\in\mathcal{D}} p^\alpha(d) \sum_{t\in\mathcal{T}} p_\lambda(t|d) f_i(t,d) \exp\left[\delta_i f_+(t,d)\right] - \bar{f}_i^\alpha, \qquad i\in S, \tag{68}$$

$$B''_\alpha(\lambda,\delta_i) = \sum_{d\in\mathcal{D}} p^\alpha(d) \sum_{t\in\mathcal{T}} p_\lambda(t|d) f_i(t,d) f_+(t,d) \exp\left[\delta_i f_+(t,d)\right], \qquad i\in S. \tag{69}$$

Jeżeli zbiór indeksów funkcji S zawiera wszystkie funkcje charakterystyczne, tzn. $S=\{1,...,k\}$, ryzykuje się znaczne przeuczenie otrzymanego modelu prawdopodobieństwa. Dla wielu funkcji f_i, $i\in\{1,...,k\}$, ich średnie wartości $\bar{f}_i^{\rm tr}$ na danych treningowych mogą być zupełnie inne niż średnie wartości $\bar{f}_i^{\rm test}$ na danych testowych. Aby wyznaczyć optymalny rozkład $p(t|d)$ należy połączyć modelowanie maksimum entropii z konstrukcją zbioru S zawierającego indeksy tylko takich funkcji, które bez ryzyka przeuczenia można wykorzystać do modelowania rozkładu $p(d|t)$. W tym celu stosuje się heurystyczny algorytm zachłanny, do którego potrzeba także danych dodatkowych $(D^{\rm ho}, Q(D^{\rm ho}))$, gdzie $D^{\rm ho}=(D_1^{\rm ho}, D_2^{\rm ho}, ..., D_{L_{\rm ho}}^{\rm ho})$. Postać tego algorytmu przedstawia diagram 3 [3]. Podany algorytm rozpoczyna od pustego zbioru indeksów S i w każdej iteracji pętli powiększa zbiór S o indeks takiej funkcji charakterystycznej f_i, której włączenie do zbioru S skutkuje największym zmniejszeniem się wartości lagranżjanu. Wyjście z pętli następuje przy pierwszym odnotowanym przypadku przeuczenia się. Ponieważ z założenia $f_i(t,d)\in\{0,1\}$, więc w celu minimalizacji lagranżjanu $L_{\rm tr}(p_\lambda,\lambda)$ można stosować ulepszony algorytm iteracyjnego skalowania (opisany w podrozdziale 1.10).

Diagram 3: **Algorytm zachłannego wyboru funkcji charakterystycznych**

podstaw $S\leftarrow\emptyset$;
podstaw $\hat\lambda\leftarrow\arg\min_\lambda L_{\rm tr}(p_\lambda,\lambda)$, $\hat L_{\rm tr}\leftarrow L_{\rm tr}(p_{\hat\lambda},\hat\lambda)$;
powtarzaj {
  podstaw $\hat\lambda^{\rm new}\leftarrow\hat\lambda$, $\hat L_{\rm tr}^{\rm new}\leftarrow\hat L_{\rm tr}$;
  podstaw $S^{\rm old}\leftarrow S$;
  dla każdego $i^{\rm prop}\in\{1,...,k\}\setminus S^{\rm old}$ wykonaj {
    podstaw $S\leftarrow S^{\rm old}\cup i^{\rm prop}$;
    podstaw $\hat\lambda^{\rm prop}\leftarrow\arg\min_\lambda L_{\rm tr}(p_\lambda,\lambda)$, $\hat L_{\rm tr}^{\rm prop}\leftarrow L_{\rm tr}(p_{\hat\lambda^{\rm prop}},\hat\lambda^{\rm prop})$;
    jeśli $\hat L_{\rm tr}^{\rm prop}<\hat L_{\rm tr}^{\rm new}$ wykonaj {
      podstaw $i^{\rm new}\leftarrow i^{\rm prop}$, $\hat\lambda^{\rm new}\leftarrow\hat\lambda^{\rm prop}$, $\hat L_{\rm tr}^{\rm new}\leftarrow\hat L_{\rm tr}^{\rm prop}$;
    }
  }
  podstaw $S\leftarrow S^{\rm old}$;
  opuść pętlę jeśli $L_{\rm ho}(p_{\hat\lambda^{\rm new}},\hat\lambda^{\rm new})\geqslant L_{\rm ho}(p_{\hat\lambda},\hat\lambda)$;
  podstaw $S\leftarrow S\cup i^{\rm new}$, $\hat\lambda\leftarrow\hat\lambda^{\rm new}$, $\hat L_{\rm tr}\leftarrow\hat L_{\rm tr}^{\rm new}$;
}

Podaną procedurę uczenia się klasyfikacji można zaadaptować do problemu uczenia się tagowania. Załóżmy, że przypisywanie ciągowi tokenów ciągu tagów jest modelowane optymalnie niestacjonarnym procesem Markowa n-tego rzędu. Znaczy to, że aby optymalnie przewidzieć wartość tagu wystarczy znać wartości wszystych tokenów oraz tylko n poprzednich tagów. W praktyce, by operować skończonymi przestrzeniami przeszukiwania i sumowania, zakłada się, że liczba tokenów, od których tag może zależeć bezpośrednio, jest także ograniczona. Wówczas zachodzi odpowiedniość

$$M(W)_j = Q(D)_j, \quad \text{gdzie} \quad D_j = (W_{j-C},...,W_{j+C}, M(W)_{j-1},...,M(W)_{j-n}). \tag{70}$$

Dla danych treningowych i dodatkowych, uczenie się rozkładu $p(T_j|D_j)=p(T_j|W_{j-C},...,W_{j+C},T_{j-1},...,T_{j-n})$ przeprowadza się zgodnie z podaną procedurą uczenia się opartą na modelowaniu maksimum entropii. Kiedy dysponujemy już modelem prawdopodobieństwa $p(T_j|D_j)$, najbardziej prawdopodobny ciąg tagów $\hat T=(\hat T_1,...,\hat T_L)$ dla nowego tekstu segmentowanego $W=(W_1,...,W_L)$ wyznaczamy stosując algorytm Viterbiego (opisany w podrozdziale 1.6). Macierz przejścia dla rozpatrywanego modelu ma postać

$$\phi_j(T_{j-n},\Sigma_j) = -\log p(T_j|W_{j-C},...,W_{j+C},T_{j-1},...,T_{j-n}), \qquad j\in\{1,...,L\}. \tag{71}$$

Skuteczność modelowania maksimum entropii w zastosowaniach praktycznych zależy od wyboru ciągu funkcji charakterystycznych $(f_i)_{i=1}^k$ oraz przestrzeni kontekstów $\mathcal{D}$. Zbiór funkcji charakterystyczbych f_i oraz przestrzeń

kontekstów $\mathcal{D}$ używane w bardzo dobrym algorytmie do tagowania morfosyntaktycznego tekstów w języku angielskim zostały wyspecyfikowne w artykule [34]. Zwykle w zastosowaniu do tagowania kładzie się w równaniu (70) wartość $C = 2$.

1.12 Tagowanie oparte na transformacjach

Najpopularniejszym typem tagerów nieopartych na modelach probabilistycznych są tagery wykorzystujące uczenie się oparte na transformacjach i sterowane liczbą błędów (transformation-based error-driven learning) [5]. Tagery należące do tego typu, nazywane są tagerami opartymi na transformacjach bądź też tagerami Brilla, autora cytowanej pracy [5].

Tagery oparte na transformacjach rozpoczynają tagowanie tekstu segmentowanego W od wyznaczenia pewnego trywialnego tagowania początkowego $T^{(0)} = (t_0, ..., t_0)$, gdzie $t_0 \in \mathcal{T}$, a następnie obliczają

$$T_i^{(n+1)} = F_{n+1}(D_i^{(n)}), \tag{72}$$

gdzie $(F_1, ..., F_N)$ jest ciągiem przekształceń $\mathcal{D} \to \mathcal{T}$, $\mathcal{D} = (\mathcal{W} \cup \{\epsilon\})^{2C+1} \times (\mathcal{T} \cup \{\epsilon\})^{2C+1}$, $T_i^{(n)} = W_i = \epsilon$ dla $i \notin \{1, ..., L\}$ oraz $D_i^{(n)} = (W_{i-C}, ..., W_{i+C}, T_{i-C}^{(n)}, ..., T_{i+C}^{(n)})$. Ostateczne tagowanie zwracane przez tager równe jest $T^{(N)}$. Dodatkowo zakłada się, że każde przekształcenie F_n sprowadza się do pojedynczej reguły postaci: „Zamień wartość tagu $t_n \in \mathcal{T}$ na wartość $y_n \in \mathcal{T}$, jeżeli kontekst spełnia warunek $\phi_n \in \Phi$". To znaczy,

$$F_n(D_i) = \begin{cases} y_n, & \text{jeśli } T_i = t_n \text{ i } \phi_n(\tilde{D}_i), \\ T_i, & \text{w innym wypadku,} \end{cases} \tag{73}$$

gdzie $D_i = (W_{i-C}, ..., W_{i+C}, T_{i-C}, ..., T_{i+C})$, $\tilde{D}_i = (W_{i-C}, ..., W_{i+C}, T_{i-C}, ...T_{i-1}, T_{i+1}, ..., T_{i+C})$. Przestrzeń warunków Φ używana w oryginalnej wersji algorytmu do tagowania morfosyntaktycznego tekstów w języku angielskim została wyspecyfikowana w artykule [5]. Każde przekształcenie F_n postaci (73) może być modelowane przez skończony automat przepisujący (transduktor, ang. transducer). Oznaczmy $\tilde{F}_n : (T, W) \mapsto (\tilde{T}, W)$, gdzie $\tilde{T}_i = F_n(W_{i-C}, ..., W_{i+C}, T_{i-C}, ..., T_{i+C})$. Złożenie $\tilde{F}_N \circ ... \circ \tilde{F}_1$ skończonej liczby transduktorów $\tilde{F}_n$ jest także pewnym transduktorem. Ponieważ istnieje efektywny algorytm konstrukcji transduktora $\tilde{F}_N \circ ... \circ \tilde{F}_1$ dla zadanych F_i, odpowiednio można otrzymać bardzo szybki tager implementujący ten transduktor. Szczegółowa postać algorytmu konstrukcji transduktora podana została w pracy [35].

Algorytm uczenia się tagera opartego na transformacjach to algorytm poszukujący odpowiednie funkcje F_n spośród zadanego skończonego podzbioru $\mathcal{F}$ zbioru przekształceń $\mathcal{D} \to \mathcal{T}$ postaci (73). Algorytm ten przypomina algorytm wyboru funkcji charakterystycznych przedstawiony na diagramie 3. W n-tej iteracji wybierana jest taka funkcja F_n, której wybór maksymalizuje pewną funkcję oceniającą. W algorytmie uczenia się opartego na transformacjach funkcją oceniającą jest akuratność $\mathrm{Acc}(A)$ tagera $A = \tilde{F}_n \circ ... \circ \tilde{F}_1(T^{(0)}, \cdot)$ dla danych treningowych W^{tr}. Dla prostych przekształceń F_n nie obserwuje się przeuczenia, więc algorytm nie potrzebuje danych dodatkowych. Ostatecznie, algorytm uczenia się sprowadza się do postaci z diagramu 4. W przedstawionym algorytmie zakłada się, że przepisanie wszystkich tagów następuje jednocześnie — w użyciu są także algorytmy, w których przepisania poszczególnych tagów następują sekwencyjnie.

Diagram 4 przedstawia algorytm uczenia się w wersji, w jakiej został on zaimplementowany w lingwistyce komputerowej po raz pierwszy [5]. Ma on złożoność $O(NL^{\mathrm{tr}}|\mathcal{F}|)$, więc jest on bardzo wolny, podobnie jak jego dostępna implementacja. Wykorzystując w przeszukiwaniu reguł dynamiczne indeksowanie reguł i pozycji w tekście anotowanym, do których się te reguły stosują, otrzymuje się znacznie szybszy algorytm uczenia się równoważny z algorytmem podanym. Szczegółową postać tego algorytmu podaje praca [32].

1.13 Tagowanie oparte na pamięci

Rzadziej wzmiankowana w lingwistyce komputerowej, jakkolwiek bardzo skuteczna klasa tagerów wykorzystuje technikę zwaną uczeniem się opartym na pamięci (memory-based learning) [10]. Spotykane niekiedy terminy „uczenie się leniwe" (lazy learning) i „uczenie oparte na przykładach" (example-based learning) są synonimami. Tagery wykorzystujące uczenie się oparte na pamięci nazywamy w skrócie tagerami opartymi na pamięci.

Tager oparty na pamięci wyznacza tagowanie $M_{\hat{K}}(W) = \hat{T}$ jako ciąg tagów, w którym każde $\hat{T}_i$ jest tagiem T_k^{tr} pewnej pary $(D_k^{\mathrm{tr}}, T_k^{\mathrm{tr}})$ z tablicy wcześniej zapamiętanych przykładów $(D^{\mathrm{tr}}, T^{\mathrm{tr}})$, dla której wcześniej określona odległość $\Delta(D_k^{\mathrm{tr}}, \hat{D}_i)$ od bieżącego kontekstu $\hat{D}_i$ jest najmniejsza. To znaczy,

$$\hat{T}_i = T_k^{\mathrm{tr}}, \quad \text{gdzie} \quad k = \operatorname*{arg\,min}_{j \in \{1,...,L_{\mathrm{tr}}\}} \Delta(D_j^{\mathrm{tr}}, \hat{D}_i) \tag{74}$$

oraz $\hat{D}_i = (W_{i-C}, ..., W_{i+C}, \hat{T}_{i-1}, ..., \hat{T}_{i-n})$ dla pewnego $n < \infty$. Ponadto $\Delta : \mathcal{D} \times \mathcal{D} \to R$, $\mathcal{D} = \mathcal{W}^{2C+1} \times \mathcal{T}^n$, spełnia dla wszystkich $d, d', d'' \in \mathcal{D}$ warunki: $\Delta(d, d) = 0$, $\Delta(d, d') > 0$ dla $d \neq d'$, $\Delta(d, d') + \Delta(d', d'') \geqslant \Delta(d, d'')$.

Diagram 4: **Algorytm uczenia się transformacji na tekście anotowanym jednoznacznie**

podstaw $n \leftarrow 0$;
podstaw $U \leftarrow 0$;
dla każdego $i \in \{1, ..., L_{\rm tr}\}$ wykonaj {
  podstaw $T_i \leftarrow t$;
  podstaw $U \leftarrow U + 1$ jeśli $T_i = M(W^{\rm tr})_i$;
}
powtarzaj {
  podstaw $n \leftarrow n + 1$;
  podstaw $U^{\rm new} \leftarrow U$;
  dla każdego $F^{\rm prop} \in \mathcal{F}$ wykonaj {
    podstaw $U^{\rm prop} \leftarrow 0$;
    dla każdego $i \in \{1, ..., L_{\rm tr}\}$ wykonaj {
      podstaw $U^{\rm prop} \leftarrow U^{\rm prop} + 1$ jeśli $F^{\rm prop}(W^{\rm tr}_{i-C}, ..., W^{\rm tr}_{i+C}, T_{i-C}, ..., T_{i+C}) = M(W^{\rm tr})_i$;
    }
    jeśli $U^{\rm prop} > U^{\rm new}$ wykonaj {
      podstaw $F^{\rm new} \leftarrow F^{\rm prop}$, $U^{\rm new} \leftarrow U^{\rm prop}$;
    }
  }
  opuść pętlę jeśli $U^{\rm new} = U$;
  podstaw $F_n \leftarrow F^{\rm new}$;
  dla każdego $i \in \{1, ..., L_{\rm tr}\}$ wykonaj {
    podstaw $T^{\rm old}_i \leftarrow T_i$;
  }
  dla każdego $i \in \{1, ..., L_{\rm tr}\}$ wykonaj {
    podstaw $T_i \leftarrow F_n(W^{\rm tr}_{i-C}, ..., W^{\rm tr}_{i+C}, T^{\rm old}_{i-C}, ..., T^{\rm old}_{i+C})$;
  }
}

Uczenie się tagera opartego na pamięci jest bardzo proste. W tym celu algorytm uczenia odczytuje dane treningowe $(W^{\rm tr}, M(W^{\rm tr}))$ i zapamiętuje je jako zbiór $(D^{\rm tr}, T^{\rm tr})$, gdzie $D^{\rm tr} = (D^{\rm tr}_1, ..., D^{\rm tr}_{L_{\rm tr}})$, $T^{\rm tr} = (T^{\rm tr}_1, ..., T^{\rm tr}_{L_{\rm tr}})$, $D^{\rm tr}_i = (W^{\rm tr}_{i-C}, ..., W^{\rm tr}_{i+C}, M(W^{\rm tr})_{i-1}, ..., M(W^{\rm tr})_{i-n})$, $T^{\rm tr}_i = M(W^{\rm tr})_i$.

Warto zauważyć, że posługując się regułą (74), tager wyznacza tagowanie dla następnego tokenu dopiero po ustaleniu tagów dla tokenów poprzednich. Korzystniejsze może być wyznaczanie ciągu $\hat{T}$ jako minimalizującego całkowitą odległość od tablicy przykładów. Zakładając, że całkowita odległość jest sumą odległości dla pojedynczych tagów, $\hat{T}$ wyrażałoby się wtedy wzorem

$$\hat{T}_m = T^{\rm tr}_{k_m}, \quad \text{gdzie} \quad (k_1, ..., k_L) = \mathop{\arg\min}_{(j_1 ... j_L) : j_i \in \{1,...,L_{\rm tr}\}} \sum_{i=1}^{L} \Delta(D^{\rm tr}_{j_i}, D_i(T^{\rm tr}_{j_{i-1}}, ..., T^{\rm tr}_{j_{i-n}})),$$

$$D_i(T_{i-1}, ..., T_{i-n}) = (W_{i-C}, ..., W_{i+C}, T_{i-1}, ..., T_{i-n}). \tag{75}$$

Do znajdowania ciągu $\hat{T}$ zdefiniowanego wzorem (75) można wykorzystać algorytm Viterbiego (opisany w podrozdziale 1.6). W tym celu należy zdefiniować macierz przejścia jako

$$\phi_i(T_{i-n}, \Sigma_i) \quad = \quad \begin{cases} \min\limits_{j \in \{1,...,L_{\rm tr}\} : T^{\rm tr}_j = T_i} \Delta(D^{\rm tr}_j, D_i(T_{i-1}, ..., T_{i-n})), & \text{jeśli } \exists\, j \in \{1, ..., L_{\rm tr}\} : T^{\rm tr}_j = T_i, \\ \infty, & \text{w innym wypadku.} \end{cases} \tag{76}$$

Możliwość zdefiniowania macierzy przejścia dla tagerów opartych na pamięci, sugeruje że na odległość ciągu tagów od tablicy przykładów można patrzeć jak na pewną miarę prawdopodobieństwa ciągu tagów. Artykuł [12] omawia paralele między uczeniem się opartym na pamięci a modelowaniem probabilistycznym od nieco innej strony.

W dotychczasowych zastosowaniach uczenia się opartego na pamięci w lingwistyce komputerowej [9, 10, 13] przyjmowano, że odległość Δ wyraża się wzorem

$$\Delta(d, d') = \sum_{i=1}^{N} w_i \delta(f_i(d) = f_i(d')), \tag{77}$$

gdzie f_i są pewnymi wybranymi funkcjami cech (features) $f_i : \mathcal{D} \to \mathcal{V}_i$, $|\mathcal{V}_i| < \infty$. Kwestia wyboru funkcji f_i nie została szerzej przedyskutowana. W artykułach [9, 10, 13] funkcje te nie są dobierane w drodze automatycznego przeszukiwania jak w algorytmie na diagramie 4, lecz przyjmowane jako $f_i(D_j(T_{i-1}, ..., T_{i-n})) = W_{j+k}$, $k \in \{-C, ..., C\}$, lub $f_i(D_j(T_{i-1}, ..., T_{i-n})) = T_{j-l}$, $l \in \{1, ..., n\}$. Wagi w_i są zdefiniowane jako $w_i = 1$ w wersji algorytmu tagowania zwanej IB1, bądź jako zyski informacji $w_i = I(\mathcal{T}; \mathcal{V}_i)/H(\mathcal{V}_i)$ w wersji algorytmu zwanej IB1-IG [13]. Odpowiednie definicje dla IB1-IG wyrażają się wzorami

$$w_i = \frac{I(\mathcal{T}; \mathcal{V}_i)}{H(\mathcal{V}_i)}, \tag{78}$$

$$I(\mathcal{T}; \mathcal{V}_i) = -\sum_{t \in \mathcal{T}} \sum_{v_i \in \mathcal{V}_i} p_{\mathcal{T}\mathcal{V}_i}(t, v_i) \log \frac{p_{\mathcal{T}}(t) p_{\mathcal{V}_i}(v_i)}{p_{\mathcal{T}\mathcal{V}_i}(t, v_i)}, \tag{79}$$

$$H(\mathcal{V}_i) = -\sum_{v_i \in \mathcal{V}_i} p_{\mathcal{V}_i}(v_i) \log p_{\mathcal{V}_i}(v_i). \tag{80}$$

Prawdopodobieństwa $p_{\mathcal{T}\mathcal{V}_i}(t, v_i)$, $p_{\mathcal{T}}(t)$, $p_{\mathcal{V}_i}(v_i)$ szacowane są w oparciu o dane treningowe i estymację najbardziej wiarogodną, tzn.

$$p_{\mathcal{T}\mathcal{V}_i}(t, v_i) = \frac{1}{L_{\text{tr}}} \sum_{j=1}^{L_{\text{tr}}} \delta\left((t, v_i) = (T_j^{\text{tr}}, f_i(D_j^{\text{tr}}))\right), \tag{81}$$

$$p_{\mathcal{T}}(t) = \frac{1}{L_{\text{tr}}} \sum_{j=1}^{L_{\text{tr}}} \delta\left(t = T_j^{\text{tr}}\right), \tag{82}$$

$$p_{\mathcal{V}_i}(v_i) = \frac{1}{L_{\text{tr}}} \sum_{j=1}^{L_{\text{tr}}} \delta\left(v_i = f_i(D_j^{\text{tr}})\right). \tag{83}$$

Algorytm IB1-IG działa zdecydowanie lepiej niż IB1 [13]. Zyski informacji znane są także z zastosowań w drzewach decyzyjnych [31].

Podstawowym problemem w stosowaniu algorytmów IB1 i IB1-IG jest konieczność porównywania przy tagowaniu bieżącego kontekstu $\hat{D}_i$ ze wszystkimi kontekstami D_j^{tr} sczytanymi z danych treningowych (L_{tr} porównań dla każdego tagowanego tokenu). Przechowując w pamięci wyłącznie tablicę wartości $(f_1(D_j^{\text{tr}}), ..., f_N(D_j^{\text{tr}}), T_j^{\text{tr}})$, liczbę porównań bez zmiany wyniku tagowania można ograniczyć do co najwyżej $|\mathcal{V}_1|...|\mathcal{V}_N|$ — nie jest to jednak redukcja zadowalająca. Często proponowanym w literaturze maszynowego uczenia się rozwiązaniem problemu jest usunięcie z tablicy $(D^{\text{tr}}, T^{\text{tr}})$ znacznej części przykładów, począwszy od tych, których konteksty D_j^{tr} są najbardziej lub najmniej typowe bądź najsilniej lub najsłabiej przewidują tagi T_j^{tr}.

Daelemans i inni w pracy [10] szczegółowo przebadali wpływ usuwania przykładów na akuratność działania IB1-IG i jego modyfikacji dla 4 różnych zadań (generowanie transkrypcji fonetycznej tekstu, tagowanie częściami mowy, ujednoznacznianie nadrzędników fraz przyimkowych, oznaczanie fraz nominalnych). W czterech typach prób usuwane były różne ułamki przykładów cechujących się najmniejszą/największą typowością/siłą przewidywania tagu. Zarówno typowość jak siła przewidywania tagu zdefiniowane były formalnie. Przykładowo, siła przewidywania tagu (class prediction strength) CPS_k dla przykładu $(D_k^{\text{tr}}, T_k^{\text{tr}})$ wyraża się jako

$$\text{CPS}_k = \frac{\sum_{m=1}^{L_{\text{tr}}} \delta(k = \arg\min_{j \in \{1,...,L_{\text{tr}}\} : j \neq m} \Delta(D_m^{\text{tr}}, D_j^{\text{tr}})) \delta(T_k^{\text{tr}} = T_m^{\text{tr}})}{\sum_{m=1}^{L_{\text{tr}}} \delta(k = \arg\min_{j \in \{1,...,L_{\text{tr}}\} : j \neq m} \Delta(D_m^{\text{tr}}, D_j^{\text{tr}}))}. \tag{84}$$

Szczegółowy wpływ usuwania przykładów na akuratność działania klasyfikatorów zależał od zadania. Ogólnie, autorzy pracy [10] stwierdzili, że jakiekolwiek usuwanie przykładów o najmniejszej typowości lub sile przewidywania tagu jest szkodliwe. Można natomiast bez większego spadku akuratności usunąć do 20% przykładow o największej typowości lub sile przewidywania tagu.

Aby przyspieszyć tagowanie bez usuwania przykładów, w pracy [13] zaproponowany został heurystyczny algorytm uczenia i tagowania nazwany IGTree. Algorytm IGTree w teście przeprowadzonym w artykule [13] osiągał akuratność równą akuratności IB1-IG, działając 100 razy szybciej i zużywając 30 razy mniej pamięci. Algorytm uczenia się przedstawiamy na diagramie 5, a algorytm tagowania — na diagramie 6. Algorytmy IGTree stanowią pewną modyfikację znanych algorytmów uczenia i przeszukiwania drzew decyzyjnych. Od algorytmów tych algorytm IGTree odróżnia to, że niezależnie od przebiegu testów równości dla poszczególnych funkcji cech f_i kolejność testowania tych funkcji jest zawsze ta sama. To znaczy, funkcje f_i uporządkowane są według malejących wartości w_i na zbiorze wszystkich danych treningowych, a nie według malejących w_i na odpowiednich podzbiorach tychże danych jak w typowych algorytmach dla drzew decyzyjnych (ID3, C4.5 i poźniejsze) [31].

Ponieważ w kolejnych decyzjach (testach/przepisaniach) algorytm IGTree (diagram 5) — a także algorytm Brilla (diagram 4) nie wnioskują wyłącznie na podstawie coraz mniejszych (mniej reprezentatywnych) podzbiorów danych, dzięki temu oba te algorytmy są znacznie bardziej odporne na przeuczenie niż algorytm ID3 [10, 5].

Diagram 5: **Algorytm uczenia się IGTree**

podstaw $(g_1, ..., g_N) \leftarrow$ posortuj $(f_1, ..., f_N)$ malejąco według wartości $(w_1, ..., w_N)$;
podstaw $B_{\mathrm{tr}} \leftarrow$ BuildIGTree$((g_1(D_1), ..., g_N(D_1), T_1), ..., (g_1(D_{L_{\mathrm{tr}}}), ..., g_N(D_{L_{\mathrm{tr}}}), T_{L_{\mathrm{tr}}}))$;

procedura BuildIGTree$((V_1^1, ..., V_1^n, T_1), ..., (V_L^1, ..., V_L^n, T_L))$ {
    podstaw $c[\cdot] \leftarrow 0$, $l[\cdot] \leftarrow ()$;
    podstaw $m \leftarrow 0$, $\sigma \leftarrow$ prawda;
    dla każdego $i \in \{1, ..., L\}$ wykonaj {
        podstaw $c[T_i] \leftarrow c[T_i] + 1$;
        podstaw $m \leftarrow c[T_i]$, $t \leftarrow T_i$ jeżeli $c[T_i] > m$;
        podstaw $\sigma \leftarrow$ fałsz jeżeli $t \neq T_i$;
    }
    zwróć $(t, ())$ jeżeli $n = 0$ lub σ;
    dla każdego $i \in \{1, ..., L\}$ wykonaj {
        podstaw $l[V_i^1] \leftarrow$ dodaj $(V_i^2, ..., V_i^n, T_i)$ na koniec listy $l[V_i^1]$;
    }
    podstaw $r \leftarrow ()$;
    dla każdego $v \in$ zbiór indeksów $l[\cdot]$ wykonaj {
        podstaw $r \leftarrow$ dodaj $(v, \mathrm{BuildIGTree}(l[v]))$ na koniec listy r;
    }
    zwróć (t, r);
}

Diagram 6: **Algorytm tagowania IGTree**

dla każdego $i \in \{1, ..., L\}$ wykonaj {
    podstaw $\hat{T}_i \leftarrow$ SearchIGTree$((g_1(\hat{D}_i), ..., g_N(\hat{D}_i)), B_{\mathrm{tr}})$;
}

procedura SearchIGTree$((V^1, ..., V^n), (t, r))$ {
    zwróć t jeżeli $r = ()$;
    dla każdego $(v, B) \in r$ wykonaj {
        zwróć SearchIGTree$((V^2, ..., V^n), B)$ jeżeli $v = V^1$;
    }
    zwróć t;
}

Niedawno, w pracy [23] zaproponowany został algorytm LSOMMBL. Jest to algorytm tagowania opartego na pamięci z dostępem za pośrednictwem samoorganizującej się mapy. W podejściu tym, w trakcie uczenia się oprócz tablicy zawierającej L_{tr} przykładów konstruowana jest mapa zawierająca $\sqrt{L_{\mathrm{tr}}}$ jednostek. Każdej jednostce z mapy przypisany jest pewien kontekst oraz zbiór wskaźników do tablicy przykładów, przy czym każdy przykład wskazywany jest przez dokładnie jedną jednostkę. W trakcie tagowania, najpierw znajdowana jest jednostka najbliższa bieżącemu kontekstowi. Następnie wyszukuje się taki przykład z tablicy wskazywany przez znalezioną jednostkę, którego kontekst także jest najbliży bieżącemu kontekstowi. Tag przypisywany bieżącemu kontekstowi jest tagiem tego przykładu. W rezultacie średnia ilość porównań zostaje zredukowana z L_{tr} do $2\sqrt{L_{\mathrm{tr}}}$. Akuratność dla algorytmu SOMMBL także jest zbliżona do akuratności dla algorytmu IB1-IG.

2 Praktyka i oprogramowanie

2.1 Tagowanie częściami mowy i dezambiguacja morfosyntaktyczna

Metody uczenia się klasyfikatorów (tagerów) przedstawione w rozdziale 1 stosuje się, testuje i usprawnia w praktycznych zastosowaniach do różnych dyscyplin. Wśród dyscyplin tych ważne miejsce zajmuje lingwistyka komputerowa. Spowodowane jest to nie tylko znaczną przydatnością wytrenowanych tagerów, lecz także istnieniem bardzo dużych zbiorów danych tekstowych w wersji elektronicznej, w tym anotowanych (korpusy). W obrębie lingwistyki komputerowej prototypowym zadaniem, w którym wykorzystuje się maszynowe uczenie się klasyfikatorów, jest tagowanie częściami mowy oraz dezambiguacja morfosyntaktyczna.

Tagowanie częściami mowy (part-of-speech tagging) to przypisanie każdemu okazowi (wystąpieniu) słowa graficznego w tekście etykietki określającej analizę morfosyntaktyczną tego okazu w zawierającym go zdaniu. [8] Termin „part-of-speech tagging" został ukuty na potrzeby opisu języków analitycznych (m. in. języka angielskiego), w których wystarczające analizy morfosyntaktyczne sprowadzają się w znacznej mierze do wyspecyfikowania dla danego okazu słowa odpowiadającej mu części mowy. [9] Dla języków o bogatej fleksji (m. in. polszczyzny) dostateczne analizy morfosyntaktyczne są bardziej złożone. Aby uwypuklić ten fakt, dalej będziemy używać określenia „tagowanie morfosyntaktyczne" jako synonimu terminu „tagowanie częściami mowy".

Przykład (85) ilustruje tagowanie morfosyntaktyczne dla języka angielskiego [29], przykład (86) — dla języka czeskiego [21]. Tagi odpowiednie dla okazów słów graficznych oddzielone są od nich ukośnikiem (/).

(85) The/AT representative/NN put/VBD chairs/NNS on/IN the/AT table/NN .

(86) Naše/náš/PSHS1-P1----- metoda/metoda/NNFS1-----A-- přitom/přitom/Db----------
využívá/využívat/VB-S---3P-AA- exponenciálního/exponenciální/AAIS2----1A--
pravděpodobnostního/pravděpodobnostní/AAIS2----1A-- modelu/model/NNIS2-----A-- ...

Pomiędzy tagsetem dla języka angielskiego (analitycznego) a tagsetem dla języka czeskiego (o bogatej fleksji) widać wyraźne różnice. Dla języka angielskiego zbiór wszystkich tagów $\mathcal{T}$ zawiera w zależności od systemu anotacji od około 50 do około 150 tagów. Na potrzeby uczenia się tagowania tagi te można traktować jako symbole atomowe. Oznaczenia tagów dla kilku tagsetów języka angielskiego podano w podręczniku [29]. Dla języka czeskiego tagset morfosyntaktyczny $\mathcal{T}$ jest podzbiorem iloczynu kartezjańskiego $\mathcal{C}_1 \times \mathcal{C}_2 \times ... \times \mathcal{C}_n$ pewnych zbiorów atrybutów $\mathcal{C}_j$. Tagset tego typu nazywamy tagsetem pozycyjnym. W szczególności w przykładzie (86) wartości poszczególnych atrybutów oznaczane są pojedynczymi literami, $|\mathcal{C}_j|$ jest rzędu 10, zaś $n = 13$. Kolejne atrybuty określają część mowy, podczęść mowy, rodzaj, liczbę, przypadek, rodzaj posiadacza, liczbę posiadacza, osobę, czas, stopień, negację, stronę, wariant. Kompletny opis tagsetu podaje opracowanie [27]. Pełny tagset $\mathcal{T}$ jest stosunkowo małym podzbiorem iloczynu $\mathcal{C}_1 \times \mathcal{C}_2 \times ... \times \mathcal{C}_n$, czyli $|\mathcal{T}| \ll 10^{13}$, jednak jest on znacznie większy niż odpowiedni tagset dla języka angielskiego. Przykładowo, w czeskiej powieści liczącej 100 000 tokenów zaobserwować można wystąpienie około 1000 różnych tagów [19]. Zbliżona liczba tagów pojawia się w odpowiednim tekście w języku słoweńskim, zatem i dla języka polskiego jako bardzo bliskiego gramatycznie można się spodziewać podobnych rezultatów. Omówienie planowanego tagsetu na potrzeby anotowanego korpusu pisanego języka polskiego zawiera raport [37]. Tagsety dla języków: angielskiego, rumuńskiego, czeskiego, bułgarskiego, słoweńskiego, węgierskiego i estońskiego używane w projekcie MULTEXT-EAST podaje specyfikuje [17].

Źródłem zasadniczej trudności w tagowaniu morfosyntaktycznym jest to, że wartości odpowiedniego tagu nie można ustalić posługując się wyłącznie wiedzą o danej słowoformie (słowie graficznym). W przykładach (87)–(89) słowoforma „kurze" jest kolejno: przymiotnikiem rodzaju nijakiego, liczby pojedynczej, w mianowniku (87); rzeczownikiem rodzaju męskorzeczowego, liczby mnogiej, w bierniku (88); rzeczownikiem rodzaju żeńskiego, liczby pojedynczej, w celowniku (89).

(87) Dodać do mąki jedno kurze jajko.

(88) Czy wytarłeś kurze z kominka?

(89) Wyszła na podwórze i rzuciła kurze ziarno.

O ostatecznej interpretacji słowoformy przesądza kontekst, zwykle jednak dla danej słowoformy lingwiści są w stanie podać wszystkie dopuszczalne tagi morfosyntaktyczne. Podanie zbioru interpretacji dla danej słowoformy nazywa się analizą morfologiczną. Formalnie, analiza morfologiczna (morphological analysis) G to

[8] Słowem graficznym nazywamy ciąg znaków nie rozdzielonych spacją lub znakiem przestankowym. Równolegle używamy na to samo pojęcie bliskoznacznego terminów „słowoforma". Wyrażenia „okaz słowa" oraz „token" oznaczają wystąpienie słowa graficznego (słowoformy) na konkretnej pozycji w tekście .

[9] Przykładowo, w tradycji gramatyki szkolnej dla języka polskiego części mowy dzielą się na: rzeczowniki, przymiotniki, liczebniki, zaimki, czasowniki, przysłówki, przyimki, partykuły, spójniki, wykrzykniki.

odwzorowanie $\mathcal{W} \to 2^{\mathcal{T}}$, gdzie $\mathcal{W}$ jest zbiorem słowoform, a zbiór tagów $\mathcal{T}$ jest tagsetem morfosyntaktycznym. Tagowanie morfosyntaktyczne $M : \mathcal{W}^L \to \mathcal{T}^L$ można przedstawić jako jako złożenie bezkontekstowej analizy morfologicznej $G : \mathcal{W} \to 2^{\mathcal{T}}$ i kontekstowej dezambiguacji morfosyntaktycznej $B : (2^{\mathcal{T}})^L \to \mathcal{T}^L$, gdzie dezambiguacja morfosyntaktyczna (morphosyntactic disambiguation) B to takie tagowanie ciągu analiz morfologicznych $G(W) = (G(W_1), G(W_2), ..., G(W_L))$, że $B(G(W))_i \in G(W_i)$ i $M(W) = B(G(W))$.

Analiza morfologiczna pomimo częstych idiosynkrazji jest przekształceniem dostatecznie regularnym, aby dawała się dobrze przybliżać funkcjami obliczalnymi. Analizatorem morfologicznym (morphological analyzer) nazywamy program obliczający analizę morfologiczną dla danego słowa graficznego. Dezambiguator morfosyntaktyczny (morphosyntactic disambiguer) to tager obliczający analizę morfosyntaktyczną dla danej analizy morfologicznej. Tagerem morfosyntaktycznym (morphosyntactic tagger, part-of-speech tagger) nazywamy tager mający działać jak złożenie idealnego analizatora morfologicznego z idealnym dezambiguatorem morfosyntaktycznym. To znaczy, tager ten oblicza analizę morfosyntaktyczną dla danego wyłącznie tekstu segmentowanego.

Do wyodrębnienia analizy morfologicznej jako sensownego stadium pośredniego w tagowaniu morfosyntaktycznym skłania także jej powiązanie z innym przekształceniem zwanym tagowaniem lematycznym (lematyzacją), czyli sprowadzeniem do formy hasłowej (lematu). Dla podanych przykładów słowoformy „kurze" lematy brzmią: „kurzy" (87), „kurz" (88), „kura" (89). Lematy słowoform dla przykładu czeskiego (86) umieszczone są między słowoformami a tagami morfosyntaktycznymi. Tagowanie lematyczne wymagane jest przy wszelkiego rodzaju analizie semantycznej i jej zastosowaniach (nawet bardzo powierzchownych, takich jak wyszukiwarki internetowe). Formalnie, tagowanie lematyczne to przekształcenie $\mathcal{W} \to \mathcal{L}$, gdzie $\mathcal{L}$ jest zbiorem lematów, $\mathcal{W} \subset A^*$, $\mathcal{L} \subset A^*$. Program obliczający tagowanie lematyczne nazywamy tagerem lematycznym. Warto zauważyć, że tagowanie lematyczne wymaga znajomości kontekstu. Podobnie jak tagowanie morfosyntaktyczne, tagowanie lematyczne $\mathcal{W}^L \to \mathcal{L}^L$ można przedstawić jako złożenie analizy lematycznej $\mathcal{W} \to 2^{\mathcal{L}}$ i dezambiguacji lematycznej $(2^{\mathcal{L}})^L \to \mathcal{L}^L$.

Analiza morfologiczna i lematyczna są ze sobą sprzężone — dla języków o bogatej fleksji praktyczne implementacje analizatorów morfologicznych i lematycznych wykorzystywałyby prawie identyczne słowniki. Dlatego też zwykle konstruuje się jeden program obliczający przekształcenie $\mathcal{W} \to 2^{(\mathcal{L} \times \mathcal{T})}$, który nazywany jest nadal analizatorem morfologicznym. [10] Dodatkowym uzasadnieniem takiego rozwiązania jest to, że mając tag morfosyntaktyczny ujednoznaczniony na podstawie kontekstu oraz pełen zbiór par (lemat, tag morfosyntaktyczny) dla danej słowoformy często można jednoznacznie ustalić lemat (dla homonimów nieabsolutnych jak np. „niż", „albo", „lub" itp.). Zdecydowanie rzadziej lematyzacja wymaga dodatkowej wiedzy o kontekście (dla homonimów absolutnych jak np. „zamek").

W tradycyjnym podejściu do anotacji korpusów języka angielskiego lematyzacji się nie uzwględnia i podaje przy słowoformach wyłącznie tagi morfosyntaktyczne. Dla tego języka istnieje prosty i dostatecznie dobry algorytm lematyzacji, zwany algorytmem Portera i opisany np. w podręczniku [26]. Zwykle też nie konstruuje się analizatorów morfologicznych jako osobnych programów, gdyż istnieją proste i dostatecznie dobre tagery morfosyntaktyczne wykorzystujące heurystyczną analizę morfologiczną. Dla języków fleksyjnych (w tym słowiańskich) analiza morfologiczno-lematyczna jest przekształceniem znacznie bardziej złożonym i proste anglocentryczne heurystyki w jej wypadku nie wychodzą najlepiej. Szczegółowe badania [19] przeprowadzone na równoległym korpusie MULTEXT-EAST wykazały, że uczenie dezambiguatora morfosyntaktycznego na wynikach starannie opracowanego analizatora morfologicznego jest dobrym pomysłem również dla języka angielskiego. Uczonym tagerem był dezambiguator Hajiča (opisany w podrozdziale 2.3), dane treningowe zaś liczyły tylko około 100 000 tokenów (równoległy tekst „Roku 1984" G. Orwella w 6 językach). Rozpatrywane były dwa przypadki. W pierwszym przypadku dezambiguator był uczony na czystym tekście anotowanym (dane treningowe = słowoformy + jednoznaczne tagi) — w tym celu dezambiguator został rozbudowany o zgadywanie analiz morfologicznych w oparciu o zakończenia słów. W drugim przypadku dezambiguator był uczony na tekście anotowanym przetworzonym przez dobry analizator morfologiczny (dane treningowe = pełne zbiory tagów + jednoznaczne tagi). Ułamki błędów na danych testowych dla wytrenowanych tagerów ujmuje tabela 1. Wykorzystanie dobrego analizatora morfologicznego zmniejsza ułamek błędów 2–3-krotnie, i to bardziej dla języka angielskiego niż dla języków słowiańskich!

Rozbicie tagowania morfosyntaktycznego M na bezkontekstową analizę morfologiczną G i kontekstową dezambiguację morfosyntaktyczną B jest interesującą heurystyką lingwistyczną, praktycznie niedyskutowaną w dotychczasowej literaturze maszynowego uczenia się klasyfikacji. W podrozdziałach 2.2 i 2.3 omawiamy dwie metody tagowania i uczenia się tagowania jawnie wykorzystujące to rozbicie.

Notacja używana w podrozdziałach 2.2 i 2.3 jest dość rozbudowana. Ciąg tokenów oznaczamy jako $W = (W_1, ..., W_L)$, ciąg analiz morfologicznych oznaczamy jako $A = (A_1, ..., A_L)$, $A_i = G(W_i)$, $A_i = \left\{T_i^{(1)}, ..., T_i^{(p_i)}\right\}$, ciąg analiz morfosyntaktycznych oznaczamy jako $T = (T_1, ..., T_L)$, $T = B(A)$. Dla każdego i istnieje takie $r \in \{1, ..., p_i\}$, że $T_i = T_i^{(r)}$. Mamy $W_i \in \mathcal{W}$, $A_i \in 2^{\mathcal{T}}$, $T_i \in \mathcal{T}$, podobnie oznaczamy dowolne elementy zbiorów

[10] Omówienie dotychczas opracowanych analizatorów morfologicznych dla języka polskiego i ich ocenę zawiera raport [22].

Tabela 1: Wyniki tagera Hajiča na korpusie MULTEXT-EAST [19].

język	liczba różnych tagów w danych treningowych	ułamek błędów		iloraz ułamków (a/b)
		bez osobnego analizatora (a)	z analizatorem (b)	
angielski	139	9.18%	3.58%	2.56
rumuński	486	7.76%	3.35%	2.31
czeski	970	18.83%	9.59%	1.96
słoweński	1033	16.26%	9.00%	1.80
estoński	476	13.59%	5.34%	2.54
węgierski	401	8.16%	2.58%	3.16

— $w \in \mathcal{W}$, $a \in 2^{\mathcal{T}}$, $t \in \mathcal{T}$.

W przypadku tagsetu pozycyjnego $\mathcal{T} \subset \mathcal{C}_1 \times \mathcal{C}_2 \times ... \times \mathcal{C}_n$ (podrozdział 2.3), tagi morfologiczne $a \in 2^{\mathcal{T}}$ oznaczamy jako $a = \left\{t^{(1)}, ..., t^{(p)}\right\}$, gdzie $t^{(r)} = \left(c_1^{(r)}, ..., c_n^{(r)}\right)$ są tagami morfosyntaktycznymi, $c_j^{(r)} \in \mathcal{C}_j$, zaś $p \in \{1, ..., |\mathcal{C}_1|...|\mathcal{C}_n|\}$. W ślad za [21], definiujemy klasy niejednoznaczności $e_j = \left\{c_j^{(r)} : r \in \{1, ..., p\}\right\}$. Ciąg analiz morfologicznych dla ciągu tokenów $W = (W_1, ..., W_L)$ ma postać $A = (A_1, ..., A_L)$, gdzie $A_i = \left\{T_i^{(1)}, ..., T_i^{(p_i)}\right\}$, $T_i^{(r)} = \left(C_{1i}^{(r)}, ..., C_{ni}^{(r)}\right)$, $C_{ji}^{(r)} \in \mathcal{C}_j$. Ciąg analiz morfosyntaktycznych ma postać $T = (T_1, ..., T_L)$, gdzie $T_i = (C_{1i}, ..., C_{ni})$, $C_{ji} \in \mathcal{C}_j$. Ciągi klas niejednoznaczności oznaczamy jako $E_j = (E_{j1}, ..., E_{jL})$, gdzie $E_{ji} = \left\{C_{ji}^{(r)} : r \in \{1, ..., p_i\}\right\}$.

2.2 Dezambiguacja oparta na regułach skreślania

Dezambiguacja oparta na regułach skreślania nadmiarowej anotacji jest pewną modyfikacją tagowania opartego na transformacjach (opisanego w podrozdziale 1.12). Została ona opracowana na potrzeby dezambiguacji morfosyntaktycznej dla języka angielskiego [6]. W metodzie tej najpierw tworzy się tekst anotowany nadmiarowo $(W, A^{(0)})$, gdzie $A^{(0)} = A = (A_1, ..., A_L)$, $A_i = G(W_i)$, a następnie automatycznie skreśla się nadmiarowe tagi w opaciu o pewien ciąg reguł. [11] Formalnie, analiza morfosyntaktyczna zwracana przez dezambiguator równa jest $T^{(N)}$, gdzie

$$A_i^{(n+1)} = F_{n+1}(D_i^{(n)}), \tag{90}$$

zaś $(F_1, ..., F_N)$ jest ciągiem przekształceń $\mathcal{D} \to 2^{\mathcal{T}}$, $\mathcal{D} = (\mathcal{W} \cup \{\epsilon\})^{2C+1} \times (2^{\mathcal{T}} \cup \{\epsilon\})^{2C+1}$, $A_i^{(n)} = W_i = \epsilon$ dla $i \notin \{1, ..., L\}$ oraz $D_i^{(n)} = (W_{i-C}, ..., W_{i+C}, A_{i-C}^{(n)}, ..., A_{i+C}^{(n)})$. Każde przekształcenie F_n ma postać reguły: „Zamień wartość tagu morfologicznego $a_n \in 2^{\mathcal{T}}$ na wartość $\{y_n\}$, gdzie $y_n \in a_n$, jeżeli kontekst spełnia warunek $\phi_n \in \Phi$". To znaczy,

$$F_n(D_i) = \begin{cases} \{y_n\}, & \text{jeśli } A_i = a_n,\ y_n \in a_n \text{ i } \phi_n(\tilde{D}_i), \\ A_i, & \text{w innym wypadku,} \end{cases} \tag{91}$$

gdzie $D_i = (W_{i-C}, ..., W_{i+C}, A_{i-C}, ..., A_{i+C})$, $\tilde{D}_i = (W_{i-C}, ..., W_{i+C}, A_{i-C}, ...A_{i-1}, A_{i+1}, ..., A_{i+C})$. Przestrzeń warunków Φ używaną w oryginalnej wersji algorytmu do tagowania morfosyntaktycznego tekstów w języku angielskim podano w artykule [6].

Interesującą własnością tagowania opartego na anotacji nadmiarowej i skreślaniu jest to, że opracowano dla niego skuteczny algorytm uczenia się reguł skreślania F_n bez nadzoru. To znaczy, algorytm uczenia się nie wymaga danych treningowych w postaci tekstu anotowanego nienadmiarowo [6] (przynajmniej dla tagowania morfosyntaktycznego języka angielskiego). Algorytm ten przedstawiamy na diagramie 7. Jest on wariacją na temat algorytmu z diagramu 4, a główna modyfikacja polega na wzięciu innej funkcji oceniającej regułę. W każdej iteracji, dla każdego warunku $\phi \in \Phi$ i każdego tagu morfologicznego $a \in 2^{\mathcal{T}}$, proponowana jest reguła

[11] Kompletny tager opisany w pracy [6] wykorzystuje heurystyczny analizator morfologiczny, który dla każdego $w \in \mathcal{W}$ zwraca zbiór $G(w) = \left\{T_i^{\text{tr}} : i \in \{1, ..., L_{\text{tr}}\} \text{ i } W_i^{\text{tr}} = w\right\}$.

Diagram 7: **Algorytm uczenia się reguł skreślania w tekście anotowanym nadmiarowo**

podstaw $n \leftarrow 0$;
dla każdego $i \in \{1, ..., L_{\rm tr}\}$ wykonaj {
  podstaw $A_i \leftarrow G(W_i^{\rm tr})$;
}
powtarzaj {
  podstaw $n \leftarrow n + 1$;
  podstaw $\Delta^{\rm new} \leftarrow 0$;
  dla każdego $\phi^{\rm prop} \in \Phi$ wykonaj {
    dla każdego $a^{\rm prop} \in 2^{\mathcal{T}}$ wykonaj {
      dla każdego $z \in a^{\rm prop}$ wykonaj {
        podstaw $n_1 \leftarrow 0$, $n_2 \leftarrow 0$;
        dla każdego $i \in \{1, ..., L_{\rm tr}\}$ wykonaj {
          podstaw $n_1 \leftarrow n_1 + 1$ jeśli $A_i = a^{\rm prop}$;
          podstaw $n_2 \leftarrow n_2 + 1$ jeśli $A_i = a^{\rm prop}$ i $\phi^{\rm prop}(W_{i-C}^{\rm tr}, ..., W_{i+C}^{\rm tr}, A_{i-C}, ...A_{i-1}, A_{i+1}, ..., A_{i+C})$;
        }
        podstaw $c[z] \leftarrow n_1$, $p[z] \leftarrow n_2/n_1$;
      }
      podstaw $y^{\rm prop} \leftarrow \arg\max_{z \in a} p[z]$, $r^{\rm prop} \leftarrow \arg\max_{z \in a \setminus \{y^{\rm prop}\}} p[z]$, $\Delta^{\rm prop} \leftarrow c[y^{\rm prop}]\,(p[y^{\rm prop}] - p[r^{\rm prop}])$;
      jeśli $\Delta^{\rm prop} > \Delta^{\rm new}$ wykonaj {
        podstaw $\phi^{\rm new} \leftarrow \phi^{\rm prop}$, $a^{\rm new} \leftarrow a^{\rm prop}$, $y^{\rm new} \leftarrow y^{\rm prop}$, $\Delta^{\rm new} \leftarrow \Delta^{\rm prop}$;
      }
    }
  }
  opuść pętlę jeśli $\Delta^{\rm new} \leqslant 0$;
  podstaw $\phi_n \leftarrow \phi^{\rm new}$, $a_n \leftarrow a^{\rm new}$, $y_n \leftarrow y^{\rm prop}$;
  dla każdego $i \in \{1, ..., L_{\rm tr}\}$ wykonaj {
    podstaw $A_i^{\rm old} \leftarrow A_i$;
  }
  dla każdego $i \in \{1, ..., L_{\rm tr}\}$ wykonaj {
    podstaw $A_i \leftarrow \{y_n\}$ jeśli $A_i^{\rm old} = a_n$ i $\phi_n(W_{i-C}^{\rm tr}, ..., W_{i+C}^{\rm tr}, A_{i-C}^{\rm old}, ..., A_{i-1}^{\rm old}, A_{i+1}^{\rm old}, ..., A_{i+C}^{\rm old})$;
  }
}

przepisująca tag a jako $\{y\}$, jeżeli kontekst spełnia warunek ϕ. Regule tej przypisywana jest ocena Δ. Wartości y i Δ wybierane są jako

$$y = \arg\max_{z \in a} \frac{c(\phi, z)}{c(z)}, \quad r = \arg\max_{z \in a \setminus \{y\}} \frac{c(\phi, z)}{c(z)}, \quad \Delta = c(y) \left[\frac{c(\phi, y)}{c(y)} - \frac{c(\phi, r)}{c(r)} \right], \tag{92}$$

gdzie

$$c(z) = \sum_{i=1}^{L_{\rm tr}} \delta(A_i = \{z\}), \tag{93}$$

$$c(\phi, z) = \sum_{i=1}^{L_{\rm tr}} \delta(A_i = \{z\}) \delta(\phi(W_{i-C}^{\rm tr}, ..., W_{i+C}^{\rm tr}, A_{i-C}, ...A_{i-1}, A_{i+1}, ..., A_{i+C})), \tag{94}$$

$A = (A_1, ..., A_{\rm tr})$ jest zaś bieżącym ciągiem tagów. Następnie wybierana i wykonywana jest reguła o maksymalnej ocenie Δ. Algorytm wychodzi z pętli, jeżeli dla najlepszej reguły $\Delta \leqslant 0$.

2.3 Dezambiguacja oparta na metodzie Hajiča

Dezambiguator morfosyntaktyczny Hajiča [21] został opracowany na potrzeby dezambiguacji morfosyntaktycznej dla języka czeskiego. Jego jedyny tryb pracy polega na tym, że ujednoznacznia on wyniki dostarczone przez niezależny od niego analizator morfologiczny, tzn. dla każdego tokenu wybiera jeden z kilku tagów wskazywanych jako możliwe przez niezależne źródło. Koncepcyjnie, dezambiguator Hajiča nawiązuje do tagerów opartych

na modelowaniu maksimum entropii (patrz podrozdział 1.11), jak np. [34] oraz tagera Brilla (patrz podrozdział 1.12). Główna różnica w stosunku do modelowania maksimum entropii polega na zastąpieniu pracochłonnego poszukiwania parametrów λ_i minimalizujących lagranżjan $L(p_\lambda, \lambda)$ przez naiwne założenie Bayesa (naive Bayes assumption). Modyfikacja ta pogarsza jakość modelowania dla ustalonego ciągu funkcji charakterystycznych $(f_l)_{l=1}^k$, lecz pozwala znacząco zwiększyć liczbę k rozpatrywanych funkcji charakterystycznych przy ustalonym całkowitym czasie uczenia się modelu. Potrzeba znaczącego zwiększenia liczby funkcji charakterystycznych związana jest z gigantycznymi rozmiarami tagsetu.

Dla języka czeskiego przyjęty tagset morfosyntaktyczny $\mathcal{T}$ jest podzbiorem iloczynu $\mathcal{C}_1 \times ... \times \mathcal{C}_n$. Tagi morfologiczne $a \in 2^{\mathcal{T}}$ mają postać $a = \left\{t^{(1)}, ..., t^{(p)}\right\}$, gdzie $t^{(r)} = \left(c_1^{(r)}, ..., c_n^{(r)}\right)$ są tagami morfosyntaktycznymi, $c_j^{(i)} \in \mathcal{C}_j$. Teoretycznie $1 \leqslant p \leqslant |\mathcal{C}_1|...|\mathcal{C}_n|$, praktycznie $p < 10$. W metodzie Hajiča ujednoznacznianie całego tagu morfologicznego $a = \left\{t^{(1)}, ..., t^{(p)}\right\}$ do pewnego tagu morfosyntaktycznego $t \in a$ traktowane jest jako niezależne ujednoznacznianie poszczególnych klas niejednoznaczności (ambiguity classes) $e_j = \left\{c_j^{(r)} : r \in \{1, ..., p\}\right\}$ do subtagów $c_j \in a_j$. Przybliżając proces tagowania niestacjonarnym modelem Markowa n-tego rzędu (25) z $n = 2$, zakłada się zatem także że, że prawdopodobieństwo $p(T_i|A_i, D_i)$ tagu $T_i \in A_i$ przy zadanej analizie $A_i \in 2^{\mathcal{T}}$ i kontekście $D_i = (W_{i-C}, ..., W_{i+C}, T_{i-1}, ..., T_{i-n}) \in \mathcal{D}$ wynosi

$$p(t|a,d) \approx \frac{\prod_{j=1}^n p(c_j|e_j,d)}{\sum_{t' \in a} \prod_{j=1}^n p(c'_j|e_j,d)}, \tag{95}$$

gdzie $t = (c_1, ..., c_n), t' = (c'_1, ..., c'_n) \in a$, $a \in 2^{\mathcal{T}}$, $d \in \mathcal{D}$.

Model prawdopodobieństwa $p(c_j|e_j, d)$ konstruowany jest dla każdej klasy niejednoznaczności e_j z osobna w oparciu o wybrane funkcje charakterystyczne $f_l : \mathcal{C}_j \times \mathcal{D} \to \{0, 1\}$, $l \in S(e_j)$. Zbiory indeksów wybranych funkcji charakterystycznych $S(e_j)$ są podzbiorami zbioru indeksów $\{1, ..., k\}$ pewnego maksymalnego ciągu funkcji $(f_l)_{l=1}^k$, $S(e_j) \subset \{1, ..., k\}$. [12] Aby oszacować $p(c_j|e_j, d)$, przyjmuje się naiwne założenie Bayesa, to znaczy kładzie się

$$p(c_j|e_j,d) \approx \frac{\prod_{l \in S(e_j)} p(c_j|e_j,l)\delta(f_l(c_j,d) \neq 0)}{\sum_{c'_j \in e_j} \prod_{l \in S(e_j)} p(c'_j|e_j,l)\delta(f_l(c'_j,d) \neq 0)}. \tag{96}$$

Wartości $p(c_j|e_j, l)$ wyznacza się na podstawie danych treningowych $(W^{\mathrm{tr}}, A^{\mathrm{tr}}, T^{\mathrm{tr}}) = (W^{\mathrm{tr}}, G(W^{\mathrm{tr}}), M(W^{\mathrm{tr}}))$, z grubsza w oparciu o wzór

$$p(c_j|e_j,l) \approx \frac{\sum_{i=1}^{L_{\mathrm{tr}}} \delta(C_{ji}^{\mathrm{tr}} = c_j)\delta(E_{ji}^{\mathrm{tr}} = e_j) f_l(C_{ji}^{\mathrm{tr}}, D_i^{\mathrm{tr}})}{\sum_{i=1}^{L_{\mathrm{tr}}} \delta(E_{ji}^{\mathrm{tr}} = e_j) f_l(C_{ji}^{\mathrm{tr}}, D_i^{\mathrm{tr}})}. \tag{97}$$

W oryginalnym programie zamiast estymacji najbardziej wiarogodnej (97) stosuje pewną prostą interpolację liniową [21].

Przy ustalonych (wyuczonych) $S(e_j)$, $p(c_j|e_j, l)$, $p(c_j|e_j, d)$ i $p(t|a, d)$, optymalne tagowanie $\hat{T}$ dla danego ciągu tokenów W wyznacza się w oparciu o algorytm Viterbiego opisany podrozdziale 1.6. Uczenie się tagera przebiega w oparciu o algorytm uczenia się sterowanego liczbą błędów (error-driven learning) przypominający algorytmy z diagramów 3 i 4. Zbiory $S(a_j)$ inicjalizowane są jako zbiory puste. Następnie w kolejnych iteracjach powiększa się $S(a_j)$ o takie funkcje f_i, których dołączenie skutkuje największym spadkiem liczby błędów na danych treningowych. Powiększanie $S(a_j)$ przerywa się, gdy maksymalny spadek liczby błędów jest mniejszy niż pewna heurystyczna stała $c > 0$ ($c \approx 2$).

[12] Heurystycznie, w pracy [21] przyjęto, że zbiór funkcji będących elementami ciągu $(f_l)_{l=1}^k$ jest zbiorem funkcji postaci

$$\begin{aligned} f_l(C_{ji}, D_i) &= \delta(C_{ji} = c_j)\delta(\phi_1(D_i)), \\ &\vdots \\ f_l(C_{ji}, D_i) &= \delta(C_{ji} = c_j)\delta(\phi_1(D_i))...\delta(\phi_m(D_i)), \end{aligned}$$

gdzie $m = 3$, $c_j \in \mathcal{C}_j$ zaś $\phi_1, ..., \phi_m \in \Phi$. Φ jest zbiorem warunków ϕ trzech postaci

$$\begin{aligned} \phi(D_i) &\iff \delta(W_{i+n} = w), \quad n \in \{-2, ..., 2\}, \\ \phi(D_i) &\iff \delta(E_{j(i+n)} = e_j), \quad n \in \{-2, ..., 2\}, \\ \phi(D_i) &\iff \delta(C_{j(i+n)} = c_j), \quad n \in \{-2, -1\}. \end{aligned}$$

Przyjęto także, że funkcje f_l dodawane są do $S(e_j)$ wyłącznie w „paczkach" (batches) zawierających wszystkie funkcje o identycznej zależności od kontekstu D_i. Znaczy to, że warunek $f_l(C_{ji}, D_i) = \delta(C_{ji} = c_j)\delta(\phi_1(D_i))...\delta(\phi_k(D_i)) \in S(e_j)$ zachodzi dla pewnego $c_j \in \mathcal{C}_j$ wtedy i tylko wtedy, gdy zachodzi dla każdego $c_j \in \mathcal{C}_j$.

Dla języka angielskiego sumaryczna liczba paczek we wszystkich $S(e_j)$ w wyuczonym dezambiguatorze wynosi około 1000, dla czeskiego — 2500, dla rumuńskiego — 3000, dla słoweńskiego — 3500 [19].

2.4 Zróżnicowanie akuratności tagerów morfosyntaktycznych

Współcześnie używane tagery morfosyntaktyczne wykorzystują różne metody spośród opisanych w rozdziale 1. Tagery te trenowane na dużych zestawach danych anotowanych osiągają maksymalną akuratność rzędu 96.6% dla języka angielskiego [34] i 93.8% dla języka czeskiego [20]. [13] Zaobserwować można wyraźne zróżnicowanie akuratności tagerów, które ma wielorakie przyczyny. W szczególności akuratność wytrenowanego tagera morfosyntaktycznego silnie zależy od:

- języka naturalnego, rozmiaru i złożoności kontekstu wymaganego do prawidłowej dezambiguacji,
- rozmiaru tagsetu,
- rozmiaru danych treningowych,
- klasy tagera, tzn.
  - metody uczenia się i tagowania,
  - metody wygładzania estymatorów prawdopodbieństw,
  - doboru przestrzeni funkcji charakterystycznych i transformacji,
  - szczegółów obsługi analizy morfologicznej,
- tego, czy akuratność jest liczona
  - tylko dla tokenów, które pojawiły się/nie pojawiły się w danych treningowych,
  - tylko dla tokenów, których analiza morfologiczna zawiera więcej niż jeden tag morfosyntaktyczny,
  - tylko dla wybranych typów tokenów (np. niebędących znakami interpunkcyjnymi),
  - dla wszystkich tokenów łącznie.

Nie jest znana żadna metoda ścisłego przewidywania akuratności w zależności od wymienionych czynników, jakkolwiek można zaobserwować pewne prawidłowości.

Zależność akuratności tagera od języka naturalnego, do którego jest on przeznaczony, sprowadza się do zależności akuratności od rozmiaru i złożoności kontekstu wymaganego do prawidłowej dezambiguacji. Głównym problemem w opisie teoretycznym i przetwarzaniu języka naturalnego jest występowanie w nim bardzo dużej liczby zależności bardzo dalekiego zasięgu. Są to zależności zarówno formalne jak i statystyczne. Ważnym wynikiem w lingwistyce komputerowej było odkrycie, że do prawidłowego tagowania morfosyntaktycznego w przeważającej większości wypadków wystarcza modelowanie języka jako procesu stochastycznego z krótką pamięcią. Tak jest w przypadku języka angielskiego i innych języków analitycznych. W przypadku języków słowiańskich poważny problem stwarzają ujednoznacznianie przypadka oraz rodzaju. Ujednoznacznianie tych atrybutów (zwłaszcza dla homonimicznych form biernika) wyraźnie wymaga znacznie dłuższego kontekstu, przy czym minimalna długość tego kontekstu bywa bardzo zmienna. Nie jest dotychczas znana skuteczna metoda radzenia sobie z tym problemem. W rezultacie dla języka czeskiego tager Hajiča osiąga ogólny ułamek błędów 6.2%, w tym ułamek błędów dla atrybutu „przypadek" wynosi 5.0%, a dla atrybutu „rodzaj" — 2.0%. Przy ograniczeniu się wyłącznie do zwracania atrybutu „część mowy" ułamek błędów zmniejsza się do 0.9% [20] (dla pełnego tagsetu angielskiego najlepszy wynik [4] wynosi 3.3%). Dla języka słoweńskiego wyniki są podobne (patrz tabela 2). Poszczególne atrybuty dla tych dwóch języków mogą być trochę niewspółmierne. Warto zauważyć, że trudność ujednoznaczniania przypadka i rodzaju jest specjalnością języków słowiańskich (np. polski, rosyjski, czeski, słoweński). W językach ugrofińskich (np. węgierski, fiński, estoński), także o bogatej fleksji, kategoria rodzaju nie występuje, a końcówki fleksyjne przypadków są bardziej jednoznaczne. W rezultacie trudność tagowania języków ugrofińskich jest porównywalna z trudnością tagowania analitycznych języków inodeuropejskich (patrz tabela 1).

Zależność akuratności tagerów od rozmiaru tagsetu dla tego samego języka sprowadza się do obserwacji, że im liczniejszy jest tagset, tym akuratność jest mniejsza. W zależności tej można się dopatrzeć wpływu dwóch czynników. Po pierwsze, ujednoznacznianie niektórych nowych typów niejednoznaczności tagów może wymagać dłuższego kontekstu. Po drugie, przy zachowaniu tej samej długości kontekstu, tagery probabilistyczne mogą potrzebować większej ilości danych treningowych, aby nauczyć się ujednoznaczniać większą liczbę typów niejednoznaczności (zakładając, że na nauczenie się ujednoznaczniania każdego typu niejednoznaczności potrzeba pewnej minimalnej ilości przykładów). O wpływie pierwszego czynnika pisaliśmy w poprzednim akapicie. Wpływ drugiego czynnika w oddzieleniu od pierwszego prawdopodobnie najbezpieczniej jest ocenić rozpatrując język, w którym nie pojawia się problem dalekokontekstowego ujednoznaczniania przypadka. Odpowiednie wyniki dla

[13] Przypominamy, że akuratność $\mathrm{Acc}(A)$ i ułamek błędów $\mathrm{Err}(A)$ wiąże wzór $\mathrm{Acc}(A) = 1 - \mathrm{Err}(A)$.

Tabela 2: Ułamki błędów dla niektórych atrybutów w językach słowiańskich [20, 16].

język	czeski	słoweński
tager	tager Hajiča	tager TnT
dane treningowe	160 000 tokenów	81 805 tokenów
ułamek błędów		
ogółem	6.2%	10.8%
dla atrybutu „przypadek”	5.0%	6.9%
dla atrybutu „rodzaj”	2.0%	2.4%
dla atrybutu „część mowy”	0.9%	3.4%

Tabela 3: Ułamki błędów dla języka szwedzkiego i różnych rozmiarów tagsetu.

Dane odczytane z dokładnością 0.25% z wykresu w pracy [30].				
	tager			
ułamek błędów	IGTree	MXPOST	tager Brilla	TnT
139 tagów	10.75%	8.75%	11.00%	**6.50%**
44 tagi	8.75%	7.00%	8.00%	**5.75%**
26 tagów	8.25%	6.25%	6.75%	**5.00%**

języka szwedzkiego, który spełnia ten warunek cytujemy w tabeli 3. Dane pochodzą z pracy [30]. Porównywane były cztery tagery: oparty na pamięci IGTree (implementacja własna), oparty na modelowaniu maksimum entropii MXPOST [34], oparty na transformacjach tager Brilla [5] i tager trigramowy TnT [4]. Przy zwiększaniu liczby tagów z 26 do 139 (44), tzn. przy 5.34-krotnym (1.69-krotnym) powiększeniu tagsetu obserwuje się n-krotny wzrost ułamka błędów, gdzie n wynosi 1.30 (1.06) dla IGTree, 1.40 (1.12) dla MXPOST, 1.62 (1.18) dla tagera Brilla, 1.30 (1.15) dla TnT. Zachowanie najbardziej odporne na zwiększanie tagsetu wykazują tagery oparte na ukrytych łańcuchach Markowa i pamięci, nieco gorsze są tagery modelujące maksimum entropii, zaś najgorzej skaluje się tager Brilla.

Dane podane w pracy [30] dotyczą tagsetów 10-krotnie mniejszych od stosowanych dla języków słowiańskich. W tabeli 4 zebraliśmy dane odnośnie akuratności kilku podstawowych klas tagerów dla czterech języków (wraz z odnośnikami do źródeł). Widać, że rezultaty dla szwedzkiego przy zbliżonych rozmiarem danych treningowych są przeciętnie dwa razy gorsze niż dla angielskiego dla wszystkich tagerów oprócz tagera trigramowego (TnT). Dzieje się tak po części dlatego, że wszystkie testowane tagery z wyjątkiem TnT przeprowadzają heurystyczną analizę morfologiczną, biorąc pod uwagę tylko 3-4 ostatnie litery słowa (TnT bierze pod uwagę 10 ostatnich liter). Jest to wystarczające dla angielskiego, pod który to język tagery te były implicite pisane, lecz niewystarcza do wychwycenia regularności dla dłuższych zakończeń słów w języku szwedzkim [30]. Dla słoweńskiego, wyniki tagerów (zbadanych tylko na niewielkim zestawie danych) są gorsze od angielskich około cztery razy. W przypadku angielskiego, szwedzkiego a także niderlandzkiego [11], najlepszy okazuje się być tager najprostszy, czyli tager trigramowy TnT [4].

Autorzy czescy badając różne metody tagowania, w tym na zredukowanych tagsetach ([21], [20]), doszli do wniosku, że dla języka czeskiego oraz słoweńskiego najlepiej sprawdza się dezambiguator Hajiča. Dezambiguator ten jest modyfikacją tagerów opartych na modelowaniu maksimum entropii dostosowaną do współpracy z dużym tagsetem pozycyjnym oraz z zewnętrznym analizatorem morfologicznym. Dla języka czeskiego zbadane modyfikacje tagerów Brilla i trigamowego są wyraźnie gorsze. Dla języka słoweńskiego używane były powszechnie używane wersje tych tagerów, a zróżnicowanie akuratności jest zdecydowanie mniejsze. Warto zauważyć, że dla słoweńskiego prosty tager TnT jest drugi co do jakości. Zadziwiająco znacznie gorsze wyniki tagera trigramowego dla czeskiego mogą wynikać z gorszej procedury interpolacji [4] bądź ze znacznie większych rozmiarów tagsetu.

Bardzo duży wpływ na akuratność tagera przy ustalonej jego klasie ma jakość analizy morfologicznej (symulowanej przez tager bądź opartej na zewnętrznym analizatorze), co omawialiśmy w podrozdziale 2.1. Podob-

Tabela 4: Najlepsze ułamki błędów dla niektórych języków i klas tagerów.

Err(A) (L_{tr})	tager oparty na					tager Hajiča
	pamięci	maksimum entropii	transformacjach	skreśleniach	trigramach	
angielski	3.6% [13] (2 000 000)	3.4% [34] (962 687)	3.4% [5] (950 000)	4.0% [6] (350 000)	**3.3%** [4] (1 000 000)	3.6% [19] (99 903)
szwedzki	6.0% [30] (1 000 000)	5.9% [30] (500 000)	8.8% [30] (500 000)	—	**4.0%** [30] (1 000 000)	—
czeski	—	—	20.25% [20] (37 892)	—	18.86% [20] (621 015)	**6.20%** [21] (160 000)
słoweński	13.58% [16] (81 805)	13.64% [16] (81 805)	14.05% [16] (81 805)	—	10.78% [16] (81 805)	**9.00%** [19] (94 457)
Err(A) – ułamek błędów, L_{tr} – liczba tokenów w danych treningowych						

Tabela 5: Ułamki błędów dla języka szwedzkiego i różnych rozmiarów danych treningowych [30].

ułamek błędów	tager			
	IGTree	MXPOST	tager Brilla	TnT
1000 tokenów	37.1%	46.6%	**17.5%**	32.0%
10 000 tokenów	20.7%	21.9%	15.7%	**14.9%**
100 000 tokenów	11.1%	9.3%	9.9%	**7.7%**
500 000 tokenów	7.8%	5.9%	8.8%	**5.1%**
1 000 000 tokenów	6.0%	—	—	**4.0%**

nie oddziałującym czynnikiem jest wybór rodzaju interpolacji liniowej dla tagerów n-gramowych oraz dobór przestrzeni transformacji, warunków i funkcji charakterystycznych dla tagerów przeszukujących te przestrzenie (tagery oparte na transformacjach, regułach skreślania i modelowaniu maksimum entropii, tager Hajiča). W publicznie dostępnych implementacjach tagerów (nie wyuczonych do przetwarzania konkretnego języka), wymienione wybory często są zaszyte w kodzie źródłowym. Rzadziej, jak np. w sytemie μ-TBL [28], użytkownikowi zostawiona jest możliwość wygodnego modyfikowania przestrzeni poszukiwań. W szczególności można mniemać, że niektóre klasyczne wolnodostępne implementacje tagerów do samodzielnego wytrenowania (tager Brilla, MXPOST Rathnaparkhiego, TiMBL) są implicite optymalizowane do analizy morfologicznej i dezambiguacji morfosyntaktycznej dla języka angielskiego. Efekty nieproporcjonalnego pogorszenia akuratności tych tagerów w zastosowaniu do innych języków europejskich były raportowane w pracach [30] i [19].

Interesująco przedstawia się zależność akuratności wytrenowanego tagera od rozmiaru danych treningowych dla różnych klas tagerów. Odpowiednie dane (dla języka szwedzkiego) zaczerpnęliśmy z pracy [30] i przytoczyliśmy w tabeli 5. Widać, że dla tagera Brilla, który jest najlepszy dla mikroskopijnych zestawów danych (1000 tokenów), następuje wysycenie przy zwiększaniu ilości danych treningowych, podczas gdy tagery probabilistyczne (łącznie z opartymi na pamięci) systematycznie poprawiają swoją akuratność i prześcigają tager Brilla. Przypomnijmy, że również tager Brilla był najgorszy pod względem odporności na powiększanie rozmiarów tagsetu.

Ułamki błędów można zliczać nie tylko biorąc pod uwagę wszystkie tokeny (i atrybuty tagów pozycyjnych), ale także dla różnych typów tokenów i atrybutów tagów z osobna. Pewien chaos spowodowany jest tym, że akuratność dla wszystkich tokenów można liczyć, uwzględniając znaki interpunkcyjne jako tokeny bądź nie.[14] Niektórzy autorzy podają statystyki dla obu przypadków — inni nie. Niektórzy nie piszą, jaki przypadek uwzględniają.

[14]W języku angielskim i francuskim problem ten jest mniej dotkliwy niż w języku niemieckim i słowiańskich, w których znacznie częściej używa się przecinka.

Tabela 6: Ułamki błędów dla tokenów znanych i nieznanych w języku szwedzkim i słoweńskim [30, 16].

	tager			
	IGTree	MXPOST	tager Brilla	TnT
język szwedzki	500 000 tokenów treningowych			
$\mathrm{Err}(A)_{\mathrm{known}}$	5.9%	4.7%	4.3%	3.7%
$\mathrm{Err}(A)_{\mathrm{unk}}$	28.2%	18.8%	32.5%	14.1%
$\mathrm{Err}(A)_{\mathrm{unk}}/\mathrm{Err}(A)_{\mathrm{known}}$	4.8	4.0	7.6	3.8
język słoweński	81 805 tokenów treningowych			
$\mathrm{Err}(A)_{\mathrm{known}}$	7.1%	8.4%	6.4%	4.9%
$\mathrm{Err}(A)_{\mathrm{unk}}$	54.6%	44.1%	55.5%	45.1%
$\mathrm{Err}(A)_{\mathrm{unk}}/\mathrm{Err}(A)_{\mathrm{known}}$	7.7	5.3	8.7	9.2
$\mathrm{Err}(A)_{\mathrm{known}}$ – ułamek błędów dla tokenów znanych, $\mathrm{Err}(A)_{\mathrm{unk}}$ – ułamek błędów dla tokenów nieznanych				

Interesujące bywa porównanie akuratności dla wszystkich tokenów z akuratnością dla tokenów, które pojawiły się w danych treningowych (tokenów znanych, known tokens) i tych, które nie pojawiły się tamże (tokenów nieznaych, unknown tokens). Dane takie pozwalają ocenić jakość analizy morfologicznej stosowanej w tagerze (zwłaszcza heurystycznej).[15] W tabeli 6 zestawiliśmy odpowiednie dane dla czterech popularnych tagerów i dwóch języków: szwedzkiego i słoweńskiego. W języku szwedzkim przy danych treningowych wielkości 500 000 tokenów, nieznanych tokenów w danych testowych było 8.1% (15.6% dla 100 000, 20.8% dla 50 000) [30]. W języku słoweńskim przy danych treningowych 81 000 nieznanych testowych było 17.1% [16]. Widać, że znów tager Brilla jest najgorszy, jeżeli chodzi o zgadywanie tagów dla tokenów nieznanych, choć całkiem dobrze radzi on sobie z tokenami znanymi.

Z punktu widzenia skuteczności dezambiguacji, od akuratności tagera na wszystkich tokenach (które można liczyć w różny sposób) bardziej interesująca jest akuratność wyłącznie na tokenach niejednoznacznych (tokenach o więcej niż jednym tagu morfosyntaktycznym zwracanym przez analizator morfologiczny bądź o więcej niż jednym tagu pojawiającym się przy wystąpieniach odpowiedniej słowoformy w danych testowych). Jeżeli akuratność $\mathrm{Acc}(A)_{\mathrm{ambig}}$ dla tokenów niejednoznacznych nie jest znacznie większa od średniej odwrotności liczby tagów morfologicznych dla tokenów niejednoznacznych $\langle 1/|G(\cdot)|\rangle_{\mathrm{ambig}}$, można wnioskować, że dezambiguator ujednoznaczniając tagi zwracane przez analizator nie kieruje się kontekstem ani częstością tagów, lecz wybiera je na chybił trafił. Wielkości $\mathrm{Acc}(A)_{\mathrm{ambig}}$, $\langle 1/|G(\cdot)|\rangle_{\mathrm{ambig}}$ definiujemy odpowiednio jako

$$\mathrm{Acc}(A)_{\mathrm{ambig}} = \frac{\sum_{i=1}^{L} \delta(|G(W_i)|>1)\delta(M_A(W)_i = M(W)_i)}{\sum_{i=1}^{L} \delta(|G(W_i)|>1)}, \tag{98}$$

$$\langle 1/|G(\cdot)|\rangle_{\mathrm{ambig}} = \frac{\sum_{i=1}^{L} \delta(|G(W_i)|>1)/|G(W_i)|}{\sum_{i=1}^{L} \delta(|G(W_i)|>1)} \leqslant \frac{1}{2}. \tag{99}$$

Niestety nie udało nam się znaleźć źródeł w literaturze, na podstawie których można by dostatecznie dokładnie wyliczyć i porównać te wielkości dla znanych implementacji tagerów (prace, jak np. [7], podają co najwyżej łączną akuratność dla tokenów nieznanych i o więcej niż jednym tagu).

2.5 Proste metody lepszego wykorzystania tagerów

Sporo uwagi w literaturze statystycznego przetwarzania języka naturalnego w ostatnich latach poświęca się prostym metodom poprawienia akuratności tagowania poprzez kombinowanie wyników kilku różnych tagerów bądź poprzez optymalne łączenie anotacji automatycznej i ręcznej (przez ekspertów).

[15] Warto zauważyć, że podział na tokeny znane i nieznane jest bardzo zgrubny. Zgodnie z prawem Zipfa, przeciętnie około połowa wszytkich typów tokenów pojawiających się w tekście pojawia się tylko raz [15]. Dla takich słowoform pełna analiza morfologiczna raczej nie może być przybliżona wyłącznie przez zbiór tagów pojawiających się w danych, jak dla słowoform częstych.

Dla pary tagerów A i B definiuje się ułamek komplementarności $\mathrm{comp}(B|A)$ jako akuratność tagera B dla tokenów błędnie tagowanych przez tager A. Jawnie

$$\mathrm{comp}(B|A) = \frac{\sum_{i=1}^{L} \delta(M_A(W)_i \neq M(W)_i)\delta(M_B(W)_i = M(W)_i)}{\sum_{i=1}^{L} \delta(M_A(W)_i \neq M(W)_i)}, \tag{100}$$

przy czym $\mathrm{comp}(B|A) \not\equiv \mathrm{comp}(A|B)$. Automatyczne wyznaczanie tagu w opraciu o sugestie kilku różnych tagerów A_1,...,A_k ma sens tylko wtedy, gdy błędy popełniane przez te tagery nie pokrywają się, tzn. gdy $\mathrm{comp}(A_i|A_j) > 0$ dla $i \neq j$.

Dla języka angielskiego ułamki komplementarności dla tagerów trigramowego, MXPOST (opartego na maksimum entropii) i tagera Brilla wynoszą przeciętnie $\mathrm{comp}(A_i|A_j) \approx 30\%$ dla $i \neq j$ [7]. Stosując prostą procedurę głosowania dla tych trzech tagerów, autorzy pracy [7] zmniejszyli ułamek błędów z 3.2% (ich wynik dla najlepszego pojedynczego tagera, tzn. MXPOST) do 3.0% (na korpusie Wall Street Journal). Procedura prostego głosowania polega na tym, że dla ustalonego tokenu wybierany jest ten tag, który jest zwracany przez największą liczbę tagerów. Jeżeli tagów takich jest kilka, wybiera się ten tag spośród nich, który jest zwracany przez tager o przeciętnie największej akuratności. Dobrym pomysłem jest potraktowanie procedury tagowania w oparciu o sugestie kilku tagerów A_1,...,A_k jako zwykłego problemu automatycznego tagowania w przypadku, gdy ciąg tokenów ma postać $W' = (W'_1, ..., W'_L)$, gdzie $W'_i = (M_{A_1}(W)_i, ..., M_{A_k}(W)_i)$, zaś tekstem anotowanym jest $(W', M(W))$. Po zastosowaniu do tego problemu uczenia się opartego na pamięci (patrz podrozdział 1.13), autorom pracy [7] udało się zmniejszyć ułamek błędów do 2.8%. Podobne eksperymenty na tym samym korpusie WSJ przeprowadził autor strony [36]. Stosując proste głosowanie dla własnej implementacji ICOPOST trzech tagerów (trigramowy, oparty na transformacjach i oparty na maksimum entropii) plus TnT [4], zmniejszył ułamek błędów z 3.3% dla najlepszego w pojedynkę TnT do 2.8%.

Innym ciekawym pomysłem jest połączenie anotowania automatycznego z anotowaniem ręcznym tylko tych tokenów, którym tager automatyczny nie przypisuje najlepszego tagu z dostatecznie dużym prawdopodobieństwem [4]. Metoda ta pozwala osiągnąć akuratność tagowania rzędu 99.5% (dla języka angielskiego i tagera TnT). Ponieważ wymaga ona niewielkiej współpracy anotatora-eksperta, metoda może być praktycznie stosowana jedynie przy anotacji dużych, statycznych korpusów tekstów. Jej zaletą jest to, że pozwala oszczędzić na pracy ekspertów, którzy zamiast anotować cały korpus, anotują tylko około 20% najtrudniejszych tokenów.[16] Metodę anotacji półautomatycznej można stosować wyłącznie dla tagerów probabilistycznych, które przypisują każdemu tagowi $T_i \in \mathcal{T}$ przy zadanym kontekście $D_i \in \mathcal{D}$ prawdopodobieństwo $p(T_i|D_i)$. Wówczas definiuje się

$$\hat{T}_i^{(1)} = \operatorname*{arg\,max}_{T_i \in \mathcal{T}} p(T_i|D_i), \quad \hat{T}_i^{(2)} = \operatorname*{arg\,max}_{T_i \in \mathcal{T} \setminus \left\{\hat{T}_i^{(1)}\right\}} p(T_i|D_i), \quad \Delta_i = \frac{p(\hat{T}_i^{(1)}|D_i)}{p(\hat{T}_i^{(2)}|D_i)}. \tag{101}$$

Jeżeli $\Delta_i \geqslant \theta$, gdzie $\theta \geqslant 1$ jest pewnym przyjętym progiem, tokenowi W_i przypisywany jest tag $\hat{T}_i^{(1)}$, w przeciwnym przypadku decyzja zostawiana jest anotatorowi. Przykładowe dane odnośnie zależności między θ a akuratnością podaje tabela 7.

Tabela 7: Zależność między wartością progu θ a akuratnością tagera TnT dla języka angielskiego [4].

θ	1	10	100	1000	10 000
tokeny anotowane automatycznie					
ułamek błędów dla tokenów o $\Delta_i \geqslant \theta$	3.4%	1.0%	0.5%	0.5%	0.6%
tokeny anotowane ręcznie					
ułamek błędów dla tokenów o $\Delta_i < \theta$	—	31.1%	17.4%	11.2%	8.4%
ułamek tokenów o $\Delta_i < \theta$	0%	7.8%	16.5%	26.2%	35.5%

2.6 Dostępne oprogramowanie

W tabeli 8 zebraliśmy dane techniczne dotyczące popularnych i niektórych mniej znanych implementacji tagerów.

[16] Można zresztą przypuszczać, że sami eksperci zgadzają się między sobą rzadziej niż w rzeczonych 99.5% przypadków.

Tabela 8: Zestawienie dostępnych tagerów morfosyntaktycznych.

nazwa (autor)	klasa tagera	analiza morfologiczna	przetwarzanie tagsetu pozycyjnego	samodzielny wybór przekształceń, funkcji, cech	czas uczenia się	dostęp do wersji nietrenowanej	ładowanie z WWW (artykuł)	cena niekomercyjna (licencja)	język
TnT (Brants)	trigramy	heurystyka (10 liter)	nie	—	kilka sekund	TAK	nie ([4])	gratis (≠GNU)	C
ICOPOST (Schröder)	trigramy	???	nie	—	???	TAK	TAK ([36])	gratis (GNU)	C
MXPOST (Rathnaparkhi)	maksim. entropii	heurystyka (pierwsze i ostatnie 4 litery)	nie	nie	około 1 dnia	TAK	TAK ([34])	gratis (≠GNU)	C, Java
ICOPOST (Schröder)	maksim. entropii	???	nie	nie	???	TAK	TAK ([36])	gratis (GNU)	C
tager Hajiča (Hajič)	naiwny Bayes	oddzielna (dezambiguator)	TAK	nie	około 1 dnia	nie	nie ([21])	gratis (≠GNU)	C
TiMBL (Daelemans i in.)	pamięć IGTree	oddzielna, heurystyka	TAK (?)	TAK	kilka minut	TAK	TAK ([14])	gratis (≠GNU)	C
tager Brilla (Brill)	transformacje	heurystyka (pierwsze i ostatnie 4 litery)	nie	nie	kilka dni	TAK	TAK ([5])	gratis (≠GNU)	C
FastTBL (Florian & Ngai)	transformacje	???	TAK	???	kilkanaście minut	TAK	??? ([18])	gratis (???)	C
μ-TBL (Lager)	transformacje	odzielna, heurystyka	TAK (?)	TAK	około 1 godz.	TAK	TAK ([28])	gratis (brak)	Prolog
ICOPOST (Schröder)	transformacje	???	nie	nie	???	TAK	TAK ([36])	gratis (GNU)	C

Linkografia

- `http://www-2.cs.cmu.edu/~aberger/maxent.html`
  Materiały dydaktyczne A. Bergera dotyczące modelowania maksimum entropii.
- `http://ufal.mff.cuni.cz/~hladka/tagging_page.html`
  Zbiór artykułów poświęconych tagowaniu języka czeskiego.
- `http://nl.ijs.si/ME/bib/mte-bib/`
  Bibliografia projektu MULTEXT-East.
- `http://www.coli.uni-sb.de/~thorsten/tnt/`
  Witryna tagera trigramowego TnT T. Brantsa.
- `http://www.cis.upenn.edu/~adwait/statnlp.html`, `ftp://ftp.cis.upenn.edu/pub/adwait/jmx`
  Witryny tagera MXPOST A. Ratnaparkhiego, opartego na modelowaniu maksimum entropii.
- `http://www.cs.jhu.edu/~brill/code.html`
  Witryna tagera E. Brilla, opartego na transformacjach.
- `http://nlp.cs.jhu.edu/~rflorian/fntbl/`

Witryna tagera FastTBL R. Floriana, opartego na transformacjach.

- `http://stp.ling.uu.se/~lager/mutbl.html`
  Witryna tagera μ-TBL T. Lagera, opartego na transformacjach.
- `http://ilk.kub.nl/software`
  Witryna klasyfikatora TiMBL (Tilburg Memory Based Learner), opartego na pamięci.
- `http://nats-www.informatik.uni-hamburg.de/~ingo/icopost/`
  Witryna systemu ICOPOST I. Schrödera, zawierającego komplet trzech tagerów: oparty na modelowaniu maksimum entropii, trigramowy i oparty na transformacjach.
- `http://www-nlp.stanford.edu/links/statnlp.html`
  Katalog dostępnego oprogramowania: Statistical NLP/corpus-based computational linguistic resources.
- `http://nlp.postech.ac.kr/~jwcha/tools.html`
  Katalog dostępnego oprogramowania: Software tools for NLP.
- `http://www.cis.upenn.edu/~adwait/penntools.html`
  Katalog dostępnego oprogramowania: PennTools.

Literatura

[1] S. Armstrong, K. Church, et al. (red.). *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Publishers, 1999.

[2] A. L. Berger. The improved iterative scaling algorithm: A gentle introduction. Carnegie Mellon University, 1997.

[3] A. L. Berger, S. A. D. Pietra, V. J. D. Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:39, 1996.

[4] T. Brants. TnT - a statistical part-of-speech tagger. W: *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*. Seattle, 2000.

[5] E. Brill. Transformation-based error-driven learning and natural language processing: A case study of part-of-speech tagging. *Computational Linguistics*, 21:543, 1995.

[6] E. Brill. Unsupervised learning of disambiguation rules for part of speech tagging. W: *Proceedings of the Third Workshop on Very Large Corpora*. Association for Computational Linguistics, 1995.

[7] E. Brill, J. Wu. Classifier combination for improved lexical disambiguation. W: *Proceedings of the Joint Conference COLING/ACL-98*. Montréal, 1998.

[8] S. F. Chen, J. Goodman. An empirical study of smoothing techniques for language modeling. W: *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*. Morgan Kaufmann Publishers, 1996.

[9] W. Daelemans. Abstraction considered harmful: Lazy learning of language processing. W: J. van den Herik, T. Weijters (red.), *Benelearn-96. Proceedings of the 6th Belgian-Dutch Conference on Machine Learning*. 1996.

[10] W. Daelemans, A. van den Bosch, J. Zavrel. Forgetting exceptions is harmful in language learning. *Machine Learning*, 34:11, 1999.

[11] W. Daelemans, J. Zavrel. Bootstrapping a tagged corpus through combination of existing heterogeneous taggers. W: *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC)*. Ateny, 1993.

[12] W. Daelemans, J. Zavrel. Memory-based learning: Using similarity for smoothing. W: *Proceedings of 35th Annual Meeting of the ACL*. Madryt, 1997.

[13] W. Daelemans, J. Zavrel, P. Berck, S. Gills. MBT: A memory-based part of speech tagger-generator. W: *Proceedings of the Fourth Workshop on Very Large Corpora*. Kopenhaga, 1996.

[14] W. Daelemans, J. Zavrel, K. van der Sloot, A. van den Bosch. TiMBL: Tilburg Memory Based Learner, version 4.0. Reference Guide. ILK Technical Report 01-04. `http://ilk.kub.nl/downloads/pub/papers/ilk0104.ps.gz`, 2001.

[15] Ł. Dębowski. Zipf's law: What and why? `http://www.ipipan.waw.pl/~ldebowsk`, 2000.

[16] S. Džeroski, T. Erjavec, J. Zavrel. Morphosyntactic tagging of Slovene: Evaluating PoS taggers and tagsets. Department for Intelligent Systems, Jozef Stefan Institute, Ljubljana, 1999.

[17] T. Erjavec, M. Monachini, (red.). *Specifications and Notation for Lexicon Encoding, COP Project 106 Multext-East Work Package WP1 - Task 1.1. Deliverable D1.1 F. Final Report.* `http://nl.ijs.si/ME/CD/docs/mte-d11f/`, 1997.

[18] R. Florian, G. Ngai. Multidimensional transformation-based learning. `http://xxx.lanl.gov/abs/cs.CL/0107021`, 2001.

[19] J. Hajič. Morphological tagging: Data vs. dictionaries. W: *Proceedings of ANLP-NAACL Conference.* Seattle, 2000.

[20] J. Hajič, B. Hladká. Probabilistic and rule-based tagging of an inflective language — a comparison. W: *Proceedings of the 5th Conference on Applied Natural Language Processing.* Washington, 1997.

[21] J. Hajič, B. Hladká. Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. W: *Proceedings of COLING-ACL Conference.* Montréal, 1998.

[22] E. Hajnicz, A. Kupść. Przegląd analizatorów morfologicznych dla języka polskiego. Prace IPI PAN. Instytut Podstaw Informatyki PAN, (w przygotowaniu).

[23] J. Hammerton, E. F. T. K. Sang. Combining a self-organising map with memory-based learning. `http://xxx.lanl.gov/abs/cs.CL/0107018`, 2001.

[24] E. T. Jaynes. *Probability Theory—The Logic of Science.* `http://bayes.wustl.edu/etj/node2.html`.

[25] F. Jelinek. *Statistical Methods for Speech Recognition.* The MIT Press, 1997.

[26] D. Jurafsky, J. H. Martin. *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall, 2000.

[27] J. Kocek, M. Kopřivová, K. Kučera (red.). *Český národní korpus. Úvod a příručka uživatele.* Ústav Českého národního korpusu, Univerzita Karlova, 2000.

[28] T. Lager. The μ-TBL system: Logic programming tools for transformation-based learning. W: *Third International Workshop on Computational Natural Language Learning (CoNLL'99).* Bergen, 1999.

[29] C. D. Manning, H. Schütze. *Foundations of Statistical Natural Language Processing.* The MIT Press, 1999.

[30] B. Megyesi. Comparing data-driven learning algorithms for PoS tagging of Swedish. W: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2001).* Carnegie Mellon University, 2001.

[31] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[32] G. Ngai, R. Florian. Transformation-based learning in the fast lane. `http://xxx.lanl.gov/abs/cs.CL/0107020`, 2001.

[33] L. A. Ramshaw, M. P. Marcus. Exploring the nature of transformation-based learning. W: *The Balancing Act: Combining Symbolic and Statistical Approaches to Language.* The MIT Press, 1994.

[34] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. W: *Proceedings of the First Conference on Empirical Methods in Natural Language Processing (EMNLP-1996).* University of Pennsylvania, 1996.

[35] E. Roche, Y. Schabes. Deterministic part-of-speech tagging with finite state transducers. *Computational Linguistics*, 21:227, 1995.

[36] I. Schröder. ICOPOST — Ingo's collection of POS taggers. `http://nats-www.informatik.uni-hamburg.de/~ingo/icopost/`.

[37] M. Woliński, A. Przepiórkowski. Projekt sposobu morfosyntaktycznego anotowania korpusu języka polskiego. Prace IPI PAN. Instytut Podstaw Informatyki PAN, (w przygotowaniu).

Pracę zgłosił: Jacek Koronacki

Adres autora: Łukasz Dębowski
Instytut Podstaw Informatyki PAN
ul. Ordona 21, 01-237 Warszawa
e-mail: ldebowsk@ipipan.waw.pl

Symbole klasyfikacji rzeczowej: I.2.7, I.2.6

Nakład 140 egzemplarzy. Papier kserograficzny klasy III. Oddano do druku w listopadzie 2001.

Wydawnictwo IPI PAN. ISSN 0138-06-48.